

From Encoding to Decoding: Advancing Spiking Neural Network Technology for Improved Learning

A thesis submitted to Auckland University of Technology in
fulfilment of the requirements for the degree of
Doctor of Philosophy

Supervisors
Professor Ajit Narayanan
Dr Michael J Watts

June 2026

by
Akshay Raj Gollahalli
School of Engineering, Computers and Mathematical
Sciences

Abstract

Spiking Neural Networks (SNNs) process information through discrete spike events, yet most existing encoders focus solely on converting data into spikes without verifying whether the original signal can be faithfully recovered. Reconstruction matters for SNNs because spike encoding is inherently lossy: continuous-valued inputs must be discretised into binary events, and without a decoding pathway, there is no way to measure how much information is preserved by this conversion. If critical signal features are lost during encoding, the network cannot recover them regardless of its architecture or training. A reconstruction capability, therefore, separates encoding quality from network quality, revealing whether classification failures stem from a poor encoder or a poor learner. It also enables applications beyond classification, such as neuromorphic signal processing and sensory data transmission, where the encoded signal must be recovered rather than merely classified.

This thesis presents the Dynamic Neural Encoder and Decoder (DyNED), a unified encoding system that converts both static and dynamic data into spike trains while providing a matched decoding pathway for reconstruction. DyNED applies Fast Fourier Transform preprocessing followed by sigma-delta quantisation with iterative error feedback, enabling frequency-domain spike encoding of images and time-domain encoding of speech signals. A companion compressor, DyNED_c, combines run-length encoding with Huffman coding to losslessly compress the resulting binary spike trains.

DyNED is evaluated on four datasets - two conventional (CIFAR-10 natural images, Google Speech Commands) and two neuromorphic extensions (CIFAR10-DVS event streams, Spiking Speech Commands cochlear spikes) - using reconstruction metrics (SSI, SNR, R^2), compression ratio, and classification accuracy within convolutional and recurrent spiking architectures. Ablation studies across quantisation levels (2-512) show that spike density scales from 0.55 to 0.99 for images and 0.23 to 0.94 for speech, with reconstruction fidelity reaching 98-99%. Comparative experiments against uniform quantisation and mu-law companding baselines demonstrate that DyNED's error feedback mechanism produces up to 74% denser spike trains at low quantisation levels while maintaining superior reconstruction quality. DyNED_c achieves compression ratios of 0.904 (9.6% space saved) for images at 256 quantisation levels and 0.007 (99.3% space saved) for speech, matching or exceeding standard compression methods. Classification

accuracy reaches 88.07% on CIFAR-10 and 92.79% on Speech Commands, exceeding a matched non-spiking CNN baseline operating on identical raw spectral features (76.59%) by approximately 16 percentage points. Furthermore, the framework achieves 76.3% and 72.96% on the neuromorphic CIFAR10-DVS and SSC extensions, respectively. This confirms that the same encoder generalises across both signal modalities and pre-spiked event streams.

These results establish that spike encoding and faithful reconstruction can coexist within a single framework, providing both the sparse representations that SNNs require and the quantifiable fidelity guarantees that principled encoder design demands. By making the encoder’s information capacity explicit and measurable, DyNED provides researchers with a principled way to attribute classification failures to encoding rather than to learning, rather than relying on estimation. This capability accelerates the design of SNN architectures across modalities. The framework opens direct applications beyond classification: bandwidth-constrained sensor-to-network transmission via DyNED_c compression, energy-efficient edge inference through sparse spike representations, and faithful signal recovery for neuromorphic processing pipelines where the encoded signal must later be decoded.

Table of Contents

Abstract	II
Glossary	XVII
Attestation of Authorship	XIX
Publications	XX
Acknowledgements	XXII
1 Introduction	24
1.1 Academic Motivation	25
1.1.1 Research Problem	26
1.2 Personal Motivation	27
1.3 Research Hypotheses and Goals	28
1.3.1 Goals	28
1.3.2 Research Questions	28
1.3.3 Hypotheses	29
1.3.3.1 Hypothesis 1	29
1.3.3.2 Hypothesis 2	30
1.3.3.3 Hypothesis 3	30
1.3.3.4 Hypothesis 4	30
1.3.3.5 Hypothesis 5	30
1.3.3.6 Hypothesis 6	30
1.3.3.7 Hypothesis 7	30
1.3.3.8 Hypothesis 8	31
1.3.4 Scope of Hypotheses	31
1.4 Evaluation Measures	31
1.5 Original Contribution of the Research	32
1.6 Structure of Thesis	33
1.7 Summary	35
2 Background	36
2.1 Biological Inspiration	37
2.2 Spike Encoding	37
2.2.1 Address Event Representation (AER) Encoder	37
2.2.2 Ben's Spiker Algorithm (BSA) Encoder	38
2.2.3 Population Rank Encoding	39
2.3 Spiking Neural Network (SNN) Models	40
2.3.1 Leaky Integrate and Fire	41
2.3.2 Adaptive Exponential Integrate and Fire	42
2.3.3 Izhikevich	43
2.3.4 Adaptive Leaky Integrate and Fire	44
2.3.5 Constrained Adaptive Leaky Integrate and Fire	45
2.4 SNN Learning Rules	46
2.4.1 Spike Timing Dependent Plasticity (STDP)	46
2.4.2 Surrogate Gradient Learning with Backpropagation (SGLBP)	47
2.5 SNN Supervised Learning	48
2.5.1 Spike Pattern Association Neuron (SPAN)	49
2.5.2 Dynamic Evolving Spiking Neural Network (deSNN)	49

2.5.3	Multi Layer Feed Forward SNN	50
2.5.4	NeuCube	51
2.5.5	Deep Learning in SNN	52
2.6	Optimisation	52
2.6.1	Decoupled Weight Decay Regularisation	53
2.6.2	Tree-structured Parzen Estimator (TPE)	53
2.7	Data Compression	55
2.7.1	Information Entropy and Source Coding	55
2.7.2	Run-Length Encoding (RLE)	56
2.7.3	Huffman Coding	56
2.7.4	Arithmetic Coding	56
2.7.5	Types of Data Compression	57
2.7.5.1	Lossless Compression	57
2.7.5.2	Lossy Compression	58
2.7.6	Applications in Modern Technology	58
2.7.7	Biological Relevance	58
2.8	Platforms	59
2.8.1	Raspberry Pi	59
2.8.2	General Purpose Computing on Graphical Processing Unit (GPGPU)	59
2.8.3	Neuromorphic Computing	60
2.9	Signal Processing	61
2.9.1	Analogue Signals	61
2.9.2	Digital Signals	63
2.9.3	Quantisation	64
2.9.3.1	Sigma-Delta Quantisation	65
2.9.4	Interpolation	66
2.9.4.1	Radial Basis Function (RBF) Interpolation	67
2.9.5	Analogue-to-Digital Converter (ADC)	68
2.9.6	Digital-to-Analogue Converter (DAC)	69
2.9.7	Nyquist-Shannon Sampling Theorem	70
2.9.7.1	Nyquist Frequency	71
2.9.7.2	Nyquist Rate	71
2.9.8	Anti-Aliasing Filter	72
2.9.9	Fast Fourier Transform (FFT)	72
2.9.10	Spectrogram	73
2.9.10.1	Mel-Spectrogram	74
2.9.11	Spectral Gate	75
2.9.12	Band-Pass Filter	75
2.10	Case Studies	76
2.10.1	CIFAR-10 Dataset	76
2.10.2	Speech Command Dataset	76
2.10.3	CIFAR10-DVS Dataset	76
2.10.4	Spiking Speech Commands (SSC) Dataset	77
2.11	Conclusion	77
3	DyNED: Dynamic Neural Encoder and Decoder	79
3.1	Introduction	79
3.2	Background and Theory	80
3.2.1	Importance of Encoders and Decoders	80
3.2.2	Challenges in Encoding for SNNs	81

3.3	Design Consideration	82
3.3.1	Criteria for Effective Encoding and Decoding	83
3.3.2	Adaptability, Efficiency, and Strategy Selection	83
3.4	Implementation Details	84
3.4.1	Introduction to the Encoder's Implementation	85
3.4.2	Development Environment and Tools	85
3.4.3	Modifications and Enhancements	87
3.4.3.1	Integration of Advanced Audio Processing Techniques	87
3.4.3.2	Data Handling and Device Management	88
3.4.4	Final Algorithm Design	88
3.4.5	Implementation Challenges and Solutions	92
3.4.6	Improvements Over Existing Encoders	93
3.5	Testing and Validation	94
3.5.1	Quantitative Metrics for Signal Reconstruction	94
3.5.2	Summary of Reconstruction Performance	94
3.5.3	Validation and Testing of SNN Models	95
3.6	Conclusion	96
4	DyNED_c: Dynamic Neural Encoder and Decoder Compressed	98
4.1	Introduction	98
4.2	Background and Theory	99
4.2.1	Importance of Data Compression in Neural Networks	100
4.3	Design Consideration	100
4.3.1	Criteria for Effective Compression	100
4.3.2	Challenges in Designing for SNN	102
4.4	Implementation Details	103
4.4.1	Development Environment and Tools	103
4.4.2	Modification and Enhancements	104
4.4.3	Final Algorithm Design	105
4.4.4	Implementation Challenges	109
4.4.5	Improvement over Existing Techniques	110
4.5	Testing and Validation	112
4.5.1	Quantitative Metrics for Signal Decompression and Testing	112
4.5.2	Summary of Compression Performance on Real Data	113
4.6	Conclusion	114
5	Case Study - CIFAR Dataset	116
5.1	Introduction	116
5.2	Relevance	117
5.3	Methodology	118
5.3.1	System Integration and Performance	119
5.3.1.1	Per-Sample Encoding and Compression Cost	120
5.3.2	Preprocessing	121
5.3.3	Spike Trains and Sigma-Delta Internals	124
5.3.3.1	Effect of Quantisation Levels on Image Encoding	126
5.3.3.2	DyNED vs Baseline Quantisation Methods	128
5.3.4	Compression	129
5.3.5	Classification	130
5.3.5.1	Architecture for CIFAR-10 Classification with Convolutional cAdLIF SNN and TET Loss	130

5.3.5.2 Architecture for CIFAR-10 Classification with DyNED_c	
Compression	134
5.4 Results	136
5.4.1 DyNED Encoding Quality	136
5.4.2 DyNED Classification Results	139
5.4.2.1 Per-Class Performance	140
5.4.2.2 Confusion Matrix	141
5.4.2.3 Network Dynamics	142
5.4.2.4 Per-Class Spike Patterns	144
5.4.2.5 Representation Quality	145
5.4.3 DyNED_c Classification Results	146
5.4.3.1 Per-Class Performance	147
5.4.3.2 Confusion Matrix	148
5.4.3.3 Network Dynamics	149
5.4.3.4 Representation Quality	150
5.4.3.5 DyNED_c Compression During Training	150
5.4.4 Comparison with Related Work	151
5.5 CIFAR10-DVS - Neuromorphic Extension	153
5.5.1 Dataset	153
5.5.2 DyNED Encoding Pipeline	153
5.5.2.1 Per-Sample Encoding and Compression Cost	154
5.5.3 Encoding Quality	155
5.5.4 CIFAR10-DVS Classification	157
5.5.5 CIFAR10-DVS Classification with DyNED_c Compression	160
5.5.5.1 DyNED_c compression on cAdLIF spike outputs	162
5.6 Conclusion	163
6 Case Study - Speech Commands	164
6.1 Introduction	164
6.2 Relevance	165
6.3 Methodology	166
6.3.1 System Integration and Performance	167
6.3.1.1 Per-Sample Encoding and Compression Cost	168
6.3.2 Preprocessing	169
6.3.3 Spike Trains and Sigma-Delta Internals	172
6.3.3.1 Effect of Quantisation Levels on Speech Encoding	173
6.3.3.2 DyNED vs Baseline Quantisation Methods	175
6.3.4 Compression	177
6.3.5 Verification	179
6.3.6 Postprocessing	182
6.3.7 Classification	185
6.3.7.1 Architecture for Speech Commands Classification	185
6.3.7.2 Architecture for Speech Commands Classification with DyNED_c	
Compression	188
6.4 Results	190
6.4.1 DyNED Encoding Quality	190
6.4.2 DyNED Classification Results	192
6.4.2.1 Per-Class Performance	197
6.4.2.2 Confusion Matrix	198
6.4.2.3 Network Dynamics	199

6.4.2.4	Per-Class Spike Patterns	204
6.4.2.5	Representation Quality	205
6.4.3	DyNED_c Classification Results	206
6.4.3.1	Per-Class Performance	207
6.4.3.2	Confusion Matrix	208
6.4.3.3	Network Dynamics	209
6.4.3.4	Representation Quality	210
6.4.3.5	DyNED_c Compression in the Data Path	210
6.4.4	Comparison with Related Work	211
6.5	SSC - Neuromorphic Extension	213
6.5.1	Dataset	213
6.5.2	DyNED Encoding Pipeline	214
6.5.2.1	Per-Sample Encoding and Compression Cost	214
6.5.3	Encoding Quality	216
6.5.4	SSC Classification	218
6.5.5	SSC Classification with DyNED_c Compression	221
6.5.5.1	DyNED_c Compression on cAdLIF Spike Outputs	223
6.6	Conclusion	224
7	Discussion and Conclusion	226
7.1	Introduction	226
7.2	Summary of Key Findings	227
7.3	Revisiting Research Questions and Hypotheses	228
7.3.1	Research Question 1 and Hypothesis 1	229
7.3.2	Research Question 2 and Hypothesis 2	229
7.3.3	Research Question 3 and Hypothesis 3	230
7.3.4	Research Question 4 and Hypothesis 4	231
7.3.5	Research Question 5 and Hypothesis 5	231
7.3.6	Research Question 6 and Hypothesis 6	232
7.3.7	Research Question 7 and Hypothesis 7	233
7.3.8	Research Question 8 and Hypothesis 8	233
7.3.9	Summary of Hypothesis Outcomes	234
7.3.10	Cross-Dataset Summary	236
7.4	Contributions to SNN Knowledge	241
7.5	Methodological and Theoretical Implications	242
7.6	Rate Coding, Temporal Coding, and Neuromorphic Deployment	244
7.6.1	Rate Coding versus Temporal Coding	244
7.6.2	Biological Plausibility	245
7.6.3	Neuromorphic Deployment Implications	247
7.7	Practical and Applied Considerations	248
7.8	Limitations of the Study	251
7.9	Directions for Future Research	252
7.9.1	Neuromorphic Hardware	252
7.9.2	Streaming Spikes	253
7.9.3	Compressed Stream Data Transmission	254
7.9.4	Real-Time SNN Learning	254
7.9.5	Deeper SNN Architectures	254
7.10	Ethical and Societal Implications	255
7.10.1	Energy Efficiency and Environmental Sustainability	255
7.10.2	Privacy Considerations	255

7.10.3 Bias, Fairness, and Accessibility	256
7.10.4 Dual-Use Considerations	256
7.11 Returning to the Research Problem	257
7.12 Concluding Remarks	258
Bibliography	260
A Appendix A	278
A.1 DyNED Encoder	278
A.2 DyNED Decoder	280
A.3 Image Encoder and Decoder	281
A.4 Speech Encoder and Decoder	283
A.5 Boosting Signals	286
A.6 DyNED _c Algorithm	286
A.7 CIFAR-10 DyNED Hyperparameter Optimisation Analysis	292
A.8 CIFAR-10 DyNED _c Hyperparameter Optimisation Analysis	294
A.9 CIFAR10-DVS DyNED Hyperparameter Optimisation Analysis	296
A.10 CIFAR10-DVS DyNED _c Hyperparameter Optimisation Analysis	298
A.11 Speech Commands cAdLIF DyNED Hyperparameter Optimisation Analysis	300
A.12 SSC DyNED Hyperparameter Optimisation Analysis	302
A.13 SSC DyNED _c Hyperparameter Optimisation Analysis	303
B Appendix B	306

Figures

Figure 1 An overview of the thesis structure.	34
Figure 2 Example of AER spike-train generation	38
Figure 3 An example of BSA (Schrauwen & Van Campenhout, 2003, p. 2827).	39
Figure 4 An example of an overlapped ganglion (Hubel et al., 1995, p. 10).	40
Figure 5 An example of Population Rank Encoding.	40
Figure 6 I&F circuit model (Paugam-Moisy & Bohte, 2012, p. 344)	42
Figure 7 LIF spike pattern (Cai et al., 2022, p. 670)	42
Figure 8 Izhikevich neuron models (Izhikevich, 2003, p. 1570).	44
Figure 9 STDP: change in synaptic strength as a function of spike timing	47
Figure 10 A structure of multilayered neural network.	50
Figure 11 NeuCube reservoir connectivity before unsupervised learning	51
Figure 12 NeuCube three-component architecture	51
Figure 13 TPE optimisation loop	54
Figure 14 SpiNN-3 board and one of its processor's architecture.	61
Figure 15 A two second analogue signal of a sine wave above origin.	62
Figure 16 A simple one-cycle square wave	63
Figure 17 An example of Uniform Quantisation	65
Figure 18 An example of Linear Interpolation	67
Figure 19 Digital signal constructed from an analogue sine wave via uniform sampling	69
Figure 20 Analogue signal constructed from a digital step wave via cubic spline interpolation	70
Figure 21 Digital signal constructed from a random analogue signal	71
Figure 22 A Spectrogram of <i>Bird</i> speech from Warden (2018).	73
Figure 23 Mel-Spectrogram of <i>Bird</i> speech	74
Figure 24 Spectral gating applied to <i>Bird</i> speech	75
Figure 25 Compression ratio comparison across quantisation levels	111
Figure 26 Visual pathways in the brain	118
Figure 27 Methodology for image classification using Spiking Neural Networks (SNN).	118
Figure 28 DyNED + DyNED_c encoding pipeline for a single CIFAR-10 sample ...	122
Figure 29 CIFAR-10 + DyNED_c encoding pipeline for all 10 classes	123
Figure 30 DyNED encoding internals for a CIFAR-10 image	125
Figure 31 Effect of quantisation levels on reconstruction quality and spike density .	126
Figure 32 Spike density and reconstruction quality vs quantisation levels for CIFAR-10	127

Figure 33 DyNED vs uniform vs mu-law quantisation on CIFAR-10	129
Figure 34 Convolutional feature maps across all ten CIFAR-10 classes	131
Figure 35 Network architecture for CIFAR-10 classification	132
Figure 36 CIFAR-10 classification architecture with DyNED _c compression	135
Figure 37 Training loss and test accuracy on CIFAR-10	139
Figure 38 Confusion matrix for CIFAR-10 classification at epoch 250.	141
Figure 39 Evolution of firing rates across network layers during training.	142
Figure 40 Spike raster plots across network layers showing temporal firing patterns.	143
Figure 41 Weight distributions across cAdLIF layers at the end of training.	143
Figure 42 Per-class output-layer spike patterns for CIFAR-10	144
Figure 43 t-SNE visualisation of CIFAR-10 learned representations	145
Figure 44 Training loss and test accuracy on CIFAR-10 with DyNED _c compression	146
Figure 45 Confusion matrix for CIFAR-10 classification with DyNED _c compression at epoch 249.	148
Figure 46 Firing-rate evolution on CIFAR-10 with DyNED _c compression	149
Figure 47 t-SNE visualisation of DyNED _c learned representations	150
Figure 48 DyNED encoding pipeline for 10 CIFAR10-DVS samples	155
Figure 49 CIFAR10-DVS classification architecture with DyNED and cAdLIF	158
Figure 50 CIFAR10-DVS classification architecture with DyNED _c compression	161
Figure 51 Front and cross-section views of the larynx	166
Figure 52 Methodology for speech classification using Spiking Neural Networks (SNN).	166
Figure 53 A sample of raw audio of <i>bird</i> from the Speech Commands dataset.	169
Figure 54 Spectral-gate denoised <i>bird</i> sample from Speech Commands	170
Figure 55 Comparison of audio cleaning techniques for <i>bird</i> speech	171
Figure 56 DyNED quantisation and spike generation for <i>bird</i> speech	172
Figure 57 Spike density and reconstruction quality vs quantisation levels for Speech Commands	175
Figure 58 DyNED vs uniform vs mu-law quantisation on Speech Commands	177
Figure 59 Compression technique comparison for <i>bird</i> spike trains	178
Figure 60 Comparison of interpolation techniques for <i>bird</i> speech	180
Figure 61 Reconstructed <i>bird</i> audio scaled by 10x	182
Figure 62 Postprocessing techniques for <i>bird</i> speech	184
Figure 63 Speech classification architecture with DyNED and cAdLIF	186
Figure 64 Speech classification architecture with DyNED _c compression	189
Figure 65 Training loss and test accuracy on Speech Commands	192

Figure 66	Learning-rate schedule for Speech Commands training	193
Figure 67	DyNED + DyNED_c encoding pipeline for a single Speech Commands sample	193
Figure 68	DyNED + DyNED_c encoding pipeline for 10 speech commands	195
Figure 69	Confusion matrix for Speech Commands classification showing per-class accuracy.	198
Figure 70	Network activity during inference on a speech command	199
Figure 71	Evolution of firing rates across network layers during training.	200
Figure 72	Spike raster plots across network layers showing temporal firing patterns.	201
Figure 73	Per-neuron spike-count and membrane-potential distributions for Speech Commands	202
Figure 74	Trained weight distributions across cAdLIF layers for Speech Commands	203
Figure 75	Per-class average spike patterns at the output layer for all 35 speech commands.	204
Figure 76	t-SNE of learned representations for Speech Commands	205
Figure 77	Training progress for Speech Commands with DyNED_c compression ...	206
Figure 78	Confusion matrix for Speech Commands with DyNED_c compression ...	208
Figure 79	Firing-rate evolution with DyNED_c compression	209
Figure 80	t-SNE for Speech Commands with DyNED_c compression	210
Figure 81	DyNED encoding pipeline for 10 SSC samples	216
Figure 82	SSC classification architecture with DyNED and cAdLIF	219
Figure 83	SSC classification architecture with DyNED_c compression	222
Figure 84	DyNED reconstruction fidelity across all four datasets	239
Figure 85	Standalone DyNED_c compression across all four datasets	240
Figure 86	Fully neuromorphic vision-to-control pipeline for autonomous drone flight	250
Figure 87	History of Neuromorphic hardware (Kudithipudi et al., 2025)	253
Figure 88	CIFAR-10 DyNED Optuna HPO analysis across 56 trials (50 epochs each).	293
Figure 89	CIFAR-10 DyNED_c Optuna HPO analysis across 50 trials (50 epochs each).	295
Figure 90	CIFAR10-DVS DyNED Optuna HPO analysis across 50 trials (50 epochs each).	297
Figure 91	CIFAR10-DVS DyNED_c Optuna HPO analysis across 70 trials (50 epochs each).	299
Figure 92	Speech cAdLIF DyNED SNN Optuna HPO analysis across 257 trials (50 epochs each).	301

Figure 93 SSC DyNED SNN Optuna HPO analysis across 65 trials (50 epochs each).	302
Figure 94 SSC DyNED _c Optuna HPO analysis across 85 trials (50 epochs each). . . .	304

Tables

Table 1	Parameter ranges for AdLIF and cAdLIF neurons	46
Table 2	DyNED reconstruction quality at selected quantisation levels	95
Table 3	DyNED_c vs baseline compressors on DyNED-encoded spike trains	111
Table 4	DyNED_c compression performance across the four case-study datasets ...	113
Table 5	Resource-usage comparison for DyNED, BSA, and TBR on CIFAR-10	119
Table 6	Per-sample encoding time on CIFAR-10	120
Table 7	Effect of quantisation levels on CIFAR-10 spike encoding	127
Table 8	DyNED vs uniform and mu-law quantisation on CIFAR-10	128
Table 9	Hyperparameter search space and best values for CIFAR-10 classification . .	134
Table 10	Hyperparameter search space and best values for CIFAR-10 with DyNED_c compression	136
Table 11	Comparison of various interpolation techniques for DyNED encoded image data.	137
Table 12	Mean compression results for DyNED encoded CIFAR-10 spike data ($n =$ 100 , $levels = 256$).	138
Table 13	Per-class test accuracy for CIFAR-10	140
Table 14	Per-class test accuracy for CIFAR-10 with DyNED_c compression	147
Table 15	CIFAR-10 classification comparison with published methods	152
Table 16	Per-sample encoding time on CIFAR10-DVS	154
Table 17	DyNED encoding quality and DyNED_c compression for CIFAR10-DVS . .	156
Table 18	Per-class test accuracy for CIFAR10-DVS	159
Table 19	CIFAR10-DVS classification comparison with published SNN methods . . .	159
Table 20	Hyperparameter search space and best values for CIFAR10-DVS classification.	160
Table 21	Per-class test accuracy for CIFAR10-DVS with DyNED_c compression	161
Table 22	DyNED_c compression on CIFAR10-DVS cAdLIF spike outputs	162
Table 23	Resource-usage comparison for DyNED, BSA, and TBR on Speech Commands	167
Table 24	Per-sample encoding time on Speech Commands	168
Table 25	Effect of quantisation levels on Speech Commands spike encoding	174
Table 26	DyNED vs uniform and mu-law quantisation on Speech Commands	176
Table 27	Hyperparameter search space and best values for Speech Commands classification	188
Table 28	Interpolation comparison on a Speech Commands sample	190
Table 29	Compression results for <i>bird</i> from the Speech dataset.	191
Table 30	Per-class test accuracy for Speech Commands	197

Table 31	Per-class test accuracy for Speech Commands with DyNED_c compression	207
Table 32	DyNED_c compression on Speech Commands ON/OFF spike streams . . .	211
Table 33	Classification results for Speech Commands dataset.	212
Table 34	Per-sample encoding time on SSC	215
Table 35	DyNED encoding quality and DyNED_c compression for SSC ($n = 10$, one sample per class, levels = 256).	217
Table 36	Per-class test accuracy for SSC	220
Table 37	SSC classification comparison with published SNN methods	220
Table 38	Hyperparameter search space and best values for SSC classification	221
Table 39	Per-class test accuracy for SSC with DyNED_c compression	223
Table 40	DyNED_c compression on SSC cAdLIF spike outputs	224
Table 41	Summary of hypothesis outcomes with measurable criteria and supporting evidence.	235
Table 42	Cross-dataset classification results for the DyNED and DyNED_c pipelines	236
Table 43	DyNED reconstruction fidelity across all four datasets at three quantisation levels	238
Table 44	Standalone DyNED_c compression across all four datasets	240
Table 45	Comparison of signal boosting techniques on a Speech Commands sample	286
Table 46	Best CIFAR-10 DyNED+cAdLIF+TET hyperparameters.	294
Table 47	Best CIFAR-10 DyNED_c cAdLIF+TET hyperparameters.	296
Table 48	Best CIFAR10-DVS DyNED hyperparameters.	298
Table 49	CIFAR10-DVS DyNED_c search space and hyperparameter values used. . .	300
Table 50	Best Speech Commands cAdLIF+DyNED SNN hyperparameters.	301
Table 51	Best SSC cAdLIF+DyNED SNN hyperparameters.	303
Table 52	Best SSC DyNED_c hyperparameters.	305

Algorithms

Algorithm 1 Pseudocode for Dynamic Evolving Spiking Neural Network (deSNN) algorithm (Dhoble et al., 2012).	50
Algorithm 2 DyNED Quantisation.	89
Algorithm 3 Step signal generation from quantised data.	90
Algorithm 4 DyNED encoder algorithm for Spiking Neural Networks.	90
Algorithm 5 Radial Basis Function (RBF) interpolation using Gaussian function.	91
Algorithm 6 Decoder algorithm for Spiking Neural Networks.	91
Algorithm 7 Encoder for Speech Commands.	91
Algorithm 8 Decoder for Speech Commands.	92
Algorithm 9 Hybrid Binary Compression: Adaptive Mode Selection	107
Algorithm 10 Hybrid Binary Decompression: Mode-Based Reconstruction	109

Glossary

R^2	Coefficient of Determination
ADC	Analogue-to-Digital Converter
AER	Address Event Representation
AI	Artificial Intelligence
AMP	Automatic Mixed Precision
ASP	Analogue Signal Processing
AdLIF	Adaptive Leaky Integrate-and-Fire
ArcTan	Arctangent surrogate gradient
BSA	Ben's Spiker Algorithm
BTDP	Burst Time-Dependent Plasticity
BWT	Burrows-Wheeler Transform
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CR	Compression Ratio
CUDA	Compute Unified Device Architecture
DA	Decompression Accuracy
DAC	Digital-to-Analogue Converter
DSNN	Deep Spiking Neural Network
DVS	Dynamic Vision Sensor
EEG	Electroencephalography
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GPGPU	General Purpose Computing on Graphical Processing Unit
HH	Hodgkin-Huxley model
HPO	Hyperparameter Optimisation
HSA	Hough Spiker Algorithm
I&F	Integrate-and-Fire
ITU	International Telecommunication Union
IoT	Internet of Things

LIF	Leaky Integrate-and-Fire
LTD	Long-Term Depression
LTP	Long-Term Potentiation
LZ77	Lempel-Ziv 1977
LZW	Lempel-Ziv-Welch
MAE	Mean Absolute Error
MATE	Mean Absolute Timing Error
MFCC	Mel-Frequency Cepstral Coefficients
OCR	Optical Character Recognition
OpenCL	Open Computing Language
RAM	Random Access Memory
RBF	Radial Basis Function
RLE	Run-Length Encoding
RMSE	Root Mean Square Error
RO	Rank Order
SNN	Spiking Neural Network
SNR	Signal-to-Noise Ratio
SPAN	Spike Pattern Association Neuron
SSC	Spiking Speech Commands
SSI	Structural Similarity Index
SSM	State-Space Model
STDP	Spike-Time Dependent Plasticity
STFT	Short-Time Fourier Transform
TBR	Threshold-Based Representation
TET	Temporal Efficient Training
TPE	Tree-structured Parzen Estimator
VGGSNN	VGG-style Spiking Neural Network
VLSI	Very Large Scale Integration
aEIF	Adaptive Exponential Integrate-and-Fire
cAdLIF	Constrained Adaptive Leaky Integrate-and-Fire
deSNN	Dynamic Evolving Spiking Neural Network
pp	Percentage Points
t-SNE	t-distributed Stochastic Neighbour Embedding

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Akshay Raj Gollahalli

Publications

Khansama, R. R., Ravi, V. R., Gollahalli, A. R., Sengupta, N. R., Kasabov, N. K., & Bilbao-Quintana, I. K. (2024). Stock Market Movement Prediction Using Evolving Spiking Neural Networks. In *Handbook of Big Data Analytics Volume 2: Applications in ICT, Security and Business Analytics* (pp. 285-312). The Institute of Engineering and Technology. https://doi.org/10.1049/PBPC037G_ch14

This publication underscores the practical application and performance of evolving Spiking Neural Network (SNN) architectures, particularly utilising the NeuCube framework in financial forecasting. It demonstrates the broader real-world applicability of SNN and reinforces the robustness of SNN advocated in this thesis, particularly highlighting its capacity for complex predictive analytics. My contribution involved conducting experiments on NeuCube using Leaky Integrate-and-Fire (LIF) neurons and Spike-Timing-Dependent Plasticity (STDP) algorithms, which are utilised extensively within this thesis.

Gholami Doborjeh, M., Gholami Doborjeh, Z., Gollahalli, A. R., Kumarasinghe, K., Breen, V., Sengupta, N., Ramos, J. I. E., Hartono, R., Capecci, E., Kawano, H., Othman, M., Zhou, L., Yang, J., Bose, P., & Ge, C. (2018). From von Neumann Architecture and Atanasoff's ABC to Neuromorphic Computation and Kasabov's NeuCube. Part II: Applications. In V. Sgurev, V. Jotsov, & J. Kacprzyk (Eds.), *Practical Issues of Intelligent Innovations* (pp. 17-36). Springer International Publishing. https://doi.org/10.1007/978-3-319-78437-3_2

This publication illustrates the practical applications and efficacy of SNN using NeuCube as an example. It applies SNN to various complex problems in spatio-temporal data analysis. It provides evidence of the adaptability and applicability of the SNN NeuCube, reinforcing the thesis' aim of demonstrating SNN technology as an efficient tool for real-world data modelling. My contribution to this work was developing a cognitive game featuring online classification using NeuCube, employing a continuous data encoding method that is elaborated upon within this thesis.

Sengupta, N., Ramos, J. I. E., Tu, E., Marks, S., Scott, N., Weclawski, J., Gollahalli, A. R., Doborjeh, M. G., Doborjeh, Z. G., Kumarasinghe, K., Breen, V., & Abbott, A. (2018). From von Neumann Architecture and Atanasoffs ABC to Neuro-Morphic Computation and Kasabov's NeuCube: Principles and Implementations. In V. Sgurev, V. Piuri, & V. Jotsov (Eds.), *Learning systems: From theory to practice* (pp. 1'28). Springer International Publishing. https://doi.org/10.1007/978-3-319-75181-8_1

This paper provides a foundation, context and technical background to the shift from traditional logic-based computing architectures to Neuromorphic computing. It elaborates on the structural and functional advantages of SNN using NeuCube as an example, supporting the thesis' exploration of improved learning methodologies for neuromorphic architectures. My specific contribution involved implementing NeuCube on Neuromorphic hardware (SpiNNaker), including continuous data encoding into spikes and training using the NeuCube architecture.

Acknowledgements

I want to begin by expressing my deepest gratitude to my supervisor, **Professor Ajit Narayanan**. Whenever I found myself hitting a road block, his patience, wisdom and genuine belief in my potential have guided me to overcome every obstacle with confidence and ingenuity. Alongside him, I owe an enormous thanks to **Dr Michael J Watts**, who walked with me every step of the way and never failed to support me, even on the days when I couldn't find my conviction. And to **Professor Nikola Kasabov**, who opened the doors to this remarkable field, thank you for setting me on a path that has genuinely transformed my life.

I am forever indebted to my wife, **Dr Keertana Tipparaju**, who has spent countless weekends at home with me while I work on this thesis. She put her own ambitions on hold to be my pillar of strength, offering her unconditional optimism every day. I equally owe my boundless love and respect to my parents, **Dr Vijayalakshmi Muirusupalli and Ravi Raj Nageswara Gollahalli**, who have always supported my every decision and empowered me with their unwavering faith. Their love and willingness to give me time and freedom to chase my dreams laid the foundation for all that I have accomplished.

My heartfelt appreciation also goes to **Joyce D'Mello**, who has been so much more than an adviser or confidante. She has been my sounding board, my motivator, and my anchor. Her advice and friendship have guided me through personal and professional trials-without her unwavering support, I might never have had the courage to undertake this endeavour.

I cannot overstate the significance of **Dr Josafath Israel Espinosa Ramos** in my life. Not only has he been a mentor of the highest calibre, but he has also proven himself to be a true friend. In my darkest moments, Israel was the constant reassurance that kept me moving forward, and I will forever be grateful for the warmth and kindness he has shown me.

I extend my thanks as well to **everyone at KEDRI**. Each member of this exceptional team has played a vital role in making my academic journey not just productive but also joyous and engaging. The camaraderie and collective enthusiasm at KEDRI have made even the most challenging days more bearable and, often, genuinely fun.

Finally, I am incredibly thankful to **AUT University** for granting me the opportunity to study in this beautiful country and for providing the resources and environment

necessary to excel. Completing this PhD would not have been possible without the academic support, facilities, and continuous encouragement I received here.

I would also like to extend my sincere thanks to the **examiners of this thesis**, whose careful reading and constructive feedback have strengthened the work and shaped it into its final form.

From the bottom of my heart, I thank each and every individual who believed in me, stood by me, and gave me the strength to see this through. This achievement belongs to us all.

Chapter 1

Introduction

The rapid advancements in computational neuroscience have led to significant developments in neural network models, particularly Spiking Neural Networks (SNNs). These networks, designed to mirror the complex dynamics of biological neural systems, have shown great potential across a range of applications, from image recognition to natural language processing. The brain itself, an intricate and powerful organ, serves as the primary inspiration for these models. Its ability to process information efficiently and adaptively drives the ongoing quest to replicate similar functionalities in computational systems.

Recent advances in SNN research, as referenced in works by Guo et al. (2023); Hussain & Thounaojam (2024); Niu et al. (2023); Tanviruzzama & Mehfuz (2023); Xiaoxue et al. (2023), span a wide range of applications including image classification, computer vision, medical data analysis, and speech recognition. New frameworks like SPAIC (Hong et al., 2024) and established ones like NeuCube (Kasabov, 2014), Brian SNN simulator (Stimberg et al., 2019) and snnTorch (Eshraghian et al., 2023a) utilise various encoding methods to convert input data into spikes. However, despite this, there needs to be greater clarity on the impact of encoding strategies on SNN performance and the completeness of data representation in the encoded spikes. This thesis aims to address this gap in the literature.

However, the field faces challenges, particularly in optimising these networks for complex, dynamic datasets. Traditional computational models, while inspired by the brain's structure, often fall short of capturing its actual processing efficiency and adaptability.

This thesis addresses a critical gap in the current understanding and application of SNNs: the efficient encoding of input data and the effective utilisation of computational resources, such as GPUs. While SNNs offer a biologically plausible model of computation, inspired by the brain's neural networks, their practical applications are often hindered by suboptimal encoding strategies and underutilised computational architectures.

Through a series of focused research questions and hypotheses, this study aims to explore novel encoding techniques, assess the role of GPUs in improving SNN efficiency, and investigate the adaptability of SNN architectures to diverse data inputs. The significance of this research lies in its potential to advance the practical applications of SNNs in computational neuroscience and artificial intelligence, enhancing our understanding and emulation of the brain's complex processing mechanisms.

1.1 Academic Motivation

The intricate workings of the human brain have long been a source of inspiration and curiosity, driving relentless pursuit in the field of computational neuroscience. The development of Spiking Neural Networks (SNNs), with their promise to emulate the brain's natural neural processes, represents a significant stride in this quest (Kasabov et al., 2013). These networks, with their potential applications ranging from image recognition to complex data analysis, are crucial to revolutionising our approach to artificial intelligence.

Spiking Neural Networks (SNNs) are a class of artificial neural networks inspired by the functionality of biological brains, in which neurons communicate via discrete events called spikes. Unlike traditional neural networks, which use activation functions to process data, SNNs operate on membrane potential dynamics, where incoming spikes influence the neuron's state and trigger a spike once a specific threshold is reached (Maass, 1997). This mechanism allows SNNs to process continuous-time data inherently, as they do not rely on predefined discrete time steps. The input and output of an SNN are always in the form of spikes, making them particularly adept at handling spatio-temporal data. A significant advantage of SNNs over traditional neural networks is their potential for high computational efficiency and lower power consumption, primarily when implemented on neuromorphic hardware, making them suitable for real-time processing in power-sensitive applications (Hu et al., 2023; Maass, 1997).

Despite substantial progress and exciting breakthroughs in SNN research, several challenges still need to be addressed, particularly in data encoding and computa-

tional efficiency. The way input data is encoded into spikes - a process fundamental to the operation of SNNs - is still a subject of ongoing research and debate. The efficiency and completeness of this encoding process critically influence the performance of SNNs, and yet, the field lacks a comprehensive understanding of this aspect. Additionally, the effective utilisation of computational resources, such as GPUs, to enhance SNN performance has not been fully explored, leaving ample room for optimisation and advancement.

My academic motivation for this research stems from a desire to bridge these gaps, deepen our understanding of SNNs, and push the boundaries of what these advanced neural network models can achieve. By focusing on novel encoding techniques, assessing the role of GPUs, and exploring the adaptability of different SNN architectures, this thesis aims to unlock new possibilities and improve the practical applicability of SNNs. The potential to enhance the emulation of the brain's complex processing mechanisms and to advance computational neuroscience and artificial intelligence is a driving force behind this study.

This research contributes to the development of more efficient and adaptive data encoding for neuromorphic systems. Informed by the principles of biological neural processing, this thesis develops novel encoder-decoder (DyNED) and compression (DyNED_c) techniques that bridge the gap between real-world data and the spike-based processing paradigm of SNNs. This work advances the field of computational neuroscience through scientifically grounded innovations in spike encoding, signal reconstruction, and data compression.

1.1.1 Research Problem

Spiking Neural Networks (SNNs) are a biologically inspired alternative to conventional artificial neural networks. Still, their practical adoption is limited by the encoding step that converts real-world data - frame-based images, audio waveforms, asynchronous neuromorphic events - into the binary spike trains that SNNs operate on. Existing encoders prioritise spike generation over reconstruction fidelity, treat each data modality with a bespoke pipeline, and produce dense spike streams that are not designed to be losslessly compressed. As a result, encoded representations cannot be reliably decoded back to the original signal, cannot be cleanly compared across data types, and cannot be efficiently stored or transmitted. Together, these gaps prevent SNNs from being deployed in the practical pipelines (remote sensing, edge inference, end-to-end neuromorphic deployment) where their energy-efficiency advantages would matter most.

RPS – Research Problem Statement: No published method performs reconstruction-faithful encoding and decoding of spike trains from multi-modal sources, and no lossless compressor exists for the resulting binary spike streams - preventing SNN deployment in real-world pipelines.

This thesis addresses that limitation by developing (i) a unified, reconstruction-faithful encoder-decoder for SNNs (DyNED) that operates without architectural change across both static and dynamic data and both conventional and neuromorphic data sources, and (ii) a lossless compressor tailored to the resulting binary spike streams (DyNED_c). The significance is twofold: closing the encoding-decoding loop with a quantified reconstruction metric ($R^2 \geq 0.95$) makes the information cost of spike encoding measurable in this study, and demonstrating modality-agnostic operation across four datasets removes a long-standing engineering obstacle to deploying SNNs in heterogeneous real-world pipelines.

The Research Problem Statement (RPS) decomposes into the research questions in Section 1.3.2, each operationalised by a testable hypothesis in Section 1.3.3 and evaluated against the criteria set out in Section 1.4.

1.2 Personal Motivation

For many centuries the brain has captivated researchers and laypersons alike, given its extraordinary capacity to enable feats ranging from skyscraper construction to musical composition. My journey into computational neuroscience began during my Master’s studies, when I developed a three-dimensional gaming environment in which objects could be moved using EEG signals (Gollahalli, 2015). This early work laid the foundation for the research that would become the focus of my doctoral thesis.

Since then, my investigations into spiking neurons have primarily involved converting real-world data into higher-dimensional spike trains. As I progressed, two fundamental questions repeatedly arose: first, whether I was performing the conversion correctly, and second, whether the higher-dimensional data legitimately reflected the original. These concerns led me to explore analogue-to-digital converters (ADCs) in depth.

When examining ADCs, I initially studied turntables - specifically, how the stylus follows the groove of a vinyl record to produce sound. Reflecting on how music consumption has evolved from gramophones to iPods and, more recently, to applications on smartwatches highlights our reliance on “digital brains” that learn and recommend content. It became evident that spiking neurons require an encoder that swiftly trans-

forms continuous data into discrete values while preserving the key characteristics of the original signal.

1.3 Research Hypotheses and Goals

1.3.1 Goals

The primary goal of this thesis is to explore and advance the understanding of Spiking Neural Networks (SNNs) in the context of image and speech recognition, focusing on the efficiency and effectiveness of various encoding techniques. This involves evaluating the performance of lossy versus lossless image processing in SNN encoders, assessing the reliability of GPUs for processing large datasets in SNNs, and determining the necessity and versatility of encoders for both static and dynamic data, drawn from both conventional and neuromorphic sources.

By conducting comprehensive case studies across four datasets - CIFAR-10 and Speech Commands (conventional image and audio) together with CIFAR10-DVS and Spiking Speech Commands (their neuromorphic event-based counterparts) - this research aims to demonstrate how minimal information loss during encoding influences overall system performance and classification accuracy, and to explore the integration capabilities of the Deep Spiking Neural Network with encoded spikes outside of its native architecture. The same supervised learning rule - surrogate-gradient backpropagation through time with the ArcTan surrogate (Fang et al., 2021) - is used in every case study, so the cross-dataset comparison isolates the effect of modality and encoder choice rather than the training algorithm. Ultimately, this thesis seeks to establish the reliability of SNNs across both static and dynamic modalities and across both conventional and neuromorphic data domains, potentially paving the way for more efficient and versatile neural network models in computational neuroscience and artificial intelligence.

1.3.2 Research Questions

This thesis investigates eight specific research questions that follow from the research problem stated in Section 1.1.1. Each research question maps one-to-one onto a testable hypothesis presented in Section 1.3.3, where the question frames the inquiry and the hypothesis operationalises it as a measurable prediction that can be supported or rejected by the experimental work in Chapter 3 through Chapter 6.

RQ1 – Research Question 1: How effective is frequency-domain transformation (FFT for images, STFT for speech) before sigma-delta spike quantisation at producing

spike representations with high reconstruction fidelity and competitive classification accuracy?

RQ2 – Research Question 2: How does sigma-delta quantisation with iterative error feedback affect spike density and reconstruction error compared to uniform quantisation and mu-law companding baselines?

RQ3 – Research Question 3: How well can a single DyNED encoder, without structural modification, encode both static image data (CIFAR-10) and temporal speech data (Speech Commands) into spike trains when paired with an SNN classifier?

RQ4 – Research Question 4: How much temporal and spectral structure do DyNED-encoded spike representations of speech retain for downstream SNN classification, compared to a non-spiking CNN baseline operating on identical raw spectral features?

RQ5 – Research Question 5: How effective is DyNED_c at losslessly compressing DyNED spike trains while maintaining classification accuracy, and how does compression effectiveness scale with spike sparsity?

RQ6 – Research Question 6: How does the hybrid DyNED_c compression method (RLE followed by Huffman coding) compare to each method applied individually and to general-purpose compressors (LZW, LZ77, Delta) on binary spike trains?

RQ7 – Research Question 7: How well can an SNN trained with surrogate gradients on DyNED-encoded data classify CIFAR-10 and Speech Commands compared to a non-spiking CNN baseline with the same convolutional front-end?

RQ8 – Research Question 8: How well does a single DyNED + DyNED_c encoder-compressor pipeline, without structural modification to either component, generalise across conventional and neuromorphic data and across static and dynamic modalities?

1.3.3 Hypotheses

The eight hypotheses below operationalise the research questions of Section 1.3.2. Each hypothesis states a measurable prediction whose support or rejection by the experimental work in Chapter 3 through Chapter 6 constitutes the answer to the corresponding research question.

1.3.3.1 Hypothesis 1

Encoding data via frequency-domain transformation (FFT for images, STFT for speech) before sigma-delta spike quantisation produces spike representations that achieve signal reconstruction fidelity of $R^2 \geq 0.95$ and enable competitive classification accuracy on both image and speech benchmarks.

1.3.3.2 Hypothesis 2

DyNED’s sigma-delta quantisation with iterative error feedback produces denser and more temporally structured spike trains than uniform quantisation and mu-law companding baselines, yielding lower reconstruction error (RMSE) at equivalent quantisation levels.

1.3.3.3 Hypothesis 3

A single DyNED encoder architecture, without structural modification, can encode both static image data (CIFAR-10) and temporal speech data (Speech Commands) into spike trains that achieve classification accuracy above 85% when paired with an SNN classifier.

1.3.3.4 Hypothesis 4

DyNED-encoded spike representations of speech retain sufficient temporal and spectral structure to support downstream SNN classification, achieving substantially higher accuracy than a non-spiking CNN baseline operating on identical raw spectral features (no DyNED encoding) on the Speech Commands dataset.

1.3.3.5 Hypothesis 5

Lossless compression of DyNED-encoded spike trains via DyNED_c reduces storage while maintaining classification accuracy within 2% of the uncompressed baseline on both image and speech tasks, with compression effectiveness scaling with spike sparsity.

1.3.3.6 Hypothesis 6

The hybrid DyNED_c compression method (RLE followed by Huffman coding) matches or exceeds the compression ratio of either method applied individually on binary spike train data, and outperforms general-purpose compressors (LZW, LZ77, Delta compression).

1.3.3.7 Hypothesis 7

An SNN trained with surrogate gradients on DyNED-encoded data achieves classification accuracy within 5 percentage points of a non-spiking CNN baseline with the same convolutional front-end, on both CIFAR-10 and Speech Commands datasets.

1.3.3.8 Hypothesis 8

A single DyNED + DyNED_c encoder-compressor pipeline, without structural modification to either component, generalises across both conventional and neuromorphic data and across both static and dynamic modalities.

1.3.4 Scope of Hypotheses

The hypotheses formulated in this thesis are scoped within the field of computational neuroscience, explicitly focusing on Spiking Neural Networks (SNNs) and their applications in image and speech recognition. The hypotheses are tested and evaluated under the following conditions:

- The comparative efficiencies of lossy and lossless compressed images and audio processing techniques provided to the SNN encoders are explored and evaluated across CIFAR-10, Speech Commands, CIFAR10-DVS, and Spiking Speech Commands.
- The reliability and performance of GPUs in handling large encoding datasets are assessed, acknowledging the computational constraints and capabilities of current GPU technology.
- The necessity and versatility of encoders for SNNs are examined across both static and dynamic data types and across both conventional and neuromorphic data sources, within the constraints of current encoding methodologies.
- The impact of information loss in encoding on the overall performance of SNNs is evaluated, considering various levels of data fidelity and processing complexity.
- The adaptability of various architectures, including NeuCube, Brian SNN simulator and snnTorch, to process encoded spikes that are not inherently native to their systems, meaning spikes generated externally by DyNED rather than by the framework's own built-in encoder, thereby underscoring the flexibility and compatibility of these SNN architectures.
- The generalisability of SNNs in image and audio recognition tasks is explored across all four datasets, comparing their performance with traditional neural network models under similar test conditions.

1.4 Evaluation Measures

The research is evaluated against the following measures, each aligned with the research questions in Section 1.3.2 and the hypotheses in Section 1.3.3:

Efficiency of Encoding: Success in demonstrating that lossy image encoding in SNNs leads to better performance than lossless encoding, as measured by processing speed

and accuracy in image recognition tasks using the CIFAR-10 dataset and its neuromorphic counterpart CIFAR10-DVS.

GPU Performance: Proving the reliability and efficiency of GPUs in processing large encoder datasets, with success indicated by improved computation times and reduced error rates across the Speech Commands and Spiking Speech Commands datasets.

Encoder Necessity and Versatility: Establishing the essential role of encoders in SNNs across both static and dynamic data and across both conventional and neuromorphic data, with success shown through comparative analysis of network performance with and without specialised encoders, using all four datasets (CIFAR-10, Speech Commands, CIFAR10-DVS, Spiking Speech Commands).

Information Loss Impact: Demonstrating that minimal information loss during encoding correlates with improved SNN performance, evaluated through comparative accuracy and reliability metrics across the four datasets.

SNN Toolbox Adaptability: Successful integration and functional performance of encoded spikes not native to NeuCube’s architecture, the Brian SNN simulator, or snnTorch, assessed through compatibility tests and performance metrics, particularly with data derived from the Speech Commands dataset.

SNN Reliability across Modalities and Data Domains: Establishing the reliability of SNNs in image recognition tasks (CIFAR-10, CIFAR10-DVS) and in speech recognition tasks (Speech Commands, Spiking Speech Commands), with success measured against benchmarks set by traditional neural network and SNN models on both the conventional and neuromorphic versions of each modality.

Each measure is designed to be quantifiable and achievable within the scope of the research, providing a clear benchmark for the work, and tracks back to the research questions in Section 1.3.2 (and their paired hypotheses in Section 1.3.3) that it operationalises.

1.5 Original Contribution of the Research

This thesis contributes several original aspects to the field of computational neuroscience, mainly focusing on the application and optimisation of Spiking Neural Networks (SNNs) for image and speech recognition:

Innovative Encoding Techniques: We have developed novel encoding strategies for static and dynamic data, significantly enhancing the efficiency and adaptability of SNNs. This approach offers a substantial improvement in handling complex datasets such as CIFAR-10 and Speech Commands compared to traditional encoding methods.

GPU Utilisation in Encoders and Decoders: An essential contribution of this research is demonstrating the potential of GPUs to improve computational efficiency and accuracy in the specific context of SNN encoders and decoders. This exploration into the GPU's role within SNN encoding processes represents a relatively uncharted area in the field.

Versatility of Encoders: The study provides new insights into the necessity and versatility of specialised encoders in SNNs, contributing an enhanced understanding of how different data types can be effectively processed in neural networks.

Impact of Information Loss: Empirical evidence shows that minimal information loss during encoding improves overall SNN performance, advancing understanding of data integrity in neural network processing.

Compatibility with Other Architectures: My work explores the adaptability of other architectures to encode spikes that are not native to their systems, offering innovative perspectives on the flexibility and applicability of this SNN architecture.

Compatibility with DSNNs: The research investigates the compatibility of Deep Spiking Neural Networks (DSNNs) with encoded spikes, demonstrating the potential to integrate encoded data into sophisticated neural network architectures.

Benchmarking SNNs in Recognition Tasks: The research also establishes new benchmarks for SNN performance in image and speech recognition tasks, challenging existing perceptions and underscoring their potential in these domains.

These contributions address specific gaps identified in existing literature and open new avenues for practical applications and future research in computational neuroscience and artificial intelligence.

1.6 Structure of Thesis

In the introduction chapter of this thesis, I begin by setting the context and outlining the primary research questions and hypotheses. Following this, I delve into a detailed structure of the thesis, presented as follows:

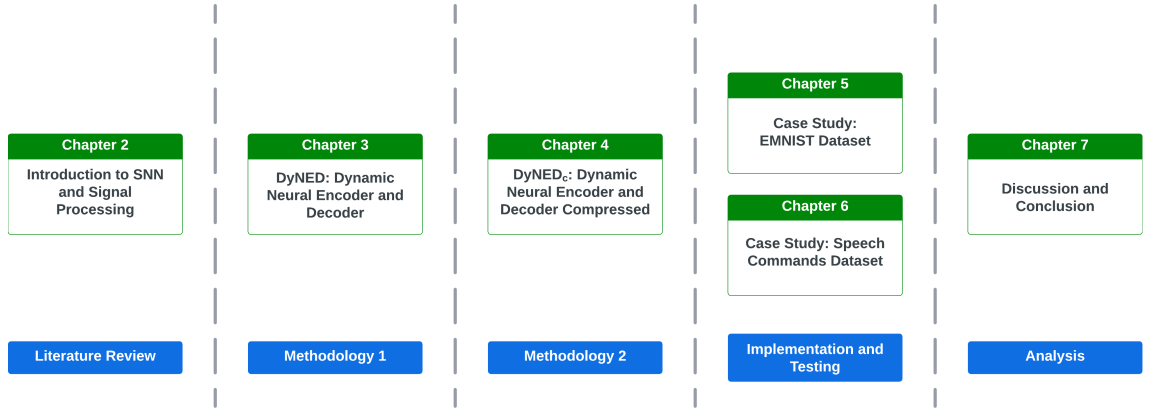


Figure 1: An overview of the thesis structure.

Chapter 2: This chapter introduces the foundational concepts crucial to the research. It covers the intricacies of Spiking Neural Networks (SNNs) and differentiates among various types. I explore both supervised and unsupervised learning paradigms within the context of SNNs. Additionally, the chapter discusses the platforms that support SNN functionalities, data compression techniques, and the signal-processing algorithms pertinent to the study. This background surveys the state of the art on each of the components named in the RPS - encoders, decoders, spike compression, and SNN learning across static and dynamic data - and identifies the gaps that the research questions in Section 1.3.2 will go on to address experimentally.

Chapter 3: Here I introduce the novel encoders and decoders developed as a part of this thesis, called Dynamic Neural Encoder and Decoder (DyNED), tailored for both static and dynamic data. The chapter not only details the design and functioning of these encoders and decoders but also justifies their significance and application in the realm of SNNs.

Chapter 4: In this chapter, I introduce a novel extension of Chapter 3, called Dynamic Neural Encoder with Compression (DyNED_c). This encoder leverages advanced compression techniques to improve the efficiency and performance of SNNs, particularly when handling large datasets. The chapter elaborates on the design, implementation, and evaluation of DyNED_c, showcasing its potential in real-world applications.

Chapter 5: Utilising the algorithms developed in Chapter 3 and Chapter 4, this chapter presents a comprehensive case study on the CIFAR-10 dataset and its neuromorphic event-based counterpart CIFAR10-DVS. The focus is on applying the methodologies and analysing their effectiveness across both conventional and neuromorphic visual data, showcasing the practical implications of the research in a controlled environment.

Chapter 6: Similarly, this chapter applies the algorithms to the Speech Commands dataset and its neuromorphic counterpart Spiking Speech Commands (SSC), which substitutes the conventional audio waveform for cochlear spike events from a biologically detailed auditory nerve model. The comparative analysis across the four datasets (conventional and neuromorphic, image and speech) highlights the adaptability and robustness of the algorithms, providing insights into their broader applicability.

Chapter 7: The concluding chapter synthesises the findings of the thesis, reflecting critically on the research’s implications, its limitations, and directions for future work. It revisits each of the eight research questions (Section 1.3.2) against the experimental evidence and returns explicitly to the RPS to summarise which aspects have been resolved by this work and which remain open for future research.

1.7 Summary

At the heart of this thesis lies a thorough examination of encoders in Spiking Neural Networks (SNNs). This fundamental component bridges the gap between raw input data and the spike-based processing paradigm of SNNs. The study is dedicated to addressing the pressing challenge of efficiently encoding data for SNNs - through the development of DyNED and DyNED_c - and to harnessing computational resources, such as GPUs, to optimise this process. Despite significant strides in the field of SNNs, the nuanced effects that various encoding strategies have on network performance remain poorly understood. Furthermore, the potential of GPUs to substantially enhance the computational efficiency of SNNs, particularly through encoder optimisation, remains an underexplored avenue, poised for discovery and innovation. This thesis, through case studies on CIFAR-10 and Speech Commands and their counterparts, demonstrates that thoughtful encoding and compression can unlock the full potential of SNNs for real-world recognition tasks.

Chapter 2

Background

Spatio-temporal data classification and interpretation are among the most challenging problems in machine learning techniques. The sheer scale and complexity of the processing units and connections within the human brain lead to highly non-linear, non-separable patterns, which make the classification of these patterns complex at best. To this end, a novel computational environment, the NeuCube, has been proposed in prior literature by Kasabov (2014).

Over the past few decades, an enormous amount of data about the structure and functionality of the brain has been collected (Abeles, 1991; Evans et al., 1993; Mai et al., 2015; Talairach & Tournoux, 1988; Toga et al., 2006; Zilles & Amunts, 2010). Large quantities of spatiotemporal data have been collected, such as Electroencephalography (EEG) (Fiasché et al., 2012), RN43, Earthquake (Zhang & Thurber, 2003), and Air Pollution (Wark & Warner, 1981). However, such types of datasets are difficult to analyse. Traditional methods such as Support Vector Machines, Neural Networks, Multi Linear- and Logistic Regression, and Decision Trees have been used with limited success (Gerstner et al., 2012) for classifying spatiotemporal data or EEG data (Fiasché et al., 2012).

The SNN neurons are similar to biological neurons in that they communicate with other neurons via electrical pulses and spikes (Maass, 1997). At the same time, previous generations of networks were encoded based on continuous values and mean firing rates.

Spiking Neural Networks have many positive features, including compact representation of space and time, fast information processing, and time- and frequency-based information representation. Therefore, they are more appropriate for spatio- and spectro-

temporal data. Amongst them are two types of classifiers deSNN by Kasabov et al. (2013) (discussed briefly in Section 2.5.2) and SPAN by Mohemmed et al. (2012) (discussed briefly in Section 2.5.1).

2.1 Biological Inspiration

Details of the brain's function have been described in considerable depth in many textbooks. Those by Kandel et al. (2000) and Bear et al. (2007) are some of the best sources on this subject, and they are summarised below. And also, its function is described in depth in Gollahalli (2015).

A typical neuron is made up of three parts: the soma (cell body), dendrites and axon. A signal from one neuron is received through the soma & dendrites and transmitted to other neurons through the axon. Typically, the neurons are positioned in such a way that the axon of one neuron and a dendrite (or soma) of another come in contact via a synapse. Synapses are of two types: excitatory (promoting neural activity) and inhibitory (suppressing neural activity). When the net excitation of a receiving neuron exceeds a certain firing threshold, the soma generates a brief change in membrane potential called an action potential (spike), which rapidly travels down the axon and triggers further synaptic transmission to downstream neurons. A continuous sequence of such spikes over time forms a spike train, which is the fundamental information-carrying signal that spike encoders produce and SNNs operate on.

2.2 Spike Encoding

Encoding algorithms convert a series of real numbers into a binary number that is represented as spikes. There are a number of encoding algorithms that have been developed. Here, I present three of the most commonly used algorithms: Address Event Representation (AER) (Lichtsteiner et al., 2008); Ben's Spiker Algorithm (BSA) (Schrauwen & Van Campenhout, 2003); and Population Encoding (Bohte, 2004).

2.2.1 Address Event Representation (AER) Encoder

AER is the asynchronous spike-encoding scheme used by event-based vision and auditory sensors such as the silicon retina (Lichtsteiner et al., 2008) and silicon cochlea. Each pixel (or channel) independently emits a spike - addressed to the host - whenever its log-intensity changes by more than a fixed threshold, encoding *temporal contrast* rather than absolute illumination.

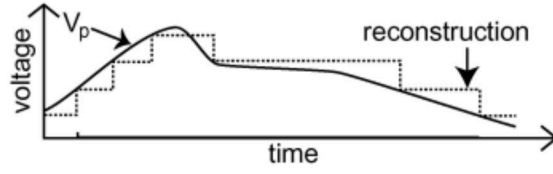


Figure 2: An example of AER. Creating a step signal (spikes) in accordance to the actual signal.

Figure 2 illustrates how AER converts a continuously varying input signal into the resulting ON/OFF event stream. The pixel responds to the temporal contrast of the input photocurrent $I(t)$:

$$\text{TCON} = \frac{1}{I(t)} \frac{dI(t)}{dt} = \frac{d \ln(I(t))}{dt}$$

An ON event is emitted whenever the change in $\ln I$ exceeds a positive threshold θ_{on} ; an OFF event when it falls below a negative threshold $-\theta_{\text{off}}$:

$$s_t = \begin{cases} +1 & \text{(ON) if } \Delta \ln I > \theta_{\text{on}} \\ -1 & \text{(OFF) if } \Delta \ln I < -\theta_{\text{off}} \\ 0 & \text{otherwise} \end{cases}$$

The resulting event rate is approximately proportional to the rate of log-intensity change:

$$f(t) \approx \frac{\text{TCON}(t)}{\theta} = \frac{1}{\theta} \frac{d \ln(I)}{dt}$$

2.2.2 Ben's Spiker Algorithm (BSA) Encoder

BSA was first introduced in Schrauwen & Van Campenhout (2003) and was primarily designed for sound analysis, but it can also be used for EEG data. It is based on the Hough Spiker Algorithm (HSA) (Hough et al., 1999), which encodes analogue signals into spike trains by iteratively subtracting a fixed filter kernel from the input signal whenever a spike is generated. HSA uses a threshold-based mechanism: at each time step, if the signal exceeds a threshold, a spike is emitted, and the filter is subtracted from the signal, reducing the residual. BSA improves upon HSA by using an adaptive threshold and a smoother filter kernel, resulting in reconstructed signals with better preserved frequency and amplitude characteristics. Studies have shown that BSA offers an improvement of 10dB-15dB Signal-to-Noise Ratio (SNR) over the HSA encoding scheme.

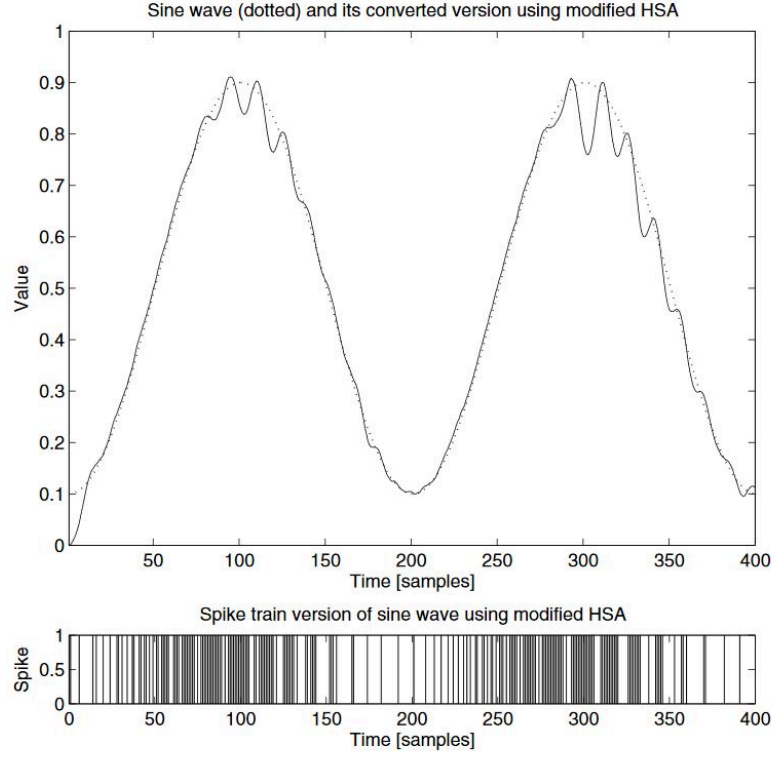


Figure 3: An example of BSA (Schrauwen & Van Campenhout, 2003, p. 2827).

Figure 3 shows a representative BSA encoding, with the source waveform aligned above the resulting spike train so the timing and density of spikes can be read directly against the underlying signal. BSA can be defined as follows. The stimulus is estimated from the spike train by:

$$s_{\text{est}} = (h \times x)(t) = \int_{-\infty}^{+\infty} x(t - \tau)h(\tau)d\tau = \sum_{k=1}^N h(t - t_k) \quad (1)$$

Where t_k is the neuron firing time, $h(t)$ is the linear filter's impulse response, and $x(t)$ is the spike of the neuron which can be calculated as

$$x(t) = \sum_{k=1}^N \delta(t - t_k) \quad (2)$$

When the spike train $x(t)$ is applied with discrete Finite Impulse Response (FIR) filter, the Equation 2 can be rewritten as

$$o(t) = (h \times x)(t) = \sum_{k=0}^M x(t - k)h(k)$$

2.2.3 Population Rank Encoding

Also known as Population Encoding, it is a technique based on the overlapping of the receptive fields. For example, the receptive fields of ganglion cells in the retina of the eye

are always overlapping each other, which means that the combined information sent to the brain via hundreds of cells is coincident. The brain filters out unwanted information and takes in only what is necessary (see Figure 4) (Hubel et al., 1995).

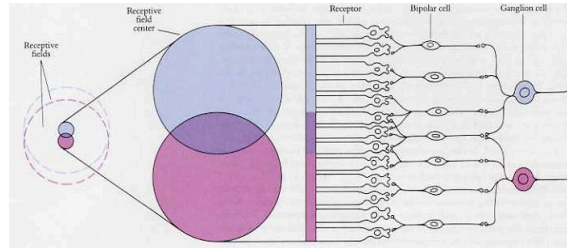


Figure 4: An example of an overlapped ganglion (Hubel et al., 1995, p. 10).

For each continuous input value, a set number of neurons is created, each with an identical (commonly Gaussian-shaped) receptive field that overlaps with the others. These neurons are uniformly distributed between two boundary values, ensuring that the input lies within this range. When the input is positioned across these overlapping receptive fields, it triggers spikes in those neurons whose receptive field curves intersect with the input, according to the distance along the curve (Benuskova & Kasabov, 2008; Bohte, 2004). In this way, a continuous input is converted into a series of delayed spike times across the defined set of neurons.

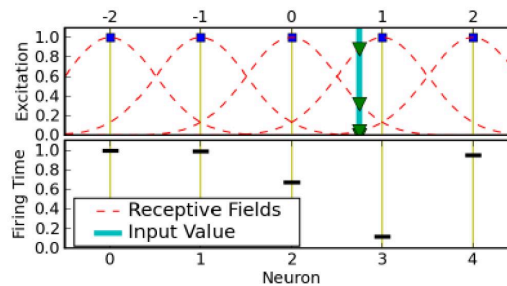


Figure 5: An example of Population Rank Encoding.

Figure 5 shows the Gaussian receptive fields tiled across the input range and the spike times they produce for a single input value, making the mapping from a scalar input to a delayed-spike pattern across the neuron population concrete.

2.3 Spiking Neural Network (SNN) Models

As discussed in Section 2.1, the brain is a complex organ which is capable of making decisions. We are what our brain is. The brain is a collection of neurons, which send

signals to each other in the form of spikes. Using principles of biological neurons, the 3rd generation of neural networks was developed by Maass (1997). This artificial spiking neuron model is known as the Spiking Neural Networks (SNN).

Several models have been proposed both at the macro and micro levels of abstraction. Microscopic models describe neural behaviours by the flow of ions through membrane channels. The Nobel Prize-winning Hodgkin-Huxley model (Hodgkin & Huxley, 1952) is perhaps the best-known of these. This type of model is one of the most biologically plausible but computationally expensive, and for this reason, it will be excluded from this research. At a macroscopic level, all the models are considered rather “black boxes”, meaning spikes are received or sent based on predefined conditions (Andrew, 2003), which are most relevant to this research. Here I have described some commonly used SNN models, such as Leaky Integrate and Fire (Abbott, 1999), Adaptive Exponential Integrate and Fire (aEIF) (Brette & Gerstner, 2005) and Izhikevich (Izhikevich, 2003). Each of these models corresponds to a different representation of biological neurons.

2.3.1 Leaky Integrate and Fire

Leaky Integrate-and-Fire (LIF) is one of the most popular models for simulating Spiking Neural Networks. According to Abbott (1999), LIF was first described in the paper by Lapicque (1907), which was published over a century ago. The model is more modernly described in Abbott (1999) and Paugam-Moisy & Bohte (2012). Due to its development, biological plausibility was sacrificed for a compact design, making it less computationally intensive.

Before Leaky Integrate-and-Fire (LIF) was developed, Integrate-and-Fire (I&F) neurons were proposed. While computationally simpler, the I&F model lacks the membrane resistance term that allows charge to decay over time. The leak was introduced to better reflect the biological behaviour of real neurons, where the membrane potential naturally decays towards rest in the absence of input (Abbott, 1999).

The equation for Integrate and Fire (I&F) is given by:

$$C \frac{du}{dt} = -\frac{1}{R}(u(t) - u_{\text{rest}}) + I(t) \quad (3)$$

In Equation 3, C is the membrane capacitance, R is the membrane resistance, $u(t)$ is the membrane potential, u_{rest} is the resting potential, and $I(t)$ is the input current. The membrane potential $u(t)$ is reset to u_{rest} when it reaches the threshold u_{th} .

The spike firing time $t^{(f)}$ is given by:

$$u(t^{(f)}) = \vartheta \text{ with } u'(t^{(f)}) > 0 \quad (4)$$

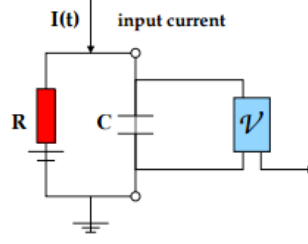


Figure 6: I&F circuit model (Paugam-Moisy & Bohte, 2012, p. 344)

The most widely used type of I&F neuron is the Leaky Integrate-and-Fire (LIF) model. The main difference between the LIF and Hodgkin-Huxley (HH) models is that, in the LIF neuron, every spike is a uniform event defined only by the time it appears. In contrast, in the HH model, the shape of the action potential is also taken into account (Paugam-Moisy & Bohte, 2012). In this model, a Resistor R is connected in parallel to Capacitor C as shown in figure Figure 6.

The membrane potential for LIF is described with the first-order differential equation:

$$\tau_m \frac{du}{dt} = u_{\text{rest}} - u(t) + RI(t) \quad (5)$$

Where $\tau_m = RC$ is the membrane time constant, which gives the voltage leakage using the threshold value shown in equation 4. Once a spike has reached it's maximum value, the membrane potential is reset to the resting potential u_{rest} . This can be seen in Figure 7.

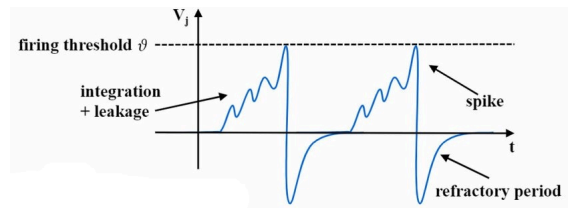


Figure 7: LIF spike pattern (Cai et al., 2022, p. 670)

2.3.2 Adaptive Exponential Integrate and Fire

Adaptive Exponential Integrate-and-Fire (aEIF) was first introduced in Brette & Gerstner (2005). Using the LIF as its base, three generalisations have been proposed -

- Firstly, to extend the current LIF to either quadratic or exponential,
- Secondly, adding the second variable for resonances or adaptation,

- And finally, changing the way the current is propagated to the neuron.

aEIF is based on two equations, starting with the I&F model, with adaptation defined by:

$$C \frac{dV}{dt} = f(V) - w + I \quad (6)$$

Where, C is the membrane capacitance, $f(V)$ is a function of passive properties and spiking mechanism, w is adaptation variable and I is synaptic current. Following Equation 6 shown in Fourcaud-Trocmé et al. (2003), for $f(V)$ combination of linear and exponential functions we get

$$f(V) = -g_L(V - E_L) + g_L \Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right)$$

Where, g_L is the leak conductance, E_L is the leak reversal potential, Δ_T is the slope factor, and V_T is the spike threshold. This is said to spike when the potential V rapidly increases to infinity.

Adaptation current w is given by:

$$\tau_w \frac{dw}{dt} = (a(V - E_L) - w)$$

Where, a is the sub-threshold adaptation conductance, τ_w is the adaptation time constant. The spike occurs when the potential V rapidly increases to infinity.

In the real world, a spike is triggered when the potential reaches $V_{\text{peak}} \approx 20$ mV, once a spike is formed the membrane potential resets to V_r and the adaptation variable is incremented: $V \leftarrow V_r$ and $w \leftarrow w + b$, where b controls the spike-triggered adaptation strength.

This model has been implemented as Very Large Scale Integration (VLSI) microchips in Millner et al. (2010) and Neuromorphic processors in Indiveri (2003) and Moradi & Indiveri (2014).

2.3.3 Izhikevich

Izhikevich's neuron models combine the biological plausibility of Hodgkin-Huxley (Hodgkin & Huxley, 1952) with the computational efficiency of Integrate-and-fire neurons (Izhikevich, 2003). According to Izhikevich (2003), his neuron model can reproduce the firing rate of thousands of cortical neurons in real time, depending on the parameters chosen.

The dynamics of Izhikevich models are based on two equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \quad (7)$$

$$\frac{du}{dt} = a(bv - u) \quad (8)$$

Where, v is the membrane potential, u is the membrane recovery variable with activation of Potassium⁺ and inactivation of Sodium⁺ ions, a is the time scale of the recovery variable u , b is the sensitivity of the recovery variable u to the sub-threshold fluctuations of the membrane potential v , and I is the input current.

There is a fixed threshold value of $v = 30\text{mV}$; when the membrane potential reaches this limit, the neuron spikes and resets u and v . The reset is given by:

$$\text{if } v \geq 30 \text{ mV then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}$$

Depending on the parameters a (decay rate of the membrane potential), b (sensitivity of membrane recovery), c (reset value of v) and d (reset value of u), many different types of cortical neurons can be modelled. Care should be taken when doing so, as even slight variations in the parameters can have a significant impact on how the models are produced. Figure 8 shows the different types of neurons that can be modelled using the Izhikevich model.

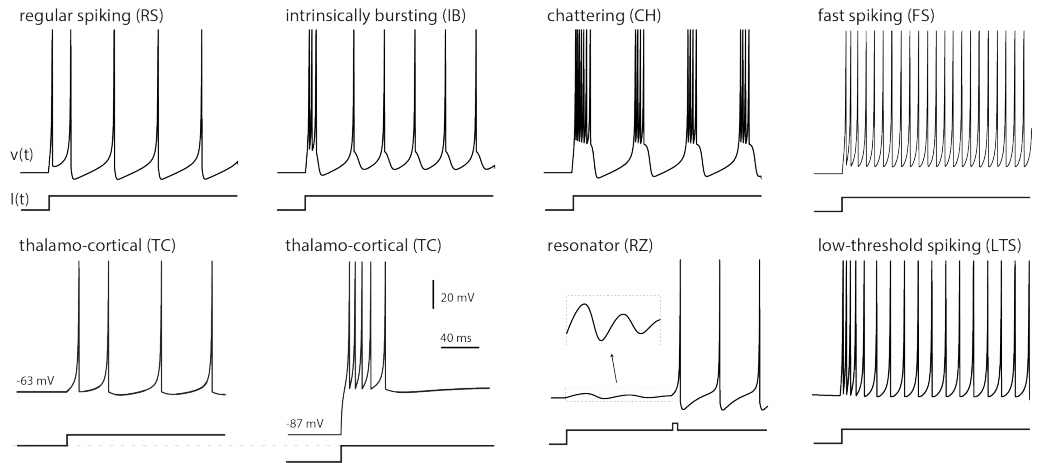


Figure 8: Izhikevich neuron models (Izhikevich, 2003, p. 1570).

2.3.4 Adaptive Leaky Integrate and Fire

Although the LIF model is computationally efficient, its fixed membrane dynamics cannot capture the diverse temporal patterns found in biological neurons, such as spike-frequency adaptation, bursting, and delayed responses. The Adaptive LIF (AdLIF) neuron, introduced by Bittar & Garner (2022), extends the LIF model by adding a learnable adaptation current w that modulates the membrane potential. Using forward-Euler discretisation with $\Delta t = 1 \text{ ms}$, the dynamics are:

$$u[t] = \alpha u[t - 1] + (1 - \alpha)(I[t] - w[t - 1]) - \theta s[t - 1] \quad (9)$$

$$w[t] = \beta w[t - 1] + (1 - \beta)au[t - 1] + bs[t - 1] \quad (10)$$

$$s[t] = \Theta(u[t] - \theta) \quad (11)$$

where u is the membrane potential, w is the adaptation current, I is the input current, s is the binary spike output, Θ is the Heaviside step function, and θ is the firing threshold. Four per-neuron parameters are learnable:

- $\alpha = \exp(-\Delta t/\tau_u)$ - controls the membrane leak rate;
- $\beta = \exp(-\Delta t/\tau_w)$ - controls the adaptation leak rate;
- a - couples the subthreshold membrane potential to the adaptation current;
- b - controls spike-triggered adaptation.

In the original AdLIF formulation (Bittar & Garner, 2022), these parameters are initialised uniformly within $\alpha \in [0.60, 0.96]$, $\beta \in [0.96, 0.99]$, $a \in [-1, 1]$, and $b \in [0, 2]$.

Compared to the standard LIF, the adaptation current w provides two forms of negative feedback. The spike-triggered term $bs[t - 1]$ increases w after each spike, reducing the membrane potential in subsequent timesteps and producing spike-frequency adaptation. The subthreshold coupling term $(1 - \beta)au[t - 1]$ allows adaptation to continuously track the membrane potential, enabling the neuron to respond differently depending on its recent history.

2.3.5 Constrained Adaptive Leaky Integrate and Fire

The Constrained AdLIF (cAdLIF), introduced by Deckers et al. (2024), modifies the AdLIF parameter constraints to improve stability when co-learning synaptic delays, weights, and adaptation. The equations remain the same (Equation 9 - Equation 11), but two key changes are made to the parameter ranges. First, the subthreshold coupling a is constrained to be positive ($a \in [0, 1]$), preventing the adaptation current from amplifying rather than dampening neural activity. Second, the membrane decay range is widened ($\alpha \in [0.36, 0.96]$) to allow faster membrane dynamics.

Table 1: Parameter ranges for AdLIF (Bittar & Garner, 2022) and cAdLIF (Deckers et al., 2024) neurons. The key difference is the positive constraint on a and the wider α range in cAdLIF.

Parameter	AdLIF Range	cAdLIF Range
α	[0.60, 0.96]	[0.36, 0.96]
β	[0.96, 0.99]	[0.96, 0.99]
a	$[-1, 1]$	$[0, 1]$
b	$[0, 2]$	$[0, 2]$

These constraints prevent parameter drift during training - without them, jointly optimising delays, weights, and neuron dynamics can lead to gradient instability and degenerate solutions (Deckers et al., 2024). The separation of timescales ($\tau_u < \tau_w$) is biologically motivated: membrane integration operates on fast timescales, while adaptation reflects slower processes such as calcium-dependent potassium currents.

In addition to the constrained neuron parameters, Deckers et al. (2024) introduces per-synapse learnable delays, allowing each connection to learn an optimal temporal offset. This enables the network to explicitly correlate temporally distant spike trains, a capability absent in standard feed-forward SNNs with fixed or no delays.

2.4 SNN Learning Rules

Learning rules adjust synaptic connections between neurons according to specific rules and parameters. Here, I describe one of the most commonly used algorithms, Spike Timing-Dependent Plasticity (STDP) (Sjöström & Gerstner, 2010).

2.4.1 Spike Timing Dependent Plasticity (STDP)

STDP is an unbalanced temporal form of the Hebbian rule, which has a temporal interrelationship between a pre- and a post-synaptic neuron. It is widely believed that synaptic plasticity underlies learning and information storage in the brain. In STDP, there are two types of synaptic results: Long-Term Potentiation (LTP) and Long-Term Depression (LTD). LTP is when a spike in a pre-synaptic neuron occurs a few milliseconds before a post-synaptic neuron's potential. LTD is when a spike in a pre-synaptic neuron occurs after a post-synaptic potential. So, STDP can be defined as the relative timing between the pre-synaptic spike and the post-synaptic potential (Sjöström & Gerstner, 2010).

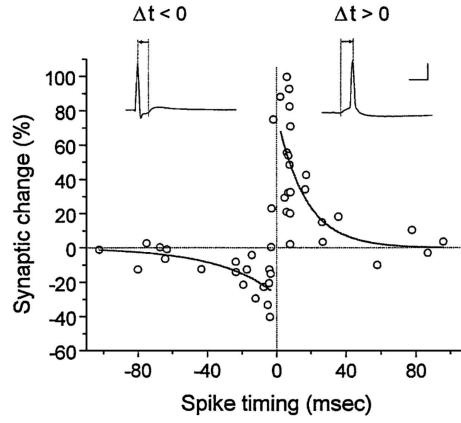


Figure 9: An example of change in excitatory post-synaptic potential amplitude as a function of the time difference $\Delta t = t_{\text{post}} - t_{\text{pre}}$ between pre- and post-synaptic spike. Pairing protocol: 60 spike pairs at 1 Hz (Senn & Pfister, 2015, p. 3263).

Let's consider Δw_j as the change in weight of a synapse from pre-synaptic neuron j ; and let's consider the pre-synaptic spike arrival time for neuron j as t_j^f , where $f = 1, 2, 3, \dots, N$, which are the counts of pre-synaptic spikes. And let's consider the firing time for the post-synaptic neuron as t^n , where $n = 1, 2, 3, \dots, N$. So, the weight change in the network Δw_j with a stimulation protocol (that stimulates action potential in a neuron) is given by:

$$\Delta w_j = \sum_{f=1}^N \sum_{n=1}^N W(t^n - t_j^f) \quad (12)$$

Where, $W(x)$, in which $x = W(t^n - t_j^f)$, is one type of STDP function, which is also known as learning window, shown in the Figure 9. Popular choice for STDP functions are:

$$W(x) = A_+ \exp\left(\frac{-x}{\tau_+}\right) \text{ for } x > 0$$

$$W(x) = -A_- \exp\left(\frac{x}{\tau_-}\right) \text{ for } x < 0$$

Here A_+ and A_- are the dependent current value of the weights w_j ; τ_+ and τ_- are the time constraints, which are in the order of $10ms$ each.

STDP is already implemented in SpiNNaker for hardware acceleration (Diehl & Cook, 2014).

2.4.2 Surrogate Gradient Learning with Backpropagation (SGLBP)

Introduced in Neftci et al. (2019), it is a type of learning rule for SNNs based on backpropagation. Compared to a traditional backpropagation, which uses differentiable activation functions, an SNN uses non-differentiable threshold functions. Surrogate gra-

dients replace the non-differentiable gradient with a continuous approximation, enabling gradient-based optimisation.

A spike is generated using the Heaviside step function, which is denoted by Θ and is given by Equation 13

$$\Theta(v_t - v_{th}) = \begin{cases} 1 & \text{if } v_t \geq v_{th} \\ 0 & \text{if } v_t < v_{th} \end{cases} \quad (13)$$

Where v_{th} is the threshold, and v_t is the membrane potential. The derivative of the Heaviside step function is the Dirac delta function, which is not defined at $v_t = v_{th}$ and its gradient is almost always 0. To overcome this, a surrogate gradient is used, which is a continuous approximation of the derivative of the Heaviside step function. When a standard backpropagation is used, it will fail if we use

$$\frac{\delta z_t}{\delta v_t} \quad (14)$$

Where z_t is the spike generated at time t . In this case, to make gradient-based learning work, a Surrogate Gradient approximation is used. This could be Sigmoid, Fast Sigmoid, ArcTan or any smooth approximation around zero. In this study, we have used the arctangent surrogate, popularised for deep SNN training by Fang et al. (2021), as the Surrogate Gradient. This is given by:

$$\psi(x) = \arctan(\alpha x)$$

The derivative of the ArcTan function is given by:

$$\frac{d}{dx} \arctan(\alpha x) = \frac{\alpha}{1 + (\alpha x)^2}$$

Hence, the arctan surrogate gradient for a spiking neuron can be written as:

$$\frac{\delta z_t}{\delta v_t} \approx \sigma_{\arctan}(v_t - v_{th}) = \frac{\alpha}{1 + [\alpha(v_t - v_{th})]^2}$$

where α is a hyperparameter controlling the “sharpness” around the threshold. This can be used in the backpropagation algorithm to train the network.

2.5 SNN Supervised Learning

SNN learning is a complicated process due to the network’s asynchronous data processing, requiring a high-performance software or hardware environment for simulations to be performed. Also, a recurrent, self-referential network topology is used in SNNs. Despite the fact that these difficulties arose, some learning algorithms have been proposed. All these techniques perform similarly by making changes to the synapse (synaptic

weights); they simulate variations in the amount of ions passing from the pre-synaptic neuron to the post-synaptic neuron.

Some existing learning rules are STDP (see Section 2.4.1), ReSuMe (Ponulak, 2005), SDSP (Fusi et al., 2000), Tempotron (Gütig & Sompolinsky, 2006), Chronotron (Florian, 2012), INST (Gardner & Grüning, 2016), and FILT (Gardner & Grüning, 2016). In this section, we will concentrate on two primary algorithms, Spike Pattern Association Neuron (SPAN) and Dynamic Evolving Spiking Neural Network (deSNN)

2.5.1 Spike Pattern Association Neuron (SPAN)

SPAN was introduced in Mohemmed et al. (2012). It states that the synaptic weights of each neuron are iteratively updated to develop a desired input-output spike-train mapping for that neuron. It is an extension of the Widrow-Hoff rule with a modification for spike-based neural communication. The input, actual and desired output spike trains are convolved with an alpha kernel in order to convert them to a continuous and comparable function. The difference between the actual and desired outputs is taken and cross-multiplied with the input to each synapse, forming the change in synaptic weights. This weight is applied, and the next training sample is presented, creating an iterative learning rule.

2.5.2 Dynamic Evolving Spiking Neural Network (deSNN)

deSNN (Dhoble et al., 2012) evolved from eSNN (Wysoski et al., 2006). The problem with eSNN is that it uses a rank order encoding method. The weight adjustments in this approach are made only once, which is only capable of classifying static data but is not very efficient for Spatio- and Spectro-Temporal Data (SSTD). Due to this problem, the weights must be tuned for each training session; for this reason, STDP was used in addition to Rank Order (RO) learning, which is based on the order of incoming spikes.

deSNN states that for each vector represented in a system, a new neuron should be evolving. The synaptic weights required to cause this neuron to spike upon presentation of its defined class pattern are calculated, and this neuron is added to the population of neurons representing its specified class. This neuron is then compared to the other neurons in the class. If that neuron's synaptic weight is within some specified distance of another neuron in the same class, the new neuron is merged with the existing neuron, and their weights are averaged.

In this way, any one neuron may be responsible for spiking when its particular data pattern is presented, while a population as a whole evolves to represent a class.

Algorithm 1: Pseudocode for Dynamic Evolving Spiking Neural Network (deSNN) algorithm (Dhoble et al., 2012).

```

input: deSNN parameters
output: deSNN()
1 forall input  $i$  represented as AER do:
2   create a new output neuron  $j$ ;
3   calculate values of connection weights using RO learning rule;
4   adjusting connections weights  $w_j$  for consecutive spikes with STDP;
5   calculate PSP;
6   calculate threshold values;
7   if  $w_j$  is similar to existing  $w_j$  then:
8     | merge neurons;
9   end
10 end

```

2.5.3 Multi Layer Feed Forward SNN

A feed-forward neural network is the simplest form of network representation and flow of learning. According to Minsky & Papert (1969), a two-layer neural network was shown not to be feasible in terms of useful representation outside a particular class. In simple terms, it is not viable for complicated classes, those that are not linearly separable. To obtain a correct approximation of the class, Hornik et al. (1989) developed a multilayer feed-forward network that could handle more complex problems. Figure 10 shows a representative multilayer feed-forward topology, where each input unit fans out to every neuron in a stack of hidden layers before reaching the output layer.

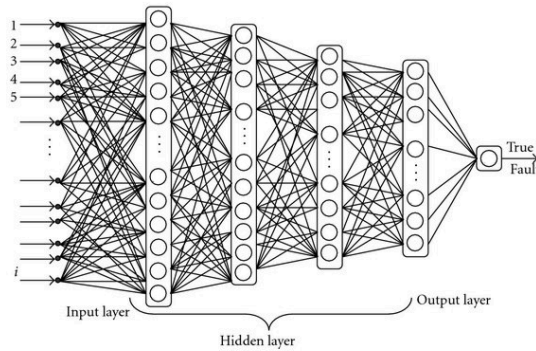


Figure 10: A structure of multilayered neural network.

2.5.4 NeuCube

The NeuCube (Kasabov, 2014) is an SNN architecture designed for modelling spatio-temporal brain data such as EEG. It consists of three main components: an input encoding layer that converts continuous signals into spike trains, a 3D recurrent SNN reservoir (the “cube”) whose neuron positions are mapped to brain regions using the Talairach brain atlas (Talairach & Tournoux, 1988), and an output classification layer. The reservoir neurons are initialised with connectivity based on spatial proximity, reflecting the brain’s structural organisation. During training, the reservoir learns temporal patterns through STDP, while the output layer is trained using deSNN (Section 2.5.2) or other supervised learning rules. This architecture makes NeuCube particularly well-suited for classifying spatio-temporal data, such as brain activity patterns in response to stimuli.

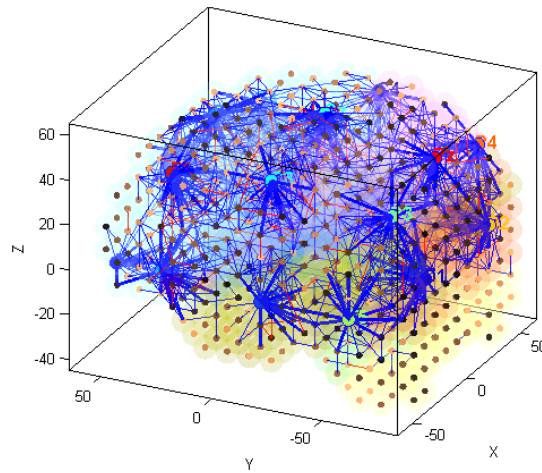


Figure 11: Connectivity generated before performing unsupervised learning for EEG data in NeuCube (Capecci et al., 2016, p. 1364).

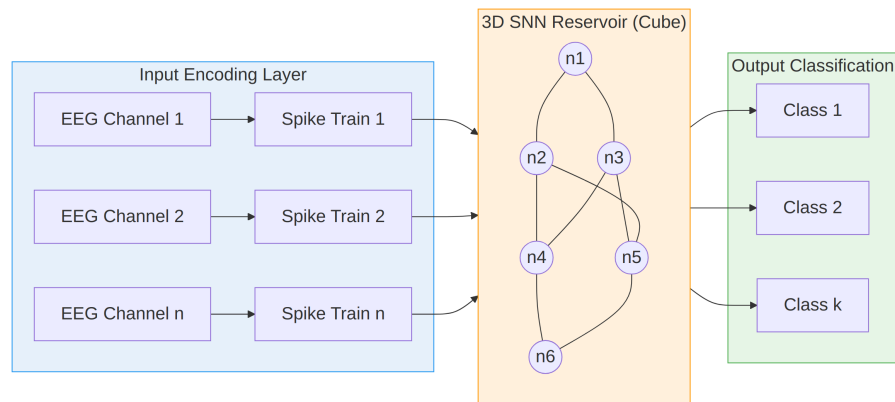


Figure 12: NeuCube three-component architecture: input encoding converts EEG channels to spike trains, a 3D SNN reservoir with recurrent connectivity learns temporal patterns via STDP, and an output layer performs classification.

Figure 11 shows the internal reservoir connectivity of NeuCube, where neurons are spatially arranged according to the Talairach brain atlas and connected based on proximity. Figure 12 illustrates the overall three-stage pipeline: continuous EEG signals are first encoded into spike trains, then processed by the 3D reservoir, which learns temporal patterns through STDP, and finally classified by the output layer.

2.5.5 Deep Learning in SNN

Deep learning has changed the way we model our data. The precision with which it can learn multi-disciplinary data has been a game-changer. Among the most widely used deep learning architectures is the Convolutional Neural Network (CNN) (LeCun et al., 1998), which has produced state-of-the-art results in image classification and object detection. Deep learning has been extensively applied in traditional artificial neural networks.

SNN doesn't use the same learning rules as traditional neural networks do. The learning rules are based on the spike timing dependent plasticity (STDP) (Sjöström & Gerstner, 2010), which is a form of Hebbian learning; they are also based on the relative timing of pre- and post-synaptic spikes. This makes it difficult to train deep SNNs, as the backpropagation algorithm used in traditional neural networks cannot be directly applied to them.

However, some techniques have been proposed to train deep SNNs. One of the most popular techniques is the Surrogate Gradient Learning with Backpropagation (SGLBP) (Section 2.4.2). This technique uses a surrogate gradient to approximate the gradient of the non-differentiable threshold function used in SNNs. This enables the backpropagation algorithm to train deep SNNs.

Deep learning networks consist of multiple layers of neurons, where each layer is responsible for learning a particular feature of the input data. The first layer learns the low-level features, such as edges and corners, while the subsequent layers learn higher-level features, such as shapes and objects. The last layer is responsible for making the final decision based on the learned features.

Figure 35 in Chapter 5 and Figure 63 in Chapter 6 are some of the examples of deep learning in SNNs used in this study.

2.6 Optimisation

Optimisation techniques help in finding the best possible solution to a problem. In SNN, optimisation is performed to find the best possible synapse weights by tuning hyperpa-

rameters. Although the tuning can be done manually and the model trained after each change, this approach is not sufficiently feasible. To overcome these problems, various optimisation techniques are used. Let's look at two optimisation techniques: Decoupled Weight Decay Regularisation and Tree-structured Parzen Estimator.

2.6.1 Decoupled Weight Decay Regularisation

Also known as AdamW, it is a variant of the Adam optimiser (Kingma, 2014) that decouples the weight decay from the optimisation steps. This technique was introduced in Loshchilov & Hutter (2017). The Adam optimiser is a popular optimisation algorithm that combines the benefits of the AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman, 2012) algorithms. It uses the first and second moments of the gradients to adapt learning rates for each parameter.

AdamW update rule (Loshchilov & Hutter, 2017) is given by:

$$\theta_t = \theta_{t-1} - \eta_t \left(\alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} + \lambda \theta_{t-1} \right)$$

with bias-corrected moments

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Where θ_t is the parameter at time t , α is the base learning rate, η_t is the schedule multiplier supplied by an external scheduler (e.g.

cosine annealing or one-cycle), m_t and v_t are the first and second moments of the gradients with their bias-corrected counterparts $\hat{m}_t$ and $\hat{v}_t$, β_1 and β_2 are the moment decay rates (defaults 0.9 and 0.999), ε is a small constant to prevent division by zero, and λ is the weight decay coefficient. The crucial property of AdamW relative to Adam is that the weight-decay term $\lambda \theta_{t-1}$ is decoupled from the gradient step but still scaled by the same schedule multiplier η_t .

2.6.2 Tree-structured Parzen Estimator (TPE)

The Tree-structured Parzen Estimator (TPE) is a Bayesian optimisation method that efficiently searches for optimal hyperparameters in machine learning models. It was introduced by Bergstra et al. (2011) and is particularly effective for high-dimensional, non-convex, and expensive-to-evaluate objective functions. Unlike traditional grid search or random search, which inefficiently explore hyperparameter spaces, TPE models the search space using probabilistic density estimation, making it more efficient and adaptive (Watanabe, 2023).

TPE works by maintaining two density models over the hyperparameter space. Given a set of observed trials and a performance threshold y^* (typically the best quantile of observed values), TPE constructs:

- $l(x)$: a density model over hyperparameter configurations that produced *good* results ($f(x) < y^*$)
- $g(x)$: a density model over configurations that produced *poor* results ($f(x) \geq y^*$)

The next candidate hyperparameter configuration is selected by maximising the ratio $l(x)/g(x)$, which approximates the Expected Improvement (EI) acquisition function. This directs the search towards regions of the hyperparameter space where good performance has been observed, while avoiding regions associated with poor results. Figure 13 illustrates this iterative process.

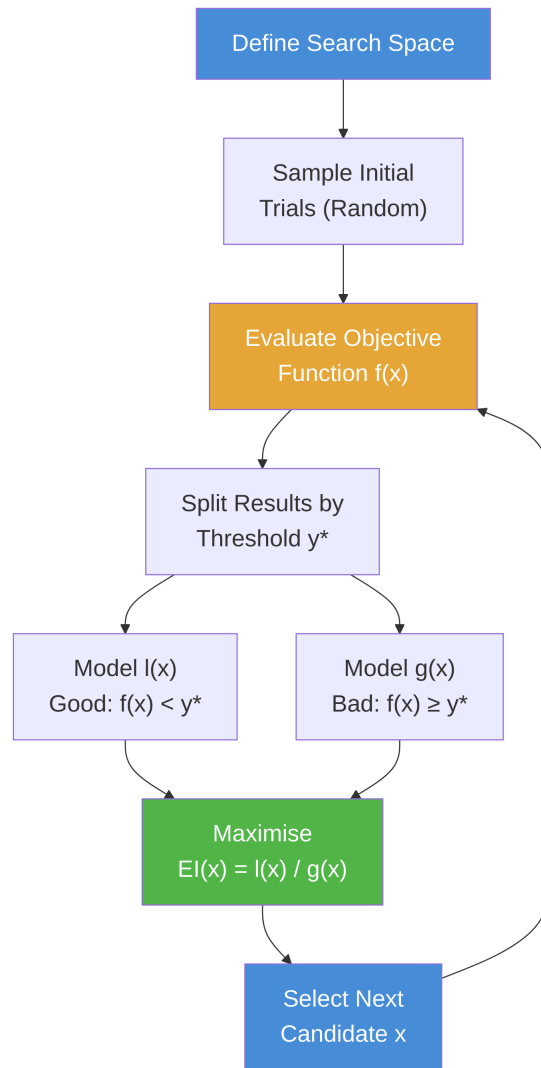


Figure 13: The TPE optimisation loop. After initial random sampling, observed results are split by a performance threshold into good and bad groups. Density models $l(x)$ and $g(x)$ are fitted to each group, and the next candidate is chosen by maximising $l(x)/g(x)$.

In this study, TPE is used via the Optuna framework (Akiba et al., 2019) to optimise the hyperparameters of the SNN architectures in both the CIFAR-10 (Chapter 5) and Speech Commands (Chapter 6) case studies. Optuna implements TPE with multivariate kernel density estimation and automated trial pruning, which stops underperforming trials early to reduce computational cost.

2.7 Data Compression

Data compression refers to the process of reducing the size of data while retaining its essential information. This enables more efficient storage, transmission, and processing of data. In the modern era, data compression plays a crucial role in managing the massive amounts of data generated by applications like video streaming, cloud computing, and artificial intelligence.

Compression achieves its goal by removing redundancies and encoding data more compactly. It is widely used across various domains, including multimedia, file storage, and communication systems.

2.7.1 Information Entropy and Source Coding

The theoretical foundation for data compression was established by Shannon (1948), who introduced the concept of *information entropy*, which quantifies the average amount of information produced by a source. A *symbol* in this context is one element of the source's alphabet - the atomic unit the source emits and that the compressor encodes; for text it is a character, for a binary spike train it is a bit (0 or 1), and for a multi-level DyNED encoder output it is one of the available quantisation levels. For a discrete source with symbols $S = \{s_1, s_2, \dots, s_n\}$ occurring with probabilities $\{p_1, p_2, \dots, p_n\}$, the entropy H is defined as:

$$H = - \sum_{i=1}^n p_i \log_2 p_i \quad (15)$$

Entropy represents the minimum average number of bits required to encode each symbol from the source. A source where all symbols are equally likely has maximum entropy, while a highly predictable source has low entropy.

Shannon's *source coding theorem* (also known as the noiseless coding theorem) states that the average code length of an optimal lossless compression scheme cannot be shorter than the entropy of the source (Shannon, 1948). In other words, entropy defines a fundamental lower bound on compression: no lossless encoding can represent data using fewer bits per symbol than H on average. Huffman coding (Section 2.7.3) approaches

this limit, achieving an average code length within one bit of the entropy for any source distribution.

This theoretical framework is directly relevant to the DyNED_c compressor developed in this thesis, which combines run-length encoding with Huffman coding on the resulting run lengths. The sigma-delta quantisation in DyNED produces spike trains with highly non-uniform symbol distributions (dominated by zeros), resulting in low entropy and enabling high compression ratios.

2.7.2 Run-Length Encoding (RLE)

RLE is the simplest form of lossless data compression, which replaces consecutive but identical data values with a symbol and a count. For example, the string “AAAABBBCC” can be compressed as “4A3B2C”. RLE is efficient for data with long sequences of repeated values, such as images with large areas of uniform colour (Salomon, 2002).

2.7.3 Huffman Coding

First introduced by Huffman (1952), Huffman coding is a lossless data compression algorithm. It assigns variable-length codes or patterns to symbols so that the most frequent symbols have the shortest codes, and the least frequent have the longest codes. This technique is widely used in file compression formats like ZIP and JPEG.

Let's consider a set of symbols $S = \{s_1, s_2, \dots, s_n\}$ that appears with a probability $\{p_1, p_2, \dots, p_n\}$, where $\sum_{i=1}^n p_i = 1$.

When each symbol s_i is assigned a code with length l_i bits, the expected code length L of the entire message is given by:

$$L = \sum_{i=1}^n p_i l_i \quad (16)$$

The goal of Huffman coding is to construct a prefix-free code that minimises L . For an optimal code, the most frequent symbols should have the shortest codes, and the least frequent symbols should have the longest codes.

2.7.4 Arithmetic Coding

Arithmetic coding (Howard & Vitter, 1994; Rissanen & Langdon, 1979; Witten et al., 1987) is a lossless entropy-coding method that, unlike Huffman coding, does not assign a separate codeword to each input symbol. Instead, the entire input message is represented by a single half-open subinterval of the unit interval $[0, 1)$. The current interval is progressively narrowed in proportion to each symbol's probability as the message is

processed, and the final compressed output is just enough bits to identify the resulting subinterval unambiguously (Witten et al., 1987).

For a source with symbols $S = \{s_1, s_2, \dots, s_n\}$ and probabilities $\{p_1, p_2, \dots, p_n\}$, the message $s_{i_1} s_{i_2} \dots s_{i_k}$ is encoded by progressively narrowing the unit interval. After all k symbols have been processed, the current interval has a width:

$$\prod_{j=1}^k p_{i_j} \quad (17)$$

The number of bits required to identify this interval unambiguously is therefore on the order of:

$$-\sum_{j=1}^k \log_2 p_{i_j} \quad (18)$$

Which approaches Shannon's entropy bound H from Equation 15 as the message lengthens (Howard & Vitter, 1994).

This is the key practical distinction from Huffman coding (Section 2.7.3). Huffman emits an integer number of bits per symbol, so it is optimal only when every symbol probability is an integral power of $1/2$; outside that case, Huffman acquires up to one extra bit per symbol, with the worst case being a source dominated by a single high-probability symbol (Witten et al., 1987). Arithmetic coding has no per-symbol integer-bit floor and can spend, on average, fractional bits per symbol, attaining a code length arbitrarily close to the entropy bound at the cost of higher computational and arithmetic-precision requirements (Howard & Vitter, 1994). Variants such as range coding and asymmetric numeral systems share the same theoretical basis and are used in contemporary image and video codecs, including H.264/H.265 (via CABAC) and JPEG 2000.

2.7.5 Types of Data Compression

Data compression techniques can be broadly categorised into two types - Lossless and Lossy Compression.

2.7.5.1 Lossless Compression

Lossless compression ensures that all original data can be perfectly reconstructed. It is used in scenarios where precision is critical, such as text files, medical imaging, and financial data.

Common techniques:

- **Huffman Coding:** Assigns shorter codes to frequent symbols (Huffman, 1952).

- **Arithmetic Coding:** Encodes entire messages into a single number based on probabilities (Witten et al., 1987).
- **Lempel-Ziv (LZ):** Forms the basis of popular formats like ZIP and PNG (Ziv & Lempel, 1977).

2.7.5.2 Lossy Compression

Lossy compression reduces file size by discarding less perceptible information, prioritising compression ratios over perfect fidelity. It is commonly used in multimedia applications such as images, audio, and video.

Such as:

- **JPEG:** Image compression using Discrete Cosine Transform (DCT) (Wallace, 1992).
- **MP3:** Audio compression that removes inaudible frequencies (Brandenburg, 1999).
- **H.264:** Video compression standard widely used in streaming services (Wiegand et al., 2003).

2.7.6 Applications in Modern Technology

Data compression is integral to various modern technologies:

- **File Compression:** Formats like ZIP, RAR, and 7z reduce file sizes for storage and sharing.
- **Multimedia Compression:** JPEG for images, MP3 for audio, and H.264/HEVC for videos to enable efficient streaming and storage.
- **Web Communication:** Protocols like HTTP/2 and HTTP/3 use Gzip and Brotli to speed up website loading (Bishop, 2022).
- **Cloud Computing:** Reduces storage costs and accelerates data transfers.

2.7.7 Biological Relevance

Studies suggest the brain employs compression-like mechanisms to store, retrieve, and process information efficiently. For instance:

- **Sparse Coding:** The visual cortex activates only a few neurons to represent complex images (Olshausen & Field, 1997).
- **Predictive Coding:** The brain minimises redundant processing by predicting sensory inputs (Rao & Ballard, 1999).
- **Memory Consolidation:** During sleep, memories are compressed and reorganised for efficient storage (Walker & Stickgold, 2006).

These mechanisms demonstrate parallels between biological and artificial data compression, inspiring new computational approaches.

2.8 Platforms

Dedicated neuromorphic hardware platforms can offer significant performance and energy-efficiency advantages over software-based SNN simulation, particularly for large-scale, real-time workloads, due to their event-driven architecture and hardware-level parallelism (Furber et al., 2014). In this section, I introduce two such hardware platforms that can be used either in a hardware cluster or standalone: SpiNNaker and Raspberry Pi.

There are many SNN simulators; some that are well known include Brian (Goodman & Brette, 2009), CARLsim3 (Beyeler et al., 2015), ANNarchy (Vitay et al., 2015), NEST (Gewaltig & Diesmann, 2007), PCSIM (Pecevski et al., 2009), NEURON (Carnevale & Hines, 2006), PyNN (Davison et al., 2008) and snnTorch (Eshraghian et al., 2023a). These are software-based simulators that can optionally leverage GPU acceleration. Many of them support learning mechanisms: snnTorch uses surrogate gradient descent for supervised training, while Brian, NEST, and CARLsim support spike-timing-dependent plasticity (STDP) and other biologically inspired learning rules. ANNarchy supports a range of neuron models beyond LIF, including Izhikevich, adaptive exponential integrate-and-fire, and Hodgkin-Huxley. However, these software simulators lack the real-time, low-power characteristics of dedicated neuromorphic hardware.

2.8.1 Raspberry Pi

Raspberry Pi is a tiny credit-card-sized computer that runs the Linux operating system. Its heart is a Raspberry Pi, a Broadcom BCM2835 system-on-chip that provides this system with a Graphical Processing Unit (GPU), general-purpose processing, and Random Access Memory (RAM) (Upton & Halfacree, 2014).

2.8.2 General Purpose Computing on Graphical Processing Unit (GPGPU)

First termed by Mark Harris, recognised in critical publications such as “A Survey of General-Purpose Computation on Graphics Hardware” (Owens et al., 2007) and “hiCUDA: High-level GPGPU Programming” (Han & Abdelrahman, 2011), GPGPU refers to the use of Graphics Processing Units (GPUs) for general-purpose computing, not limited to graphics rendering. GPUs are inherently parallel in their architecture, unlike the serial instruction set of Central Processing Units (CPUs). This parallel nature allows GPUs

to efficiently handle complex computations, particularly those involving floating-point operations.

NVIDIA capitalised on this capability by architecting the Compute Unified Device Architecture (CUDA), a proprietary framework that harnesses the power of GPUs for general-purpose computing. Introduced in November 2006 alongside the GeForce 8800 GTX (G80 architecture), CUDA represents a significant milestone in the history of GPGPU (Lindholm et al., 2008).

Following Nvidia's CUDA, the Khronos Group introduced an open standard alternative, the Open Computing Language (OpenCL), with the 1.0 specification released in December 2008. OpenCL is a cross-platform heterogeneous computing accelerator compatible with GPUs and CPUs, Field-Programmable Gate Arrays (FPGAs), and various custom hardware platforms. Notably, Nvidia's graphics cards also support OpenCL (Bensalem et al., 2023).

This study uses PyTorch (Paszke et al., 2019), a Python-based library that supports GPGPU. PyTorch's flexibility and ease of use make it an ideal choice for leveraging GPU capabilities across a wide range of applications, from neural network training to advanced data processing.

2.8.3 Neuromorphic Computing

Neuromorphic hardware refers to computational systems designed to mimic the structure and function of the brain (Furber et al., 2014). Unlike conventional von Neumann architectures, neuromorphic processors use event-driven, asynchronous computation that is well-suited to spiking neural networks. Notable neuromorphic platforms include:

- **SpiNNaker** (Furber et al., 2014) is based on ARM cores and is massively parallel, designed to simulate spiking neurons in real time. The full SpiNNaker machine at the University of Manchester contains over one million cores and can model up to one billion neurons.
- **IBM TrueNorth** (Merolla et al., 2014) is a digital neuromorphic chip containing 4,096 neurosynaptic cores, capable of simulating one million neurons and 256 million synapses while consuming only 70 milliwatts of power.
- **Intel Loihi** (Davies et al., 2018) is a neuromorphic research chip featuring 128 neuromorphic cores with on-chip learning capabilities, supporting programmable synaptic learning rules including STDP. Its successor, **Loihi 2** (Orchard et al., 2021), introduced fully programmable neuron models with graded spikes and approximately 10x performance improvement over the original.

- **BrainScaleS-2** (Pehle et al., 2022) takes a mixed-signal approach, using analogue circuits to emulate neural dynamics approximately 1,000 times faster than biological real time, with embedded digital processors for on-chip plasticity.

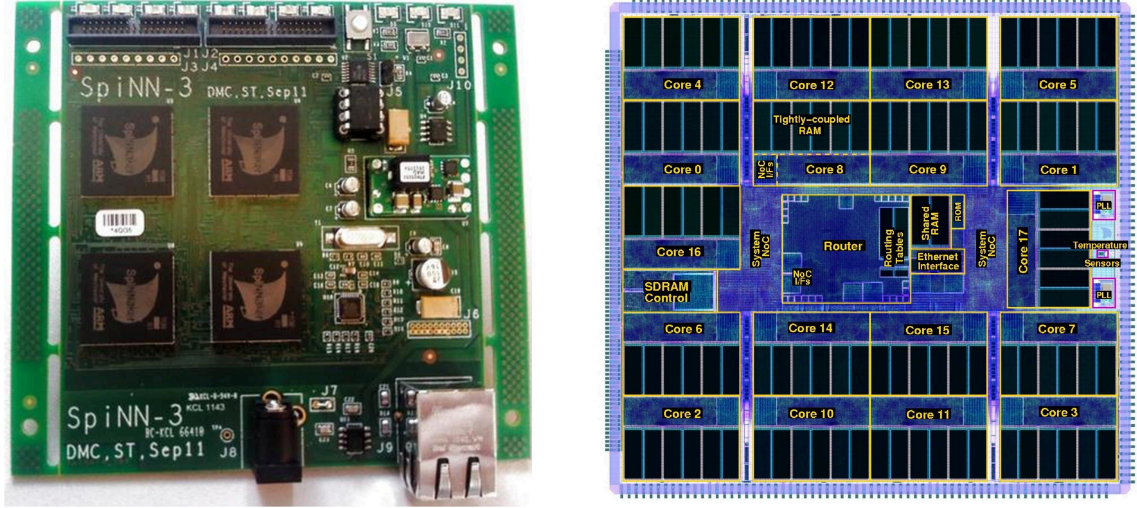


Figure 14: SpiNN-3 board and one of its processor's architecture.

Figure 14 shows a smaller SpiNN-3 board with four ARM9 chips, each with 18 cores (17 usable, 1 reserved as monitor processor), giving an estimated capacity of approximately 17,340 neurons (Furber et al., 2014).

2.9 Signal Processing

Signal processing is a field within science and applied mathematics that involves filtering, extracting, or manipulating signals, which may be based on discrete-time or continuous (analogue) systems (Kuo & Gan, 2005; Schlichthärle, 2011). Generally, signal processing is divided into two main categories: Analogue Signal Processing (ASP) and Digital Signal Processing (DSP). ASP concerns the manipulation of continuous signals, while DSP focuses on discrete signals. In this section, we will look at several prevalent signal-processing techniques that are employed within the scope of this thesis.

2.9.1 Analogue Signals

Analogue signals are continuous, time-varying signals represented by mathematical functions of time, usually depicted as continuous waveforms like sine waves, though they can take many other forms. These signals are continuous in both time and value, meaning they are defined at every point within a specific range and can take on any value within their amplitude range; in contrast, digital signals are discrete in both time and value. Common characteristics of analogue signals include amplitude, frequency, and phase,

with periodic signals such as sine waves repeating their values at regular intervals over time. They find wide application in technology, especially in traditional broadcasting and analogue electronics, and as the initial form of real-world data before it is converted into a digital format for modern systems. However, analogue signals are more susceptible to noise and degradation over long distances and time, which is one reason for the transition to digital formats that offer precision and reproducibility. Despite this, analogue signals are fundamental in understanding real-world phenomena. They remain vital because they naturally represent varying data, with all digital systems interfacing with the real world through some form of analogue-to-digital or digital-to-analogue conversion. The study of analogue signals is crucial for designing and working with various technologies and remains a foundational concept in electronics, communications, and signal processing (Smith, 1999).

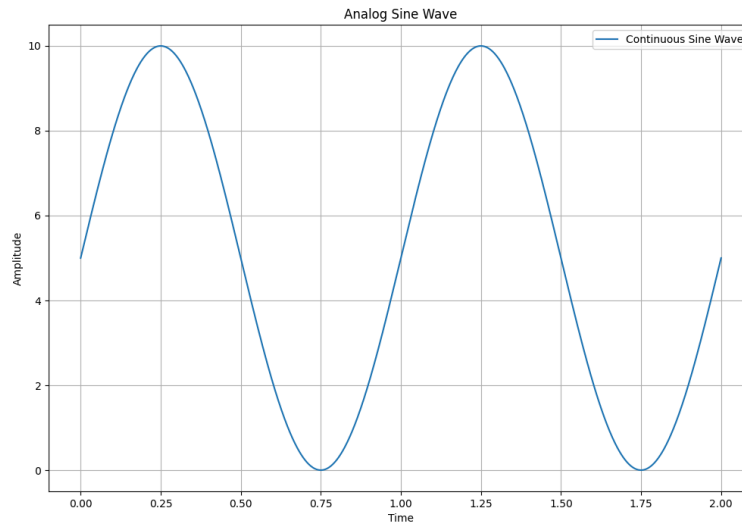


Figure 15: A two second analogue signal of a sine wave above origin.

Figure 15 shows a two-second analogue signal of a sine wave with an amplitude of $A = 5$ at a frequency $f = 1$, the cycle (period) is given by:

$$T = \frac{1}{f}$$

And the sine wave is given by:

$$s(t) = A \sin(2\pi ft)$$

Where $s(t)$ is the amplitude of the sine wave at time t . A is the peak amplitude. $2\pi ft$ is the oscillation frequency.

2.9.2 Digital Signals

Digital signals are discrete, time-varying signals represented by a sequence of discrete values, often depicted as a series of square or rectangular pulses. Unlike analogue signals, which are continuous in time and amplitude, digital signals are defined only at specific intervals and can assume only a finite set of values. Characteristics of digital signals include logical levels, frequencies, and phases, with some forms, such as binary square waves, repeating in a regular cycle. They find extensive applications in modern electronics, communications, and digital systems due to their robustness against noise, ease of processing, and precise reproducibility. Digital signals are fundamental in modern technology and are typically interfaced with the real world through analogue-to-digital or digital-to-analogue conversion. The study of digital signals is crucial for designing and working with various technologies, providing a foundation for electronics, communications, and signal processing (Smith, 1999).

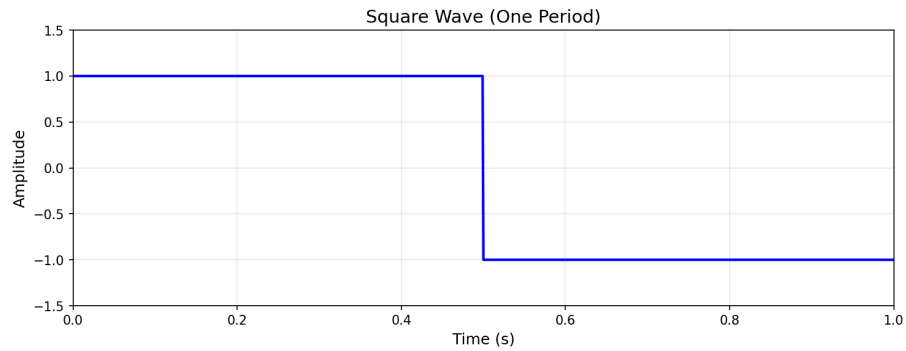


Figure 16: A simple one-cycle square wave

Figure 16 shows a simple one-cycle square wave with a frequency of $f = 1$ and a duty cycle of 50%, is given by:

$$s(t) = \text{sgn}(\sin(2\pi ft))$$

Where $s(t)$ is the value of the square wave at time t . $\sin(2\pi ft)$ is the sine function, where f is the frequency. sgn is the signum function, which returns the sign of the argument and, it is given by:

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

2.9.3 Quantisation

Quantisation is a technique in digital signal processing in which a continuous range of values is converted into a finite range of discrete values (Gray & Neuhoﬀ, 1998). This process is essential for transforming analogue signals into digital signals, a fundamental step in applications such as audio, video, and image processing. A quantiser, the function or device performing this task, maps input values from a large or continuous set to a smaller, discrete set. The discrete values resulting from the quantisation process are known as quantisation levels, whose number is often determined by the resolution or number of bits used in the digital representation. For example, an 8-bit quantiser can produce 256 distinct levels. During this conversion, quantisation error or noise typically occurs, representing the difference between the original analogue value and the quantised digital value. Despite this inherent error, quantisation is critical for the eﬃcient storage, processing, and transmission of digital information, enabling continuous signals such as sound to be represented and manipulated digitally.

For a uniform quantisation, the equation is given by:

$$Q(x) = \min_val + \Delta \cdot \left\lfloor \frac{x - \min_val}{\Delta} \right\rfloor$$

Where $Q(x)$ is the quantised value of the original signal value x , $\min_val$ is the minimum value of the input signal's range. Δ is the quantisation step size, which is calculated as $\Delta = \frac{\max_val - \min_val}{N}$, where $\max_val$ is the maximum value of the input signal's range, and N is the number of quantisation levels. Furthermore, $\left\lfloor \frac{x - \min_val}{\Delta} \right\rfloor$ calculates how many quantisation steps away x is from $\min_val$, and the floor function rounds this value down to the nearest integer. Figure 17 visualises this mapping, with the continuous input on the horizontal axis and the resulting staircase of equally spaced output levels - and the residual quantisation error at each step - on the vertical axis.

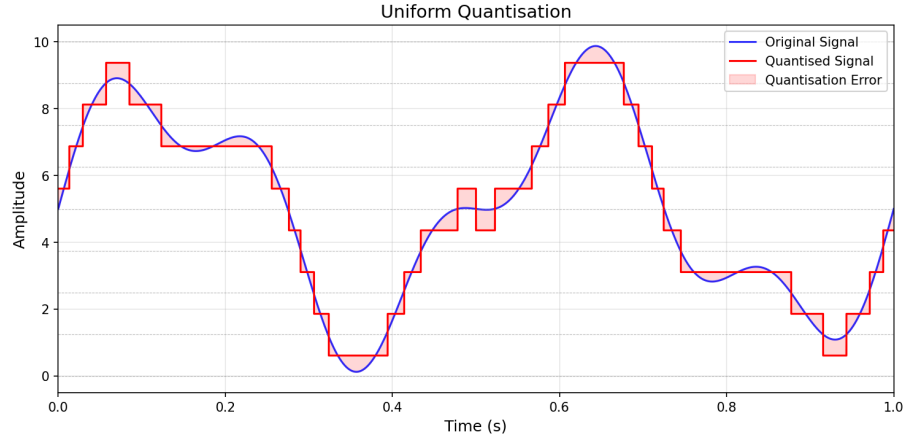


Figure 17: An example of Uniform Quantisation

2.9.3.1 Sigma-Delta Quantisation

Sigma-Delta modulation is a type of quantisation technique that uses a feedback loop to convert an analogue signal into a digital signal with high resolution and low noise. This technique is particularly effective in applications requiring high precision, such as audio and other data acquisition systems, where maintaining signal integrity and reducing quantisation noise are paramount. Sigma-Delta quantisation operates by oversampling the input signal at a higher rate, then passing it through a low-pass filter to remove high-frequency components. The filtered signal is compared to a reference voltage, and the difference is fed back to the input through a digital-to-analogue converter. This feedback loop effectively reduces quantisation noise by shifting it to higher frequencies, resulting in a high-resolution digital signal with minimal noise (Benedetto et al., 2006).

The Sigma-Delta quantisation process can be described by the following equation:

$$\hat{x}[n] = \left\lceil \text{round} \left(\frac{x[n] + e[n-1] - x_{\min}}{x_{\max} - x_{\min}} \cdot (L - 1) \right) \right\rceil \cdot \frac{x_{\max} - x_{\min}}{L - 1} + x_{\min} \quad (19)$$

Where:

- $x[n]$ is the input signal
- $\hat{x}[n]$ is the quantised output
- $e[n-1]$ is the previous error term
- L is the number of quantisation levels
- $x_{\max}$ and $x_{\min}$ are the maximum and minimum values of the input signal

In Equation 19, $e[n]$ is the error term calculated as:

$$e[n] = (x[n] + e[n-1]) - \hat{x}[n] \quad (20)$$

The input signal $x[n]$ is combined with the previous error term $e[n-1]$ to form $y[n] = x[n] + e[n-1]$. This combined signal is then quantised to the nearest quantisation level to produce $\hat{x}[n]$. The quantisation error $e[n]$ is calculated as the difference between the combined input $y[n]$ and the quantised output $\hat{x}[n]$, such that: $e[n] = y[n] - \hat{x}[n] = (x[n] + e[n-1]) - \hat{x}[n]$

This error feedback mechanism is key to sigma-delta modulation as it:

1. Shapes the quantisation noise by pushing it to higher frequencies (noise shaping)
2. Improves the effective resolution of the quantisation through oversampling
3. Accumulates the quantisation error over time, which is then used to correct subsequent samples

2.9.4 Interpolation

Interpolation, a critical technique in digital signal processing often associated with up-sampling, plays a vital role in approximation theory. Fundamentally, it involves estimating unknown values that fall between known data points. This process is essential when increasing the sample rate of a discrete signal, a practice commonly referred to as up-sampling, or in reconstructing a signal from a set of discrete samples. The essence of interpolation lies in constructing a continuous curve that either passes through or closely approximates the original discrete data points, thereby enabling a higher resolution or finer representation of the signal (Mastroianni & Milovanovic, 2008).

The application of interpolation extends beyond simply filling data gaps; it encompasses a range of mathematical techniques, such as linear, polynomial, spline, and Lanczos interpolation. Each method varies in complexity and suitability depending on factors such as desired accuracy, computational efficiency, and the specific characteristics of the data. For instance, linear interpolation connects data points with straight lines for simplicity, whereas spline interpolation employs piecewise polynomials to achieve a smoother, more natural-looking curve. This versatility makes interpolation a cornerstone in fields such as numerical analysis, computer graphics, and data analysis, where it helps approximate complex functions and create detailed visual and audio representations. Moreover, unlike extrapolation, which predicts values outside the known data range, interpolation is generally more reliable as it operates within the bounds of existing data points, thereby making fewer assumptions about the function's behaviour beyond the observed points.

For a linear interpolation between two known points (x_0, y_0) and (x_1, y_1) , where to find the value of y at some point x , the equation is given by:

$$y = y_0 + (x - x_0) \cdot \frac{y_1 - y_0}{x_1 - x_0}$$

Here, x_0 and x_1 are the known points of x -coordinates, y_0 and y_1 are the known values at y -coordinates. x is the point at which the value of y is to be found. And $\frac{y_1 - y_0}{x_1 - x_0}$ is the slope of the line between the two known points. Figure 18 shows the geometric intuition: the two known samples are joined by a straight line, and the interpolated value at any intermediate x is the height of that line at x .

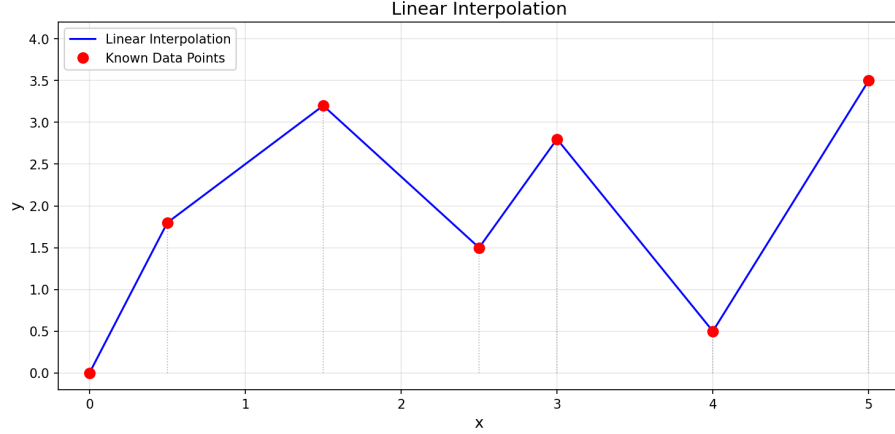


Figure 18: An example of Linear Interpolation

2.9.4.1 Radial Basis Function (RBF) Interpolation

Radial Basis Function (RBF) interpolation is a sophisticated technique utilised across signal processing, machine learning, and numerical analysis to approximate complex functions and interpolate scattered data points. Unlike conventional interpolation methods that primarily rely on polynomials or splines, RBF interpolation employs radial basis functions. These functions are uniquely characterised by their dependence solely on the distance from a central point or origin. This distinctive attribute enables RBFs to capture complex patterns and relationships within the data. As a result, RBF interpolation is particularly effective in addressing high-dimensional and non-linear problems, where traditional methods might falter due to inherent limitations in handling non-linearity and multi-dimensional data spaces (Buhmann, 2000; Rippa, 1999).

RBF for interpolation is given by:

$$f(x) = \sum_{i=1}^N a_i \Phi(\|x - x_i\|), x \in \mathbb{R}^d \quad (21)$$

Where, a_i are the weights, x_i are the centres, Φ is the radial basis function, and $\|x - x_i\|$ is the Euclidean distance between x and x_i .

The selection of the radial basis function is pivotal in RBF interpolation, as it critically influences the shape and adaptability of the interpolant. Commonly utilised RBFs include the Gaussian, Multiquadric, and Inverse-Multiquadric functions, each distinguished by unique properties that affect their performance in different applications. We can achieve precise and efficient approximations of complex functions by correctly choosing an appropriate RBF and meticulously optimising the interpolation parameters.

In Equation 21, the Gaussian RBF is given by:

$$\Phi(r) = \exp(-\varepsilon r^2)$$

Where, ε is the shape parameter that determines the width of the Gaussian function. And r is the Euclidean distance between the point x and the centre x_i .

2.9.5 Analogue-to-Digital Converter (ADC)

An Analogue-to-Digital Converter (ADC) is both a process and a device essential for converting continuous analogue signals into discrete digital signals (Smith, 1999). This conversion is fundamental in audio, video, and image processing applications, where real-world analogue inputs must be transformed into a format suitable for digital processing and analysis. The ADC process involves two main steps: sampling and quantisation (See Section 2.9.3). During sampling, the continuous-time signal is measured at discrete intervals, effectively converting it into a discrete-time signal. The subsequent quantisation step maps the continuous amplitudes of these samples to a set of discrete levels, known as quantisation levels. The ADC's resolution determines the number of these levels, typically represented in bits. For example, an 8-bit ADC can distinguish 256 (2^8) distinct levels. An inherent aspect of this process is the introduction of quantisation error or noise, arising from approximating analogue values with a limited set of discrete levels. Despite this, the ADC process is indispensable for digital systems, enabling them to efficiently store, process, and transmit information derived from continuous real-world signals such as sound and light.

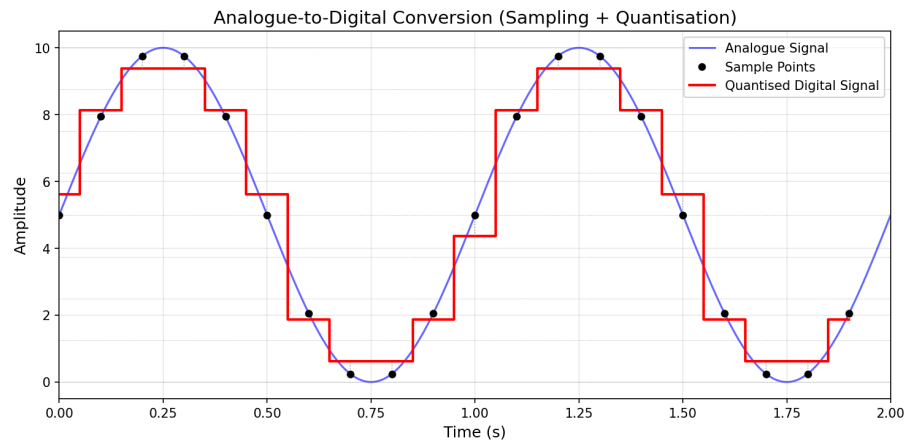


Figure 19: A constructed digital signal of an analogue sine wave using uniform sampling

Figure 19 shows a constructed digital signal of a sine wave as shown in Figure 15, using uniform sampling.

2.9.6 Digital-to-Analogue Converter (DAC)

A Digital-to-Analogue Converter (DAC) is an essential component in digital systems, serving as the bridge that converts digital, discrete-time signals back into analogue, continuous-time signals (Smith, 1999). This conversion is crucial in numerous applications where digital data must be rendered into a form that can be physically perceived or utilised, such as audio speakers, video monitors, or other analogue devices.

The process of a DAC involves translating a digital signal, typically represented in binary form, into analogue signals such as voltages or currents. Each digital value, consisting of a certain number of bits, corresponds to a specific analogue output. For example, in a 16-bit DAC, 65,536 (2^{16}) possible digital values are mapped to unique voltage levels within the device's output range. The higher the bit resolution of the DAC, the more precise the conversion can be, allowing for a more accurate representation of the original signal.

As the DAC receives digital data, it generates a series of analogue levels corresponding to these digital values. This results in a stepped output that approximates the original analogue waveform. Filtering methods are often employed to smooth out these steps and produce a continuous analogue signal. Therefore, the quality of the DAC depends not only on its resolution but also on the effectiveness of its filtering techniques.

DACs play a critical role in digital electronics by making digital content accessible and usable in the real world, effectively translating the abstract world of digital data into tangible, perceivable outputs. They are fundamental in multimedia systems,

communications, and any application where digital data needs to be converted to a usable analogue form.

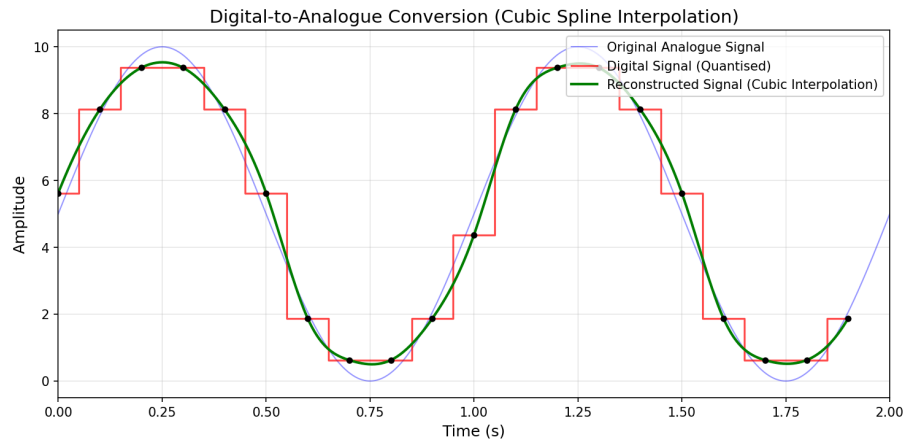


Figure 20: A constructed analogue signal of a digital step wave using cubic spline interpolation (Sun et al., 2023)

2.9.7 Nyquist-Shannon Sampling Theorem

The Nyquist-Shannon Sampling Theorem, a foundational concept in signal processing, defines the minimum sampling rate required to reconstruct a signal accurately without any information loss. First formulated by Harry Nyquist (Nyquist, 1928) and later expanded by Claude Shannon (Shannon, 1949), the theorem stipulates that a signal must be sampled at least twice the highest frequency component present in the signal to avoid aliasing. This rate is known as the Nyquist Rate, a crucial parameter in the design of Analogue-to-Digital Converters (ADCs) and other digital signal processing systems.

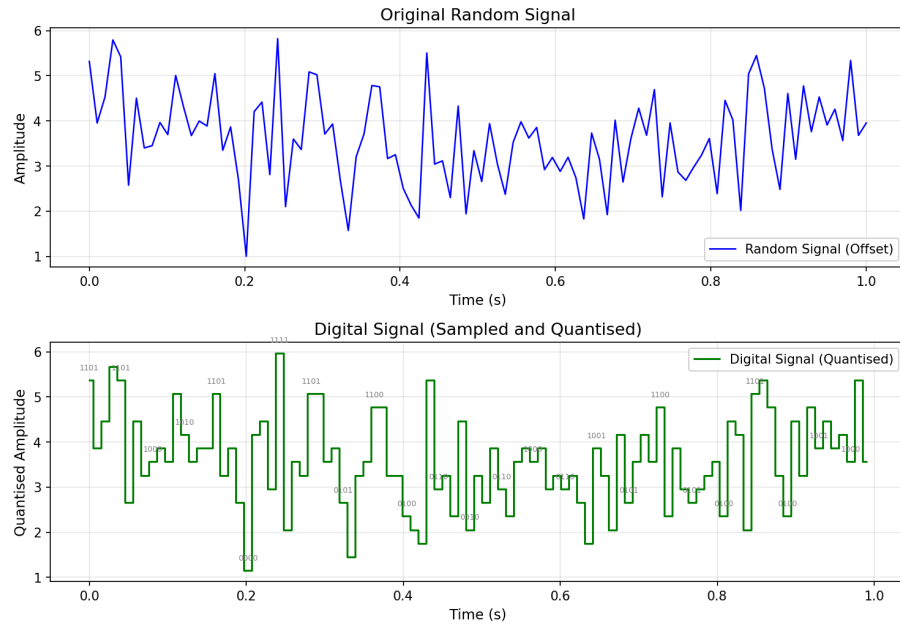


Figure 21: A constructed digital signal from a random signal. The digital signal shows binary output

Figure 21 demonstrates Section 2.9.7.1 and Section 2.9.7.2 on a random signal.

2.9.7.1 Nyquist Frequency

The Nyquist frequency, also known as the folding frequency, is defined as half of the sampling frequency. This fundamental concept was introduced by Harry Nyquist (Nyquist, 1924). The Nyquist frequency is given by:

$$f_N = \frac{f_s}{2}$$

Where, f_N is the Nyquist frequency and f_s is the sampling frequency.

2.9.7.2 Nyquist Rate

Whereas the Nyquist frequency (Section 2.9.7.1) characterises the sampler itself - the highest signal frequency that a given sample rate can faithfully resolve - the Nyquist rate characterises the requirement the sampler must satisfy for a given signal. It is the minimum sampling rate needed to capture the signal without information loss and is set by the signal's highest frequency component, $f_{\max}$ (Nyquist, 1924). The Nyquist rate is given by:

$$R_N = 2f_{\max}$$

Where, R_N is the Nyquist rate and $f_{\max}$ is the maximum frequency in a signal.

2.9.8 Anti-Aliasing Filter

Anti-aliasing filters are crucial in digital signal processing for removing high-frequency components from a signal before sampling. Their primary function is to prevent aliasing, a form of signal distortion that occurs when frequencies higher than the Nyquist frequency are incorrectly interpreted as lower frequencies in the sampled signal. These filters, typically low-pass, attenuate frequencies above the cutoff point. The cutoff frequency of an anti-aliasing filter is set at or slightly below the Nyquist frequency, as established in Section 2.9.7.1; this ensures that the sampled signal is free of aliasing-induced distortions, thereby maintaining the integrity of its frequency content.

2.9.9 Fast Fourier Transform (FFT)

First introduced by Cooley & Tukey (1965), the Fast Fourier Transform (FFT) is a pivotal algorithm in digital signal processing for efficiently computing the Discrete Fourier Transform (DFT) of a signal or sequence (Rao et al., 2010). DFT is a mathematical function that translates a signal from the time domain, where it is usually captured, to the frequency domain, revealing its frequency components. The FFT significantly reduces the computational complexity of the DFT from $O(N^2)$ to $O(N \log N)$ (Rao et al., 2010), enabling rapid processing of large datasets and real-time signal analysis. This breakthrough has been fundamental in advancing various fields, such as digital communications, audio and image processing, and spectral analysis.

The DFT equation is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j(\frac{2\pi}{N})kn}, \quad \text{where } k = 0, 1, \dots, N-1 \quad (22)$$

Where, $X(k)$ is the DFT of the k^{th} element, representing the frequency component at frequency index k . $x(n)$ is the input time domain sequence. N is the total number of samples in a sequence, and n is the number of samples in the signal.

The twiddle factor is used to reduce the complexity to form faster DFTs when using FFT and it is denoted by:

$$W_N = e^{-j(\frac{2\pi}{N})} \quad (23)$$

The Equation 22 can be rewritten as:

$$X(K) = \sum_{n=0}^{N-1} x(n)W_N^{kn}, \quad \text{where } k = 0, 1, \dots, N-1$$

FFT takes advantage of the fact that the twiddle factor is periodic and repeating; this is given by Equation 23. Based on that, it can compute DFTs much faster than the DFT equation (Kuo & Gan, 2005).

2.9.10 Spectrogram

A spectrogram, also known as a sonogram, is a visual representation of a signal's frequency spectrum as it varies over time (Basak et al., 2023). This two-dimensional depiction plots time on the x-axis and frequency on the y-axis, effectively mapping the frequency content of a temporal signal over its duration. In a spectrogram, the intensity or colour saturation indicates the amplitude of different frequency components at each time point, with brighter colours typically representing higher amplitudes.

Spectrograms are essential tools across a variety of signal processing contexts, including audio processing, speech analysis, and other fields where understanding a signal's frequency characteristics is crucial. The utility of a spectrogram lies in its ability to reveal detailed information about a signal's frequency content over time, making it an invaluable resource for detecting patterns, trends, and anomalies in complex datasets. Through this visualisation, analysts and researchers can gain insights into the temporal dynamics of signals, assisting with tasks ranging from diagnostics to enhancing audio experiences. Figure 22 shows the spectrogram of a one-second *Bird* utterance from the Speech Commands dataset (Warden, 2018).

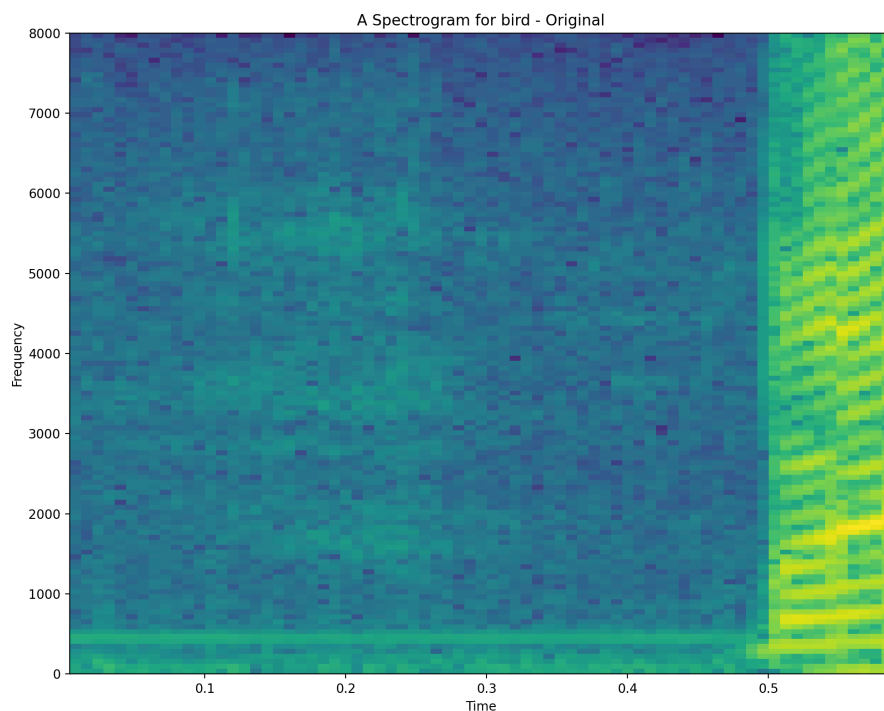


Figure 22: A Spectrogram of *Bird* speech from Warden (2018).

2.9.10.1 Mel-Spectrogram

The Mel-Spectrogram is an advanced variant of the conventional spectrogram, tailored to align more closely with human auditory perception by incorporating the Mel scale (Setianingrum et al., 2023). This perceptual scale reflects the human ear's non-linear sensitivity to frequencies, being more sensitive to lower frequencies than higher ones. The transformation of the frequency axis from a linear to a Mel scale in a Mel-Spectrogram provides a representation that more closely approximates human hearing.

This adaptation makes the Mel-Spectrogram exceptionally valuable in fields like audio processing and speech recognition, where accurate emulation of human auditory processing enhances the performance of analytical and predictive models. By providing insights into how sound is perceived rather than merely detected, Mel-Spectrograms facilitate more effective analyses of audio signals, particularly in applications which require robust feature extraction for tasks such as speaker identification, speech synthesis, and audio classification. Figure 23 contrasts the raw waveform of the same *Bird* utterance with its Mel-spectrogram representation.

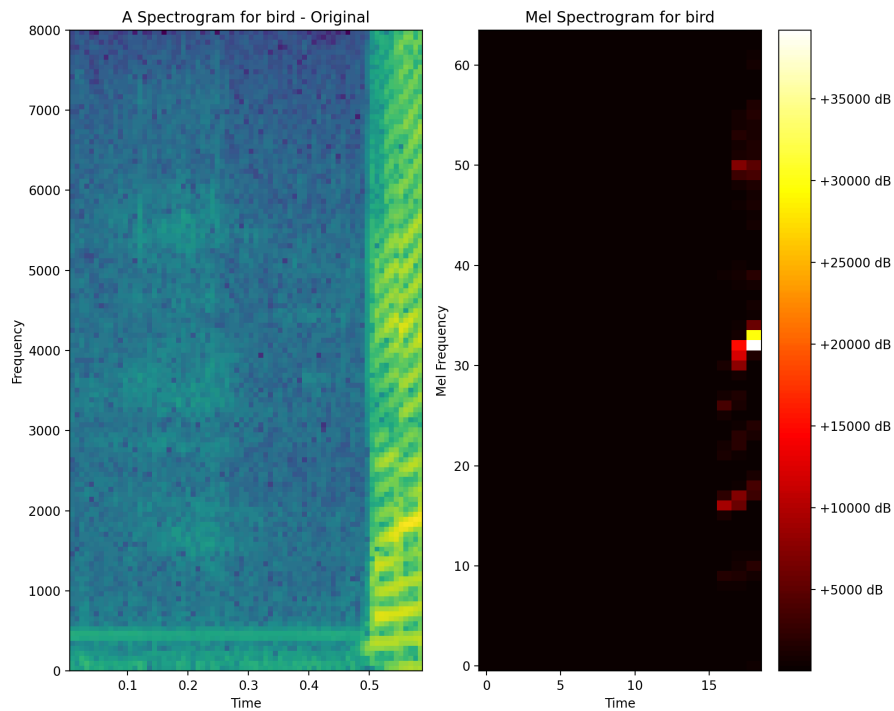


Figure 23: A Mel-Spectrogram of *Bird* speech from Warden (2018). *A* is the original signal, *B* is the signal after Mel-Spectrogram.

2.9.11 Spectral Gate

Spectral gating is a powerful signal processing technique that selectively filters specific frequency components of a signal, akin to a gate which allows certain elements to pass while blocking others (Sainburg & Gentner, 2021). It is used to suppress noise, enhance desired frequency bands, and isolate particular signal components for further analysis. By establishing thresholds or criteria for frequency components, spectral gating effectively controls the flow of information within a signal, enabling precise processing and manipulation of its spectral content. Figure 24 shows the same *Bird* utterance before and after spectral gating, with the silent regions audibly cleaned up.

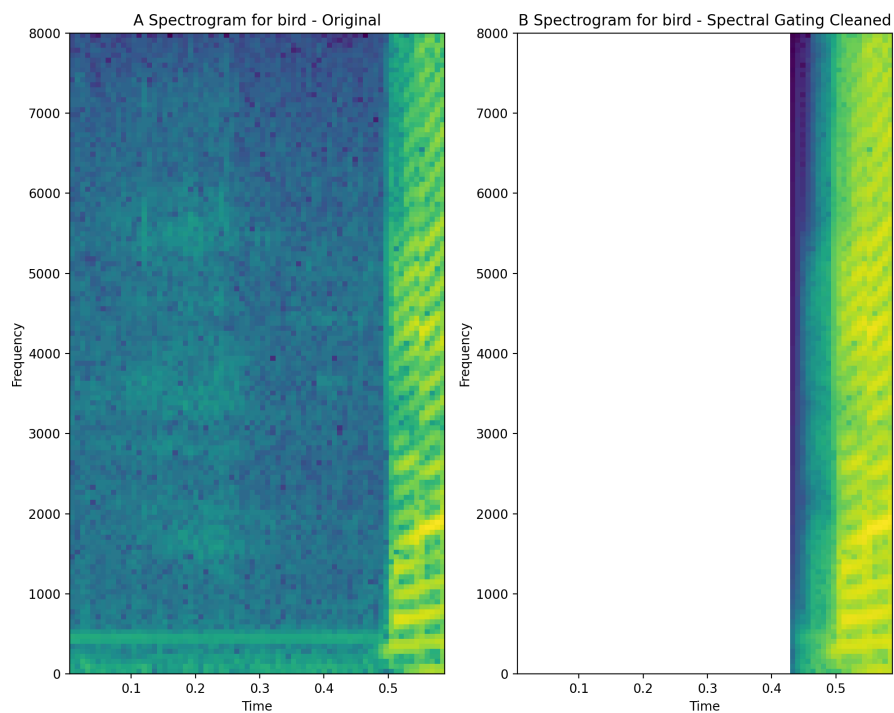


Figure 24: Cleaning a *Bird* speech from Warden (2018) using spectral gating. *A* is the original signal, *B* is the signal after spectral gating.

2.9.12 Band-Pass Filter

A band-pass filter is a frequency filter that allows signals within a specified frequency range, known as the passband, to pass through while attenuating signals outside this range (Nouri et al., 2024). This type of filter is designed to isolate and amplify signals within a particular frequency band, effectively reducing unwanted noise or interference. Band-pass filters are essential in various applications, such as audio processing and communications, where precise control over frequency components is crucial. By adjusting

the passband and filter parameters, we can optimise the filter's response to meet specific requirements, enhancing the quality and clarity of the processed signal.

2.10 Case Studies

In this section, I introduce the case studies that will empirically address the research questions in Section 1.3.2 and their corresponding hypotheses in Section 1.3.3, using SNNs together with the DyNED and DyNED_c encoders developed in this thesis.

2.10.1 CIFAR-10 Dataset

The CIFAR-10 dataset (Krizhevsky et al., 2009) forms a foundational case study in this thesis, emphasising the application of Spiking Neural Networks (SNNs) in image recognition. CIFAR-10 comprises 60,000 colour images across 10 classes, each presented as a 32x32 pixel RGB image. This dataset is instrumental for evaluating the effectiveness of SNNs for object recognition, serving as a benchmark for assessing their pattern recognition capabilities. The diversity of natural images in the dataset makes it an excellent testbed for demonstrating the proficiency of SNNs in deciphering visual information, a critical aspect of many AI and machine learning applications (Krizhevsky et al., 2009).

2.10.2 Speech Command Dataset

Another pivotal case study in this thesis utilises the Google Speech Commands dataset (Warden, 2018) to explore the potential of Spiking Neural Networks (SNNs) in processing auditory signals, specifically in the context of speech recognition. The dataset, consisting of 65,000 one-second-long utterances of 30 short words spoken by numerous speakers, provides a rich resource for training and evaluating SNNs for voice command recognition. It is particularly relevant for developing voice-activated systems and virtual assistants. This dataset provides an opportunity to demonstrate how SNNs, in conjunction with the universal encoder, can effectively interpret and respond to voice commands, showcasing the adaptability and efficiency of these networks in handling real-world audio data. The insights gained from this case study have significant implications for advancing voice recognition technology in AI systems.

2.10.3 CIFAR10-DVS Dataset

CIFAR10-DVS (Li et al., 2017) is the neuromorphic event-based counterpart of the CIFAR-10 dataset. The 10,000 CIFAR-10 images are converted into asynchronous spike events using a Dynamic Vision Sensor (DVS), with each event encoded as a tuple of $(x,$

y, polarity, timestamp) rather than as a pixel intensity. The dataset retains the same 10 object classes as CIFAR-10, providing a direct comparison between conventional image data and its event-based representation. Because the events are sparse, asynchronous, and inherently temporal, CIFAR10-DVS is widely used as a benchmark for SNN architectures designed to exploit the spatio-temporal structure of neuromorphic data (Deng et al., 2022; 2026; Fang et al., 2021; Lee et al., 2025; Yang et al., 2026; C. Yu et al., 2026; T. Zhang et al., 2025). In this thesis, CIFAR10-DVS is used to test whether the DyNED + DyNED_c pipeline can operate on already-spiking visual data without structural change to the encoder or compressor.

2.10.4 Spiking Speech Commands (SSC) Dataset

The Spiking Speech Commands (SSC) dataset (Cramer et al., 2020) is the neuromorphic counterpart of the Google Speech Commands dataset. The conventional audio waveforms are passed through a biologically detailed model of the inner ear (a 700-channel auditory nerve model) to produce asynchronous cochlear spike events, with each event a tuple of channel and timestamp. The dataset retains the same 35 word classes as Speech Commands, enabling a direct comparison between conventional audio and cochlear-spike audio. Like CIFAR10-DVS, SSC is used in this thesis to test whether the DyNED + DyNED_c pipeline generalises from conventional audio waveforms to already-spiking auditory data.

2.11 Conclusion

This chapter has established the theoretical and technical foundations upon which the remainder of this thesis builds. It reviewed the biological inspiration behind Spiking Neural Networks, the principal neuron models (LIF, aEIF, Izhikevich, AdLIF, cAdLIF), and the learning rules (STDP, surrogate gradient) that govern their behaviour. Signal processing techniques - including quantisation, interpolation, FFT, and spectral analysis - were introduced as the building blocks for encoding real-world data into spike-compatible formats. Existing spike encoding methods, such as AER, BSA, and Population Encoding, were discussed, highlighting a gap in the literature: most encoders focus on efficient spike generation but neglect faithful reconstruction of the original data.

Read against the RPS, the survey shows that a wide variety of encoding and decoding methods have been published but no single method dominates - each is tailored to a particular data modality, prioritises spike generation over reconstruction, or produces dense spike streams that do not lend themselves to lossless compression.

The compression literature treats RLE, Huffman, and the LZ family as general-purpose tools without optimising for the binary structure of spike trains. This lack of a unified, reconstruction-faithful, compressible encoding pipeline - across both static and dynamic data and both conventional and neuromorphic sources - is precisely the gap that the research questions in Section 1.3.2 are designed to close.

The concepts covered here directly inform the design of DyNED (Chapter 3), which combines sigma-delta quantisation with step signal generation and RBF interpolation to produce binary spike trains while preserving data fidelity. The data compression techniques reviewed - particularly RLE and Huffman coding - provide the basis for DyNED_c (Chapter 4), which compresses these spike trains for efficient storage and processing. The SNN architectures and optimisation methods discussed are applied across four case-study datasets: CIFAR-10 and Speech Commands as conventional image and audio benchmarks (Chapter 5 and Chapter 6), together with their neuromorphic event-based counterparts CIFAR10-DVS and Spiking Speech Commands.

Chapter 3

DyNED: Dynamic Neural Encoder and Decoder

3.1 Introduction

This chapter looks into the critical role of encoders and decoders in Spiking Neural Networks (SNNs). Encoders in SNNs serve as the primary interface between the network and the external world, converting continuous or high-dimensional data into discrete data points. The SNN then interprets these data points as spikes, the fundamental information transmission format, within these networks.

The design and functionality of encoders are pivotal. They must efficiently translate real-world data into spike patterns while preserving the integrity and richness of the original information. This translation is not trivial; it involves capturing the essential features of the input data and representing them in a way that the SNN's spike dynamics can effectively process. The challenge lies in balancing data fidelity and computational efficiency. A well-designed encoder minimises data loss during this translation process, thereby maintaining high data quality.

Decoders in SNNs play an equally vital role. They are tasked with interpreting the network's spike patterns and converting them into a comprehensible form, often matching the original input domain. The decoding process is crucial for extracting meaningful information from the network's spike outputs, especially in applications such as sensory data interpretation, pattern recognition, and actuator control in robotics.

Both static and dynamic encoders and decoders are explored in this chapter. Static encoders and decoders operate with fixed parameters and are generally more straightforward but can be less flexible in adapting to varying data characteristics. In contrast, dynamic encoders and decoders can adapt their parameters in response to input data, potentially offering a more nuanced, responsive data-processing capability that can be particularly beneficial for handling complex, time-varying inputs.

This chapter aims to provide a comprehensive understanding of how these encoders and decoders function within SNNs, their design considerations, and their impact on overall performance and the applicability of SNNs across various computational tasks. Exploring these components is not just a technical endeavour, but also a step towards bridging the gap between artificial intelligence systems and the remarkable efficiency and adaptability of biological neural processing.

Read against the RPS, this chapter addresses the encoding-and-decoding half of the problem identified in Chapter 1 - the absence of a unified, reconstruction-faithful method that operates across both static and dynamic data. The material covered in the remainder of Chapter 3 bears directly on Research Question 1 (RQ1), Research Question 2 (RQ2), and Research Question 3 (RQ3), each of which is taken up experimentally in Chapter 5 and Chapter 6.

3.2 Background and Theory

In Chapter 2, a variety of encoders and decoders for Spiking Neural Networks (SNNs) were introduced, providing a foundational understanding of existing methodologies within the field (see Sections 2.2.3, 2.2.2, and 2.2.1). Building upon this base, the following sections aim to explore recent advancements in encoder and decoder technologies for SNNs. This chapter further investigates the design, functionality, and unique capabilities of these newer models, highlighting how they contribute to the evolving landscape of SNN research. Each of these algorithms represents a step forward in our ability to process complex data types through SNNs, offering enhanced efficiency, accuracy, or adaptability compared to earlier approaches. This exploration sets the stage for a deeper discussion on implementing a novel encoder developed as part of this research, which will be detailed in the subsequent Section 3.4.

3.2.1 Importance of Encoders and Decoders

High-quality data is crucial for SNNs to learn effectively and make accurate predictions. However, transforming data into high-dimensional spike trains (sequences of ones and

zeros) introduces a challenge: ensuring minimal information loss during encoding and decoding.

Recent research highlights a concerning trend: data encoded in spikes can be distorted or even lost entirely. Many existing encoders prioritise efficient conversion to spikes without considering faithful reconstruction. Studies by Henkes et al. (2024) and Eshraghian et al. (2023b) demonstrate encoders focused solely on spike generation. While Guo et al. (2024) proposes a ternary spiking approach that minimises information loss, data reconstruction capabilities remain unexplored. Similarly, Shi et al. (2024) prioritises the speed and precision of spiking in their SpikingResformer architecture, at the expense of reconstruction. Chen et al. (2024) introduces a hybrid encoder that combines various techniques, but data reconstruction is not revisited.

Reviews by Rathi et al. (2023) and Cherian & others (2024) explore similar encoding methods, including BSA (Section 2.2.2), but lack evidence of successful data reconstruction.

The current focus in SNN research prioritises efficient spike encoding (adaptability, speed, precision) over ensuring the encoded representation accurately reflects the original data. This work emphasises the importance of data fidelity and efficient encoding. This study aims to bridge the gap by investigating methods which transform data into spikes while enabling the reconstruction of the original data with minimal loss.

3.2.2 Challenges in Encoding for SNNs

Spiking Neural Networks (SNNs) fundamentally differ from traditional artificial neural networks in their operation. Where conventional networks propagate continuous-valued activations through layers, SNNs communicate through binary, time-dependent spike events (Eshraghian et al., 2023a; Khan et al., 2025; Maass, 1997). This fundamental difference necessitates a tailored approach to encoding and decoding, requiring adaptations of classical processing mechanisms to fit the spiking paradigm. These adaptations introduce a set of challenges unique to SNNs, driving the need for more nuanced and specialised strategies in encoding and decoding.

One significant challenge arises from the discrete nature of spikes. Encoding for SNNs involves capturing essential attributes of input data and translating these into a spike-based format that SNNs can efficiently process (Rathi et al., 2023). This task demands an in-depth understanding of both the data's inherent characteristics and the dynamics of spike behaviour within the network. Encoders must delicately balance the preservation of data fidelity against the need for computational efficiency - rate coding

offers simplicity and noise robustness but limited temporal precision, while temporal coding yields higher information density at greater computational cost (Rathi et al., 2023; Voudaskas et al., 2025). This involves ensuring critical information is retained with minimal loss during its conversion to spikes while also optimising the resource utilisation to operate within the SNN's constraints.

A further challenge concerns the decoder. While most SNN research focuses on classification tasks where the output is a decision rather than a reconstructed signal, applications that require signal recovery - such as encoding pipelines that must verify the fidelity of their spike representation - face the additional task of accurately interpreting the generated spike patterns and converting them back into a comprehensible format (Henkes et al., 2024). As noted in Section 3.2.1, few existing encoders address reconstruction, making this an underexplored area. The DyNED encoder developed in this thesis explicitly addresses this gap by providing both encoding and decoding pathways with quantifiable reconstruction metrics.

These challenges highlight the complexity of developing encoding and decoding mechanisms for SNNs, and underscore the need for innovative approaches that accommodate the unique demands of spike-based processing. Recent surveys note the lack of standardised encoding benchmarks and unified metrics, further complicating cross-study comparisons (Khan et al., 2025; Voudaskas et al., 2025). Specialised knowledge and techniques are essential for crafting effective, efficient solutions, ensuring SNN systems meet the rigorous demands of real-world applications.

3.3 Design Consideration

The design of encoders and decoders for Spiking Neural Networks (SNNs) represents a complex challenge that demands a nuanced balance among several critical factors. These include ensuring data fidelity, maximising computational efficiency, and providing flexibility to handle diverse, dynamically changing data characteristics. This section explores the essential design considerations that underpin the development of encoding and decoding algorithms tailored for SNNs. We will discuss the selection of appropriate encoding and decoding strategies, techniques for accurately representing input data, and the necessity for algorithms to adapt to variations in input data over time. A deep understanding of these considerations is paramount for creating encoders and decoders that effectively process an extensive array of data types, thereby ensuring the integrity and quality of data throughout the computational process. Highlighting the implications of these design choices, we will illustrate their significance in enhancing the overall

performance and applicability of SNNs across various domains. This marks a critical step towards realising the full potential of these neural networks in advanced computational tasks.

3.3.1 Criteria for Effective Encoding and Decoding

The effectiveness of encoding and decoding in Spiking Neural Networks (SNNs) hinges on a crucial balance: preserving the integrity of input data while ensuring computational efficiency. This balance varies significantly depending on the nature of the data. For instance, in the realm of Analogue-to-Digital conversion, such as digitising vinyl records, maintaining high fidelity of the original analogue signal is paramount, requiring minimal data loss during the conversion. This principle ensures that the richness and nuances of the audio are faithfully preserved, mirroring the high standards set for auditory data processing.

Conversely, visual data captured by cameras may undergo a degree of compression without markedly detracting from the user's experience, thanks to the human visual system's tolerance for certain losses in detail. This discrepancy in data handling underscores a fundamental principle for SNN design. While striving for maximum data preservation is ideal, practical considerations require some data compression to optimise processing speed and resource utilisation.

In the context of SNNs, determining the optimal balance involves a nuanced understanding of the specific application's requirements and the inherent trade-offs. For instance, encoding strategies for SNNs must account for the importance of each piece of information in the dataset, ensuring that vital information is prioritised and preserved during spike conversion. This approach not only safeguards the quality and usability of the encoded data but also enhances the SNN's efficiency by allocating computational resources to the most impactful data elements.

Ultimately, the goal is to develop encoding and decoding mechanisms that adeptly navigate the tightrope between data fidelity and computational pragmatism. By tailoring these mechanisms to each data type's unique characteristics and requirements, SNNs can achieve a harmonious balance that maximises both performance and efficiency.

3.3.2 Adaptability, Efficiency, and Strategy Selection

A key objective in designing encoders for SNNs is balancing three concerns: adaptability to diverse data types, computational efficiency, and data fidelity. The encoder must handle both static data (e.g. images) and dynamic data (e.g. audio) without requiring

changes to the SNN architecture itself. This is achieved through mechanisms such as dynamic thresholding and configurable quantisation levels that adjust to the characteristics of the input data.

Computational efficiency is achieved through algorithmic optimisations and GPU-accelerated parallel processing, enabling the encoder to handle large datasets within the time constraints of real-time applications. At the same time, data fidelity is preserved by retaining the critical features of the input within the encoded spike patterns - techniques such as sigma-delta quantisation with error feedback (Section 2.9.3.1) and RBF interpolation for reconstruction (Section 2.9.4.1) ensure that no significant information is lost during encoding.

The selection of encoding strategies is guided by the data's dimensionality, variability, and temporal dynamics, as well as the SNN's architecture and computational goals. Parameters such as quantisation levels, interpolation resolution, and noise reduction thresholds are tuned through empirical testing to identify the optimal configuration for each application domain. The effect of quantisation levels on spike density and reconstruction quality is evaluated empirically in Chapter 5 (Table 7) and Chapter 6 (Table 25), where increasing the number of levels yields denser spike distributions with higher reconstruction fidelity. The contribution of the error feedback mechanism is also quantified in those chapters (Table 8, Table 26).

3.4 Implementation Details

This section explores the process of implementing a novel encoder specifically devised for this research. Designed to process complex, high-dimensional data, this encoder converts it into spike patterns, enabling effective interpretation by Spiking Neural Networks (SNNs). Here, I detail the technical aspects of the encoder's design, including the development environment and the tools used throughout the creation process. Additionally, I outline the specific encoding and decoding algorithms developed, shedding light on their operational mechanics and the rationale for their selection.

Moreover, this section covers the comprehensive testing and validation framework established to ensure the encoder's efficacy and reliability. We navigate the challenges encountered during the encoder's development, presenting the innovative solutions employed to overcome them. Through this detailed exposition, this chapter aims to provide a thorough understanding of the encoder's implementation journey, from conceptual design to practical application, highlighting the critical steps taken to bring this novel encoding solution to fruition.

3.4.1 Introduction to the Encoder's Implementation

In this encoder's development, we harness the time-tested techniques of Digital-to-Analogue Conversion (DAC) and Analogue-to-Digital Conversion (ADC), foundational pillars in the field of signal processing, and repurpose them for the innovative domain of Spiking Neural Networks (SNNs). By employing interpolation to enhance data continuity and quantisation to discretise it into manageable, spike-compatible formats, this encoder adeptly translates intricate, high-dimensional datasets into spike patterns. This allows SNNs to interpret the encoded data and the decoder to reconstruct the original signal from it. These techniques, while established, are meticulously tailored to align with the unique computational demands and data-integrity needs inherent to SNNs. This allows us to bridge conventional signal processing with the nuanced requirements of neuromorphic computing.

The encoder's design is not just another chance discovery; it represents a conceptual leap in how data is prepared for and recovered from SNN processing. Interpolation (Section 2.9.4) and Quantisation (Section 2.9.3) are chosen for their reliability and ability to preserve the fidelity of the original data amidst the conversion process, ensuring that the essence of the information remains intact and actionable. This careful balance of accuracy and efficiency is critical in a landscape where SNNs are increasingly applied to complex, real-time decision-making tasks across various sectors, from autonomous systems to advanced diagnostics in healthcare.

Furthermore, adapting DAC and ADC methodologies to SNNs underscores a significant advancement in making neuromorphic computing more accessible and effective. By addressing the challenges of encoding and decoding within this context, the encoder opens up new avenues for research and application, pushing the boundaries of what is possible with SNNs. It sets the stage for a detailed exploration of the encoder's implementation journey, from the initial concept to the final solution. This narrative not only charts the technical hurdles we have overcome but also highlights the strategic decisions and innovations. This then becomes an important tool to elevate the performance and applicability of SNNs in tackling some of today's most pressing computational challenges.

3.4.2 Development Environment and Tools

The algorithm's development leveraged the Python programming language (Van Rossum & Drake, 2009), renowned for its versatility and reliable support for scientific

computing through a rich ecosystem of libraries. Essential libraries used in this project include:

PyTorch (Paszke et al., 2019): Primarily leveraged not just as a framework for neural networks but also as an advanced replacement for NumPy, especially for tasks involving image manipulation. Its dynamic computation graph and GPU acceleration capabilities were crucial for efficiently processing complex, high-dimensional datasets.

Matplotlib (Hunter, 2007): Utilised for generating high-quality visualisations of the data and results, aiding in analysis and presentation.

NumPy (Harris et al., 2020): A fundamental package for numerical computations in Python, critical for handling and processing large datasets.

SciPy (Virtanen et al., 2020): Provides additional scientific and technical computing tools, enhancing data manipulation and analysis processes.

Plotly (Plotly, 2015): Similar to Matplotlib, Plotly was used for interactive visualisations, offering a dynamic and engaging way to explore the data and results.

UV (UV, 2024): An extremely fast Python package and project manager, written in Rust. The development was conducted on a Linux-based system, specifically Ubuntu 20.04 (Canonical, 2004), chosen for its stability and broad support within the scientific computing community. The hardware was deployed on DigitalOcean's 20 virtual CPUs with 240GB of RAM, ensuring swift computation and ample memory for handling complex data operations. A significant acceleration in the encoding process was achieved by utilising an NVIDIA H100 GPU. This high-performance GPU significantly reduced computation times, enabling more iterative testing and refinement of the encoder algorithm.

This combination of advanced software tools and powerful hardware provided a strong platform for developing and optimising the encoder. It ensures that the project can leverage cutting-edge computational techniques and efficiently process the intricate datasets characteristic of SNN applications.

The choice of Python as the primary programming language was motivated by its extensive support for scientific computing and machine learning libraries, making it an ideal environment for developing and testing the encoder algorithm. PyTorch was selected for its strong GPU support and NumPy-like programming approach, providing a comprehensive suite of tools for building Encoders and Decoders. Matplotlib was used to visualise the encoder's performance and the spike patterns generated, while NumPy and Scipy were employed for efficient array manipulation and mathematical operations.

3.4.3 Modifications and Enhancements

The developmental trajectory of encoder algorithms for Spiking Neural Networks (SNNs) demonstrates a consistent progression towards greater sophistication and precision. They are specifically tailored to enhance both the performance and accuracy in processing complex real-world audio and image data. Initially, the encoder algorithm began by integrating fundamental signal processing techniques, such as grayscale conversion and the Fast Fourier Transform (FFT, Section 2.9.9), to convert images to the frequency domain, followed by amplitude quantisation (Section 2.9.3) to manage data compression.

As algorithms matured, advancements included the introduction of uniform quantisation (Section 2.9.3), which standardised the approach to data compression by providing a systematic framework that replaced less structured amplitude quantisation techniques. This development streamlined the encoder's operational pipeline, emphasising essential transformations while eliminating extraneous processes. This iteration further refined output handling, enabling direct transformation of reconstructed data into displayable formats and enhancing the practical utility of the encoder.

Further developments included the incorporation of Radial Basis Function (RBF) interpolation (Section 2.9.4.1) directly applied to quantised frequency data. This method significantly improved the quality of the reconstructed outputs by providing dual paths for data reconstruction, utilising both interpolated and non-interpolated quantised data. This enabled precise optimisation tailored to specific investigative needs and facilitated detailed comparative analyses of the effects of interpolation on data fidelity.

In the most advanced version, the encoder algorithm employed sophisticated quantisation techniques such as sigma-delta quantisation (Section 2.9.3.1) alongside traditional uniform quantisation. This stage included an in-depth comparative analysis of the effects of various quantisation methods on the encoding quality, enhancing the encoder's ability to maintain data integrity during the compression phase. Additionally, this version expanded the visualisation toolkit to include quantitative scatter plots that demonstrate the impact of quantisation on the frequency domain, alongside traditional visual representations such as waveforms and spectrograms.

3.4.3.1 Integration of Advanced Audio Processing Techniques

Mel Spectrogram Conversion: Using torchaudio's MelSpectrogram, the algorithm transformed audio waveforms into a Mel Spectrogram representation (Section 2.9.10.1). This method involves computing a Short-Time Fourier Transform (STFT, Section 2.9.9) and mapping the power spectra onto the Mel scale. This scale is partic-

ularly effective for speech-processing tasks because it aligns with human auditory perception.

Noise Reduction and Bandpass Filtering: The script incorporated noise reduction using spectral gating (Section 2.9.11) via the TorchGate library, which removes non-stationary noise from audio signals, thereby clarifying the audio data for more accurate SNN analysis. Concurrently, a bandpass biquad (Section 2.9.12) was employed to selectively focus on frequencies most relevant to human speech, significantly reducing the impact of irrelevant frequencies and enhancing the encoder’s focus on pertinent data.

3.4.3.2 Data Handling and Device Management

Custom SPEECHCOMMANDS Dataset Class: A specialised subclass of torchaudio’s SPEECHCOMMANDS was developed to seamlessly integrate transformations into the data-loading workflow. This enabled the dynamic application of audio transformations, such as Mel Spectrogram conversion and noise reduction, directly during dataset loading, thereby optimising the workflow for SNN training and inference.

Optimised Computational Resources: The code ensured that all data transformations and manipulations were compatible across both CPU and GPU environments, maximising computational efficiency and allowing for flexible deployment across diverse hardware configurations.

These advancements not only enhanced the functionality of the encoder but also significantly broadened its applicability in neuromorphic computing. Integrating these sophisticated audio processing techniques marked a pivotal shift towards more adaptive, strong, and efficient preprocessing methods, essential for handling the complexities of real-world data in advanced computational models such as SNNs. As referenced in the earlier stages of this thesis (Figure 19, Figure 20, Figure 21), these enhancements build on the foundational methods previously discussed, linking the theoretical underpinnings with practical implementations to advance the field of digital signal processing.

These cross-modality adaptations - the audio-processing enhancements that extend DyNED beyond static image data without altering the encoder’s core structure - directly support RQ3, which is taken up experimentally in Chapter 5 and Chapter 6.

3.4.4 Final Algorithm Design

The encoder consists of multiple stages, each designed to transform the input data into a format suitable for processing by Spiking Neural Networks (SNNs). Standard prepro-

cessing steps such as grayscale conversion and flattening are applied first. The core of the DyNED algorithm then transforms the data through three stages: frequency extraction via FFT, sigma-delta quantisation with error feedback, and step signal generation to produce binary spike trains.

Within DyNED, the input data is projected into the frequency domain via the Fast Fourier Transform (FFT) and immediately passed through a quantisation loop with error feedback to produce spike-compatible values; these two steps form a single integrated encoding stage rather than a separate preprocessing pass followed by quantisation.

Algorithm 2: DyNED Quantisation.

```

input: Domain frequency data (data) and quantisation levels (levels)
output: Quantised frequency domain data
1 quantised  $\leftarrow$  array of zeros with same shape as data;
2 error  $\leftarrow$  0;
3 data_min  $\leftarrow$  min(data);
4 data_max  $\leftarrow$  max(data);
5 for i  $\leftarrow$  0 to length(data)-1 do:
6   sample_with_error  $\leftarrow$  data[i] + error;
7   quantised_sample  $\leftarrow$  round  $\left( \frac{\text{sample\_with\_error} - \text{data\_min}}{\text{data\_max} - \text{data\_min}} \times (\text{levels} - 1) \right)$ ;
8   quantised_sample  $\leftarrow$   $\frac{\text{quantised\_sample}}{\text{levels} - 1} \times (\text{data\_max} - \text{data\_min}) + \text{data\_min}$ ;
9   quantised[i]  $\leftarrow$  quantised_sample;
10  error  $\leftarrow$  sample_with_error - quantised_sample;
11 end for
12 return quantised;

```

The quantised output from Algorithm 2 still contains multi-level values. To produce binary spike data suitable for SNNs, a step signal is generated by comparing consecutive quantised samples: a spike (1) is emitted wherever the quantised value changes, and a 0 is emitted otherwise. This is shown in Algorithm 3.

Algorithm 3: Step signal generation from quantised data.

input: Quantised data (*quantised*)
output: Binary step signal (*step_signal*)

```

1 step_signal  $\leftarrow$  array of zeros with same shape as quantised;
2 for i  $\leftarrow$  1 to  $\text{length}(\text{quantised})-1$  do:
3   if quantised[i]  $\neq$  quantised[i-1] then
4     | step_signal[i]  $\leftarrow$  1;
5   end if
6 end for
7 return step_signal;

```

Using Algorithm 2 and Algorithm 3, the complete DyNED encoder can then be represented as:

Algorithm 4: DyNED encoder algorithm for Spiking Neural Networks.

input: Preprocessed data (*data*) and quantisation levels (*levels*)
output: Binary spike train

```

1 frequency_domain  $\leftarrow$  FFT(data);
2 quantised  $\leftarrow$  SigmaDeltaQuantisation(frequency_domain, levels);
3 spike_train  $\leftarrow$  GenerateStepSignal(quantised);
4 return spike_train;

```

The decoder algorithm for reconstructing the original data from the encoded spike would require interpolation and an inverse FFT. In this case, Radial Basis Function (RBF) interpolation is used. Linear interpolation was not considered for this stage because it produces piecewise-linear reconstructions with discontinuous first derivatives at each data point. When reconstructing signals from frequency-domain data, this lack of smoothness introduces artificial high-frequency artefacts that degrade the quality of the inverse FFT reconstruction. Higher-order methods such as RBF and cubic spline interpolation produce smooth, continuously differentiable curves that better preserve the spectral characteristics of the original signal.

The Gaussian-kernel RBF interpolation routine is given in Algorithm 5.

Algorithm 5: Radial Basis Function (RBF) interpolation using Gaussian function.

input: Real Data points (*real_data*), Imaginary data points (*imag_data*)
output: Interpolated data

- 1 $N \leftarrow$ array of zeros with shape (*interpolation_points*);
- 2 $x \leftarrow$ evenly spaced points between 0, $N - 1$ and N ;
- 3 $x_{reshaped} \leftarrow x[:, None]$;
- 4 $rbf_real \leftarrow \text{RBFInterpolation}(x_{reshaped}, \text{real_data}[:, None], \text{kernel}='gaussian', \text{epsilon}=1.0)$;
- 5 $rbf_imag \leftarrow \text{RBFInterpolation}(x_{reshaped}, \text{imag_data}[:, None], \text{kernel}='gaussian', \text{epsilon}=1.0)$;
- 6 $interpolated_real \leftarrow rbf_real(x_{reshaped}).flatten()$;
- 7 $interpolated_imag \leftarrow rbf_imag(x_{reshaped}).flatten()$;
- 8 $interpolated_data \leftarrow interpolated_real + interpolated_imag * 1j$;
- 9 **return** *interpolated_data*;

Using Algorithm 5, the decoder algorithm can be represented as Algorithm 6:

Algorithm 6: Decoder algorithm for Spiking Neural Networks.

input: Encoded spike pattern
output: Reconstructed data

reconstructed_frequency_domain $\leftarrow$

- 1 $\text{RBFWithGaussianInterpolation}(\text{encoded_spike_pattern})$;
- 2 $reconstructed_flattened_image \leftarrow \text{InverseFFT}(reconstructed_frequency_domain)$;
- 3 $reconstructed_image \leftarrow \text{Reshape}(reconstructed_flattened_image)$;
- 4 **return** *reconstructed_image*;

For temporal data, specifically audio data, the encoder algorithm is also converted to the frequency domain and then to quantised data points. But before that, noise reduction should be implemented, which in this case is done using Spectral Gates. The full speech encoder is given in Algorithm 7.

Algorithm 7: Encoder for Speech Commands.

input: Audio data
output: Binary spike train

- 1 $noise_reduced_audio \leftarrow \text{SpectralGate}(\text{audio})$;
- 2 $frequency_domain \leftarrow \text{FFT}(noise_reduced_audio)$;
- 3 $quantised_frequency_domain \leftarrow \text{SigmaDeltaQuantisation}(frequency_domain)$;
- 4 $spike_train \leftarrow \text{GenerateStepSignal}(quantised_frequency_domain)$;
- 5 **return** *spike_train*;

The only difference between the encoder for images and audio data is the noise reduction. In Algorithm 8, the decoder also uses RBF interpolation.

Algorithm 8: Decoder for Speech Commands.

input: Encoded spike pattern
output: Reconstructed audio data

- 1 $interpolated_fft_audio \leftarrow RBFWithGaussianInterpolation(encoded_spike_pattern);$
- 2 $reconstructed_audio \leftarrow InverseFFT(interpolated_fft_audio);$
- 3 **return** $reconstructed_audio;$

The final algorithm described in this subsection operationalises RQ1 and RQ2, both of which are evaluated experimentally in Chapter 5 and Chapter 6.

3.4.5 Implementation Challenges and Solutions

Entering the intricate domain of signal processing presented a steep learning curve, as familiarity with the subject was initially limited. The foundational step in the journey was selecting the optimal tools for developing the end algorithm. After careful consideration, Python emerged as the language of choice, primarily due to its rich ecosystem of supportive libraries, which were indispensable for this project. Although C++ with CUDA was a contender, particularly for its performance advantages, Python’s extensive library support, especially for signal processing, ultimately tipped the scales in its favour.

The initial stages of algorithm design were predicated on manipulating 1D data. However, the project’s scope expanded, necessitating the accommodation of 2D data. This transition was challenging, requiring a foray into more sophisticated dimensions of image manipulation. PyTorch proved instrumental during this phase, providing powerful tools that enabled the necessary algorithmic adaptations.

A central challenge observed was reconstructing the original data from the encoded spike trains. This was a complex process because of the need to understand in detail both the encoding mechanisms and the dynamic nature of spike responses within the Spiking Neural Network (SNN). The resolution to this conundrum was found through rigorous testing and validation, as shown in Figure 30 in Chapter 5 and Figure 56 in Chapter 6. This continuous process allowed for the refinement of the decoding algorithms, ensuring the fidelity of data reconstruction.

Resource constraints were another hurdle; minimal computational resources led to leveraging the available GPU to expedite computational tasks. This reliance on GPU processing was pivotal in maintaining the project’s momentum.

Prior experience with SNNs provided a foundational understanding, yet integrating encoders and decoders with SNNs was uncharted territory. Thus, we embarked on a journey to unravel the complexities of these components, learning their operational nuances and how they sync with the SNN framework.

Furthermore, working with the Speech Commands dataset highlighted the crucial role of data preprocessing. The audio data required meticulous preprocessing to ensure seamless compatibility with the encoder. This step was essential to transform the dataset into a suitable format for subsequent processing, underscoring the importance of initial data handling in the study's overall success.

To summarise, the study was a tapestry of challenges; each was addressed with a tailored solution that advanced it and contributed significantly to professional growth in signal processing and neural network implementation.

3.4.6 Improvements Over Existing Encoders

The encoder developed in this study introduces several design features that distinguish it from existing encoding mechanisms for Spiking Neural Networks (SNNs). The empirical evidence supporting these improvements is presented in the CIFAR-10 case study (Section 5) and Speech Commands case study (Section 6). The key design features include:

Advanced Preprocessing: Incorporation of noise reduction and band-pass filtering to improve signal clarity and quality before encoding.

Mel-Spectrogram Transformation: Use of Mel-Spectrograms to align audio signal processing more intimately with human auditory perception, enhanced the accuracy of audio-related tasks.

Custom Dataset Integration: Development of specialised dataset classes (e.g., Custom-SPEECHCOMMANDS) to seamlessly integrate transformations during data loading, ensuring streamlined workflows.

Dynamic Adaptability: The ability to handle both static and dynamic data efficiently makes the encoder versatile across different data types.

Optimised for GPUs: Designed to leverage GPU capabilities for faster processing, which is crucial for real-time applications.

Efficient Quantisation and Interpolation: Use of techniques like sigma-delta quantisation and RBF interpolation to ensure high data representation and reconstruction fidelity.

Detailed Visualisations: Enhanced visualisation tools to inspect and verify each processing pipeline stage, from raw waveforms to frequency spectra, aiding in deeper analysis and understanding.

Robust Testing Framework: Implementation of thorough testing protocols using metrics like MSE, RMSE, PSNR, SNR, and SSI to validate the effectiveness and accuracy of the encoder.

3.5 Testing and Validation

A multifaceted approach to testing and validation was implemented to provide a comprehensive evaluation of the reconstructed signals and ensure an accurate Spiking Neural Network (SNN) model. This involved using several quantitative metrics to assess signal quality and model performance aspects.

3.5.1 Quantitative Metrics for Signal Reconstruction

For signal reconstruction, the Mean Squared Error (MSE) (Wang & Bovik, 2009) and Root Mean Squared Error (RMSE) were primarily utilised to quantify the average discrepancies between the original and reconstructed signals, providing insights into amplitude fidelity. The Peak Signal-to-Noise Ratio (PSNR) (Wang & Bovik, 2009) and Signal-to-Noise Ratio (SNR) evaluated the relationship between signal strength and noise, which is particularly crucial in scenarios where background noise can obscure signal integrity. Furthermore, the Structural Similarity Index (SSI) (Wang & Bovik, 2009) was employed on image-based datasets to assess the perceptual quality of the reconstructed images by comparing their luminance, contrast, and structural features with those of the original images. Together, these metrics ensured a detailed and nuanced assessment of the quality and accuracy of the signal reconstruction process.

3.5.2 Summary of Reconstruction Performance

Table 2 presents a representative subset of DyNED's reconstruction performance across both datasets. The full ablation studies spanning nine quantisation levels are detailed in Chapter 5 (Table 7) and Chapter 6 (Table 25).

Table 2: DyNED reconstruction quality at selected quantisation levels. Bold rows indicate operating points used in classification experiments. Quality metric is SSI for images and SNR for speech.

Dataset	Levels	Spike Density	Quality	RMSE
CIFAR-10	8	0.630	SSI: 0.810	0.194
CIFAR-10	32	0.875	SSI: 0.980	0.040
CIFAR-10	256	0.983	SSI: 1.000	0.005
Speech Commands	8	0.502	SNR: 20.75 dB	0.006947
Speech Commands	32	0.717	SNR: 33.99 dB	0.001493
Speech Commands	256	0.909	SNR: 52.31 dB	0.000182

3.5.3 Validation and Testing of SNN Models

The SNN models were rigorously tested and validated to perform effectively across the CIFAR-10 and Speech Commands datasets. This involved several steps:

Dataset-Specific Preprocessing: Each dataset requires unique preprocessing strategies to optimise it for SNN processing. For CIFAR-10, this involved normalising and resizing images to ensure uniformity, while for Speech Commands, audio signals were preprocessed to normalise volume and reduce background noise.

Cross-Validation Techniques: To ensure that the models generalised well across unseen data, k -fold cross-validation was employed. This technique involved dividing the dataset into k number of subsets and training the model on $k - 1$ subsets while using the remaining subset for testing. This process was repeated so that each subset was used as the testing set at least once.

Performance Metrics: In addition to MSE, RMSE, PSNR, SNR, and SSI, classification accuracy, precision, recall, and F1-score were calculated to assess model performance in correctly categorising the inputs. These metrics provided a comprehensive overview of the model's effectiveness and ability to handle static (CIFAR-10) and dynamic (Speech Commands) inputs.

Robustness Testing: The models were subjected to various conditions that tested their robustness, including input noise variations and different types of signal distortions. This was crucial to ensuring the models remained effective across diverse, potentially challenging real-world conditions.

Comparative Analysis: The performance of the SNN models was also compared with that of traditional neural network models to benchmark their efficiency and accuracy. This comparative analysis highlighted the advantages and potential limitations of using SNNs for specific data processing tasks.

Detailed case studies are available for an in-depth exploration of these validations in real-world contexts. These studies provide comprehensive insights into the practical implementations and the outcomes of the tests conducted:

CIFAR-10 Case Study: The methodologies and specific results for the CIFAR-10 dataset are thoroughly examined in Chapter 5. This case study illustrates the application of the theoretical concepts discussed in this chapter and presents a detailed analysis of the encoder and decoder performance with static image data.

Speech Commands Case Study: Similarly, the application of SNN models to the dynamic, variable Speech Commands dataset is detailed in Chapter 6. This case study provides a closer look at how these models handle real-time audio processing challenges, demonstrating their practical effectiveness and adaptability.

These studies bolster theoretical discussions with empirical evidence and highlight the scalable, adaptable nature of SNNs in handling complex, real-world datasets. They serve as crucial references for understanding the detailed application of SNN models and offer valuable insights into their operational strengths and areas for further enhancement.

The reconstruction-quality metrics introduced in this section are the measurement instruments used to test RQ1, with per-dataset results reported in Chapter 5 and Chapter 6.

3.6 Conclusion

This chapter has explored encoders and decoders within the Spiking Neural Networks (SNNs) framework, focusing on static and dynamic models. These components are crucial for translating real-world data into the spike-based language of SNNs and back, facilitating a broad spectrum of applications from robotics to pattern recognition. The journey through the design, functionality, challenges, and implementation of these encoders and decoders highlights the complexity of such systems and their immense potential to bridge the gap between biological neural processes and artificial neural computing.

The detailed examination of various encoding and decoding strategies reveals a significant evolution from basic models to more advanced, adaptive systems capable of handling complex, dynamic inputs. The design considerations discussed underscore the need to achieve a delicate balance between data fidelity and computational efficiency. This balance is pivotal to maximising the performance and applicability of SNNs across different domains.

Moreover, the chapter addressed the challenges encountered in developing these systems, ranging from the steep learning curve of entering a new scientific domain to the technical hurdles of handling high-dimensional data and ensuring real-time processing. The solutions to these challenges were found through innovative adaptations and the strategic application of computational resources, illustrating the field's adaptive nature.

Validation and testing protocols were rigorously outlined to ensure that the encoders and decoders meet theoretical expectations and perform effectively in practical, real-world applications. Here, we saw the use of comprehensive metrics such as MSE, RMSE, PSNR, SNR, and SSI for signal reconstruction, and performance metrics for model evaluation. This provided a thorough assessment of the systems developed, ensuring their reliability and effectiveness.

As this chapter closes, it is evident that developing encoders and decoders for SNNs is a dynamic field that requires continuous innovation and adaptation. The advancements discussed herein not only enhance our understanding of neural processing but also open up new possibilities for applying SNNs in complex computational tasks. Future work in the field, beyond the scope of this thesis, will undoubtedly build on the foundations laid in this chapter, pushing the boundaries of what is possible with neuromorphic computing and further closing the gap between artificial and biological systems.

In summary, the exploration of static and dynamic encoders and decoders for SNNs not only highlights the challenges and complexities involved but also showcases the potential of these systems to revolutionise the way we process and interpret data in an increasingly digital world. As technology advances, so will our ability to create more efficient, adaptable, and intelligent computational models, potentially transforming numerous sectors and enhancing our interaction with technology.

Read against the RPS, this chapter has covered the encoding-and-decoding problems for both static and dynamic data and has laid out DyNED as a single, reconstruction-faithful pipeline addressing RQ1, RQ2, and RQ3. The remaining half of the RPS - the absence of a lossless compressor tailored to the resulting binary spike streams - is the focus of Chapter 4, where DyNED_c is introduced to address Research Question 5 (RQ5) and Research Question 6 (RQ6) on the dynamic, sequential spike output that DyNED produces.

Chapter 4

DyNED_c: Dynamic Neural Encoder and Decoder Compressed

4.1 Introduction

For any neural network, data quality is very important (Klein & Rossin, 1999). However, that does not mean the quantity is not. Often, deep-learning neural networks require a large amount of data to train and learn complex patterns. Large amounts of data usually take a significant amount of time to train and require substantial computational resources, especially when transformers are trained. This is where we could use compression techniques to combine the best of both worlds - quality and quantity.

Data compression has been around for a long time. Morse code is said to be the first form of data compression, where a combination of dashes and dots (ITU, 2009) represent each character. Later, Shannon (1948) introduced the concepts of entropy and information theory, which have become the basis for data compression techniques. Today, we use data compression in almost every aspect of our lives, from sending emails to watching videos online. As of 2024, according to the International Telecommunication Union (ITU), around 5.5 billion people use the internet, which is around 68% of the world's population (ITU, 2025). Moreover, the increase in the number of users is directly proportional to the amount of data generated. Moving large amounts of data requires a higher bandwidth, which is not always available. Data compression, such as HTTP/3 (QUIC protocol (Bishop, 2022)), helps reduce the size of data transfer, reducing the bandwidth required.

Our brain is a complex organ that can store and process large amounts of data in a tiny space. A study conducted by Bruce (2010); Micheva et al. (2010); Smith & Nichols (2018) quotes that “One synapse, by itself, is more like a microprocessor - with both memory-storage and information-processing elements - than a mere on/off switch. In fact, one synapse may contain on the order of 1,000 molecular-scale switches. A single human brain has more switches than all the computers, routers, and Internet connections on Earth”. Imagine how much data it can store and process? There have been several studies conducted by Al Roumi et al. (2023); Allen-Zhu et al. (2014); Benna & Fusi (2021); Motiwala et al. (2022); Schütt et al. (2024); Zhou et al. (2022), which suggest we use some form of data compression in our brain to store, retrieve and process memory.

This chapter gives us an insight into the use of a binary compressor called DyNED_c, which builds on top of the DyNED encoder-decoder algorithm explored in Chapter 3. We will also analyse how this compression reduces the data size while maintaining its quality. We will also look at the design considerations, implementation details, testing and validation of the algorithm. Exploring such algorithms in depth will help us understand how compression can bridge the gap between biological and artificial systems, enabling more optimised and reliable neural networks that can meet the increasing demands of modern AI applications.

Read against the RPS, this chapter addresses the lossless-compression half of the problem - the absence of a compressor tailored to binary spike streams. The material covered in the remainder of Chapter 4 bears directly on RQ5 and RQ6, both of which are taken up experimentally in Chapter 5 and Chapter 6.

4.2 Background and Theory

Data compression can be broadly classified into two categories: lossy and lossless. Lossy compression is a technique in which some data is lost during the compression process, while lossless compression is a technique in which no data is lost. Lossy compression is often used for images, videos, and audio, where some data loss is acceptable. Lossless compression is often used in text, document, and program compression, where no data loss is acceptable. More information on this can be found in 2.7. Some recent studies have used data compression techniques in neural networks to speed up learning and reduce computational resources (Sharma et al., 2025; M. Zhang et al., 2025). SNNs have been shown to be more energy- and memory-efficient than traditional neural networks (Henkes et al., 2024). However, some experiments performed using DyNED and DSNN have taken a longer duration to train the data. This is an instance where data compression

could be utilised to reduce the data size and speed up the training process. The implementation of this compression technique will be further discussed in Section 4.4.

4.2.1 Importance of Data Compression in Neural Networks

High-quality data is crucial in training neural networks, particularly deep learning SNNs. However, increased training data presents significant challenges regarding computational resources and their efficiency. While DyNED successfully addressed the challenge of minimising data loss during conversion of raw, real-valued data to higher-dimensional binary spike data, the resulting data size still required substantial computational resources for training. In response to these challenges, DyNED_c introduces data compression techniques to reduce data size while preserving data quality.

Recent research in neural network compression has primarily focused on model-centric approaches, such as compressing network weights to reduce model size and accelerate inference processing (Baietto et al., 2025; H. Liu et al., 2024; Q. Liu et al., 2024; Tang et al., 2025). A comprehensive review by Xie et al. (2024) reveals that existing SNN compression research predominantly emphasises pruning, weight quantisation, and knowledge distillation techniques. However, there remains a notable gap in research exploring the potential of compressed input data for SNNs.

This study addresses this research gap by investigating the integration of data compression techniques with SNNs, specifically focusing on compressing input data before network training. This approach aims to optimise storage requirements and computational efficiency while maintaining the network's learning capabilities.

4.3 Design Consideration

Designing a data compression for an encoder-decoder that can be used for both static and dynamic data requires careful consideration. Do we compress the raw values or the encoded values? How do we ensure that compression does not degrade data quality? How do we ensure that the compression does not affect the training process? These are some of the questions that need to be answered before designing a data compression algorithm. In this section, we will look at the design considerations for the DyNED_c algorithm, which is based on the DyNED encoder-decoder algorithm (Chapter 3).

4.3.1 Criteria for Effective Compression

An effective compression algorithm must reduce data size while preserving essential information. Depending on the application, the efficiency of a compression method

is determined by various factors, including the compression ratio, computational cost, adaptability, and its impact on data quality. Striking the right balance between compression efficiency and usability is key, especially in neural network compression, real-time data processing, and large-scale data storage.

Compression Ratio: The ratio of the compressed data size to the original data size.

A lower compression ratio indicates better compression, and lossy techniques generally achieve higher ratios than lossless methods. Lossless compression allows perfect reconstruction of the original data, making it suitable for critical applications such as text storage, executable programs, and financial records. In contrast, lossy compression sacrifices some information to achieve more significant size reductions, commonly used in multimedia applications such as image, video, and audio compression.

Computational Efficiency: It refers to both the time required to compress data and to decompress it. Compression in real-time applications such as video streaming must be fast enough to minimise latency. Similarly, an efficient compression technique can reduce computational resource consumption without degrading performance in machine learning and neural network training. Additionally, memory overhead should remain low, particularly for embedded systems, mobile devices, and edge computing environments where storage and processing capabilities are limited.

Robustness against Noise and Errors: In many cases, compressed data must be transmitted across unreliable networks where corruption can occur. Compression methods should either integrate error-detection and correction mechanisms or remain resilient to data loss by allowing partial recovery. Some modern compression algorithms employ checksums, error-correcting codes (ECC), or redundancy-based encoding to maintain data integrity in noisy environments.

Hardware and Energy Efficiency: It is another critical aspect, particularly in low-power devices and embedded systems. When deployed in Internet of Things (IoT) networks, smartphones, or battery-operated AI models, compression techniques should minimise power consumption. While high-compression algorithms often demand significant computational resources, they must be designed to balance energy efficiency and performance to be practical for real-world deployment.

Compatible with Downstream Applications: Data compression should not introduce unnecessary decoding overhead or reduce interpretability when applied to machine learning, databases, or networking protocols. Additionally, format compatibility is crucial, ensuring that the compressed data can be processed across different

platforms and integrated with existing standards, such as HTTP compression (e.g., Brotli, Gzip), as well as neural network weight quantisation methods.

Neural Network Compression: Effective techniques must ensure the model’s accuracy is preserved, while reducing its storage and computational requirements. This includes methods like weight pruning, quantisation, and knowledge distillation. An ideal compression technique for neural networks should improve inference speed and reduce energy consumption while maintaining model performance. Compression should also leverage the sparsity inherent in neural network architectures, eliminating the least important weights without compromising prediction quality. In the context of DyNED_c, the primary goal is to develop a compression method that efficiently reduces data size while preserving the integrity of neural network computations, ensuring that AI models remain lightweight and powerful.

4.3.2 Challenges in Designing for SNN

The unique way data is fed into Spiking Neural Networks (SNNs) poses a significant challenge for designing a compression algorithm. Unlike conventional neural networks, which process real-valued inputs such as images, text, or structured numerical data, SNNs operate on binary spike trains, where each input is either 0 or 1, representing the presence or absence of a spike. This fundamental difference makes it difficult to apply traditional compression methods directly, as most are optimised for real-valued data.

One of the primary challenges lies in the incompatibility of conventional compression techniques with the binary nature of spike-based data. Algorithms such as Huffman coding (Huffman, 1952) (See Section 2.7.3), Arithmetic coding (Witten et al., 1987) (See Section 2.7.4), or even standard lossy techniques like JPEG and MP3 are tailored for datasets with a wide range of values, allowing them to efficiently exploit redundancies in floating-point representations. However, binary data lacks this variability, making it harder to find optimal encoding schemes that significantly reduce data size while maintaining fidelity.

Another key challenge is that spike trains often exhibit sparsity and temporal dependencies, meaning that spikes occur at specific times rather than continuously. A well-designed compression algorithm must leverage this sparsity to achieve high compression efficiency. Run-Length Encoding (RLE) (Golomb, 1966) (See Section 2.7.2) is one approach that can be applied, as it effectively compresses sequences with long runs of repeated values. However, naive RLE may not always be optimal, especially for complex spike patterns that encode meaningful temporal information.

Moreover, preserving the structure of spike timing is critical for SNNs to function correctly. Unlike traditional neural networks that rely on static numerical values, SNNs depend on precise spike timing and order for information processing. Therefore, any compression method must ensure that it does not introduce timing distortions or cause loss of important spike sequence information, as this could negatively impact the network's ability to learn and make accurate predictions.

Finally, computational efficiency is a crucial factor when designing compression algorithms for SNNs. Since SNNs are often deployed in low-power environments, such as neuromorphic chips and embedded systems, the compression technique must be computationally lightweight and energy-efficient. Algorithms that introduce high decompression overhead may offset any benefits gained from reduced storage, making them impractical for real-time spike processing.

4.4 Implementation Details

In this section, we describe the implementation of the DyNED_c algorithm, which builds upon the DyNED encoder-decoder framework. The implementation is designed to effectively compress binary spike-based data used in Spiking Neural Networks (SNNs) while preserving critical temporal and structural information.

4.4.1 Development Environment and Tools

Similar to Section 3.4.2, DyNED_c uses Python as its primary programming language due to its extensive libraries for numerical computing, machine learning, and data processing. The development environment for DyNED_c includes the following tools:

PyTorch (Paszke et al., 2019): Primarily leveraged not just as a framework for neural networks but also as an advanced replacement for NumPy, especially for tasks involving image manipulation. Its dynamic computation graph and GPU acceleration capabilities were crucial for efficiently processing complex, high-dimensional datasets.

Matplotlib (Hunter, 2007): Utilised for generating high-quality visualisations of the data and results, aiding in analysis and presentation.

NumPy (Harris et al., 2020): A fundamental package for numerical computations in Python, critical for handling and processing large datasets.

SciPy (Virtanen et al., 2020): Provides additional scientific and technical computing tools, enhancing data manipulation and analysis processes.

Plotly (Plotly, 2015): Similar to Matplotlib, Plotly was used for interactive visualisations, offering a dynamic and engaging way to explore the data and results.

UV (UV, 2024): An extremely fast Python package and project manager, written in Rust. The development was conducted on a Linux-based system, specifically Ubuntu 20.04 (Canonical, 2004), chosen for its stability and broad support within the scientific computing community. The hardware was deployed on DigitalOcean’s 20 virtual CPUs with 240GB of RAM, ensuring swift computation and ample memory for handling complex data operations. A significant acceleration in the encoding process was achieved by utilising an NVIDIA H100 GPU. This high-performance GPU significantly reduced computation times, enabling more iterative testing and refinement of the encoder algorithm.

4.4.2 Modification and Enhancements

A few key modifications were made to make Huffman Coding and Run-Length Encoding (RLE) work effectively on binary data, particularly in the context of SNNs. Traditional compression techniques are generally designed for real-valued data such as text, images, and audio, but binary spike data presents unique challenges. This section outlines the modifications necessary to handle binary sequences efficiently, while maintaining high compression ratios and low computational overhead.

One major limitation of standard RLE when applied to binary sequences is that it often introduces overhead when encoding short run lengths. A naive approach would store fixed-bit lengths for run sizes, which wastes space when encoding short runs. To address this, the implemented algorithm uses variable-length encoding for run lengths, allowing shorter runs to be stored in fewer bits while longer runs are dynamically allocated more bits. Instead of encoding every count with a fixed 3- or 4-bit length, it was determined that the optimal bit length should be based on the actual run size, ensuring that no unnecessary bits were stored.

Additionally, standard RLE struggles with single-bit spikes, where individual 1s or 0s appear mixed. Encoding these using traditional RLE (1,1 or 1,0) can increase the data size rather than reduce it. To address this, the algorithm includes an alternating-run RLE variant that exploits the binary nature of the data: since runs of 0s and 1s always alternate, only the run lengths and a single starting value need to be stored, eliminating the per-run symbol overhead entirely. The run-length values are then Huffman-encoded directly, capturing the frequency distribution of run lengths. This approach is particularly effective for very sparse or very dense spike trains.

While RLE optimises the representation of long runs, Huffman Coding needs to be adapted to handle binary data efficiently. The problem with standard Huffman encoding is that it assigns variable-length codes based on symbol frequencies, which

works well for multi-symbol data (e.g., text characters) but is inefficient when applied directly to binary values. Since binary data has only two symbols, Huffman cannot effectively reduce the size if 0s and 1s occur with equal probability. To overcome this, this study introduced a chunk-based Huffman encoding scheme, in which individual bits are grouped into fixed-size chunks before Huffman compression. This chunking technique allows Huffman to capture repeating patterns in spike sequences, significantly improving compression performance.

Another key enhancement was applying Huffman Coding after RLE compression, rather than directly to the raw binary data. Since RLE reduces redundancy first, Huffman can assign shorter codes to more frequent RLE patterns, resulting in a more compact final encoding. This two-stage process efficiently compresses long runs and frequent patterns, achieving a better compression ratio than either method used alone.

To further improve compression efficiency, we introduced a hybrid mode that dynamically selects between four strategies based on the structure of the binary data: (1) variable-length RLE only, (2) chunk-based Huffman only, (3) RLE followed by chunk-based Huffman, and (4) alternating-run RLE with Huffman coding on the run-length values. The algorithm compresses the input using all four strategies and selects the one that produces the smallest output. This adaptive selection ensures optimal compression across varying spike densities - for example, the alternating-run mode excels on very sparse or very dense data. At the same time, chunk-based Huffman captures repeating patterns more effectively at moderate densities.

These modifications enable Huffman Coding and RLE to work effectively on binary spike data while preserving spike timing and ensuring computational efficiency. By implementing variable-length run encoding, chunk-based Huffman compression, adaptive RLE, and hybrid mode selection, we achieve a balance between high compression efficiency, low memory usage, and fast decompression. These enhancements make DyNED_c well-suited for Spiking Neural Networks, neuromorphic computing, and real-time AI applications.

4.4.3 Final Algorithm Design

The data from the DyNED algorithm is fed into the compression algorithm shown in Algorithm 9. The algorithm tries four compression strategies in parallel and selects the one that produces the smallest compressed output. The four strategies are: (1) variable-length RLE alone, (2) chunk-based Huffman alone, (3) variable-length RLE followed by chunk-based Huffman, and (4) alternating-run RLE with Huffman coding applied to the

run-length values. The compressed data is then stored along with metadata such as the original data length, compressed length, compression ratio, and the selected mode. The decompression process reverses the steps of the selected mode.

Algorithm 9: Hybrid Binary Compression: Adaptive Mode Selection

input: Binary string *binary_data* of 1s and 0s.
output: Tuple of (*compressed_string*, *compression_metadata*).
mode 1: Variable-length RLE.

```

1 | compressed_rle  $\leftarrow$  []
2 | count  $\leftarrow$  1, current  $\leftarrow$  binary_data[0]
3 | for each i  $\leftarrow$  1 to length(binary_data) - 1 do
4 |   if binary_data[i] = current then
5 |     count  $\leftarrow$  count + 1
6 |     if count = 255 then
7 |       flush count_size + count_binary + current to compressed_rle
8 |       count  $\leftarrow$  0
9 |     end if
10 |   else
11 |     if count > 0 then
12 |       flush count_size + count_binary + current to compressed_rle
13 |     end if
14 |     current  $\leftarrow$  binary_data[i], count  $\leftarrow$  1
15 |   end if
16 | end for
17 | flush remaining run
18 | candidate_1  $\leftarrow$  ".join(compressed_rle)

mode 2: Chunk-based Huffman.
19 | split binary_data into fixed-size chunks (e.g. 4 bits)
20 | build Huffman tree from chunk frequencies
21 | candidate_2  $\leftarrow$  Huffman-encode each chunk

mode 3: RLE + chunk-based Huffman (hybrid).
22 | candidate_3  $\leftarrow$  apply chunk-based Huffman to candidate_1

mode 4: Alternating-run RLE + Huffman.
23 | extract alternating run lengths and starting value from binary_data
24 | build Huffman tree from run-length value frequencies
25 | candidate_4  $\leftarrow$  Huffman-encode each run-length value

select best mode:
26 | best  $\leftarrow$  candidate with smallest length

output metadata:
27 | compressed_metadata  $\leftarrow$  { original_length: length(binary_data), compressed_length:  

   | length(best), compression_ratio: length(best) / length(binary_data), mode:  

   | selected_mode }
28 | return (best, compressed_metadata)

```

The DyNED_c algorithm combines the strengths of RLE and Huffman Coding to achieve high compression ratios on binary spike data. By trying all four strategies and selecting

the best, DyNED_c adapts to the specific characteristics of each spike train - whether sparse, dense, or structured. The alternating-run mode exploits the binary nature of the data by storing only run lengths without per-run symbols, while the chunk-based Huffman mode captures repeating bit patterns. This adaptive selection ensures optimal compression across varying spike densities without manual tuning. The code can be found in Section A.6.

Decompression is the reverse process of compression, as shown in Algorithm 10. The decompression path depends on the mode selected during compression. For RLE-only mode, the RLE stream is decoded directly. For Huffman-only mode, the Huffman codes are reversed to recover the original data. For hybrid mode, Huffman decoding is applied first to recover the RLE-compressed stream, which is then RLE-decoded. For alternating-run mode, Huffman decoding recovers the run-length values, and the binary sequence is reconstructed by alternating symbols starting from the stored starting value.

Algorithm 10: Hybrid Binary Decompression: Mode-Based Reconstruction

```

input: Compressed binary string compressed_data, mode, and huffman_codes (if applicable).
output: Original binary string binary_data.
if mode = "rle" then
1  decode variable-length RLE directly:
2  binary_data  $\leftarrow []$ , i  $\leftarrow 0$ .
3  while i < length(compressed_data) do
4    count_size  $\leftarrow \text{int}(\text{compressed\_data}[i:i+3], 2) + 1$ .
5    i  $\leftarrow i + 3$ .
6    count  $\leftarrow \text{int}(\text{compressed\_data}[i:i+\text{count\_size}], 2)$ .
7    i  $\leftarrow i + \text{count\_size}$ .
8    value  $\leftarrow \text{compressed\_data}[i]$ , i  $\leftarrow i + 1$ .
9    append value  $\times$  count to binary_data.
10 end while
    else if mode = "alt_huffman" then
11 decode Huffman to recover run lengths, then reconstruct by alternating symbols:
12 runs  $\leftarrow$  Huffman-decode compressed_data using huffman_codes
13 current  $\leftarrow \text{start\_value}$ 
14 for each length  $\in$  runs do
15   append current  $\times$  length to binary_data
16   current  $\leftarrow \text{toggle}(\text{current})$ 
17 end for
    else (huffman or hybrid)
18   step 1: huffman_decoded  $\leftarrow$  Huffman-decode compressed_data using huffman_codes
19   if mode = "huffman" then return huffman_decoded
20   step 2: decode variable-length RLE on huffman_decoded (same as rle mode above)
return binary_data.

```

The hybrid RLE-then-Huffman algorithm described in this subsection is the design under test for RQ6, evaluated per-dataset in Chapter 5 and Chapter 6.

4.4.4 Implementation Challenges

As discussed in Section 3.4.5, one of the main difficulties encountered during the implementation of DyNED_c was understanding the interactive nature of signal processing and data compression. Combining information theory with signal processing presented a unique challenge, as it required a deep understanding of both domains, the mathematical principles governing compression and the real-time constraints of GPUs.

A significant hurdle was adapting traditional compression techniques to binary spike data. Most conventional compression algorithms, including Huffman Coding and Run-Length Encoding (RLE), are optimised for real-valued data. However, spike-based data in Spiking Neural Networks (SNNs) is inherently binary and sparse, meaning that the application of standard compression methods often results in suboptimal performance or even an increase in data size. To overcome this, we introduced modifications to RLE and Huffman Coding that maintained compression efficiency without distorting spike timing, a crucial property for SNNs.

Although the conceptual design of DyNED_c was structured, implementing it efficiently required overcoming scalability and computational efficiency challenges. The availability of numerical computing libraries such as PyTorch and NumPy facilitated implementation, making matrix operations and data transformations more efficient. However, optimising the algorithm for large-scale data was still a significant challenge. Large SNN datasets contain millions of spikes per second, requiring a high-speed compression and decompression pipeline that operates in real time.

Moreover, ensuring low-latency decompression was as important as achieving high compression ratios. While efficient at reducing data size, many compression algorithms introduce high computational overhead during decompression, making them impractical for real-time SNN applications. Therefore, significant effort was devoted to ensuring that decompression remained lightweight and computationally efficient, enabling real-time signal reconstruction with minimal delay.

The key implementations include integrating signal processing with information theory, adapting compression algorithms for binary spike data, optimising computational efficiency, and ensuring real-time decompression. Addressing these requires theoretical modifications to existing algorithms and practical optimisations in their implementation to handle large-scale neuromorphic data efficiently.

These challenges were successfully overcome through a combination of theoretical insights, algorithmic modifications, and computational optimisations, resulting in a robust, efficient compression algorithm specifically tailored for Spiking Neural Networks.

4.4.5 Improvement over Existing Techniques

The modifications introduced in Section 4.4 enable DyNED_c to consistently outperform any single traditional compression method across varying spike densities. Table 3 and Figure 25 compare DyNED_c against standard RLE, chunk-based Huffman, and LZW on

DyNED-encoded spike trains at five quantisation levels. The test signal is a composite sinusoidal waveform (5Hz + 13Hz with additive Gaussian noise), sampled at 1,000 points. This synthetic signal was chosen because its combination of multiple frequency components and noise produces spike patterns representative of real-world signals - containing both structured runs and irregular transitions - making it suitable for evaluating compression across a range of spike densities. The quantisation levels refer to the number of discrete amplitude steps used by DyNED's encoding process (Chapter 3) - higher levels produce finer quantisation, resulting in denser spike trains where more samples change value between consecutive time steps.

Table 3: Compression ratio comparison on DyNED-encoded spike trains. Lower is better. DyNED_c adaptively selects the best mode, matching or outperforming all individual methods at every density.

Levels	Density	RLE ↓	Huffman ↓	LZW ↓	DyNED _c ↓	Mode
16	0.635	2.436	0.940	1.520	0.940	Huffman
32	0.740	2.142	0.820	1.392	0.820	Huffman
64	0.862	1.395	0.574	1.112	0.574	Huffman
128	0.937	0.733	0.393	0.707	0.379	Alt. RLE
256	0.970	0.398	0.315	0.553	0.178	Alt. RLE

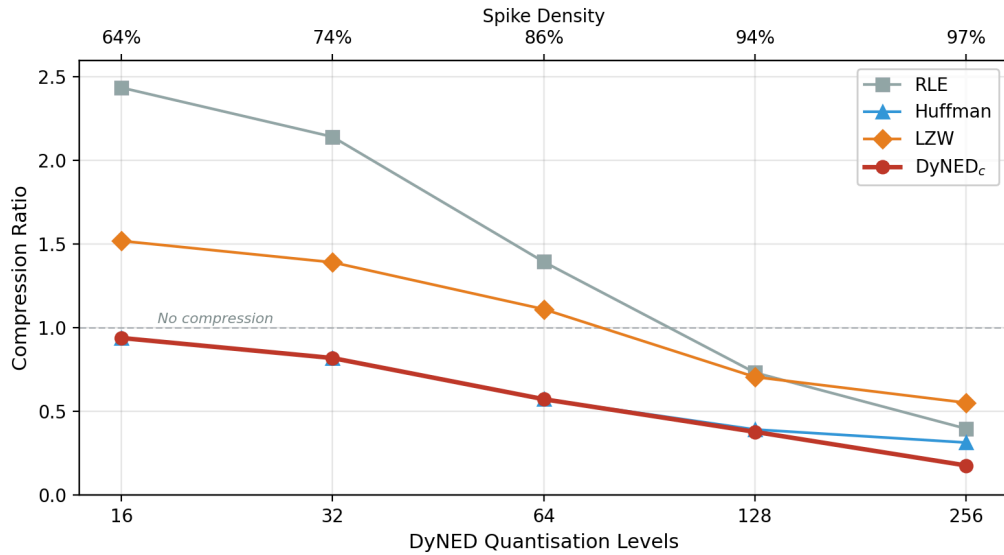


Figure 25: Compression ratio comparison across DyNED quantisation levels. DyNED_c (red) consistently achieves the lowest ratio, adaptively switching between Huffman and alternating-run modes as spike density increases.

At moderate densities (16-64 levels), the chunk-based Huffman mode is automatically selected, capturing repeating bit patterns more effectively than RLE. At higher densities (128-256 levels), the alternating-run mode is selected, exploiting the binary alternation property to achieve up to 82.2% space savings. The adaptive mode selection described in Algorithm 9 ensures that DyNED_c never performs worse than the best individual strategy for any given input, without requiring manual parameter tuning.

As a lossless compression method, DyNED_c guarantees perfect reconstruction of the original spike sequence, preserving spike timing and event-based encoding. Combined with GPU-accelerated compression and decompression, these properties make DyNED_c a scalable compression technique tailored for SNN applications. Detailed results on CIFAR-10 and Speech Commands data are presented in Chapter 5 and Chapter 6, respectively.

4.5 Testing and Validation

To ensure the effectiveness of DyNED_c, the algorithm underwent rigorous testing and validation across four datasets - Google Speech Commands and CIFAR-10, together with their neuromorphic counterparts Spiking Speech Commands and CIFAR10-DVS. The validation process focused on compression performance, decompression accuracy, computational efficiency, and real-time applicability in Spiking Neural Networks (SNNs). This section outlines the quantitative metrics for signal decompression and the testing methodology used to evaluate the system.

4.5.1 Quantitative Metrics for Signal Decompression and Testing

The effectiveness of DyNED_c is measured using several key quantitative metrics that assess the quality of decompressed spike sequences relative to the original input. These metrics ensure that the compression methods maintain data integrity while achieving high compression ratios.

Compression Ratio: As mentioned in Section 4.3.1, the compression ratio measures the reduction in data size achieved by the compression algorithm. It is given by Equation 24:

$$CR = \frac{\text{Compressed Size}}{\text{Original Size}} \quad (24)$$

A lower CR value indicates better compression efficiency, with values closer to zero representing higher compression gains.

Decompression Accuracy: Since DyNED_c is a lossless compression method, the decompressed signal should exactly match the original. The accuracy is verified using bitwise comparison between the original and decompressed sequences and is given by Equation 25:

$$DA = \left(\frac{\text{Number of Correct Bits}}{\text{Total Bits}} \right) \times 100\% \quad (25)$$

A DA value of 100% indicates perfect decompression accuracy, while lower values indicate data loss or corruption during decompression.

Mean Absolute Timing Error: Since SNNs rely on precise spike timing, any shift or loss in spike occurrence can degrade neural processing. The Mean Absolute Timing Error (MATE) metric is introduced in this thesis to evaluate timing preservation under lossless compression, where each original spike has a one-to-one correspondence with a decompressed spike (a setting where standard spike-train similarity metrics such as Victor-Purpura (Victor & Purpura, 1996) or van Rossum (van Rossum, 2001) do not apply, as they assume unrelated spike trains). It is given by Equation 26:

$$MATE = \frac{1}{N} \sum_{i=1}^N |T_i^{\text{original}} - T_i^{\text{decompressed}}| \quad (26)$$

where T_i represents the time step of each spike event. A MATE score of zero confirms perfect spike timing retention.

With these metrics, the computational speed and memory usage were also measured.

4.5.2 Summary of Compression Performance on Real Data

Table 4 presents DyNED_c compression performance on spike trains generated by DyNED from the four case-study datasets: CIFAR-10, Speech Commands, CIFAR10-DVS, and Spiking Speech Commands (SSC). Full comparisons against baseline compression algorithms are in Chapter 5 (Table 12 and Table 17) and Chapter 6 (Table 29 and Table 35).

Table 4: DyNED_c compression performance on real datasets (lower CR is better). DA = 100% and MATE = 0 on every dataset confirm lossless reconstruction.

Dataset	CR	Space Saved	DA	MATE
CIFAR-10	0.904	9.6%	100%	0
Speech Commands	0.007	99.3%	100%	0
CIFAR10-DVS	0.113	88.7%	100%	0
SSC	0.503	49.8%	100%	0

These compression-quality measurements provide the foundation for testing RQ5, evaluated in full alongside downstream classification in Chapter 5 and Chapter 6.

4.6 Conclusion

This chapter has delved into the design, implementation, and validation of the DyNED_c compression algorithm, emphasising its role in efficiently encoding and decoding binary spike data for Spiking Neural Networks (SNNs). This exploration underscores the intricate challenge of translating real-world signals into the spike-based domain without compromising temporal fidelity or incurring prohibitive computational costs. By combining variable-length RLE, chunk-based Huffman coding, alternating-run encoding, and adaptive mode selection, DyNED_c demonstrates a marked evolution beyond conventional compression schemes, offering a robust framework that preserves critical spike timing whilst significantly reducing data size.

The progression from conceptual design to practical deployment unveiled several challenges, particularly in handling sparse binary inputs and maintaining efficient performance. Addressing these issues called for innovative modifications - ranging from chunk-based Huffman approaches to adaptive RLE structures - showcasing the creative synergy between classical information theory and cutting-edge neuromorphic computing. The result is a balance between data integrity and computational efficiency, a cornerstone in maximising the applicability of SNN-driven solutions across diverse fields.

Rigorous testing and validation, including comprehensive performance metrics for compression ratio, timing accuracy, and computational overhead, provided concrete evidence of DyNED_c's effectiveness have been performed in Chapter 5 and Chapter 6. By preserving essential spike-event information, the algorithm supports an array of tasks where precise timing is paramount, thus reinforcing the potential of SNNs in real-time scenarios, from sensor networks to sophisticated AI-driven applications. Furthermore, using large-scale datasets and a GPU-accelerated implementation demonstrates that DyNED_c is not merely a theoretical construct but a scalable, real-world tool.

In summary, this chapter established DyNED_c as a lossless compressor purpose-built for the binary spike-train output of DyNED. The hybrid design - variable-length RLE, chunk-based Huffman coding, alternating-run encoding, and adaptive mode selection - was developed in response to the failure modes of off-the-shelf compressors on sparse binary data, and its decompression-accuracy and timing-fidelity guarantees were

formalised through the DA and MATE metrics. A GPU-accelerated reference implementation was produced and is used by every downstream experiment in this thesis.

With both halves of the RPS now addressed at the algorithmic level - reconstruction-faithful encoding-and-decoding in Chapter 3 and lossless spike-stream compression in this chapter - the remainder of the thesis turns to empirical validation. The next two chapters present case studies that test the full DyNED + DyNED_c pipeline end-to-end on four datasets spanning conventional and neuromorphic modalities for both image and audio. Chapter 5 takes up CIFAR-10 and its neuromorphic counterpart CIFAR10-DVS, addressing RQ1, RQ2, RQ3, RQ5, RQ6, Research Question 7 (RQ7), and Research Question 8 (RQ8) on visual data; Chapter 6 applies the same pipeline to Speech Commands and Spiking Speech Commands, addressing RQ3 and Research Question 4 (RQ4) specifically on auditory data alongside the broader RQ1, RQ2, RQ5, RQ6, RQ7, and RQ8.

Chapter 5

Case Study - CIFAR Dataset

5.1 Introduction

This chapter is the first of the two case-study chapters that empirically test the DyNED + DyNED_c pipeline assembled in Chapter 3 and Chapter 4 against the RPS. It addresses RQ1, RQ2, RQ3, RQ5, RQ6, RQ7, and RQ8 on visual data, using CIFAR-10 as the conventional benchmark and CIFAR10-DVS as its neuromorphic event-based counterpart.

Image classification in Spiking Neural Networks (SNNs) is a challenging task due to the static nature of the data. SNNs are event-driven networks designed for temporal data. In this case study, we will use the CIFAR-10 dataset (Krizhevsky et al., 2009) to classify images using a convolutional front-end combined with cAdLIF spiking layers with learnable per-neuron synaptic delays, trained with surrogate gradients and the Temporal Efficient Training (TET) loss, with the DyNED (Chapter 3) and DyNED_c (Chapter 4) architecture used for spike encoding and compression.

The CIFAR-10 dataset (Krizhevsky et al., 2009) is a widely used benchmark in machine learning research. It consists of 60,000 colour images of size 32×32 pixels, divided into 10 mutually exclusive classes: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. The dataset is split into 50,000 training images and 10,000 test images, with each class containing exactly 6,000 and 1,000 images, respectively.

Unlike character recognition datasets, CIFAR-10 poses a more challenging classification task. The images contain natural objects with complex backgrounds, varied poses, lighting conditions, and intra-class variation. For instance, the *cat* and *dog* classes share similar shapes and textures, while *automobile* and *truck* may overlap in structural

features. This diversity makes CIFAR-10 a suitable benchmark for evaluating the representational capacity of spiking neural networks on real-world visual data. Additionally, CIFAR-10's static nature directly tests whether DyNED can effectively encode non-temporal data into spike trains (Section 1.3.3.3), while the visual complexity of natural images tests whether DyNED's frequency-domain encoding improves classification accuracy over direct time-domain encoding (Section 1.3.3.1). The dataset's moderate size (60,000 images) also enables meaningful evaluation of GPU acceleration during the encoding process (Section 1.3.3.2).

5.2 Relevance

Human perception is inherently spatio-temporal (Rao et al., 2001; Treisman & Gelade, 1980). Spatial awareness involves understanding the arrangement and relationships of objects in space, which enables us to navigate and interact with our environment effectively. Temporal awareness relates to our perception of time, allowing us to sequence events, anticipate future occurrences, and recall past experiences.

These cognitive processes are deeply interconnected. For instance, when navigating a new environment, we rely on spatial memory to recall landmarks and temporal memory to estimate the duration between them. Research indicates that different sensory modalities contribute uniquely to these processes; auditory information processing shows enhanced sensitivity to temporal information but lower sensitivity to spatial information, while visual information processing shows higher sensitivity to spatial information but lower sensitivity to temporal information (Loeffler et al., 2018).

Moreover, the brain contains specialised neurons, namely *place cells* and *time cells*, which encode spatial and temporal information. Studies have shown that place cells remain consistent while navigating a virtual environment, whereas time cells change with the task, suggesting that time and space are separable neural dimensions (Aghajan et al., 2023; Schonhaut et al., 2023).

Our cortical processing involves two primary pathways - Ventral and Dorsal streams. The Ventral streams process "What" information, including object detection and recognition. The Dorsal streams process the "Where" information, which includes spatial awareness or motion detection (Joo et al., 2024; Norman, 2002).

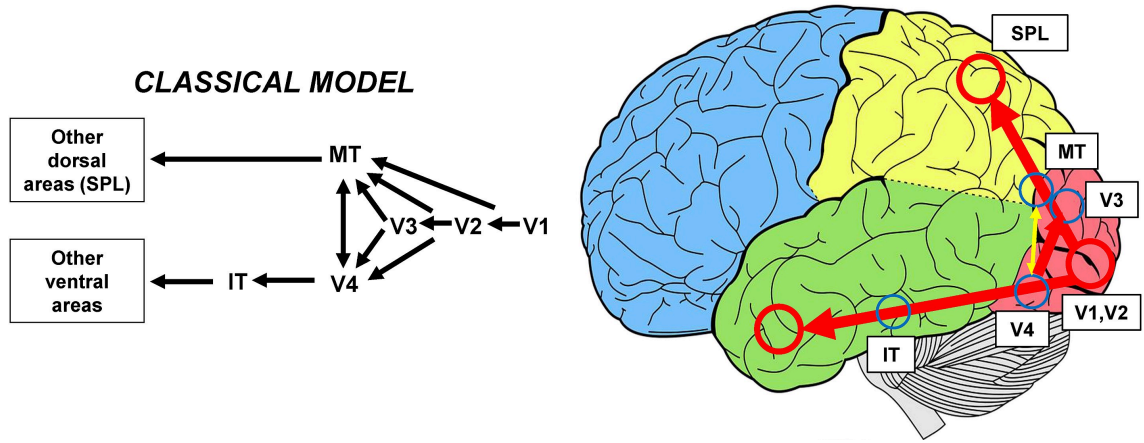


Figure 26: Visual pathways in the brain. Left side of the image shows a simplified schematic diagram and on the right is the visual aspect of it. Where V1, V2, V3 and V4 are the areas of visual cortex, MT is Middle Temporal visual area, IT is Inferior Temporal cortex and SPL is Superior Parietal Lobule (Choi et al., 2020, p. 6).

Figure 26 shows the brain’s visual pathways. In this case study, we will focus on the Ventral stream’s first process: the encoding of the image, followed by its classification.

5.3 Methodology

The proposed methodology for image classification using Spiking Neural Networks (SNN) comprises a convolutional encoder-decoder architecture that processes data through several sequential stages. This approach combines learnable convolutional feature extraction with biologically-inspired spiking dynamics to classify natural images.

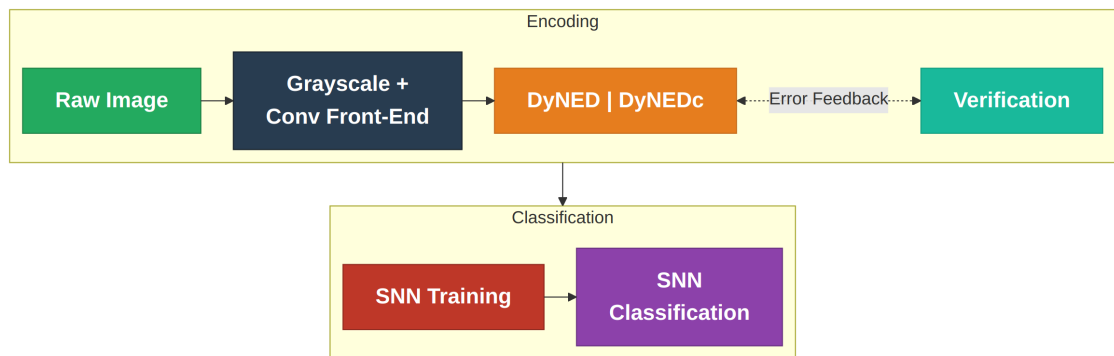


Figure 27: Methodology for image classification using Spiking Neural Networks (SNN).

The experiments begin with the raw CIFAR-10 images. The colour images are first converted to grayscale for reduced dimensionality, as colour information is not essential for the object categories in this dataset (Kanan & Cottrell, 2012). The grayscale images are then normalised and processed through a convolutional front-end to extract spatial

features, which are subsequently projected into a spiking neural network with cAdLIF layers. In separate encoding experiments, the images are transformed into the frequency domain using Fast Fourier Transform (FFT) and encoded into spike-trains using the DyNED encoder (Chapter 3), with optional compression using DyNED_c (Chapter 4).

5.3.1 System Integration and Performance

Throughout this process, the system maintains a feedback loop between the verification stage and the quantisation parameters, enabling continuous optimisation to balance classification accuracy with computational efficiency. This integrated approach ensures optimal image classification while balancing temporal precision and processing requirements. Performance is evaluated based on multiple criteria, including classification accuracy, processing latency, and energy efficiency, providing a comprehensive assessment of the system's effectiveness.

The experiments were conducted using the DigitalOcean cloud computing service, specifically a 20-core virtual CPU with 240 GB of RAM and an NVIDIA H100 GPU with 80 GB of dedicated memory. The encoder used Python 3.12 and PyTorch 2.5. Spikes (Gollahalli, 2019) library was used to generate the spike trains for BSA (Section 2.2.2) and Threshold-Based Representation (TBR) (Delbruck & Lichtsteiner, 2007).

Table 5: Comparison of memory usage, processing power, time taken, and GPU usage for DyNED, BSA, and TBR.

Encoder Type	Memory (%)	Processing (%)	Time	GPU Usage (%)
DyNED	≈ 46%	≈ 60%	≈ 4 s	≈ 63%
DyNED _c	≈ 50%	≈ 64%	≈ 7 s	≈ 65%
BSA (Gollahalli, 2019) ¹	≈ 96%	≈ 95%	≈ 8 s	✗
TBR (Gollahalli, 2019) ¹	≈ 94%	≈ 95%	≈ 3 s	✗
Delta (Eshraghian et al., 2023b)	≈ 40%	≈ 72%	≈ 1 s	≈ 52%
Latency (Eshraghian et al., 2023b)	≈ 60%	≈ 76%	≈ 4 s	≈ 73%
Rate (Eshraghian et al., 2023b)	≈ 58%	≈ 81%	≈ 6 s	≈ 68%

Table 5 reveals that for CIFAR-10, all encoders achieve fast encoding times on the order of seconds for the full 50,000-sample training set. DyNED encodes the dataset in approximately 4 seconds, with 46% memory usage, 60% CPU utilisation, and 63% GPU utilisation. The combined DyNED + DyNED_c pipeline completes in approximately 7 seconds. These times are comparable to those of the other encoders, with Delta being the fastest at approximately 1 second and BSA the slowest at approximately 8 seconds.

¹Does not use GPU. Running parallel in multi-processor CPU

The key distinction between encoders lies not in encoding speed but in resource efficiency and representation quality. BSA and TBR do not leverage GPU acceleration, relying instead on multi-processor CPUs, and consume substantially more system resources - BSA at 96% memory and 95% processing power, TBR at 94% memory and 95% processing power. In contrast, DyNED and the snnTorch encoders (Delta, Latency, Rate) achieve comparable or faster encoding times while consuming significantly less memory and processing power through GPU acceleration.

Delta operates at 40% memory and 72% processing power, finishing encoding in roughly 1 second, using 52% of GPU capacity. Latency uses 60% of memory and 76% of processing power (73% GPU usage), completing in approximately 4 seconds, and Rate uses 58% of memory and 81% of processing power (68% GPU usage), completing in approximately 6 seconds. DyNED’s advantage over these simpler encoders lies not in encoding speed but in the richness of its frequency-domain spike representation, which captures both magnitude and temporal structure through the FFT and sigma-delta quantisation pipeline.

5.3.1.1 Per-Sample Encoding and Compression Cost

To quantify the encoding overhead more precisely, all encoders were benchmarked over 1,000 CIFAR-10 samples. DyNED and DyNED_c use GPU acceleration and Numba JIT compilation for performance-critical loops. BSA and TBR use the Spikes (Gollahalli, 2019) library, while Delta, Latency, and Rate use snnTorch (Eshraghian et al., 2023b). Table 6 presents the per-sample mean encoding time and estimated full training set time for each encoder.

Table 6: Per-sample encoding time for all encoders on CIFAR-10 (32×32 grayscale images), benchmarked over 1,000 samples.

Encoder	Mean (ms)	Median (ms)	Std (ms)
DyNED	0.073	0.070	0.021
DyNED_c	0.072	0.070	0.011
DyNED + DyNED_c	0.145	0.141	0.025
BSA (Gollahalli, 2019)	0.150	0.146	0.023
TBR (Gollahalli, 2019)	0.054	0.053	0.023
Delta (Eshraghian et al., 2023b)	0.025	0.024	0.011
Latency (Eshraghian et al., 2023b)	0.087	0.085	0.015
Rate (Eshraghian et al., 2023b)	0.115	0.115	0.015

Table 6 shows that all encoders achieve sub-millisecond per-sample encoding times on CIFAR-10. DyNED encoding takes a mean of 0.073 ms per image, while DyNED_c compression adds 0.072 ms, for a combined cost of 0.145 ms per sample. Delta is the fastest encoder at 0.025 ms, followed by TBR at 0.054 ms. Latency (0.087 ms), Rate (0.115 ms), and BSA (0.150 ms) fall within a similar range to DyNED. These per-sample costs are negligible relative to the SNN training step, which includes both the forward pass through spiking layers and backpropagation, demonstrating that the encoding stages introduce minimal computational overhead to the classification pipeline.

5.3.2 Preprocessing

The process begins with the conversion of the CIFAR-10 colour images ($32 \times 32 \times 3$ RGB) to grayscale ($32 \times 32 \times 1$), reducing the input dimensionality while preserving structural information. The grayscale images are normalised with a mean of 0.5 and a standard deviation of 0.5 to centre the pixel values around zero. Data augmentation is applied during training, including random affine transformations (rotation $\pm 15^\circ$, translation $\pm 10\%$, scaling 90%-110%), random perspective distortion (scale 0.2, probability 50%), and random erasing (probability 10%).

For the DyNED encoding experiments, the grayscale images are flattened into 1D arrays and transformed into the frequency domain using the Fast Fourier Transform (FFT), which reveals the underlying frequency components and provides both magnitude and phase information. Operating in the frequency domain can reduce sensitivity to spatial shifts and aid feature extraction (Yuan et al., 2024). The zero-frequency component is shifted to the centre of the spectrum for analysis (Yuan et al., 2024).

Figure 28 visualises the complete DyNED + DyNED_c encoding pipeline applied to a single CIFAR-10 sample (an *automobile*). Panels 1 and 2 show the original 32×32 RGB image and its grayscale conversion. Panel 3 shows the 2D FFT log-magnitude spectrum, with energy concentrated near the DC component as expected for natural-image statistics. Panel 4 illustrates DyNED’s sigma-delta quantisation on a single FFT row, where the quantised step signal (orange) closely tracks the original log-magnitude profile (grey). Panel 5 shows the full DyNED spike matrix - a sparse 32×32 binary representation derived from sigma-delta state changes across all FFT bins. Panel 6 shows the same spike matrix after DyNED_c lossless compression: the compressed payload occupies only the leading region of the panel (CR ≈ 0.15 , $\approx 85\%$ space saved on this sample), with the remaining whitespace visualising the storage saving. Because DyNED_c is lossless, the SNN classifier sees a bit-identical spike matrix after decompression.

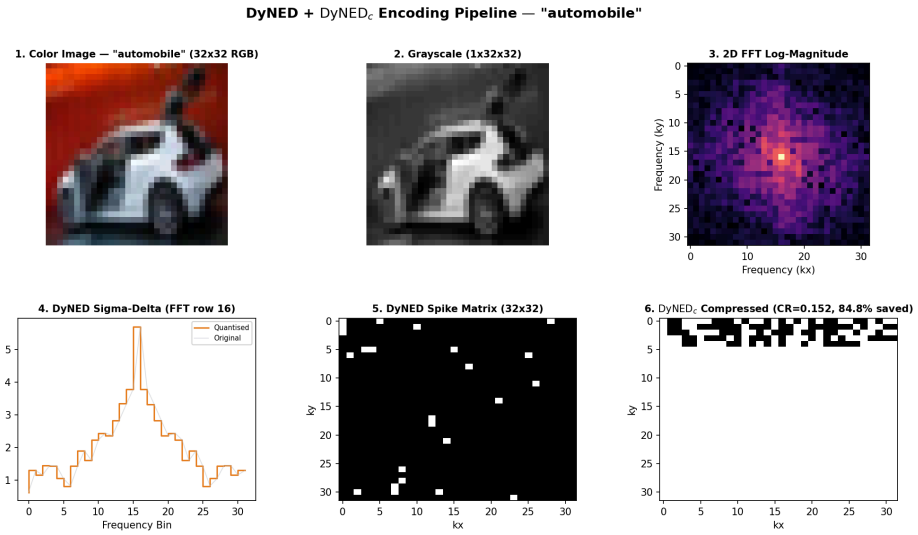


Figure 28: DyNED + DyNED_c encoding pipeline for a single CIFAR-10 sample, showing the transformation from RGB image through grayscale, 2D FFT, DyNED sigma-delta quantisation, full spike matrix, and lossless DyNED_c compression with CR and space-saved annotation.

Figure 29 shows all 10 CIFAR-10 classes processed through the complete encoding pipeline: from the original colour image, through grayscale conversion, 2D FFT frequency analysis, DyNED spike encoding, and finally DyNED_c compression. Each row represents a different class, providing a comprehensive overview of how diverse natural images are transformed into sparse neuromorphic representations.

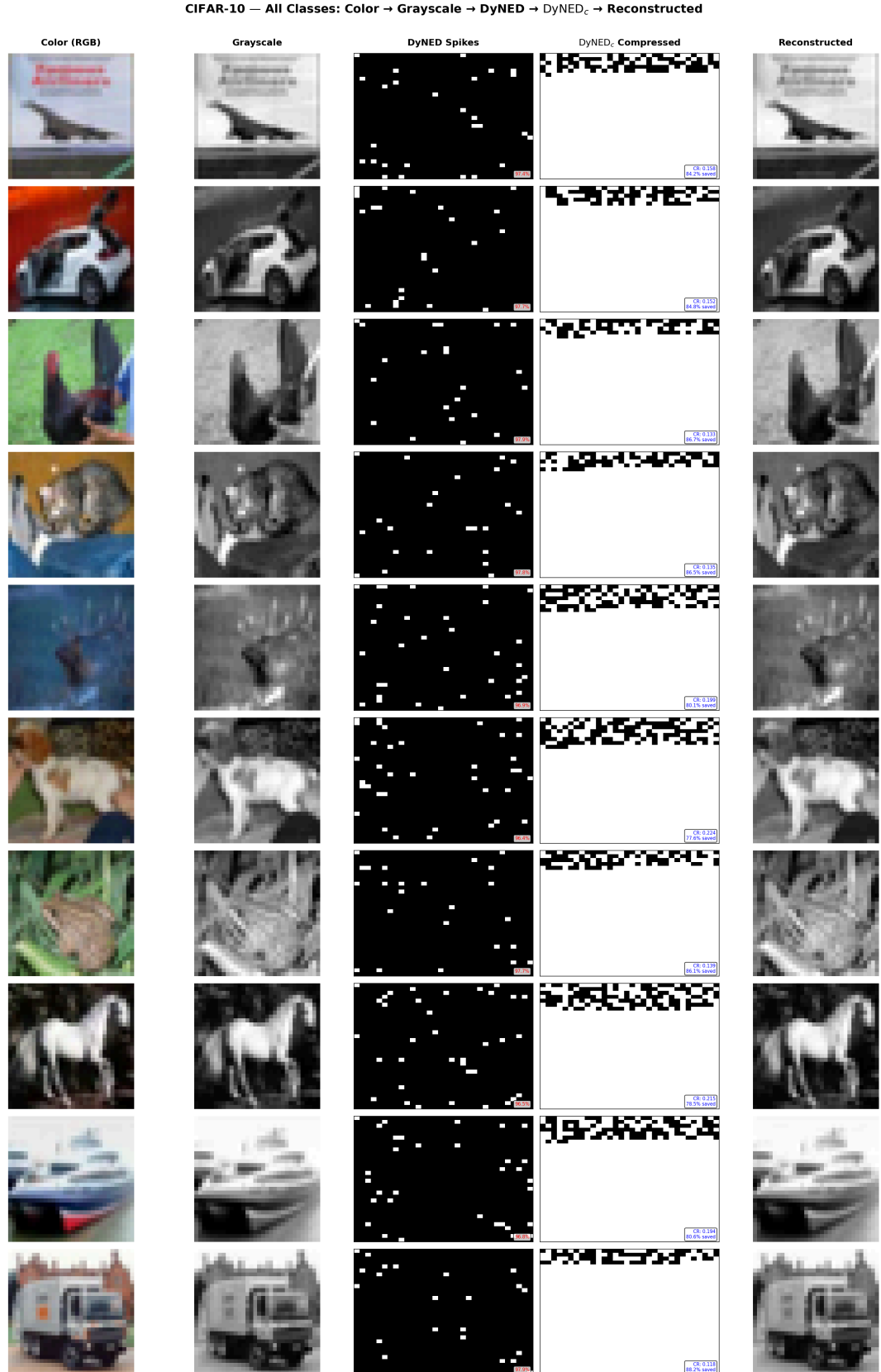


Figure 29: CIFAR-10 + DyNED_c encoding pipeline for all 10 classes. (Left to Right) Original colour image; grayscale conversion; DyNED spike matrix with density percentage; DyNED_c compressed bitstream rendered in the same shape (leading region holds the compressed payload, trailing whitespace visualises the storage saving) with CR and space-saved annotation; and image reconstructed from DyNED encoding via cubic spline interpolation and inverse FFT.

The DyNED spike trains are sparse, with spike densities typically below 10%, confirming effective sigma-delta quantisation. The DyNED_c compressed representations achieve over 90% space savings while perfectly preserving the spike information; the decompressed spikes are identical to the originals. The reconstructed column shows images recovered from the DyNED encoding by interpolating the quantised magnitude spectrum (cubic spline), reversing the log transform ($e^x - 1$), combining with the original phase, and performing inverse FFT. The reconstructed images clearly preserve the essential structural features of each class, demonstrating that the DyNED encoding retains sufficient information for faithful signal recovery.

5.3.3 Spike Trains and Sigma-Delta Internals

The DyNED algorithm (Chapter 3) converts the FFT magnitude spectrum into discrete spike trains through sigma-delta quantisation. The signal amplitude is divided into discrete levels, and a spike is emitted whenever the quantised value changes between consecutive samples. The quantisation process is governed by several key parameters, including the number of quantisation levels, the time window for spike generation, and threshold values for spike triggering.

Figure 30 shows the internal workings of the DyNED encoding process applied to a CIFAR-10 image, revealing how the FFT magnitude spectrum is transformed into spike events through quantisation, error accumulation, and change detection.

DyNED Internals — "airplane" (levels=256)

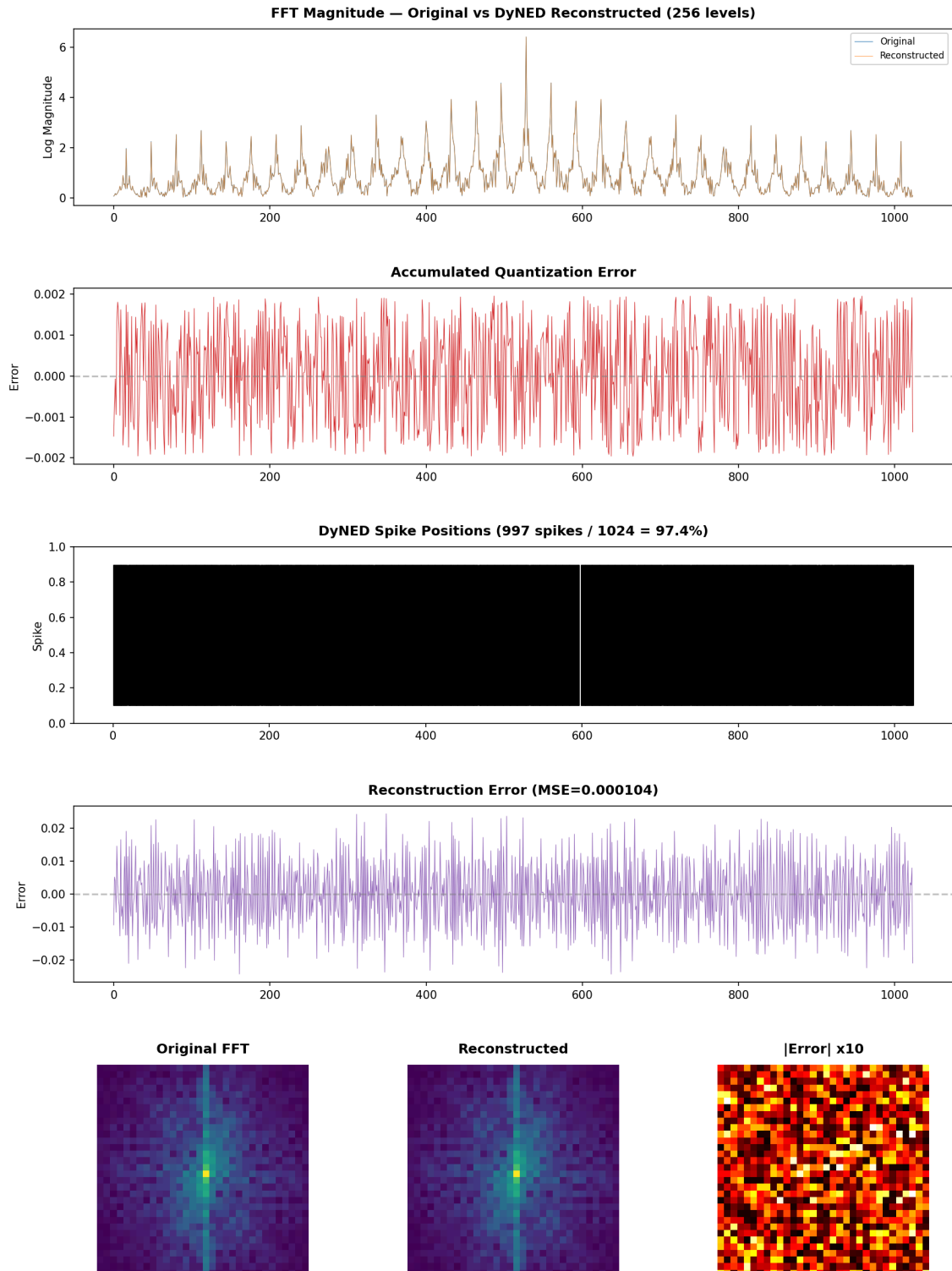


Figure 30: DyNED encoding internals for a CIFAR-10 image. (Top to Bottom) Original FFT magnitude vs DyNED reconstructed signal; accumulated quantisation error; DyNED spike positions with density; reconstruction error; and 2D views of the original FFT, reconstructed signal, and amplified error.

The first plot shows the original FFT magnitude spectrum (blue) overlaid with the DyNED reconstructed signal (orange). The two signals align closely, confirming high-fidelity quantisation. The second plot shows the accumulated quantisation error that drives the feedback loop; it oscillates around zero, demonstrating the self-correcting nature of the DyNED encoder. The spike raster plot shows the distribution of spikes, while the reconstruction error plot confirms that the encoding preserves the essential signal characteristics with minimal distortion.

5.3.3.1 Effect of Quantisation Levels on Image Encoding

Figure 31 demonstrates the effect of varying the number of quantisation levels on both the reconstructed signal quality and spike density.

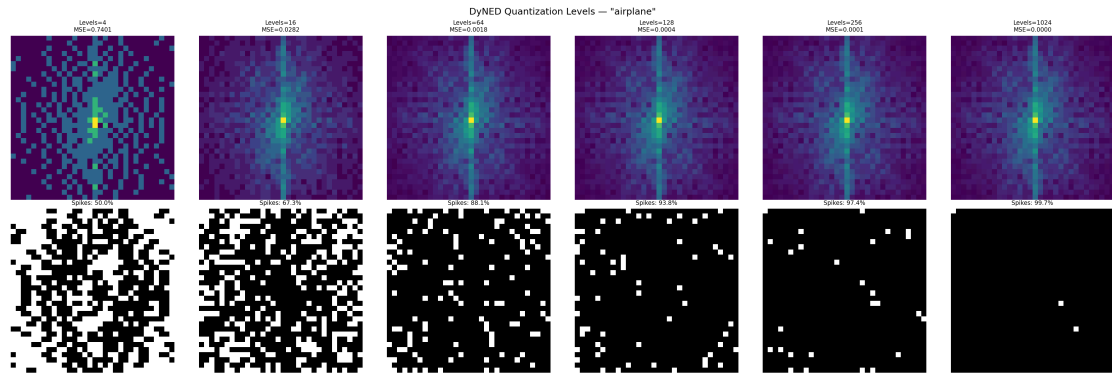


Figure 31: Effect of DyNED quantisation levels on reconstruction quality and spike density. Higher levels produce lower MSE but denser spike trains, while lower levels yield sparser representations at the cost of reconstruction fidelity.

As the number of quantisation levels increases from 4 to 1024, the reconstruction MSE decreases (higher fidelity), but the spike density increases. This trade-off is central to the DyNED design: fewer levels produce sparser, more energy-efficient representations suitable for neuromorphic hardware, while more levels preserve finer signal details. For the CIFAR-10 classification experiments, 256 levels were used to balance reconstruction quality and spike sparsity.

To quantify this trade-off, the DyNED encoder was evaluated across nine quantisation levels on 100 CIFAR-10 test images. Table 7 reports the mean spike density, Structural Similarity Index (SSIM), and Root Mean Squared Error (RMSE) at each level. Figure 32 visualises the joint relationship between spike density and reconstruction quality.

Table 7: Effect of quantisation levels on spike density and reconstruction quality for CIFAR-10 ($n = 100$). The selected operating point (256 levels) is shown in bold.

Levels	Spike Density	SSIM	RMSE
2	0.547	0.044	6.527
4	0.562	0.462	0.627
8	0.630	0.810	0.194
16	0.767	0.933	0.084
32	0.875	0.980	0.040
64	0.937	0.994	0.021
128	0.968	0.998	0.010
256	0.983	1.000	0.005
512	0.991	1.000	0.002

Spike density increases monotonically from 0.547 at 2 levels to 0.991 at 512 levels, confirming that finer quantisation granularity produces denser spike representations. Reconstruction fidelity follows the same trend: SSIM rises from 0.044 (effectively no structural similarity) at 2 levels to 1.000 at 256 levels, while RMSE drops from 6.527 to 0.005. The diminishing returns above 128 levels, where SSIM already exceeds 0.998, justify choosing 256 levels as the operating point for the classification experiments.

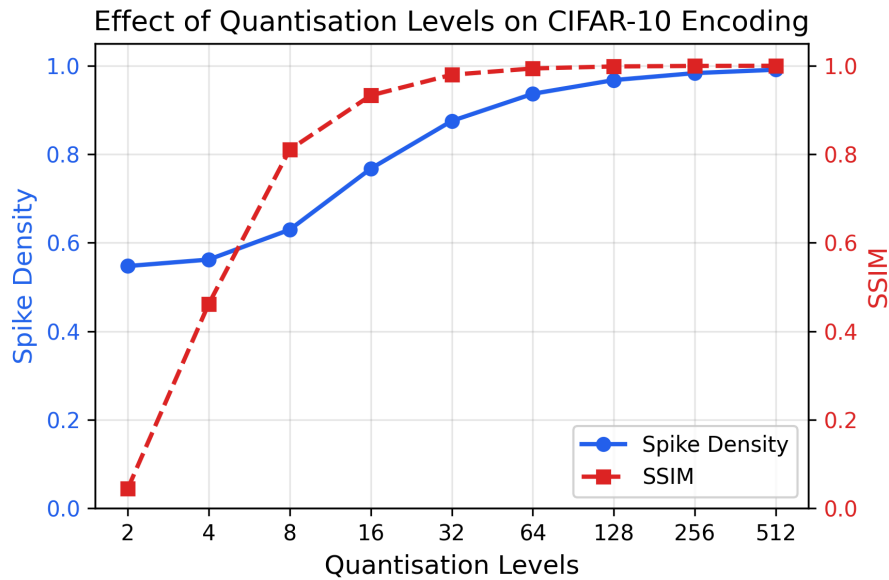


Figure 32: Spike density (blue, left axis) and reconstruction quality (red, right axis) as a function of quantisation levels for CIFAR-10. Both metrics increase with the number of levels, with diminishing returns beyond 128 levels.

5.3.3.2 DyNED vs Baseline Quantisation Methods

To isolate the contributions of DyNED’s error feedback mechanism and uniform level spacing, the encoder was compared against two baselines: simple uniform quantisation (rounding to the nearest level without error propagation) and mu-law companding quantisation (logarithmic compression without error propagation). Mu-law was selected as the non-uniform baseline because it is the most widely deployed non-uniform quantisation scheme in signal processing, standardised as ITU-T G.711 (ITU-T, 1988) and used extensively in telephony and audio compression. Uniform quantisation shares DyNED’s linear level spacing but lacks error feedback, while mu-law uses non-uniform level spacing that concentrates quantisation levels at low amplitudes. This three-way comparison isolates two factors: the effect of error feedback (DyNED vs uniform) and the effect of level spacing (uniform vs mu-law). Table 8 reports the results at four quantisation levels.

Table 8: DyNED (with error feedback) vs uniform and mu-law quantisation (both without error feedback) on CIFAR-10 ($n = 100$).

Levels	Method	Spike Density	SSIM	RMSE
8	DyNED	0.630	0.810	0.194
8	Uniform	0.531	0.822	0.188
8	Mu-law	0.411	0.185	1.677
32	DyNED	0.875	0.980	0.040
32	Uniform	0.872	0.978	0.042
32	Mu-law	0.811	0.822	0.182
128	DyNED	0.968	0.998	0.010
128	Uniform	0.968	0.999	0.010
128	Mu-law	0.950	0.974	0.043
256	DyNED	0.983	1.000	0.005
256	Uniform	0.983	1.000	0.005
256	Mu-law	0.975	0.993	0.021

Mu-law companding performs substantially worse than both DyNED and uniform quantisation on frequency-domain image data. At 8 levels, mu-law achieves an SSIM of only 0.185 - effectively no structural similarity - compared to 0.810 for DyNED and 0.822 for uniform. This is because mu-law’s logarithmic level spacing concentrates quantisation resolution at low amplitudes, which is appropriate for speech waveforms but misaligned with the log-magnitude FFT coefficients used in image encoding, where the signal energy is more uniformly distributed. Even at 256 levels, mu-law’s SSIM (0.993) and RMSE (0.021) remain noticeably worse than DyNED (1.000, 0.005) and uniform (1.000, 0.005).

DyNED consistently produces denser spike trains than both baselines, particularly at lower quantisation levels - at 8 levels, DyNED yields a spike density of 0.630 compared to 0.531 for uniform and 0.411 for mu-law. The error feedback mechanism diffuses quantisation error across consecutive samples, leading to more frequent level transitions and, consequently, more spikes. As the number of levels increases, DyNED and uniform converge in reconstruction quality. Still, DyNED's denser spike trains carry more temporal structure - a property that is valuable for SNNs that rely on spike timing for classification.

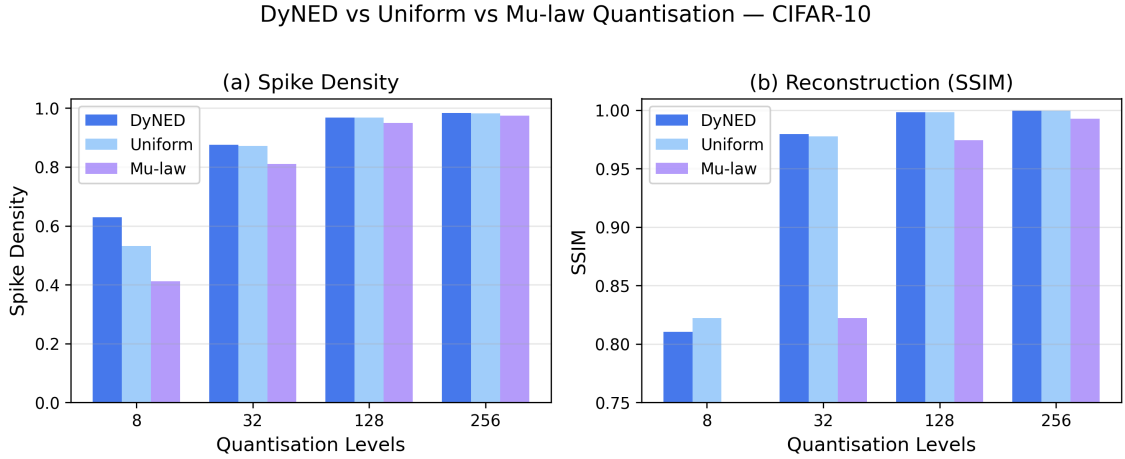


Figure 33: Comparison of DyNED, uniform, and mu-law quantisation on CIFAR-10. (a) Spike density: DyNED produces the densest spike trains. (b) Reconstruction quality (SSIM): mu-law performs poorly on frequency-domain image data, while DyNED and uniform achieve comparable fidelity.

5.3.4 Compression

Compression is part of the DyNED_c algorithm, where the spikes generated from DyNED are compressed losslessly. While developing the DyNED_c algorithm, multiple compression algorithms were tested on the DyNED-encoded spike data at 256 quantisation levels. Each technique is evaluated using two key metrics: the compression ratio (lower values indicate better compression) and the percentage of space saved (higher values indicate better efficiency).

At 256 quantisation levels, the spike density is 0.984 (Table 7), leaving little redundancy for general-purpose compressors. Among the baseline methods, only Huffman Compression achieves positive savings with a compression ratio of 0.905 (9.5% space saved) by exploiting the non-uniform distribution of spike amplitudes (See Section 2.7.3). Run-Length Encoding (2.203), LZW (1.466), Delta (1.495), BWT (1.005), and LZ77 (2.342)

all expand the data, because the dense spike trains contain few consecutive repeated values.

The DyNED_c compression represents a hybrid approach combining Run-Length Encoding and Huffman Compression (See Chapter 4). It achieves the best compression ratio of 0.904 (9.6% space saved), outperforming all baseline methods. While the absolute savings are moderate at 256 levels due to the high spike density, DyNED_c is the only method that consistently reduces storage; at lower quantisation levels where spikes are sparser, savings increase substantially (e.g., 55.3% at 16 levels).

5.3.5 Classification

The final stage of the methodology involves feeding the images through the neural network. The network architecture consists of a convolutional front-end for spatial feature extraction, followed by cAdLIF spiking layers with learnable per-neuron synaptic delays, and a leaky-integrator readout that produces the classification results. The processing mechanism incorporates spike propagation across network layers, temporal integration of input spikes, and end-to-end weight updates via surrogate gradient backpropagation. The classification output is determined based on accumulated readout-layer membrane potentials, using the Temporal Efficient Training (TET) loss to drive per-timestep classification.

5.3.5.1 Architecture for CIFAR-10 Classification with Convolutional cAdLIF SNN and TET Loss

Figure 34 shows the convolutional feature maps extracted by the network's three convolutional layers for all ten CIFAR-10 classes. Each row presents a representative image from one class, progressing from the grayscale input through increasingly abstract feature representations: 64-channel maps at full 32×32 resolution, 128-channel maps at 16×16 after max pooling, and 256-channel maps at 8×8 after the second pooling stage.

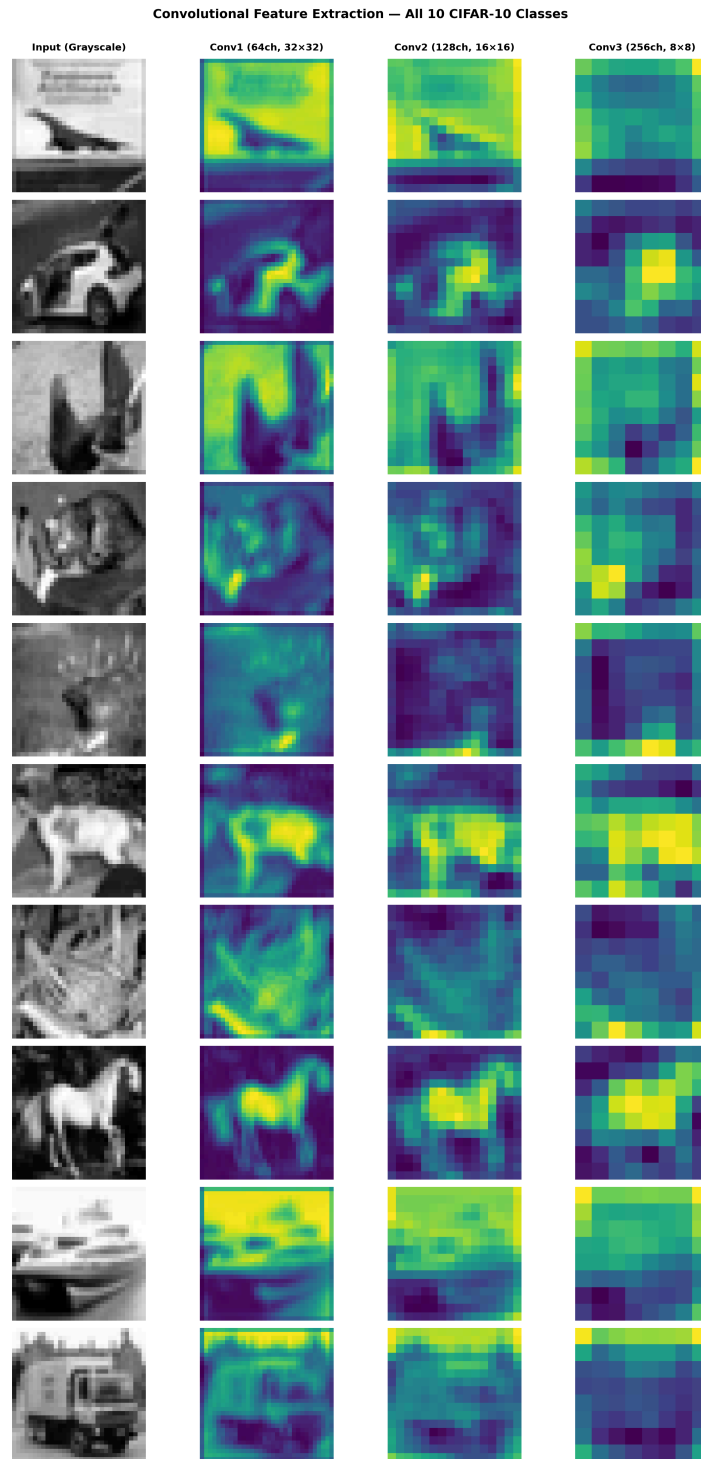


Figure 34: Convolutional feature extraction across all ten CIFAR-10 classes. Each row shows a class example progressing through three convolutional blocks: Conv1 (64 channels, 32×32), Conv2 (128 channels, 16×16), and Conv3 (256 channels, 8×8). The network progressively transforms low-level edges and textures into high-level class-discriminative representations.

Unlike the multi-stream approach with hand-crafted kernels, this architecture employs a learnable convolutional front-end that automatically discovers relevant spatial features from the data. The architecture is shown in Figure 35.

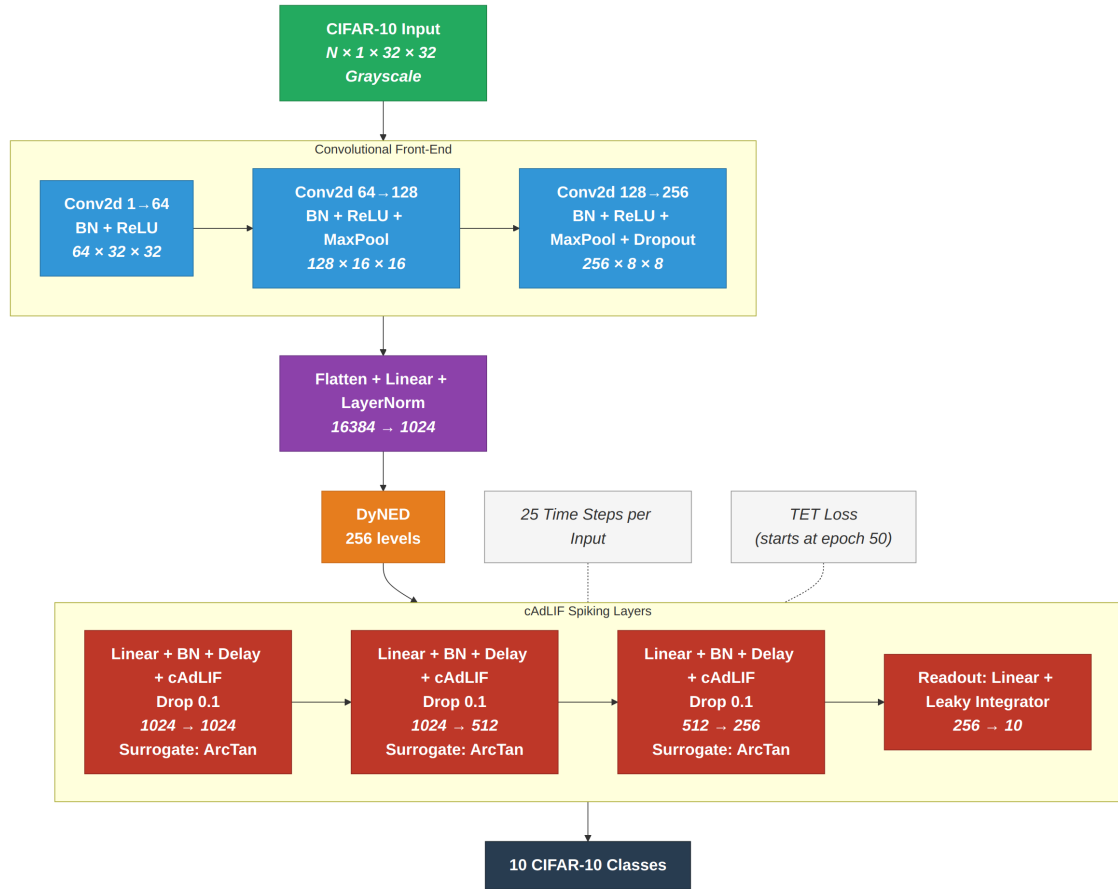


Figure 35: Network architecture for CIFAR-10 classification with convolutional front-end, DyNED encoder, and cAdLIF spiking layers.

The architecture uses the following configuration:

Convolutional Front-End: The network begins with a learnable convolutional feature extractor, replacing the fixed-kernel approach used in earlier experiments. The front-end consists of three convolutional blocks: the first block applies 64 filters of size 3×3 with padding, followed by batch normalisation and a ReLU activation. The second block applies 128 filters of size 3×3 , followed by batch normalisation, ReLU, and 2×2 max pooling (reducing the spatial dimensions to 16×16). The third block applies 256 filters of size 3×3 , followed by batch normalisation, ReLU, 2×2 max pooling (reducing to 8×8), and spatial dropout with probability 0.1. This produces a feature volume of $256 \times 8 \times 8 = 16,384$ dimensions, which is projected to 1,024 dimensions via a linear layer with layer normalisation.

cAdLIF Spiking Layers: The projected features are passed through DyNED and then fed into a stack of three cAdLIF spiking layers (Deckers et al., 2024). Each spiking layer consists of a linear projection, batch normalisation, a learnable per-neuron synaptic delay, the cAdLIF neuron, and dropout ($p = 0.1$). The first hidden layer maps $1,024 \rightarrow 1,024$ neurons, the second $1,024 \rightarrow 512$, and the third $512 \rightarrow 256$. A leaky-integrator readout maps the final 256 neurons to 10 output classes. Synaptic delays are learnable per-neuron, with a maximum of 7 timesteps, and are initialised uniformly across $[0, \text{max_delay}]$, so each neuron starts at a different value and gradients can move it in either direction. The surrogate gradient used throughout is the ArcTan function.

cAdLIF Neuron Configuration: Each cAdLIF neuron carries four learnable per-neuron parameters - membrane decay (α), adaptation decay (β), adaptation coupling (a), and spike-triggered adaptation increment (b) - all of which are constrained to physically meaningful ranges following Deckers et al. (Deckers et al., 2024). The membrane and adaptation states evolve jointly with the input current at each timestep, producing a spike when the membrane crosses the threshold; the surrogate gradient passes through the spike nonlinearity using ArcTan.

Temporal Processing: The network processes each input over 25 time steps. At each step, the convolutional features are encoded by DyNED and propagated through the cAdLIF layers; the readout integrates membrane potentials over time, and classification is based on the accumulated readout potentials across all time steps.

Optimisation: Training used the AdamW optimiser (Loshchilov & Hutter, 2017) with a learning rate of 3.04×10^{-3} , a separate delay learning rate of 1.11×10^{-2} for the synaptic-delay parameters, weight decay of 5.56×10^{-5} , and β parameters (0.9, 0.999). A one-cycle learning rate scheduler was used with 32.7% warmup, cosine annealing, initial division factor of 25, and final division factor of 1000. Gradient clipping was maintained at a maximum norm of 1.0.

Loss Function: Training used the Temporal Efficient Training (TET) loss (Deng et al., 2022). The first 50 epochs were trained with standard cross-entropy on time-averaged readout potentials to bootstrap the spiking dynamics; from epoch 50 onwards, TET took over, applying cross-entropy independently at each timestep and averaging across the temporal window, with a temporal-consistency regulariser ($\lambda_{\text{TET}} = 1.21 \times 10^{-4}$). This two-phase schedule provided stable early learning while ensuring the final solution was shaped by per-timestep gradient pressure.

Training Configuration: The network was trained for 250 epochs with a batch size of 128.

The epoch count was determined by examining the Optuna trial learning curves: at 50 epochs (the trial length), accuracy was still increasing, indicating the model had not converged. A $5 \times$ extension to 250 epochs was chosen, and the final model achieved its best accuracy at epoch 245, confirming that the training budget was necessary but not excessive. Mixed-precision training was used for computational efficiency. Data augmentation included random affine transformations (rotation $\pm 15^\circ$, translation $\pm 10\%$, scaling 90%-110%), random perspective distortion (scale 0.2, probability 50%), and random erasing (probability 10%).

Hyperparameter Optimisation: The hyperparameters listed above were optimised using the Optuna framework (Akiba et al., 2019) with the TPE sampler (Section 2.6.2). The search space and optimal values found are shown in Table 9; a six-panel parameter-importance analysis of the underlying 56-trial search is provided in Appendix A.7.

Table 9: Hyperparameter search space and best values for CIFAR-10 classification with Conv + DyNED + cAdLIF + TET, found via TPE with Optuna (56 trials, 50 epochs per trial). Best trial test accuracy at 50 epochs: 87.05%. Full 250-epoch retraining at these parameters reached 88.07% best test accuracy at epoch 245.

Hyperparameter	Search Range	Best Value
Learning rate	$1 \times 10^{-3} - 1 \times 10^{-1}$ (log)	3.04×10^{-3}
Delay learning rate	$1 \times 10^{-3} - 1 \times 10^{-1}$ (log)	1.11×10^{-2}
Weight decay	$1 \times 10^{-5} - 1 \times 10^{-3}$ (log)	5.56×10^{-5}
Hidden size	{1024, 2048}	1024
DyNED dimension	{256, 512, 1024}	512
Max delay	{2, 3, 5, 7}	7
Time steps	10 - 25	25
Dropout	0.1 - 0.4	0.101
Conv dropout	0.05 - 0.2	0.055
Batch size	{128, 256, 512}	128
Warmup fraction	0.1 - 0.4	0.327
DyNED levels	{16, 32, 64, 128, 256}	256
TET λ	$1 \times 10^{-4} - 1 \times 10^{-1}$ (log)	1.21×10^{-4}

5.3.5.2 Architecture for CIFAR-10 Classification with DyNED_c Compression

The compressed variant extends the architecture above by adding DyNED_c after the DyNED encoder, denoted as a combined DyNED + DyNED_c block in Figure 36.

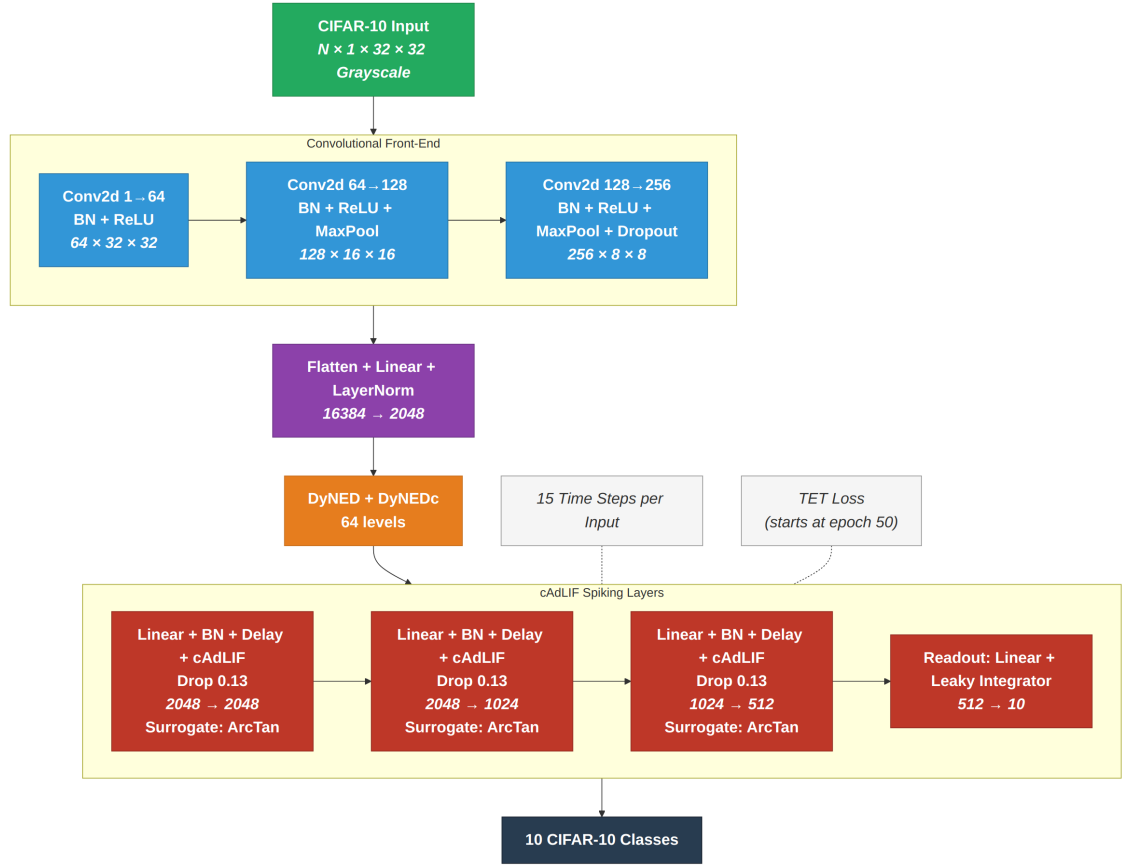


Figure 36: Architecture for CIFAR-10 classification with DyNED + DyNED_c. The combined encoder-plus-compressor pipeline is shown as a single block feeding the cAdLIF spiking backbone.

DyNED_c losslessly compresses the spike trains downstream of the DyNED encoder, reducing the memory footprint of the spike representation without affecting classification performance. This demonstrates that the spike-based encoding contains significant redundancy that can be exploited for efficient storage and transmission.

Speed and Memory Efficiency: The DyNED_c compression algorithm significantly reduces the memory footprint of the spike trains, lowering storage requirements and accelerating data transfer between encoding and classification stages. Compressed spike trains enable more efficient training and inference, making the network more scalable and adaptable for real-world deployment.

Hyperparameter Optimisation: The same Optuna-based TPE search was applied to the DyNED_c variant, with the network hyperparameters drawn from the same ranges as the DyNED experiment (Table 9). Table 10 shows the search space and best values; a parameter-importance analysis of the 50-trial search is provided in Appendix A.8.

Table 10: Hyperparameter search space and best values for CIFAR-10 classification with DyNED_c compression, found via TPE with Optuna (50 trials, 50 epochs per trial). Best trial test accuracy at 50 epochs: 86.76%. Full 250-epoch retraining at these parameters reached 87.69% best test accuracy at epoch 247.

Hyperparameter	Search Range	Best Value
Learning rate	$1 \times 10^{-3} - 1 \times 10^{-1}$ (log)	1.35×10^{-2}
Delay learning rate	$1 \times 10^{-3} - 1 \times 10^{-1}$ (log)	4.34×10^{-2}
Weight decay	$1 \times 10^{-5} - 1 \times 10^{-3}$ (log)	2.29×10^{-5}
Hidden size	{1024, 2048}	2048
DyNED dimension	{256, 512, 1024}	512
Max delay	{2, 3, 5, 7}	5
Time steps	10 - 25	15
Dropout	0.1 - 0.4	0.130
Conv dropout	0.05 - 0.2	0.094
Batch size	{128, 256, 512}	512
Warmup fraction	0.1 - 0.4	0.222
DyNED levels	{16, 32, 64, 128, 256}	64
TET λ	$1 \times 10^{-4} - 1 \times 10^{-1}$ (log)	4.26×10^{-3}

Compared to the DyNED-only experiment, the DyNED_c optimiser selected a larger hidden size (2,048 vs 1,024), a higher learning rate (1.35×10^{-2} vs 3.04×10^{-3}), a higher delay learning rate (4.34×10^{-2} vs 1.11×10^{-2}), and a shorter temporal extent ($T = 15$ vs $T = 25$). The optimiser also preferred a coarser DyNED quantisation (64 levels vs 256) and a larger batch size (512 vs 128), suggesting that the DyNED_c regime benefits from sparser, lower-resolution spike trains that compress more aggressively while remaining classifiable.

5.4 Results

5.4.1 DyNED Encoding Quality

The DyNED encoding quality was evaluated by interpolating the quantised spike trains to continuous signals and comparing the results with the original input. Four interpolation techniques were tested: Radial Basis Function (Cubic), Cubic Spline, Radial Basis Function (Linear), and Krogh interpolation. Table 11 presents a comparison of these methods using multiple metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Signal-to-Interference Ratio (SSI), and coefficient of determination (R^2).

Table 11: Comparison of various interpolation techniques for DyNED encoded image data.

Type of Interpolation	RMSE	MAE	SSI	R^2	Time (s)
Radial Basis Function (Cubic)	0.12234886093920139	0.10111915411878254	0.915554554654564	0.986880320917212	0.0144s
Cubic Spline	0.12234886093905940	0.10111915411864819	0.901654984651563	0.986880320917228	0.0008s
Radial Basis Function (Linear)	0.12234879989921617	0.10111911644760059	0.904260096909894	0.9868803169651034	0.9075s
Krogh Interpolation	0.12234885184770541	0.10111915446963873	0.904260296964433	0.9868803248714999	0.0991s

All four interpolation methods - RBF Cubic, Cubic Spline, RBF Linear, and Krogh - demonstrate remarkably similar performances, with differences appearing only at the microscopic level. The consistency across methods suggests that the encoded image features are well-suited for multiple interpolation approaches.

Error Metrics Analysis: The Root Mean Square Error (RMSE) values are remarkably consistent across all four interpolation methods, hovering around 0.122, with RBF Linear showing marginally better performance (0.12234879). Similarly, the Mean Absolute Error (MAE) values are uniformly close to 0.101, again with RBF Linear displaying slightly superior results (0.10111911). These consistently low error values indicate high accuracy in signal reconstruction across all methods.

Structural and Fit Analysis: The Structural Similarity Index (SSI) shows more noticeable variations among the methods. RBF Cubic achieves the highest SSI of 0.915, indicating superior structural preservation, while Cubic Spline shows the lowest at 0.901. The R^2 values are exceptionally consistent at approximately 0.987 across all methods, demonstrating that each interpolation technique explains about 98.7% of the variance in the data.

Computational Efficiency: The execution times reveal significant differences in computational efficiency. Cubic Spline is remarkably fast at 0.0008 seconds, followed by RBF Cubic at 0.0144 seconds. In contrast, RBF Linear and Krogh Interpolation are considerably slower, requiring 0.9075 and 0.0991 seconds, respectively. This substantial variation in execution time provides an important practical consideration despite the similar accuracy metrics.

Practical Implications: Given these results, Cubic Spline emerges as the most practical choice for this specific application. While it shows slightly lower structural similarities compared to RBF Cubic, it maintains equivalent error metrics and R^2 values while being approximately 18 times faster than RBF Cubic and over 1,000 times faster than RBF Linear. This combination of high accuracy and computational efficiency

makes it particularly suitable for applications where processing time is a critical factor.

Table 12: Mean compression results for DyNED encoded CIFAR-10 spike data ($n = 100$, levels = 256).

Compression Algorithm	Compression Type	Compression Ratio ↓	% Space Saved ↑
Run-Length Encoding	Lossless	2.203	−120.3%
LZW Compression	Lossless	1.466	−46.6%
Huffman Compression	Lossless	0.905	9.5%
Delta Compression	Lossless	1.495	−49.5%
BWT Compression	Lossless	1.005	−0.5%
LZ77 Compression	Lossless	2.342	−134.2%
DyNED_c Compression	Hybrid Lossless	0.904	9.6%

The compression comparison in Table 12 evaluates seven methods on spike data produced by DyNED at 256 quantisation levels - the same setting used for the classification experiments. At this level, the spike density is 0.984 (Table 7), leaving little redundancy for general-purpose compressors. Run-Length Encoding, LZW, Delta, and LZ77 all expand the data because the dense spike trains contain few consecutive repeated values. Huffman coding achieves a modest 9.5% saving by exploiting the non-uniform distribution of spike amplitudes. DyNED_c achieves the best compression ratio of 0.904 (9.6% space saving), outperforming all baseline methods. While the absolute savings are moderate at 256 levels, DyNED_c is the only method that consistently reduces storage for dense spike data; at lower quantisation levels where spikes are sparser, savings increase substantially (e.g., 55.3% at 16 levels).

The encoding-quality measurements reported in this section provide the visual half of the evidence for RQ1 and RQ2; per-class classification accuracy is taken up in the next section as the visual evidence for RQ3.

5.4.2 DyNED Classification Results

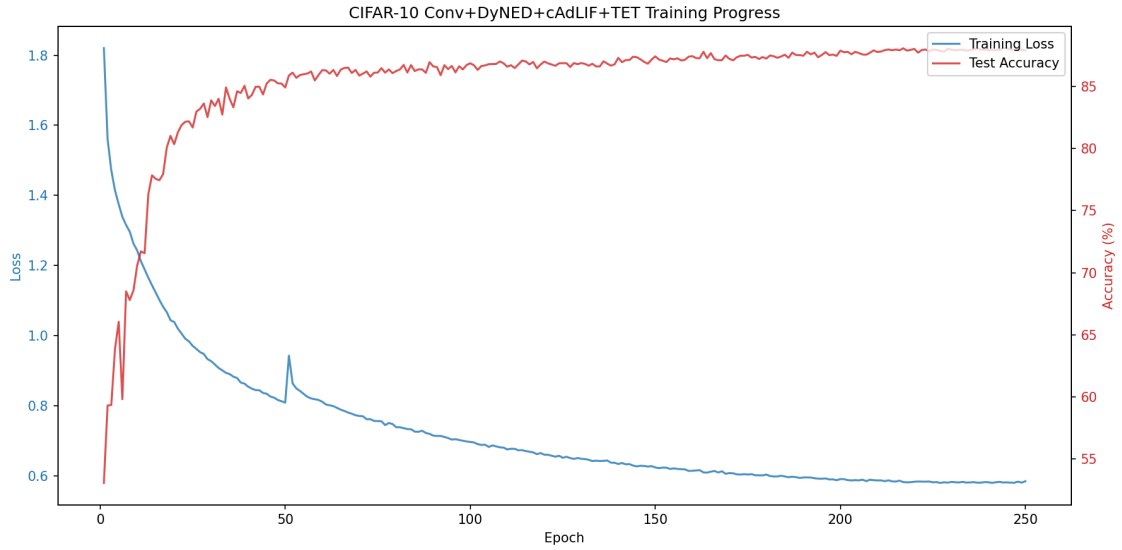


Figure 37: Training loss and test accuracy over 250 epochs for the Conv+DyNED+cAdLIF+TET SNN on CIFAR-10.

Figure 37 shows the evolution of the model’s performance over 250 epochs, tracking loss and accuracy. The training loss begins at approximately 1.79 (with label smoothing) and decreases steadily, stabilising at 0.58 by the final epoch. The test accuracy starts at 46.5% after the first epoch and reaches 81.2% by epoch 50, with the best accuracy of 88.07% observed at epoch 245 and the final accuracy at 87.88% at epoch 250.

The training dynamics reveal three distinct phases. During the initial cross-entropy phase (epochs 1-50), the network learns under mean cross-entropy loss. At the same time, the OneCycleLR scheduler ramps the learning rate up to its peak, with accuracy climbing from 46% to 81%. At epoch 50, the TET loss engages: per-timestep cross-entropy provides stronger gradient pressure on the cAdLIF temporal dynamics, producing an immediate increase in layer-3 firing rate and a brief accuracy dip before the model adapts. In the refinement phase (epochs 50-200), accuracy climbs steadily from 81% to approximately 87% as the cosine decay reduces the learning rate. The convergence phase (epochs 200-250) settles the model at 88%, with the best epoch at 245.

5.4.2.1 Per-Class Performance

Table 13: Per-class test accuracy for CIFAR-10 at the best epoch (245) of the Conv+DyNED+cAdLIF+TET SNN. Overall: 88.07%.

Class	Accuracy (%)	Samples
Airplane	91.5	1,000
Automobile	95.4	1,000
Bird	82.7	1,000
Cat	75.3	1,000
Deer	85.7	1,000
Dog	82.4	1,000
Frog	91.1	1,000
Horse	90.5	1,000
Ship	94.0	1,000
Truck	92.1	1,000
Overall	88.07	10,000

Table 13 reveals notable variation in per-class performance. The *automobile* class achieves the highest accuracy at 95.4%, likely due to its distinctive geometric structure (rectangular body, circular wheels) that produces clear edge responses in the convolutional filters. Other structured classes like *ship* (94.0%), *truck* (92.1%), *airplane* (91.5%), *frog* (91.1%), and *horse* (90.5%) also score above 90%.

The most challenging class is *cat* (75.3%), with *bird* (82.7%) and *dog* (82.4%) trailing the leaders. These three share significant visual overlap; cats and dogs have similar body proportions and fur textures, while birds can appear in varied poses against complex backgrounds. This pattern of confusion between semantically similar categories is consistent with the inherent difficulty of fine-grained visual discrimination in CIFAR-10.

5.4.2.2 Confusion Matrix

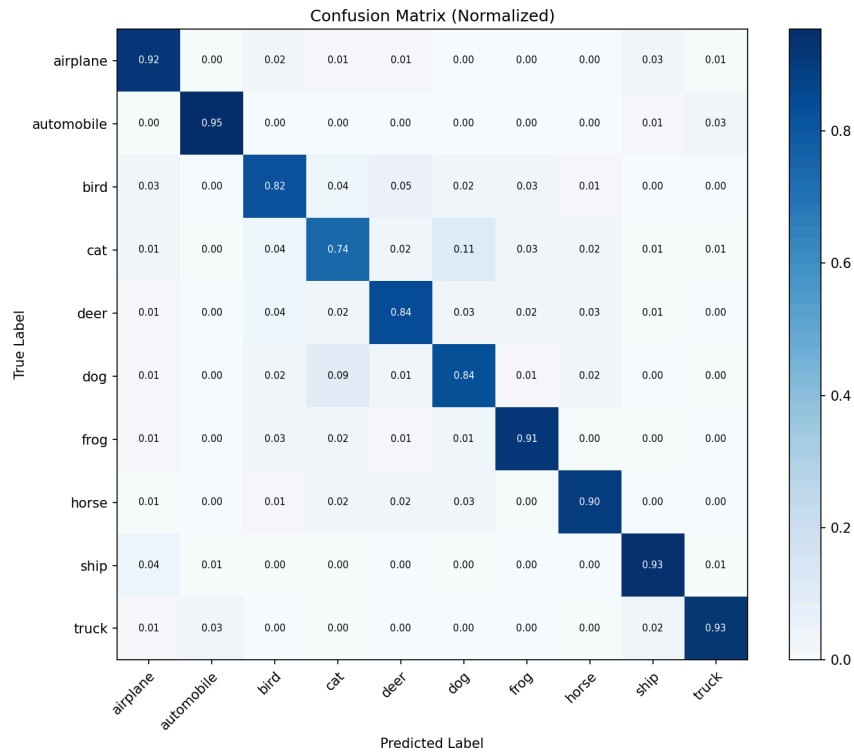


Figure 38: Confusion matrix for CIFAR-10 classification at epoch 250.

Figure 38 presents the confusion matrix, which reveals the specific misclassification patterns. The most prominent confusions occur between *cat* and *dog* (bidirectional), *deer* and *bird*, and *dog* and *cat*. These confusions align with visual similarities - animals with similar body shapes and textures are more likely to be confused. Notably, vehicle classes (*automobile*, *ship*, *truck*, *airplane*) are rarely confused with animal classes, indicating that the convolutional front-end effectively separates these broad semantic categories.

5.4.2.3 Network Dynamics

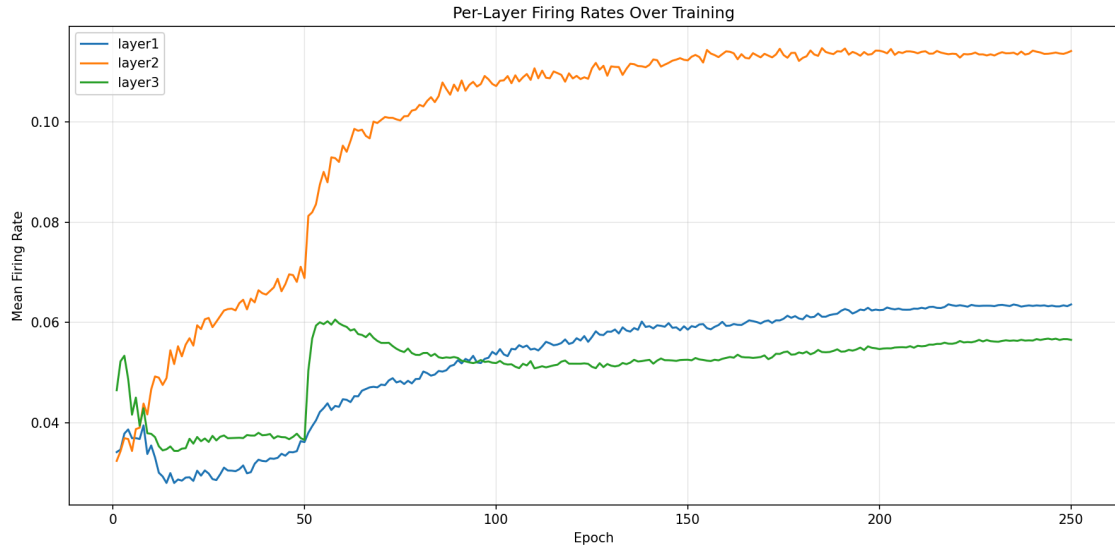


Figure 39: Evolution of firing rates across network layers during training.

Figure 39 shows the evolution of firing rates across the three cAdLIF hidden layers during training. The firing rates stabilise after the TET phase engages at epoch 50: Layer 1 operates at approximately 6.3%, Layer 2 at 11.4%, and Layer 3 at 5.7%. The lower rates in Layers 1 and 3 indicate strong feature selection at the input and pre-readout stages. At the same time, Layer 2 maintains a slightly denser representation that encodes the intermediate features handed off between them. This distribution demonstrates that the network develops a sparse, energy-efficient representation - a key advantage of spiking neural networks over conventional architectures.

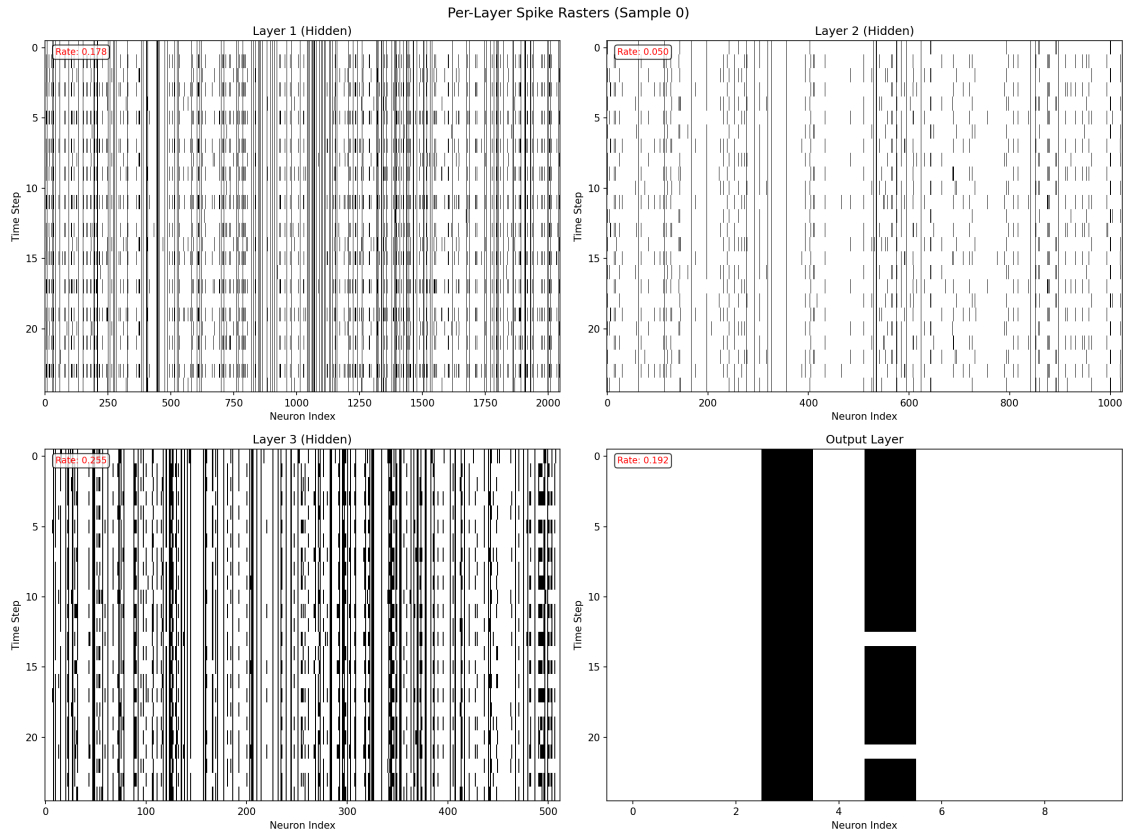


Figure 40: Spike raster plots across network layers showing temporal firing patterns.

Figure 40 displays spike raster plots for each layer, revealing the temporal dynamics of neural activity. The raster patterns show that spikes are distributed across time steps rather than concentrated in bursts, indicating effective temporal processing. The progressive sparsification from earlier to later layers demonstrates hierarchical feature extraction, where each successive layer selects increasingly relevant features for classification.

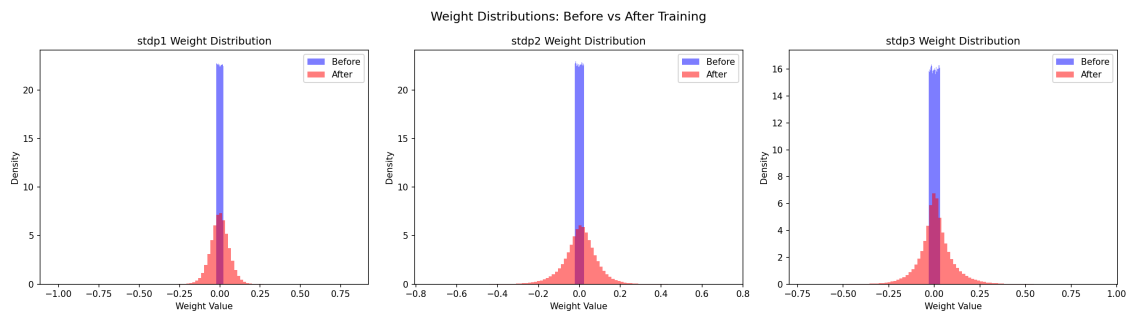


Figure 41: Weight distributions across cAdLIF layers at the end of training.

Figure 41 shows the final weight distributions across all cAdLIF layers. The distributions are approximately centred around zero with varying spreads, indicating that surrogate-gradient training has shaped the synaptic connections to be both excitatory and inhibitory. The weight distributions in the later layers show slightly broader spreads,

reflecting the need for stronger discriminative connections as the network approaches the classification output.

5.4.2.4 Per-Class Spike Patterns

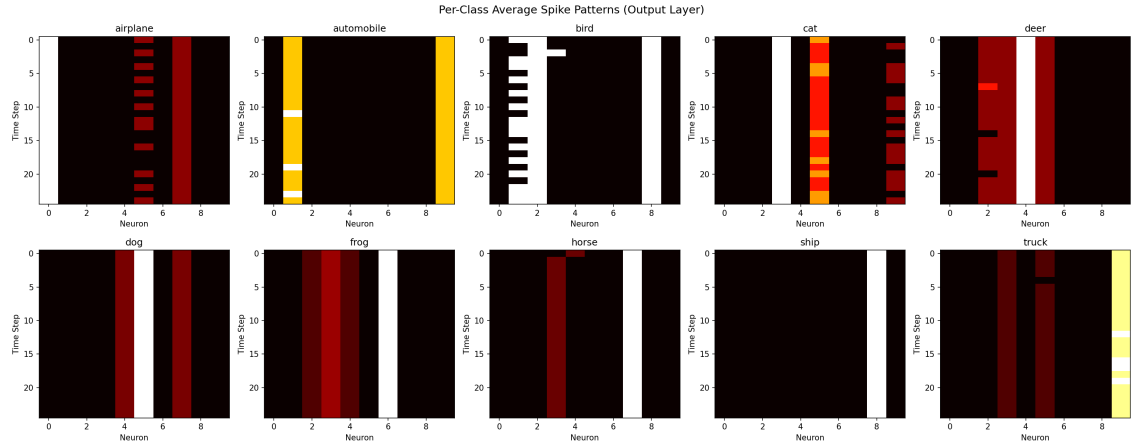


Figure 42: Per-class average spike patterns at the output layer for all 10 CIFAR-10 classes.

Figure 42 visualises the average output spike patterns for each of the 10 CIFAR-10 classes. Each subplot shows the spike activity across the 10 output neurons over the inference timesteps when a sample of that class is presented. In each case, the corresponding target neuron exhibits strong, sustained spiking activity (bright band) while all other output neurons remain largely inactive (dark regions). The consistency and selectivity of these patterns across all classes confirm that the network has developed robust class-specific representations through surrogate-gradient training with the TET loss.

5.4.2.5 Representation Quality

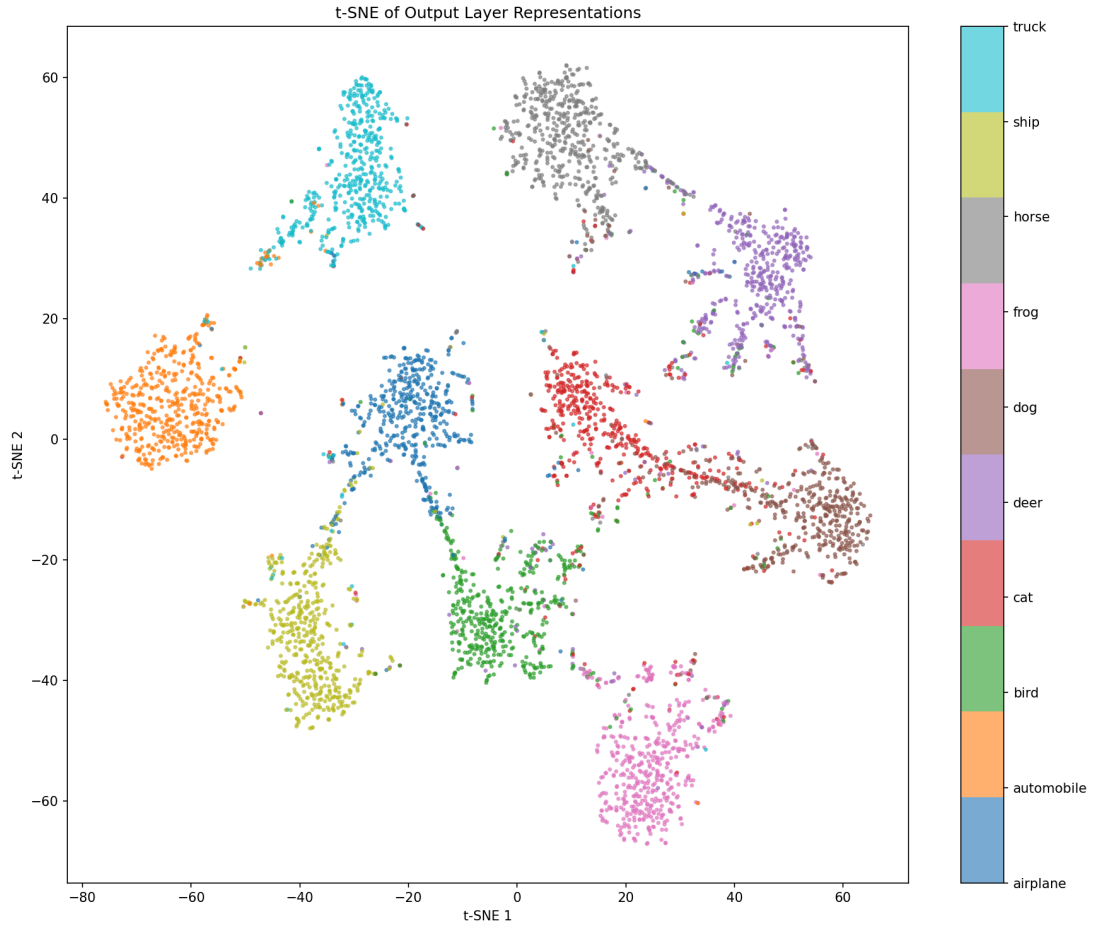


Figure 43: t-SNE visualisation of learned representations for CIFAR-10 test samples, showing class-wise clustering across network layers.

Figure 43 presents t-SNE visualisations of the network’s internal representations at different layers. t-SNE (t-distributed Stochastic Neighbour Embedding) is a dimensionality reduction technique that projects high-dimensional representations into 2D space while preserving local neighbourhood structure. The visualisation shows progressive separation of class clusters from early to later layers, confirming that the network develops increasingly discriminative representations as it deepens. Vehicle classes (automobile, ship, truck, airplane) form relatively tight, well-separated clusters, while animal classes show more overlap - consistent with the per-class accuracy differences observed in Table 13.

The classification results reported in this section constitute the visual half of the experimental evidence for RQ3.

5.4.3 DyNED_c Classification Results

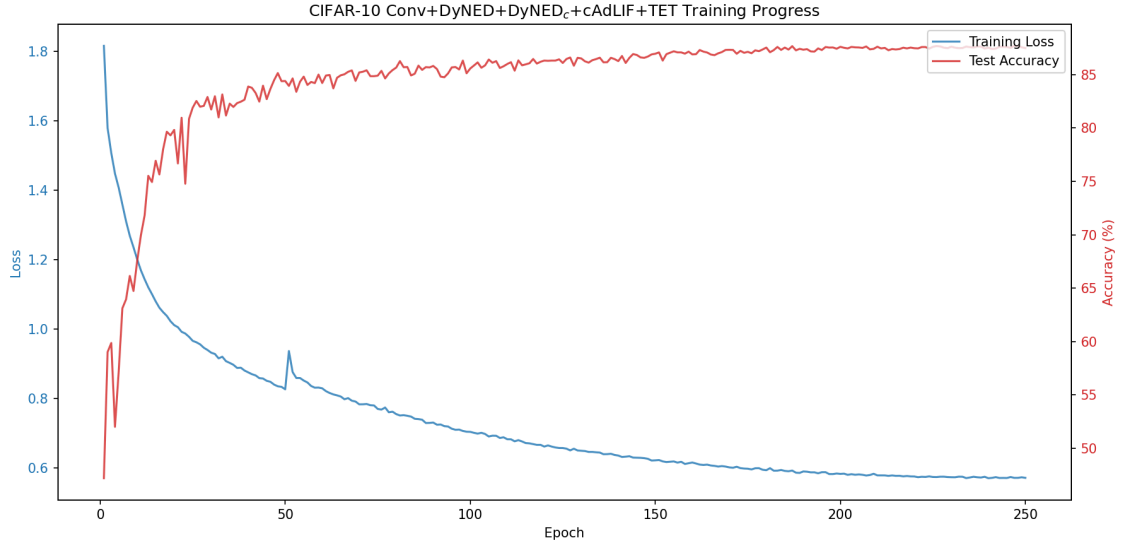


Figure 44: Training loss and test accuracy over 250 epochs for the Conv+DyNED+cAdLIF+TET SNN with DyNED_c compression on CIFAR-10.

Figure 44 shows the training progression for the DyNED_c compressed variant over 250 epochs. Test accuracy reaches 87.48% at the final epoch, with the best accuracy of 87.69% observed at epoch 247. The training dynamics follow a similar three-phase pattern to the DyNED variant: initial warmup, refinement, and convergence, with the TET loss engaging at epoch 50.

5.4.3.1 Per-Class Performance

Table 14: Per-class test accuracy for CIFAR-10 with DyNED_c compression at the best epoch (247). Overall: 87.69%.

Class	Accuracy (%)	Samples
Airplane	89.2	1,000
Automobile	96.3	1,000
Bird	81.3	1,000
Cat	74.3	1,000
Deer	85.2	1,000
Dog	82.4	1,000
Frog	90.6	1,000
Horse	91.8	1,000
Ship	94.4	1,000
Truck	91.4	1,000
Overall	87.69	10,000

Table 14 shows the per-class performance of the DyNED_c variant. The class-wise pattern closely mirrors the DyNED-only results: *automobile* (96.3%) and *ship* (94.4%) remain the strongest classes, while *cat* (74.3%) and *bird* (81.3%) are the most challenging. Compared to the DyNED-only variant (Table 13), per-class scores stay within 2 percentage points, confirming that the lossless DyNED_c compression on the cAdLIF spike outputs preserves the discriminative structure of the spike representation.

5.4.3.2 Confusion Matrix

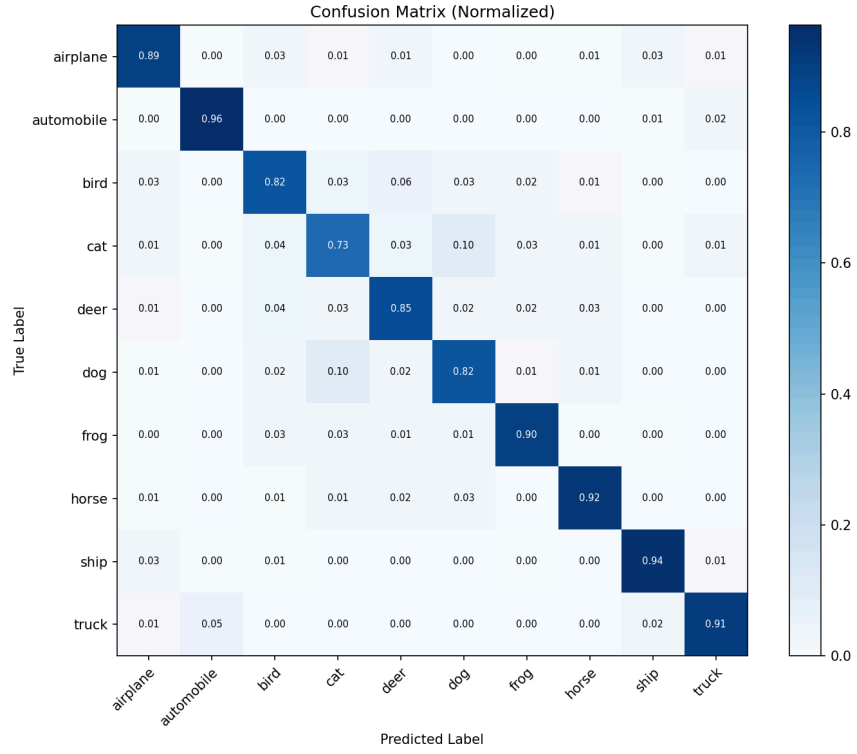


Figure 45: Confusion matrix for CIFAR-10 classification with DyNED_c compression at epoch 249.

Figure 45 shows the confusion matrix for the DyNED_c variant. The misclassification patterns are consistent with the DyNED variant: *cat-dog* bidirectional confusion remains the most prominent, and vehicle classes are rarely confused with animal classes.

5.4.3.3 Network Dynamics



Figure 46: Evolution of firing rates across network layers during training with DyNED_c compression.

Figure 46 shows the firing rate evolution for the DyNED_c variant. The network develops similar sparse representations to the DyNED-only variant: Layer 1 operates at 5.2% (vs 6.3%), Layer 2 at 10.0% (vs 11.4%), and Layer 3 at 2.6% (vs 5.7%). The lower Layer-3 firing rate enables the strong 81.4% compression that the DyNED_c algorithm achieves in this deepest layer (Table 12), reflecting the joint optimisation between the DyNED quantisation level (64) and the downstream layer sparsity.

5.4.3.4 Representation Quality

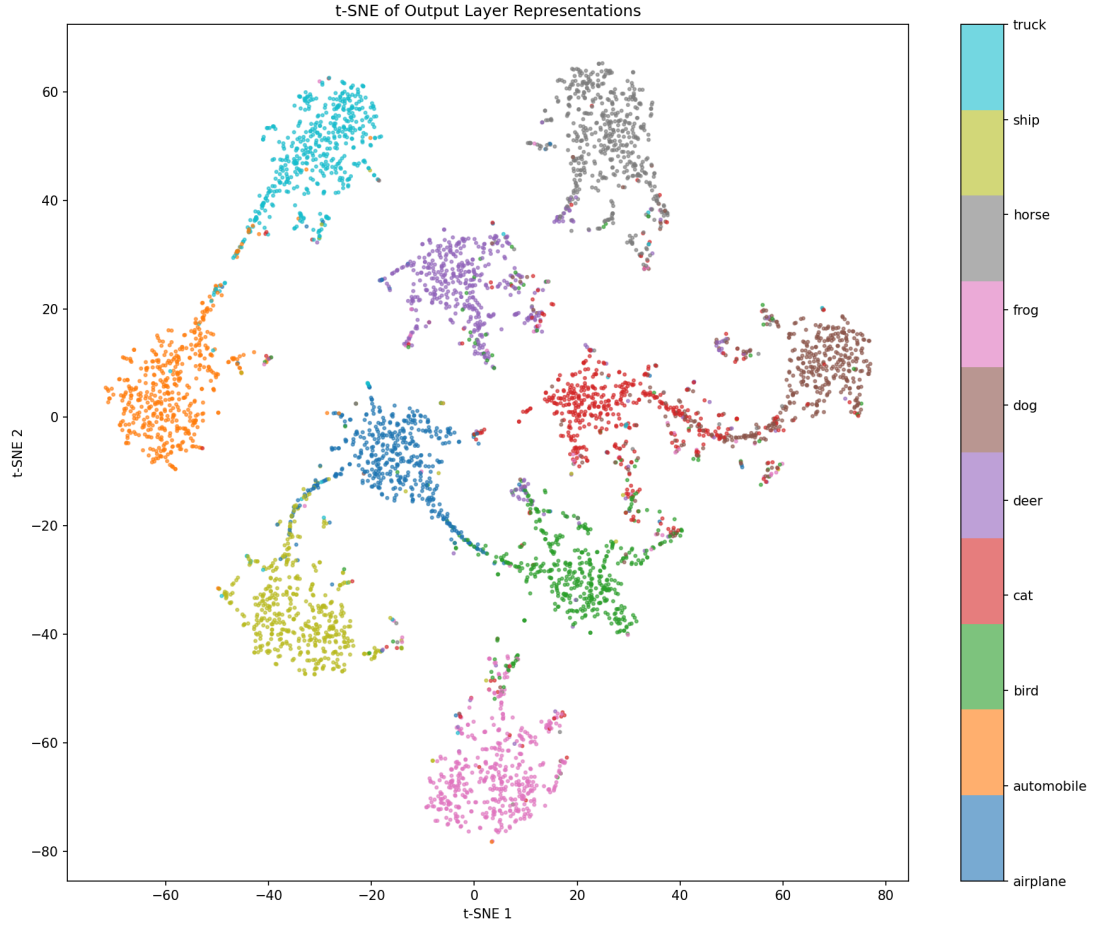


Figure 47: t-SNE visualisation of learned representations for CIFAR-10 test samples with DyNED_c compression, showing class-wise clustering.

Figure 47 presents the t-SNE visualisation for the DyNED_c variant. The class clusters show comparable separation to the DyNED variant (Figure 43), confirming that the compressed spike trains retain sufficient discriminative information for the spiking layers to develop well-separated class representations.

5.4.3.5 DyNED_c Compression During Training

Post-training DyNED_c compression on the cAdLIF spike outputs (chunk size 4) yields 65.8% overall space saving across the 10,000-sample test set. Per layer: Layer 1 at 64.2% saving (mixed Huffman/alternating-Huffman branches), Layer 2 at 51.8% (dominantly standard Huffman), and Layer 3 at 81.4% (alternating-Huffman). The deepest layer compresses best, reflecting its sparser spike output. Because DyNED_c is lossless, this compression has no effect on classification accuracy.

The DyNED_c measurements reported in this section provide the visual half of the experimental evidence for RQ5 and RQ6.

5.4.4 Comparison with Related Work

Table 15 contextualises the proposed architecture against recent SNN approaches evaluated on the same CIFAR-10 benchmark. State-of-the-art spiking transformers such as SGLFormer (Zhang et al., 2024) achieve 96.76% accuracy with only 4 timesteps, while surrogate-gradient-trained ResNets (Deng et al., 2022) reach 94.44%. These methods employ deeper architectures (ResNet-19, transformer blocks) and benefit from large-scale pre-training or highly optimised training recipes.

The proposed Conv+DyNED+cAdLIF+TET SNN combines a convolutional front-end with three layers of cAdLIF neurons, learnable per-neuron synaptic delays, and TET loss, trained with the ArcTan surrogate gradient. It achieves 88.07% test accuracy on CIFAR-10. Importantly, this result was obtained with a deliberately simple three-layer convolutional architecture without residual connections, skip layers, or pre-training - a design choice that isolates the contribution of the DyNED encoding pipeline from network depth. To provide a controlled comparison, a non-spiking CNN baseline was trained using the same convolutional front-end and identical training configuration (grayscale input, data augmentation, AdamW optimiser, one-cycle learning rate scheduler, 250 epochs), replacing only the spiking layers with standard fully connected layers and ReLU activations. A CNN was chosen over a transformer or other architecture because the goal is to measure the accuracy cost of spiking neurons specifically, not to compare different network topologies; by sharing the same convolutional feature extractor, the only variable is the classification head (spiking vs non-spiking). This baseline achieved 86.62%, placing the SNN approximately 1.5 percentage points above its non-spiking counterpart. Exceeding the matched-features non-spiking baseline confirms that the DyNED encoding pipeline preserves information effectively, and that the accuracy gap to state-of-the-art SNNs is attributable to architectural differences (deeper backbones, transformer blocks) rather than encoding quality. The DyNED_c variant achieves 87.69%, matching the DyNED-only result within run-to-run variance while adding loss-less compression that recovers 65.8% of the cAdLIF spike-output storage.

Table 15: Classification results comparison on CIFAR-10. Comparison methods are sourced from a recent survey (C. Liu et al., 2024).

Network Platform	Neural Network	Learning Mechanism	Learning Method	Accuracy	Training Epochs
Spiking (Conversion)	VGG-16 (Sengupta et al., 2019)	ANN-to-SNN weight normalisation	Supervised	91.55%	2,500 timesteps
Spiking (Hybrid)	VGG-11 (Rathi et al., 2020)	Conversion + STDB fine-tuning	Supervised	92.22%	125 timesteps
Spiking (Direct)	ResNet-19 (Zheng et al., 2021)	STBP-tdBN surrogate gradient	Supervised	92.92%	4 timesteps
Spiking (Direct)	ResNet-19 (Deng et al., 2022)	TET surrogate gradient	Supervised	94.44%	4 timesteps
Spiking (Transformer)	Spikformer-4-384 (Zhou et al., 2023)	Surrogate gradient	Supervised	95.51%	4 timesteps
Spiking (Transformer)	SGLFormer-4-384 (Zhang et al., 2024)	Surrogate gradient	Supervised	96.76%	4 timesteps
Deep Spiking (Proposed)	Conv+DyNED+cAdLIF+TET	Surrogate gradient (ArcTan) + cAdLIF neurons with learnable per-neuron synaptic delays, Conv front-end, DyNED encoder, TET loss	Supervised	88.07%	250 epochs
Deep Spiking (Proposed)	Conv+DyNED+DyNED_c+cAdLIF+TET	cAdLIF + delays + TET, Conv front-end, DyNED and DyNED_c	Supervised	87.69%	250 epochs
Non-spiking Baseline	CNN	Same Conv front-end with FC classifier (no spiking layers)	Supervised	86.62%	250 epochs

The CIFAR-10 SNN-versus-matched-CNN comparison reported in this section is the visual half of the experimental evidence for RQ7 (Section 1.3.3.7); the auditory half is taken up in Chapter 6, and the consolidated outcome appears in Table 41.

5.5 CIFAR10-DVS - Neuromorphic Extension

The preceding experiments demonstrated DyNED’s effectiveness on static CIFAR-10 images, where the encoder converts conventional pixel data into spike trains. To evaluate whether DyNED generalises to data that is already neuromorphic in origin, this section applies the same encoding and compression pipeline to the CIFAR10-DVS dataset (Li et al., 2017) - a neuromorphic version of CIFAR-10 captured directly by a Dynamic Vision Sensor (DVS).

5.5.1 Dataset

CIFAR10-DVS (Li et al., 2017) consists of 10,000 event streams (1,000 per class) recorded by a 128×128 DVS camera viewing moving CIFAR-10 images on a monitor. Unlike conventional cameras that capture full frames at fixed intervals, the DVS emits asynchronous events only when pixel-level brightness changes exceed a threshold. Each event is a tuple (x, y, t, p) encoding the spatial coordinates, timestamp, and polarity (brightness increase or decrease). The dataset retains the same 10 classes as CIFAR-10 - *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck* - enabling direct comparison of DyNED’s encoding quality across conventional and neuromorphic image data.

5.5.2 DyNED Encoding Pipeline

The DyNED encoding pipeline for CIFAR10-DVS proceeds as follows. First, the raw DVS events are spatially downsampled by a factor of 0.25 and accumulated into 16 temporal bins, producing a dense frame of shape $[16, 2, 32, 32]$ (two polarities). The polarities are summed, and the temporal bins averaged, yielding a single-channel 32×32 grayscale frame normalised to $[-1, 1]$. This frame is then transformed via 2D FFT, and the log-magnitude $\log(1 + |\text{FFT}_{2D}|)$ is flattened and passed through DyNED’s sigma-delta quantisation at 256 levels. The resulting quantised values are used to generate binary step signals (spikes), which constitute the encoded representation. Reconstruction is performed by reversing the log transform on the quantised magnitudes, combining with the original FFT phase, and applying the inverse 2D FFT.

5.5.2.1 Per-Sample Encoding and Compression Cost

The CIFAR10-DVS input is already event-based, so the comparator encoders (BSA, TBR, Delta, Latency, Rate) cannot be applied to a raw signal as in Table 6 or Table 24; instead, they were benchmarked on the dense $[16, 2, 32, 32]$ frame tensor that the SNN consumes after tonic’s event-to-frame conversion. The DyNED stage on this dataset is the frame preprocessing itself (events $\rightarrow$ downsample $\rightarrow$ frame binning $\rightarrow$ normalisation), since the sigma-delta quantisation runs inside the forward pass on convolutional features. DyNED_c was timed on a $[16, 512]$ binary tensor matching the cAdLIF spike-output shape at $\approx 10\%$ firing rate, the actual compression target downstream of the trained network.

Table 16: Per-sample encoding time for the CIFAR10-DVS pipeline, benchmarked over 500 samples. Comparator encoders (BSA, TBR, Delta, Latency, Rate) are applied to the dense $[16, 2, 32, 32]$ frame tensor that the SNN consumes.

Stage / Encoder	Mean (ms)	Median (ms)	Std (ms)
Frame preprocessing	0.068	0.064	0.074
DyNED_c (spike-output proxy)	0.787	0.779	0.064
Combined	0.856	0.843	0.114
BSA (Gollahalli, 2019)	1.528	1.485	0.502
TBR (Gollahalli, 2019)	0.140	0.133	0.117
Delta (Eshraghian et al., 2023b)	0.084	0.065	0.413
Latency (Eshraghian et al., 2023b)	1.503	2.007	0.821
Rate (Eshraghian et al., 2023b)	4.097	4.067	0.576

Table 16 shows that the dataset-side frame preprocessing is essentially free at 0.068 ms per sample, since tonic’s downsample-and-bin operations are vectorised numpy routines on a small event list. DyNED_c compression of the cAdLIF spike output dominates the per-sample budget at 0.787 ms, giving a combined cost of 0.856 ms per sample. Among comparator encoders applied to the dense frame, Delta (0.084 ms) and TBR (0.140 ms) are the fastest, while BSA (1.528 ms), Latency (1.503 ms), and Rate (4.097 ms) are slower than the full DyNED + DyNED_c pipeline. Extrapolated to the full 9,000-sample training set, frame preprocessing takes ≈ 0.6 s, and the prep + DyNED_c pipeline takes ≈ 7.7 s - negligible relative to the multi-hour SNN training step.

5.5.3 Encoding Quality

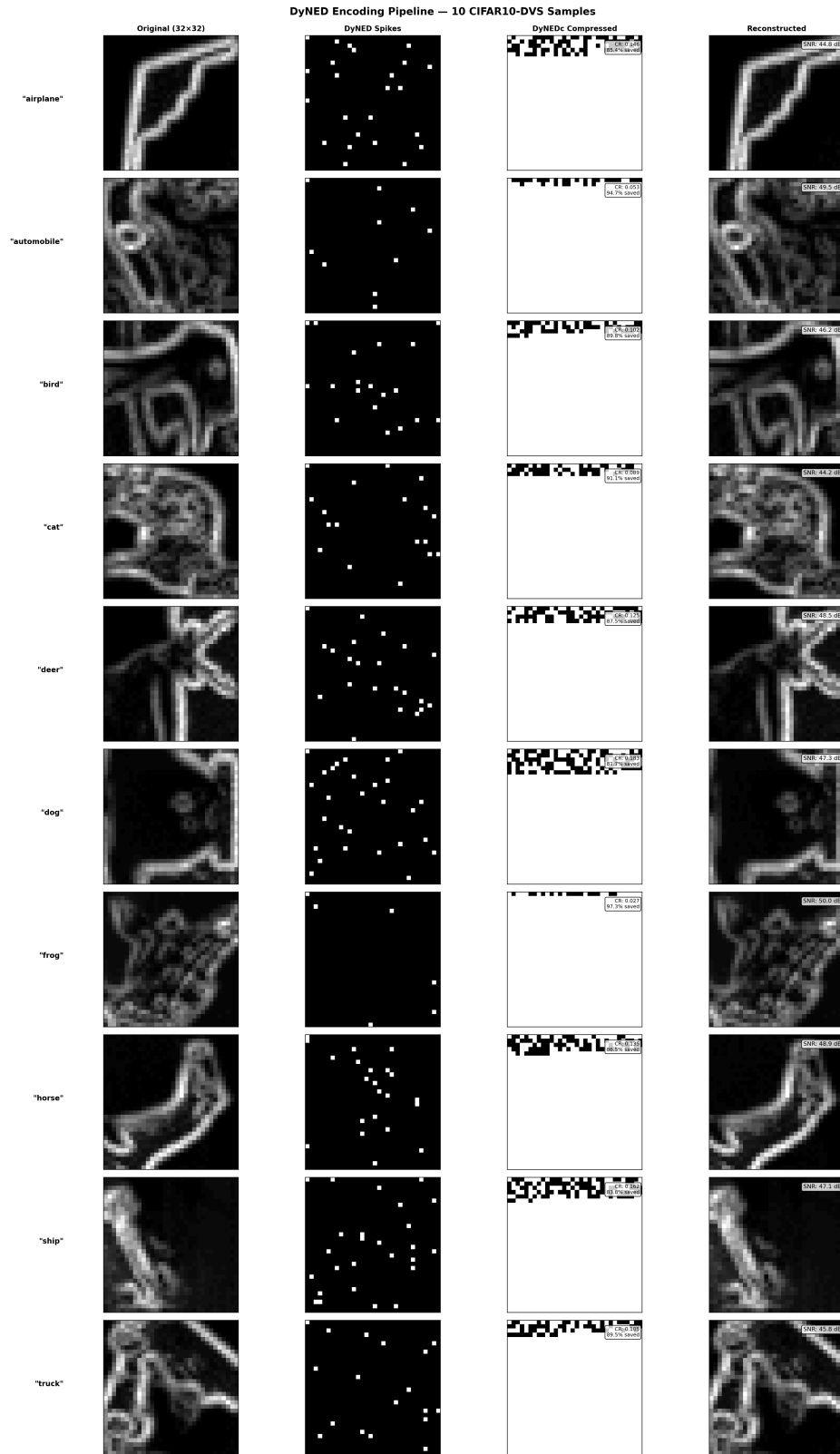


Figure 48: DyNED encoding pipeline for 10 CIFAR10-DVS samples. Each row shows (left to right): original DVS frame (32x32), DyNED spike matrix, DyNED_c compressed bitstream with compression ratio, and reconstructed frame with SNR.

Figure 48 presents the complete DyNED encoding pipeline applied to 10 CIFAR10-DVS samples, one per class. The first column shows the original 32×32 grayscale frames derived from the DVS event streams. The second column displays the DyNED spike matrices produced by sigma-delta quantisation of the FFT magnitudes. The third column shows the compressed bitstream for DyNED_c, where the compressed data occupies only a fraction of the original frame size; the remaining white space visually demonstrates the compression achieved. The fourth column presents the reconstructed frames obtained by inverting the DyNED encoding, with SNR annotations confirming reconstruction fidelity.

Table 17: DyNED encoding quality and DyNED_c compression for CIFAR10-DVS ($n = 10$, one sample per class, levels = 256).

Class	SNR (dB)	CR	Space Saved
Airplane	44.8	0.146	85.4%
Automobile	49.5	0.053	94.7%
Bird	46.2	0.102	89.8%
Cat	44.2	0.089	91.1%
Deer	48.5	0.125	87.5%
Dog	47.3	0.183	81.7%
Frog	50.0	0.027	97.3%
Horse	48.9	0.135	86.5%
Ship	47.1	0.162	83.8%
Truck	45.8	0.105	89.5%
Mean	47.2	0.113	88.7%

Table 17 summarises the encoding quality and compression performance across all 10 classes. The mean SNR of 47.2 dB confirms that DyNED preserves the essential spatial information from the DVS event streams with high fidelity. Per-class SNR ranges from 44.2 dB (*cat*) to 50.0 dB (*frog*), with classes that produce cleaner DVS frames (fewer noisy events) achieving higher reconstruction fidelity. Compared to the conventional CIFAR-10 results ($R^2 = 0.987$), the neuromorphic data achieves comparable encoding quality despite the fundamentally different input modality - DVS events rather than pixel intensities.

The DyNED_c compression achieves a mean compression ratio of 0.113, corresponding to 88.7% space savings. This is substantially better than the 9.6% savings achieved on conventional CIFAR-10 spike data at the same quantisation level (256). The difference arises because DVS-derived spike trains are inherently sparser - the sigma-delta quantiser produces fewer state changes when encoding the smooth, low-frequency content typical of downsampled DVS frames. The *frog* class achieves the highest compres-

sion (97.3% saved, CR = 0.027), while *dog* achieves the lowest (81.7% saved, CR = 0.183), reflecting variations in spatial complexity across classes.

5.5.4 CIFAR10-DVS Classification

To evaluate whether DyNED-encoded representations support downstream classification on neuromorphic data, the encoded spike trains were used as input to a deep convolutional spiking neural network. The architecture combines a non-spiking VGG-style convolutional backbone with DyNED temporal sigma-delta encoding and a three-layer constrained Adaptive Leaky Integrate-and-Fire (cAdLIF) (Deckers et al., 2024) spiking classifier with learnable synaptic delays.

Architecture: The VGG backbone consists of seven convolutional layers with channel progression $2 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 256 \rightarrow 512 \rightarrow 512 \rightarrow 512$, with batch normalisation and ReLU activations, and three average-pooling stages reducing the 32×32 input spatial dimension to 4×4 . The flattened convolutional features ($512 \times 4 \times 4 = 8192$ dimensions) are projected down to a 256-dimensional latent space, encoded by DyNED across the $T = 16$ temporal axis using sigma-delta quantisation at 16 levels, then projected up to 1024 hidden units. The spiking classifier comprises three cAdLIF layers (1024 neurons each) with learnable per-neuron synaptic delays ($\max = 2$), batch normalisation, and dropout (0.21). A leaky readout layer integrates membrane potentials over time, and softmax votes are accumulated across timesteps to produce class scores. The full architecture is shown in Figure 49.

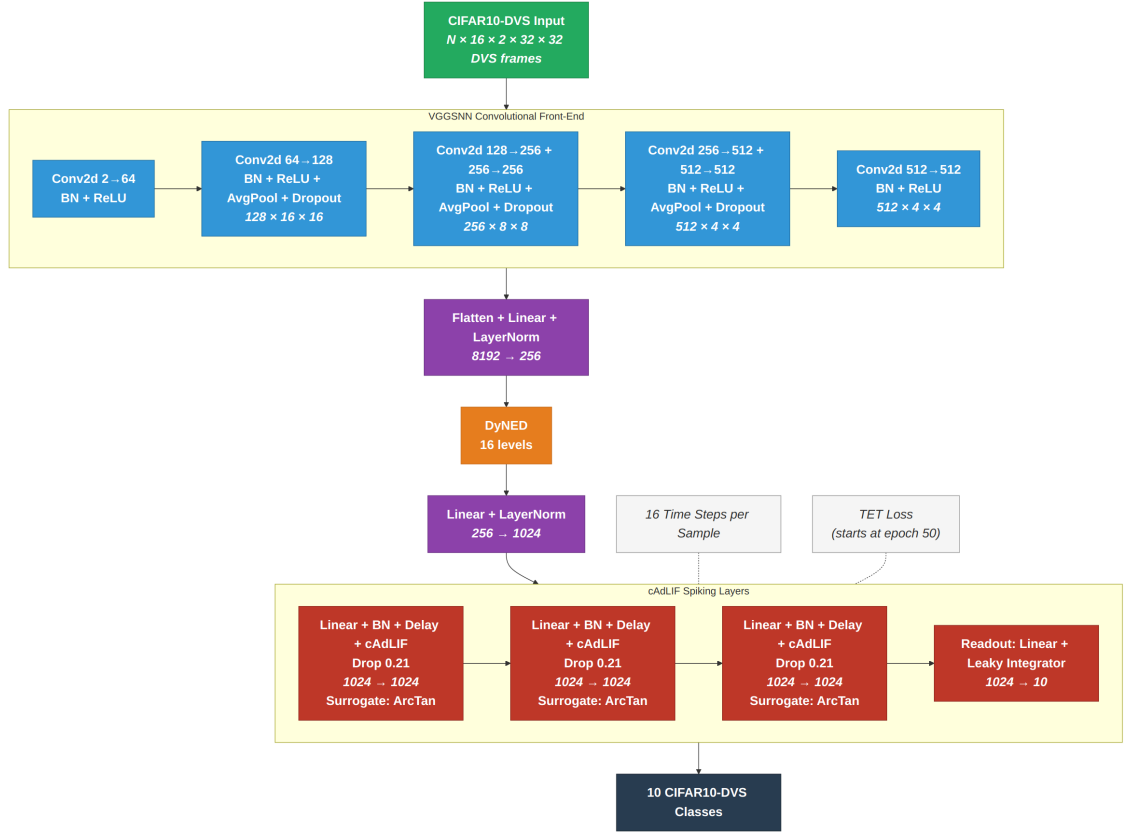


Figure 49: Network architecture for CIFAR10-DVS classification: VGGSNN convolutional front-end, DyNED temporal sigma-delta encoding across $T = 16$ DVS frames, and a three-layer cAdLIF spiking classifier with learnable per-neuron synaptic delays.

Training Protocol: The dataset was split 90%/10% for training and testing (9000 train, 1000 test), matching the standard CIFAR10-DVS evaluation protocol (Li et al., 2017). The network was trained for 300 epochs using SGD with momentum 0.9, an initial learning rate of 0.1, weight decay 5×10^{-4} , and cosine annealing to 1×10^{-6} . The cAdLIF cell parameters (α , β , a , b) and the readout decay α_{out} were placed in a separate optimiser group with weight decay disabled, since these are physically meaningful neuron parameters and decaying them toward zero degrades the cAdLIF into a vanilla LIF. Per-neuron synaptic delays were initialised uniformly in $[0, \text{max_delay}]$ so each neuron started at a different value and gradients could move it in either direction. Training followed the two-phase strategy of (Deng et al., 2022): standard cross-entropy loss on time-averaged membrane potentials for the first 50 epochs, followed by Temporal Efficient Training (TET) loss for the remaining 250 epochs. TET applied cross-entropy independently at each timestep and averaged the result, with an additional MSE temporal-consistency regulariser ($\lambda_{\text{TET}} = 1 \times 10^{-3}$). Augmentation consisted of horizontal flip, random affine, 6×6 cutout (30% proba-

bility), and EventDrop (Gu et al., 2021) (10% temporal bins, 30% probability). Mixed precision (AMP) was used throughout, with gradient clipping at $\max_{\text{norm}} = 1.0$.

Results: The network achieved 76.3% best test accuracy at epoch 260. Table 18 reports per-class accuracy, and Table 19 provides a comparison with published results on this dataset.

Table 18: Per-class test accuracy for CIFAR10-DVS at the best epoch (260). Accuracies range from 47.1% (cat) to 90.4% (automobile).

Class	Accuracy	Class	Accuracy
automobile	90.4%	horse	78.4%
ship	88.7%	deer	75.0%
airplane	87.1%	dog	69.5%
truck	86.3%	bird	58.4%
frog	83.5%	cat	47.1%

Table 19: Comparison of CIFAR10-DVS classification accuracy with published direct-trained SNN methods. The DyNED-based encoding achieves comparable accuracy to the SEW-ResNet baseline while preserving an explicit, interpretable spike representation.

Method	Architecture	Encoding	Accuracy
Fang et al. (Fang et al., 2021)	SEW-ResNet	Direct	74.4%
Deng et al. (Deng et al., 2022)	VGGSNN + TET	Direct	83.2%
This work	VGGSNN + cAdLIF + TET	DyNED	76.3%

The DyNED-based system surpasses direct-trained SEW-ResNet baselines (Fang et al., 2021) (74.4% vs 76.3%). The gap to the highest published result on this dataset (Deng et al. (Deng et al., 2022) at 83.2%) reflects the additional sigma-delta quantisation step in the DyNED pipeline, which trades approximately 7 percentage points of accuracy for a structured, compressible spike representation suitable for DyNED_c compression and neuromorphic deployment. This trade-off is consistent with DyNED’s role as an explicit encoding layer rather than a learned representation collapsed into the network weights.

Hyperparameter Optimisation: The hyperparameters were optimised using Optuna (Akiba et al., 2019) with the TPE sampler (Section 2.6.2), running 50 trials with 50 -epoch evaluation. The search space and best values are shown in Table 20; a parameter-importance analysis of the underlying TPE search is provided in Appendix A.9.

Table 20: Hyperparameter search space and best values for CIFAR10-DVS classification with VGGSNN + DyNED + cAdLIF + TET, found via TPE with Optuna (50 trials, 50 epochs per trial). Test accuracy at the full 300-epoch training schedule: 76.3%.

Hyperparameter	Search Range	Best Value
Learning rate	$1 \times 10^{-2} - 3 \times 10^{-1}$ (log)	0.133
Delay learning rate	$1 \times 10^{-3} - 1 \times 10^{-1}$ (log)	0.018
Weight decay	$1 \times 10^{-5} - 1 \times 10^{-3}$ (log)	1.63×10^{-5}
Hidden size	{256, 512, 1024}	1024
DyNED dimension	{256, 512, 1024}	256
Max delay	{2, 3, 5, 7}	2
Dropout	0.1 - 0.4	0.213
Conv dropout	0.05 - 0.2	0.132
Batch size	{64, 128, 256}	64
DyNED levels	{16, 32, 64, 128, 256}	16
TET λ	$1 \times 10^{-4} - 1 \times 10^{-1}$ (log)	0.057

5.5.5 CIFAR10-DVS Classification with DyNED_c Compression

The DyNED_c variant for CIFAR10-DVS used the same VGGSNN + DyNED + cAdLIF backbone (Figure 49) and the same training protocol described in Section 5.5, with two refinements informed by an earlier experiment in which the cAdLIF cell parameters drifted toward zero. First, the cAdLIF cell parameters (α , β , a , b) and the readout decay α_{out} were placed in a separate optimiser group with weight decay disabled, since these are physically meaningful neuron parameters and decaying them toward zero degrades the cAdLIF into a vanilla LIF. Second, the TET phase was started earlier - at epoch 50 rather than epoch 200 - so the per-timestep gradient pressure shaped the model from the beginning of training rather than only correcting it at the end. The hyperparameters used, and the corresponding TPE-Optuna search analysis are reported in Appendix A.10. The full architecture is shown in Figure 50.

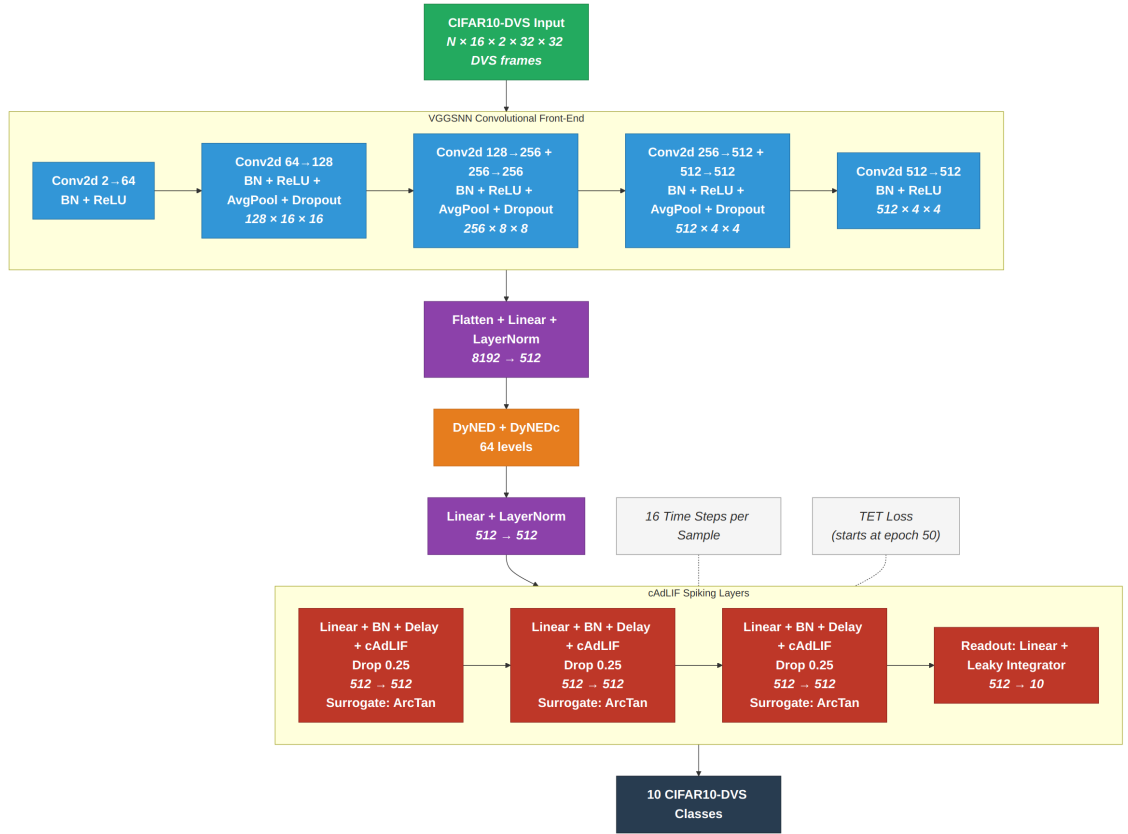


Figure 50: Architecture for CIFAR10-DVS classification with DyNED + DyNED_c. The combined encoder-plus-compressor pipeline is shown as a single block feeding the VGGSN-fronted cAdLIF spiking backbone.

Results: The network achieved 79.0% best test accuracy at epoch 264, a 2.7-percentage-point improvement over the DyNED-only variant (Section 5.5). Table 21 reports the per-class breakdown.

Table 21: Per-class test accuracy for CIFAR10-DVS with DyNED_c compression at the best epoch (264). Overall test accuracy: 79.0%. Per-class accuracy ranges from 50.0% (cat) to 92.2% (automobile).

Class	Accuracy	Class	Accuracy
automobile	92.2%	ship	85.6%
truck	91.6%	deer	71.0%
frog	90.3%	bird	69.3%
horse	86.6%	dog	68.6%
airplane	85.9%	cat	50.0%

Table 21 shows the per-class breakdown. Vehicle classes - *automobile* (92.2%), *truck* (91.6%), *airplane* (85.9%), *ship* (85.6%) - are classified consistently above 85%, while animal

classes show greater variation: *frog* and *horse* exceed 86% but *bird* (69.3%), *dog* (68.6%) and *cat* (50.0%) trail. The *cat* result is the weakest, reflecting the limited spatial discriminability of the downsampled 32×32 DVS frames for visually similar quadrupeds combined with only 1,000 samples per class.

5.5.5.1 DyNED_c compression on cAdLIF spike outputs

DyNED_c compresses the binary spike trains that the three cAdLIF hidden layers emit during inference. Table 22 reports the compression ratio (compressed size divided by raw size, lower is better) and the corresponding space saving for each layer along with an overall summary across the 1,000-sample test set.

Table 22: DyNED_c compression statistics on the binary spike outputs of the three cAdLIF hidden layers during inference, aggregated over the 1,000-sample test set. Lower compression ratio is better. All 3,000 samples selected the alternating-Huffman branch, indicating consistently sparse spike-train statistics. The compression is lossless and therefore has no effect on classification accuracy.

Layer	Compression Ratio	Space Saving	Min - Max Ratio	<i>n</i> samples
Layer 1 (cAdLIF)	0.087	91.3%	0.027 - 0.125	1,000
Layer 2 (cAdLIF)	0.131	86.9%	0.080 - 0.176	1,000
Layer 3 (cAdLIF)	0.118	88.2%	0.035 - 0.162	1,000
Overall	0.112	88.8%	0.027 - 0.176	3,000

Table 22 shows that DyNED_c compresses the cAdLIF spike trains to approximately 11% of their original size on average, an $\approx 9x$ reduction. Layer 1 is the most compressible (91.3% saving) because its spike trains are the sparsest, while Layer 2 is slightly less compressible (86.9% saving) due to higher per-neuron firing rates. All 3,000 samples select the alternating-Huffman branch, indicating that the spike-train statistics are stable enough across samples and layers for the compressor to choose a single optimal coding strategy. Together with the 79.0% classification accuracy, this confirms that the DyNED + cAdLIF + DyNED_c pipeline produces classification-relevant binary spike trains that admit an order-of-magnitude lossless reduction in memory footprint.

The CIFAR10-DVS extension reported in this section is the neuromorphic-static cell of the four-cell evaluation underlying RQ8 (Section 1.3.3.8); the neuromorphic-dynamic cell is taken up via SSC in Chapter 6, and the consolidated cross-cell verdict is recorded in Table 42.

5.6 Conclusion

This case study demonstrated the application of a convolutional spiking neural network for image classification on the CIFAR-10 dataset, achieving 88.07% test accuracy. The architecture combines a learnable convolutional front-end ($1 \rightarrow 64 \rightarrow 128 \rightarrow 256$ channels with batch normalisation, ReLU, and max pooling) with three cAdLIF spiking layers with learnable per-neuron synaptic delays ($1,024 \rightarrow 512 \rightarrow 256 \rightarrow 10$ neurons), trained with the ArcTan surrogate gradient and TET loss, processing each image over 25 time steps.

The per-class analysis revealed that structured objects like *automobile* (95.4%), *ship* (94.0%), and *truck* (92.1%) are classified with high confidence, while visually similar categories like *cat* (75.3%), *bird* (82.7%), and *dog* (82.4%) present greater challenges. The network develops sparse, energy-efficient representations with firing rates around 5-10% across layers, demonstrating the computational efficiency advantages of spiking neural networks.

The DyNED encoder was validated for image-to-spike conversion, with a reconstruction R^2 of 0.987 and quantisation error within ± 0.1 units. The DyNED_c variant achieved 87.69% classification accuracy at epoch 247, essentially matching the DyNED-only result within run-to-run variance, while losslessly compressing the cAdLIF spike outputs by 65.8% overall (81.4% on the deepest layer).

This case study shows that spiking neural networks, when augmented with convolutional feature extraction, cAdLIF adaptation, learnable synaptic delays, and TET surrogate-gradient training, can effectively classify natural images. The combination of biologically-inspired temporal processing with modern deep learning techniques provides a viable pathway toward energy-efficient visual recognition systems.

Chapter 6

Case Study - Speech Commands

6.1 Introduction

This chapter is the second of the two case-study chapters that empirically test the DyNED + DyNED_c pipeline assembled in Chapter 3 and Chapter 4 against the RPS. It addresses RQ3 and RQ4 specifically on auditory data, alongside the broader RQ1, RQ2, RQ5, RQ6, RQ7, and RQ8, using Speech Commands as the conventional benchmark and Spiking Speech Commands as its neuromorphic cochlear-spike counterpart. It is a direct progression from Chapter 5: the same DyNED + DyNED_c pipeline and the same supervised learning rule are reused without structural change, with the downstream SNN classifier swapped from a convolutional front-end to a recurrent cAdLIF stack to match the temporal nature of audio.

Spiking Neural Networks, being event-driven systems, are well-suited for processing spatio-temporal data (Zhou et al., 2024). In this study, we use an encoder-decoder architecture with data compression to classify spoken words using the Speech Commands dataset (Warden, 2018).

Speech commands by Google (Warden, 2018) is a dataset of spoken words. It is a temporal dataset containing 105,829 audio files of 35 words spoken by thousands of people. The dataset is divided into 32,540 training files, 3,768 validation files, and 3,720 test files, and it is designed for training machine learning models to recognise spoken words.

Each audio file consists of a single word, spoken by a variety of speakers, and is one second long. The audio files are stored as uncompressed 16-bit PCM WAV files at a

sample rate of 16 kHz. The dataset is designed to be easy to use while also providing a benchmark for machine learning models. Although each sample is one second long, the onset of words in the recording may vary, with some starting at the beginning and others just before the end. This slight variation in the start time of the word is intentional and is designed to make the dataset more challenging.

The Speech Commands dataset was selected because its temporal, one-dimensional nature complements CIFAR-10's static, two-dimensional data, directly testing DyNED's versatility across fundamentally different data types (Section 1.3.3.3). The variation in speakers, accents, and word-onset times tests the robustness of DyNED's encoding to real-world signal variability. At the same time, the dataset's scale (105,829 audio files) provides a meaningful benchmark for evaluating GPU-accelerated encoding performance (Section 1.3.3.2).

The history of speech recognition dates back to the 1950s when Bell Labs first introduced Audrey (Mishra et al., 2024), the Automatic Digit Recognition. Since then, the field has seen a lot of advancements, namely, the introduction of Hidden Markov Models (HMM) in the 1970s by L. R. Rabiner (Juang & Rabiner, 1991), the introduction of Recurrent Neural Networks (RNN) in the 1980s by L. Elman (Elman, 1990), and the introduction of Convolutional Neural Networks (CNN) in the 1990s by Y. LeCun (LeCun et al., 2015). In the past decade, we have seen a boom in deep neural network research, with the introduction of deep learning and Long Short-Term Memory (LSTM). This study demonstrates the use of Spiking Neural Networks, the DyNED and DyNED_c encoder to recognise spoken words.

6.2 Relevance

Speech is a complex process which involves the coordination of several muscles and organs. The larynx, commonly known as the voice box, is a key organ in the production of speech. The larynx contains the vocal cords, which are two bands of muscle that vibrate to produce sound. The pitch of the sound is determined by the tension in the vocal cords, and the volume of the sound is determined by the force of the air passing through the vocal cords. Figure 51 shows a front view and a cross-sectional view of the larynx. The brain also receives feedback from the ears to monitor speech quality. In this manner, the brain can distinguish emotions in speech.

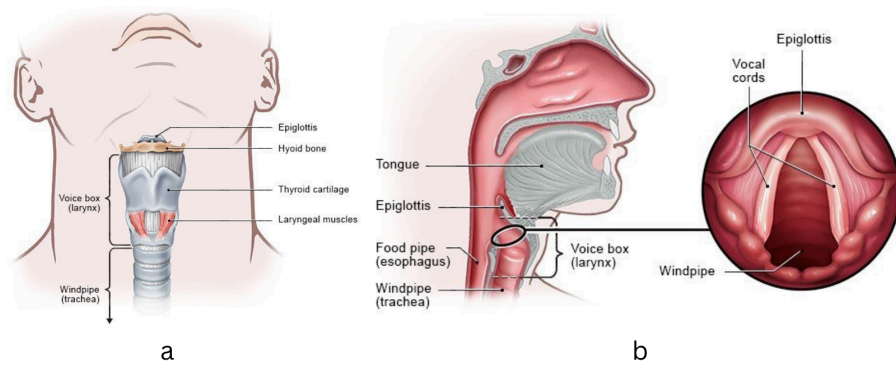


Figure 51: a) A front view of the larynx. b) A cross-sectional view of the larynx with vocal cords. (*How Does the Larynx Work?*, 2024)

Generally, women tend to speak at 168 Hz [throughout the years, the frequency of women’s speech has decreased tremendously to 100 Hz (Heister et al., 2022)], and the frequency of men’s speech has remained consistent at 110 Hz (Heister et al., 2022). Our brain is capable of isolating these tones and frequencies to distinguish the emotions.

6.3 Methodology

The proposed methodology follows the same general pattern as Chapter 5, but differs in preprocessing and SNN architecture. Preprocessing and the encoder perform several sequential steps to convert the raw audio to spike trains. Using the DyNED encoder, the raw audio is converted into spike trains so the SNN can classify spoken words accurately. An overview of the methodology is shown in Figure 52.

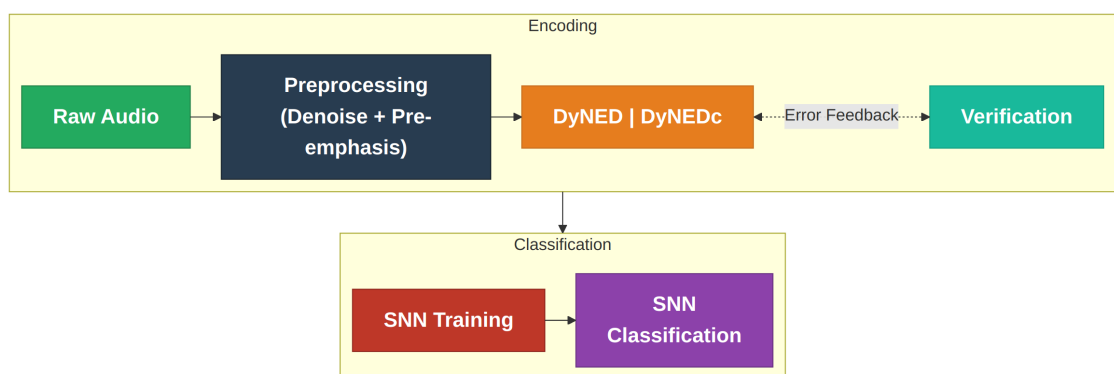


Figure 52: Methodology for speech classification using Spiking Neural Networks (SNN).

The experiment begins with raw speech audio, which is then filtered to remove any noise and unwanted frequencies. The cleaned audio is then fed to FFT for conversion to the frequency domain. These signals are fed through the DyNED encoder to generate spike trains. The spike trains are then fed to the SNN for classification. The SNN is trained on

spike trains and their corresponding labels. The trained model is then used to classify the spoken words.

6.3.1 System Integration and Performance

Unlike using DyNED on the CIFAR-10 dataset, the speech dataset requires less memory and processing power, though the same type of error-detection and feedback system is used for both. This allows for continuous optimisation to balance classification accuracy with computational efficiency.

The hardware and software environment is identical to Section 5.3.1 (DigitalOcean 20-core vCPU, 240 GB RAM, NVIDIA H100 80 GB GPU, Python 3.12, PyTorch 2.5).

Table 23: Comparison of memory usage, processing power, time taken, and GPU usage for DyNED, BSA, and TBR.

Encoder Type	Memory (%)	Processing (%)	Time	GPU Usage (%)
DyNED	$\approx 40\%$	$\approx 52\%$	≈ 1.1 min	$\approx 67\%$
DyNED _c	$\approx 48\%$	$\approx 55\%$	≈ 8.7 min	$\approx 69\%$
BSA (Gollahalli, 2019) ²	$\approx 92\%$	$\approx 94\%$	≈ 1.2 min	$\times$
TBR (Gollahalli, 2019) ²	$\approx 89\%$	$\approx 92\%$	≈ 12 s	$\times$
Delta (Eshraghian et al., 2023b)	$\approx 40\%$	$\approx 68\%$	≈ 6 s	$\approx 55\%$
Latency (Eshraghian et al., 2023b)	$\approx 58\%$	$\approx 71\%$	≈ 1 min	$\approx 62\%$
Rate (Eshraghian et al., 2023b)	$\approx 62\%$	$\approx 84\%$	≈ 4 min	$\approx 62\%$

The performance comparison in Table 23 shows that DyNED itself encodes the full 84,843-sample training set in approximately 1.1 minutes, comparable to BSA (≈ 1.2 min) and well below Rate (≈ 4 min). The mel + MFCC torch transforms run on the GPU and the sigma-delta quantisation is numba-JIT compiled, keeping per-sample cost low. The simpler encoders - Delta (≈ 6 s), TBR (≈ 12 s), and Latency (≈ 1 min) - remain the fastest as they apply lightweight transformations directly to the raw waveform.

The combined DyNED + DyNED_c pipeline takes approximately 8.7 minutes, with DyNED_c contributing roughly 7.6 of those minutes. The dominant cost in the compression stage comes from the per-row run-length analysis and Huffman packing applied to the 160-row $\times$ 51-frame index tensor produced by DyNED on each sample. DyNED maintains moderate resource consumption at 40% memory, 52% processing power, and 67% GPU utilisation.

The BSA and TBR encoders, operating exclusively on multi-processor CPU systems without GPU acceleration, demand substantially higher system resources - BSA

²Does not use GPU. Running parallel in multi-processor CPU

at 92% memory and 94% processing power, TBR at 89% memory and 92% processing power. This highlights that encoding speed and resource efficiency are distinct concerns: simpler encoders can be faster per sample, but GPU-accelerated encoders like DyNED achieve lower memory and CPU utilisation by offloading computation to the GPU.

The encoders from Eshraghian et al. (2023b) - Delta, Latency, and Rate - demonstrate fast encoding with moderate resource usage and consistent GPU utilisation around 55-62%. The overall comparison shows that DyNED encoding cost is comparable to the simpler baselines, with the additional storage saving from DyNED_c paid for by a one-off compression pass at dataset preparation time.

6.3.1.1 Per-Sample Encoding and Compression Cost

To quantify the encoding overhead more precisely, all encoders were benchmarked over 500 Speech Commands samples using the same toolchain described in Section 5.3.1. Table 24 presents the per-sample mean encoding time for each encoder.

Table 24: Per-sample encoding time for all encoders on Speech Commands (16 kHz, 1-second waveforms), benchmarked over 500 samples.

Encoder	Mean (ms)	Median (ms)	Std (ms)
DyNED	0.758	0.757	0.050
DyNED_c	5.367	5.203	4.258
DyNED + DyNED_c	6.125	5.961	4.257
BSA (Gollahalli, 2019)	0.848	0.815	0.144
TBR (Gollahalli, 2019)	0.139	0.136	0.013
Delta (Eshraghian et al., 2023b)	0.068	0.066	0.011
Latency (Eshraghian et al., 2023b)	0.705	0.705	0.068
Rate (Eshraghian et al., 2023b)	2.846	2.902	0.155

Table 24 shows that DyNED encoding takes a mean of 0.758 ms per waveform, while DyNED_c compression adds 5.367 ms, for a combined cost of 6.125 ms per sample. The mean per-sample compared to CIFAR-10 (0.145 ms) is approximately 42 times higher, reflecting two effects. First, each speech sample is downsampled to 8 kHz and passed through 80-channel mel spectrogram and 80-channel MFCC transforms before DyNED quantisation, producing a 160×51 feature tensor per sample compared to the 1,024 values in a flattened CIFAR-10 image. Second, DyNED_c acts on the flattened uint8 index representation of this tensor (8,160 values per sample) rather than the 1,024-value CIFAR-10 spike train, and so dominates the per-sample cost. DyNED itself remains in the same range as the simpler encoders (Delta 0.068 ms, TBR 0.139 ms, Latency 0.705

ms, BSA 0.848 ms, Rate 2.846 ms), and the bulk of the speech-side overhead sits in the compression stage. All encoding costs remain small relative to the SNN training step, which includes forward propagation through spiking layers and backpropagation.

6.3.2 Preprocessing

Any Speech or audio is considered temporal data; that said, it contains a lot of noise. This can take the form of background noise or noise from the recording device. These unwanted noises can be removed using various techniques. Consider an example of raw audio of *bird* from the Speech Commands dataset shown in Figure 53. It shows three plots: the first contains the raw audio, the middle contains the Mel Spectrogram revealing the phonetic structure, and the last shows the audio spectrogram.

The first plot shows that the actual audio starts around ≈ 4000 (≈ 500 ms at the 8 kHz sample rate used here), which can be confirmed by the Mel Spectrum plot. Anything before that is considered noise and can be removed.

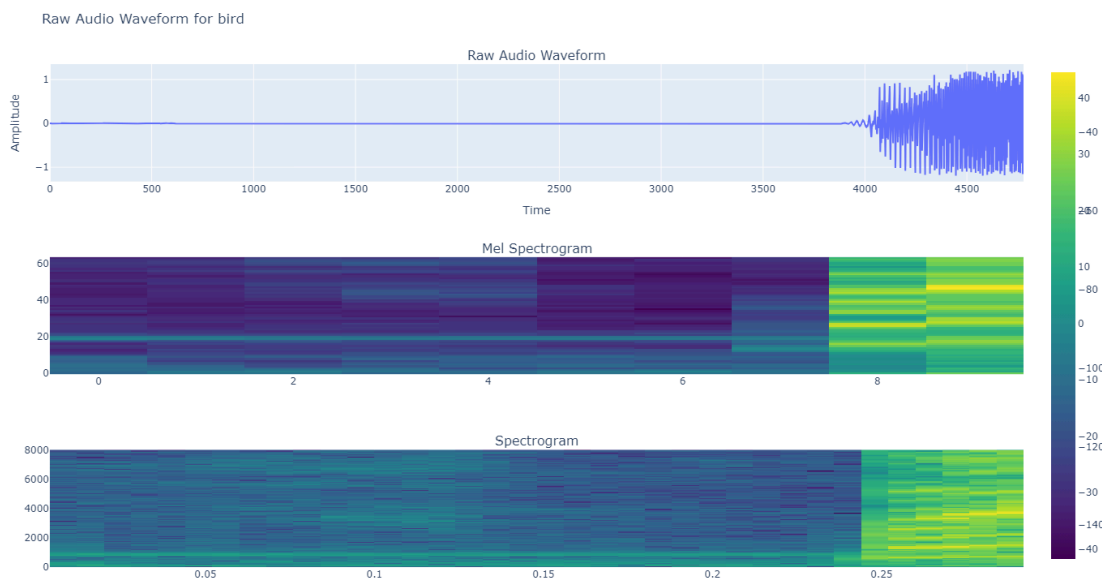


Figure 53: A sample of raw audio of *bird* from the Speech Commands dataset.

Using Spectral Gating (Section 2.9.11), the noise can be removed from the audio. The cleaned audio of *bird* from the Speech Commands dataset is shown in Figure 54. The cleaned audio visualisation of the word *bird* demonstrates the effectiveness of spectral gating as a noise reduction technique. When comparing this processed version to the original audio, several significant improvements become apparent across all three visualisation methods. The raw audio waveform reveals a markedly cleaner signal, particularly

in the sections before the onset of the speech. The background noise has been substantially reduced in these silent portions, while the crucial speech segment around time point 4000 (≈ 500 ms) maintains its fundamental structure, albeit with enhanced clarity.

The Mel Spectrogram provides the most striking evidence of the impact of spectral gating. The silent regions are now represented by more uniform dark purple areas, indicating successful noise suppression. This enhanced contrast between speech and non-speech segments makes the acoustic features of the word *bird* stand out more prominently. The spectral content of the actual word, occurring around time point 8 (≈ 160 ms; mel-frame index at hop length 160 over 8 kHz audio), remains intact but appears more distinct due to the reduction in background interference.

Several technical refinements are noteworthy in the standard spectrogram. The analysis now focuses on a narrower frequency range, capped at 4000 Hz rather than the original 8000 Hz, helping concentrate on the most relevant speech frequencies. The background noise components have been effectively attenuated across all frequency bands, resulting in a more precise visualisation of the speech formants and phonetic transitions. The temporal focus has also been adjusted, with the spectrogram beginning around 0.4 seconds to better highlight the actual speech segment.

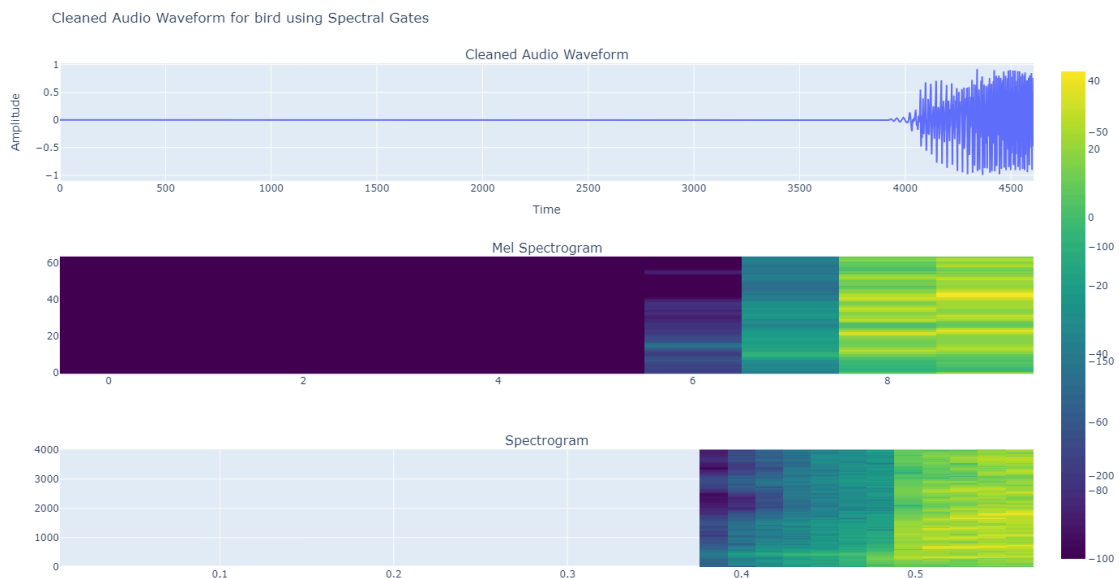


Figure 54: A sample of cleaned audio of *bird* from the Speech Commands dataset using Spectral Gates.

Before planning to use Spectral Gating as the preprocessing technique, a comparison was undertaken with the FIR filter and the Bandpass filter, as shown in Figure 55. The spectral

gating method (second row) significantly improves noise reduction while preserving the integrity of the speech signal. The spectrogram shows a clearer contrast between speech and non-speech regions, and the frequency spectrum shows a more defined structure with a reduced noise floor. The waveform maintains the essential shape of the speech signal while reducing the background variations.

The bandpass filter approach (third row) is aggressive in its filtering. While it does reduce noise, it also affects the speech signal components. This is evidenced by the more constrained frequency range in the spectrogram and the reduced amplitude in certain frequency bands. This might impact the natural quality of the speech.

The FIR (Finite Impulse Response) filter (fourth row) shows moderate noise reduction. However, it appears to retain more background noise than spectral gating, as visible in both the spectrogram and frequency spectrum. The time-domain waveform also shows more residual noise components.

Spectral gating achieves the best balance between noise reduction and speech preservation, making it the most suitable choice for this particular audio processing task. This is evident from its cleaner spectrogram, well-preserved frequency components, and natural-looking waveform structure.

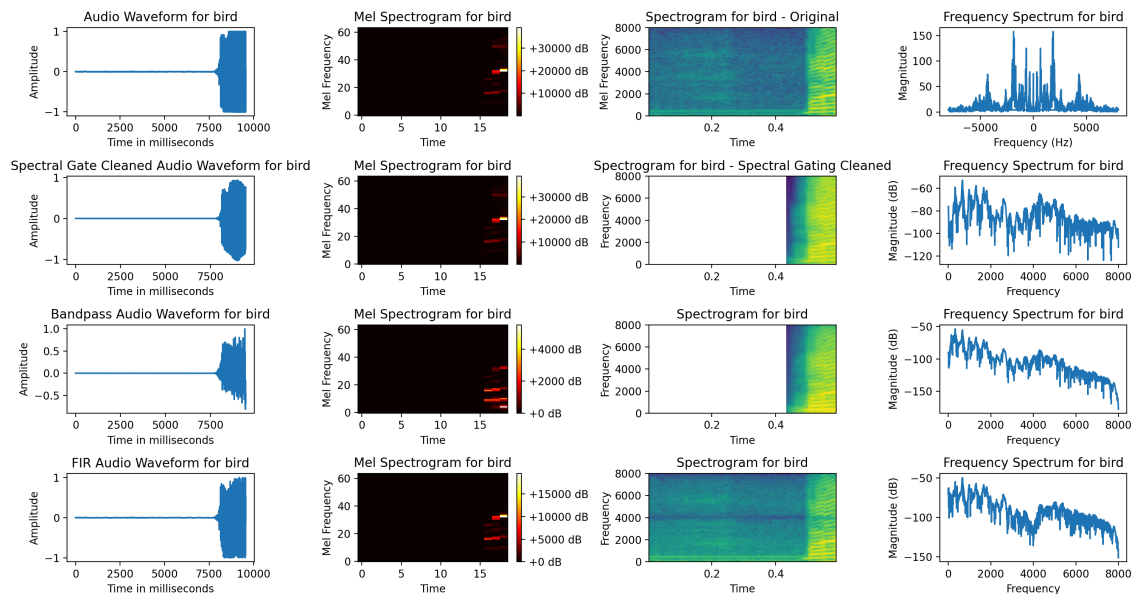


Figure 55: Comparison of various cleaning techniques for *bird* from the Speech dataset. (Top to Bottom) Raw audio, spectral gating, Bandpass filter and FIR filter.

6.3.3 Spike Trains and Sigma-Delta Internals

Once the audio is cleaned, it can then be converted to spike trains. Using the DyNED encoder, the cleaned audio of *bird* from the Speech Commands dataset is converted to spike trains. The spike trains are generated by quantising the FFT of the audio signals. The quantised signals are then converted to spike trains using the DyNED encoder. The spike trains are shown in Figure 56.

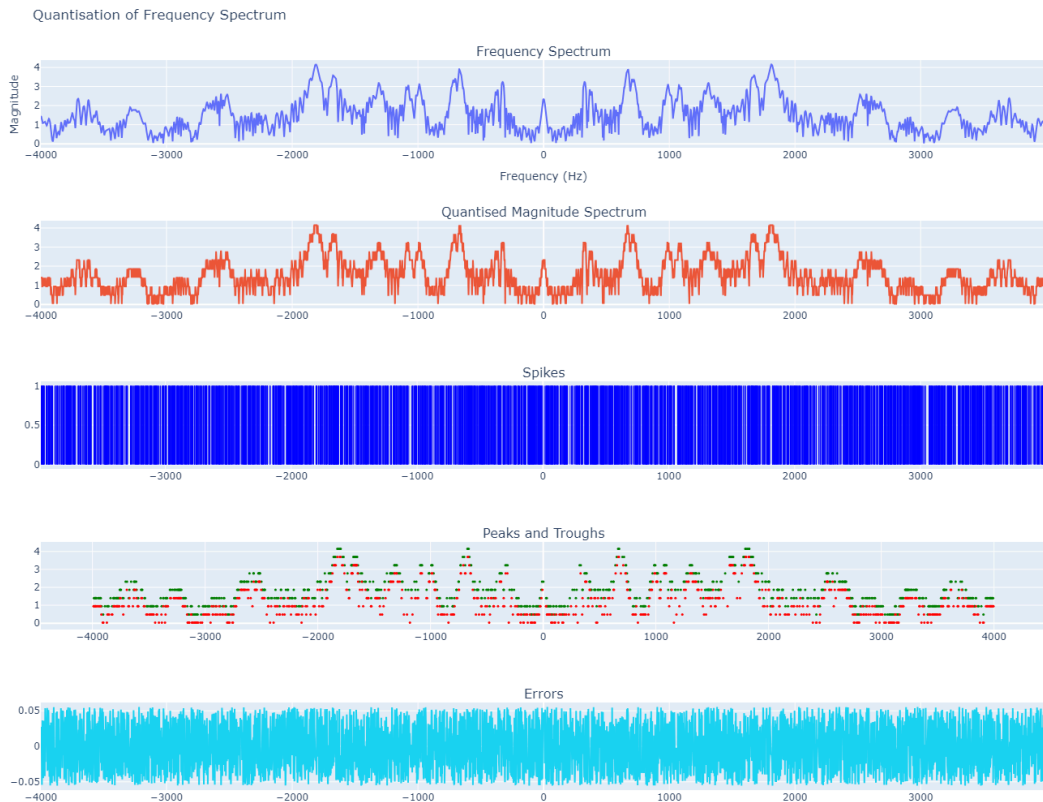


Figure 56: (Top to Bottom) Quantised signals of *bird* from the Speech Commands dataset. Spike trains are generated from the quantised signals. Peaks and troughs are shown in the quantised signals. And Quantisation error is shown in the last plot.

The visualisation presents a comprehensive breakdown of the DyNED (Dynamic Neural Encoder-Decoder) signal-processing pipeline, illustrating how an audio signal is transformed into spike trains across multiple processing stages. The topmost plot displays the original frequency spectrum in blue, spanning -4000 to 4000 Hz, capturing the input signal's natural frequency content and variations. This serves as the baseline for subsequent processing steps and provides a reference point for evaluating the quality of the transformation.

Following the original spectrum, the second plot shows the quantised magnitude spectrum in red, demonstrating how the continuous frequency information has been discretised into distinct levels. This quantisation process maintains the fundamental structure and characteristics of the original spectrum while reducing the precision to facilitate spike-based encoding. The discretised representation is an intermediate step between the continuous signal and the final spike train output.

The central plot introduces the spike train representation, depicted as vertical blue lines, which indicate the temporal occurrence of spikes. This binary encoding represents the signal's features by the presence or absence of spikes, with spike density corresponding to the intensity of frequency components in the original signal. This transformation is crucial for neural-inspired processing systems that operate on discrete events rather than continuous values.

The fourth plot provides insight into the signal structure by marking peaks (green dots) and troughs (red dots) in the quantised spectrum. This visualisation helps identify significant transitions and important features in the signal, offering a clear view of how the quantisation process preserves critical points in the frequency content. The distribution of these peaks and troughs demonstrates the preservation of signal dynamics through quantisation.

The final plot at the bottom displays the quantisation error in cyan, showing the difference between the original and processed signal. The error oscillates within a narrow band of approximately ± 0.05 , indicating high fidelity in the signal transformation process. The consistent and relatively small magnitude of these errors suggests that the DyNED system effectively preserves the essential characteristics of the original signal while converting it into a spike-based representation suitable for neuromorphic processing.

6.3.3.1 Effect of Quantisation Levels on Speech Encoding

As with the CIFAR-10 case study (Table 7), the effect of varying quantisation levels was evaluated on 100 Speech Commands test waveforms. Table 25 reports the mean spike density, Signal-to-Noise Ratio (SNR), and Root Mean Squared Error (RMSE) at each level.

Table 25: Effect of quantisation levels on spike density and reconstruction quality for Speech Commands ($n = 100$). The selected operating point (32 levels) is shown in bold and underlined.

Levels	Spike Density	SNR (dB)	RMSE
2	0.228	-4.44	0.143274
4	0.380	11.86	0.020024
8	0.502	20.75	0.006947
16	0.619	27.63	0.003125
<u>32</u>	<u>0.717</u>	<u>33.99</u>	<u>0.001493</u>
64	0.796	40.28	0.000721
128	0.860	46.36	0.000362
256	0.909	52.31	0.000182
512	0.944	58.42	0.000089

Spike density increases monotonically from 0.228 at 2 levels to 0.944 at 512 levels, while reconstruction quality improves from -4.44 dB SNR (destructive quantisation) to 58.42 dB (near-perfect reconstruction). The speech data shows a wider dynamic range than images, with meaningful improvements continuing up to 512 levels. For the Speech Commands classification experiments, 32 levels were selected via TPE optimisation (Table 27) as the best trade-off between spike sparsity and classification accuracy. At 32 levels, the SNR of 33.99 dB is sufficient for accurate classification while maintaining a moderate spike density of 0.717.

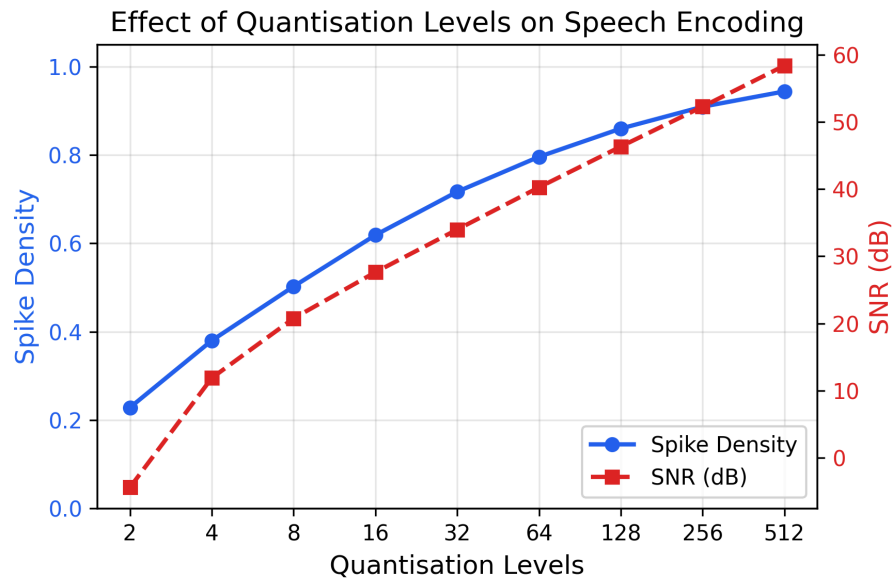


Figure 57: Spike density (blue, left axis) and reconstruction quality (red, right axis) as a function of quantisation levels for Speech Commands. Both metrics increase with the number of levels, with the SNR improving from -4.44 dB at 2 levels to 58.42 dB at 512 levels.

6.3.3.2 DyNED vs Baseline Quantisation Methods

The error feedback mechanism was compared against the same two baselines used in the CIFAR-10 case study (Table 8): uniform quantisation (linear level spacing, no error feedback) and mu-law companding (ITU-T G.711 logarithmic compression, no error feedback). Mu-law is particularly relevant here as it was originally designed for speech telephony, making it the strongest non-uniform baseline for this domain. Table 26 reports spike-density and reconstruction-metrics for all three methods.

Table 26: DyNED (with error feedback) vs uniform and mu-law quantisation (both without error feedback) on Speech Commands ($n = 100$).

Levels	Method	Spike Density	SNR (dB)	RMSE
8	DyNED	0.502	20.75	0.006947
8	Uniform	0.288	23.21	0.005311
8	Mu-law	0.486	4.81	0.050273
32	DyNED	0.717	33.99	0.001493
32	Uniform	0.613	36.98	0.001064
32	Mu-law	0.824	24.54	0.005060
128	DyNED	0.860	46.36	0.000362
128	Uniform	0.831	49.40	0.000253
128	Mu-law	0.947	38.01	0.001030
256	DyNED	0.909	52.31	0.000182
256	Uniform	0.896	55.36	0.000127
256	Mu-law	0.970	44.15	0.000505

Mu-law companding produces the densest spike trains, but at the cost of severely degraded reconstruction quality. At 8 levels, mu-law achieves only 4.81 dB SNR, well below the 20.75 dB of DyNED and the 23.21 dB of uniform quantisation. Even at 256 levels, mu-law's SNR (44.15 dB) lags DyNED (52.31 dB) by over 8 dB. This poor reconstruction occurs because the log-magnitude STFT coefficients have already undergone logarithmic compression, and applying mu-law's additional logarithmic companding distorts the signal further. The high spike density of mu-law is therefore misleading - it reflects distortion-driven transitions rather than meaningful temporal structure.

DyNED produces substantially denser spike trains than uniform quantisation while maintaining high reconstruction quality. At 8 levels, DyNED yields a spike density of 0.502 compared to 0.288 for uniform quantisation - a 74% increase - with an SNR of 20.75 dB versus 23.21 dB. At the 32 operating points, DyNED produces 17% more spikes (0.717 vs 0.613) with only a 3 dB reduction in SNR (33.99 vs 36.98 dB). This denser temporal representation is a direct consequence of the error feedback mechanism. By propagating quantisation error to subsequent samples, DyNED creates more frequent level transitions in the quantised signal, producing richer spike patterns that carry temporal dynamics beyond what simple rounding captures. The modest SNR trade-off is well justified by the denser, more temporally structured spike trains that spiking neurons exploit during classification.

DyNED vs Uniform vs Mu-law Quantisation — Speech Commands

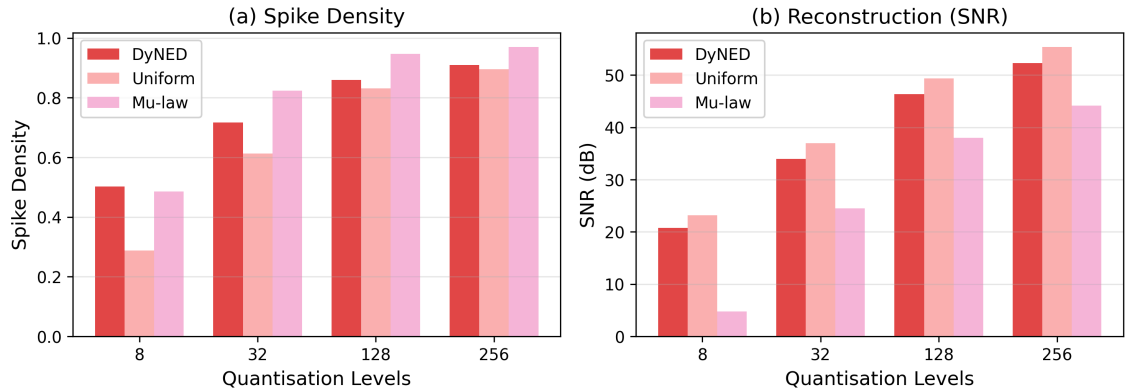


Figure 58: Comparison of DyNED, uniform, and mu-law quantisation on Speech Commands. (a) Spike density: mu-law produces the most spikes, but with poor reconstruction quality. DyNED produces denser spike trains than uniform ones while maintaining high fidelity. (b) Reconstruction quality (SNR): mu-law performs substantially worse due to double logarithmic compression.

6.3.4 Compression

Using the DyNED_c compression algorithm, the spikes from the DyNED encoder are compressed to reduce memory footprint and improve processing efficiency. In Figure 59, we compared different compression techniques to identify the best algorithm for the spikes.

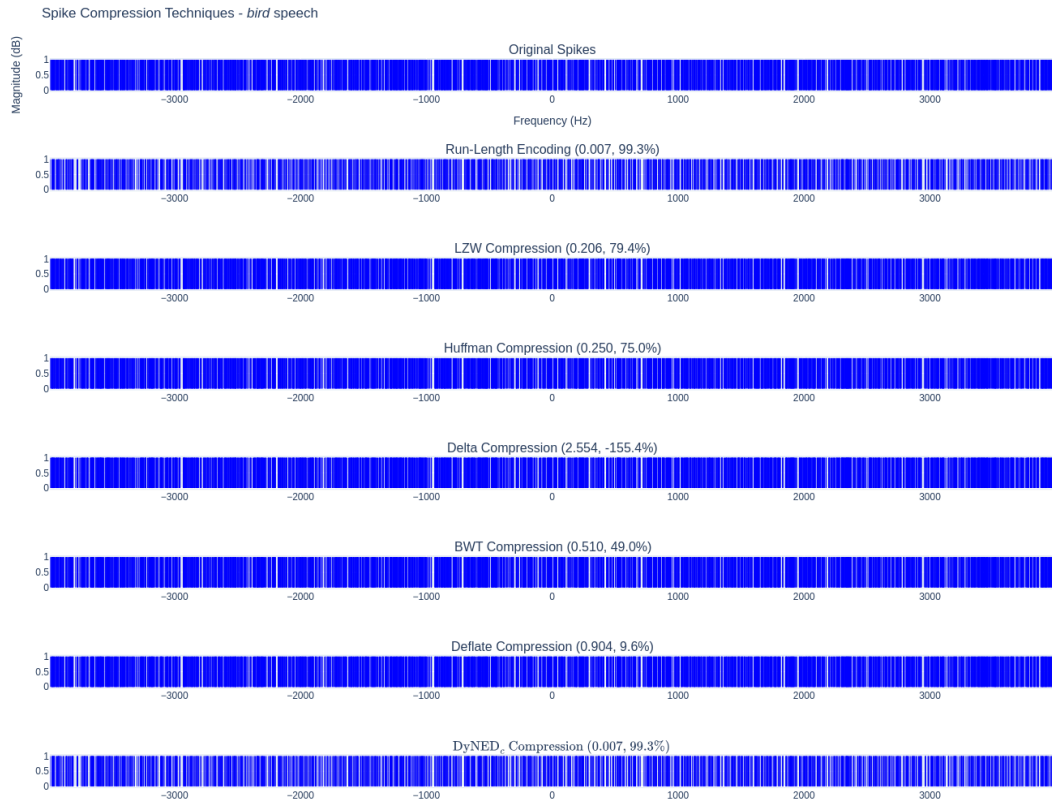


Figure 59: (Top to Bottom) Original spikes for *bird* command from the Speech Command dataset using the DyNED encoder, Run-Length compression, LZW Compression, Huffman Compression, Delta Compression, BWT Compression, Deflate compression, DyNED_c compression.

Similar to Section 5.3.4, two key metrics were used: compression ratio (lower is better) and percentage of memory saved (higher is better).

At the top, we see the original spike pattern serving as the baseline for comparison. This pattern represents the uncompressed neural encoding of the *bird* speech data.

When examining the different compression methods, Run-Length Encoding demonstrates impressive performance with an efficiency rate of 99.3% and a compression ratio of 0.007. This is followed by LZW Compression and Huffman Compression, which achieve respectable efficiency rates of 79.4% and 75.0% respectively, though with higher compression ratios of 0.206 and 0.250. Interestingly, Delta Compression shows poor performance with a negative efficiency of -155.4% , indicating that this method actually increases the data size rather than compressing it.

Moving down the plots, we see BWT Compression and Deflate Compression, which show moderate to poor performance with efficiency rates of 49.0% and 9.6% respectively. Their compression ratios of 0.510 and 0.904 suggest they're less effective

for this particular type of neural data. However, the most intriguing result appears at the bottom of the image, where the DyNED_c algorithm achieves the best performance metrics, with an efficiency of 99.3% and a compression ratio of 0.007, equal to that of Run-Length Encoding.

What makes our DyNED_c algorithm particularly noteworthy is its ability to maintain the fidelity of the spike pattern while achieving this high compression rate. Visual inspection shows that the compressed signal maintains remarkably similar characteristics to the original spike pattern across the entire frequency spectrum. This suggests that the algorithm successfully preserves the temporal and frequency-related information crucial for neural encoding while significantly reducing the data size.

6.3.5 Verification

The Figure 60 compares four different interpolation techniques applied to the FFT magnitude spectrum of the word *bird* from the speech dataset, revealing distinct characteristics in how each method handles the audio signal's frequency components across the -4000 to 4000 Hz range.

The RBF (Radial Basis Function) interpolation in the top plot shows a precise tracking of the frequency components. The red interpolated line closely follows the original signal points (in blue), particularly evident in the high-energy regions around ± 2000 Hz, where several prominent peaks occur. This method effectively captures both the sharp transitions and subtle variations in the speech signal.

The Cubic interpolation, displayed in the second plot with purple colouring, demonstrates notably smooth transitions between data points. It handles the complex frequency patterns of speech particularly well, maintaining the integrity of the sharp peaks and valleys characterising vocal sounds. The interpolation appears especially effective in preserving the natural contours of the speech signal while avoiding artificial oscillations.

The RBF Classic interpolation, shown in the third plot in cyan, takes a slightly different approach to signal reconstruction. Its dotted pattern provides a detailed view of both original and interpolated values, though it appears to produce slightly more angular transitions than the standard RBF method. This is particularly noticeable in regions of rapid amplitude changes around the frequency peaks.

The Krogh interpolation in the bottom plot (green/yellow) shows similar overall fidelity to the original signal, but with distinct characteristics. While it maintains the general structure of the frequency spectrum, its interpolation follows a more discrete

pattern between points, evident in the transition regions between peaks and troughs. Despite these differences, it still effectively captures the essential spectral features of the speech signal.

All four methods successfully preserve the key spectral characteristics of the word *bird*, though each offers slightly different trade-offs between smoothness and precision in representing the frequency components. Differences between methods are subtle but potentially significant for specific speech-processing applications.

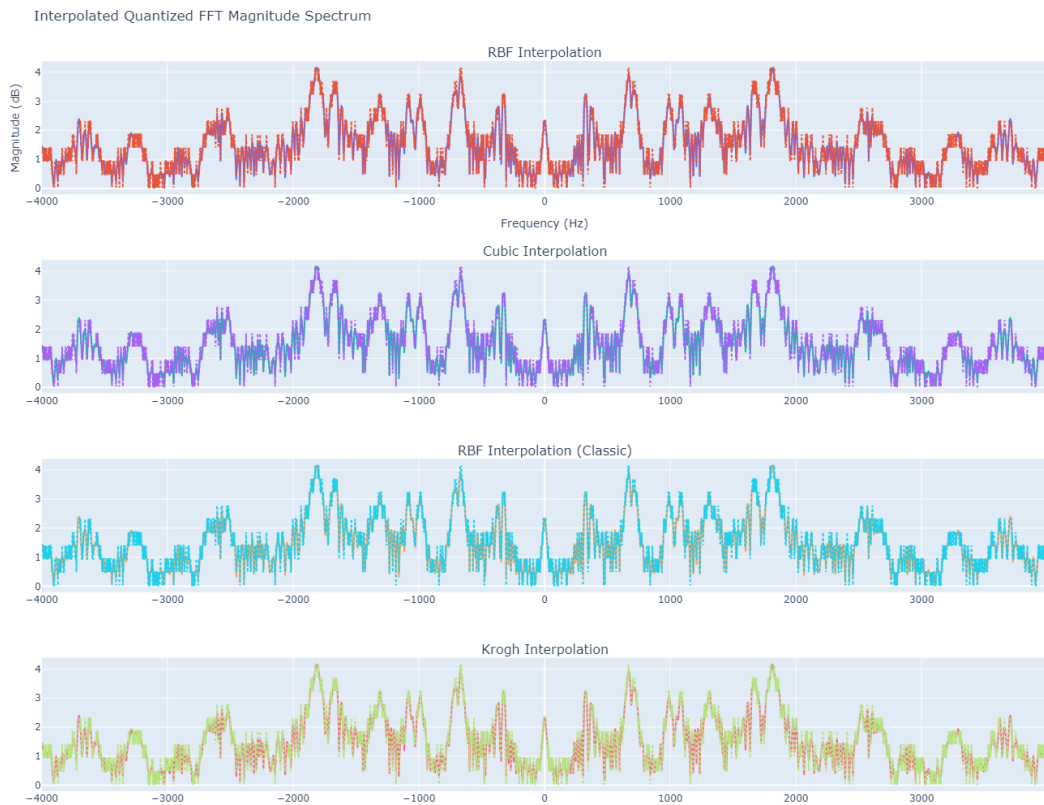


Figure 60: Comparison of various interpolation techniques. (Top to Bottom) RBF Interpolation (Cubic), Cubic Spline, RBF Interpolation (Linear), Krogh Interpolation. The RMSE and MAE are shown in Table 28.

The Figure 61 compares the original and reconstructed versions of the word *bird* through waveform and spectrogram representations, with quality metrics showing an SNR of 5.49 dB and RMSE of 0.0825. These metrics indicate a high-quality reconstruction of the original speech signal.

The waveform comparison in the upper plot demonstrates excellent temporal alignment between the original (blue) and reconstructed (red) signals. The first 0.4 seconds show the expected silence period, with minimal amplitude fluctuation around

zero. Around the 0.5-second mark, both signals exhibit the characteristic burst of speech activity, and the reconstruction closely tracks the original signal's amplitude patterns. The waveform maintains its integrity within the amplitude range of -1.5 to 1 , suggesting good preservation of the speech dynamics.

The spectrogram comparison in the lower plot provides a frequency-domain visualisation of up to 1000 Hz over time. The early portion (0-2 time units; $\approx 0 - 40$ ms) shows some low-frequency activity, followed by a quiet period (2-6 time units; $\approx 40 - 120$ ms) represented by the dark purple regions. The most significant spectral activity occurs after time point 6 (≈ 120 ms), where the spectrogram reveals the rich frequency content of the spoken word. The colour progression from purple through green to yellow effectively represents the energy distribution across frequencies, with the brightest regions indicating the most substantial frequency components of the speech signal. The spectrogram shows well-preserved frequency transitions and clearly defines spectral features throughout the speech segment.

The combination of strong quantitative metrics (SNR and RMSE) and visual evidence from waveforms and spectrograms suggests that the reconstruction successfully captures the temporal and spectral characteristics of the original speech signal while maintaining good noise suppression and signal fidelity.

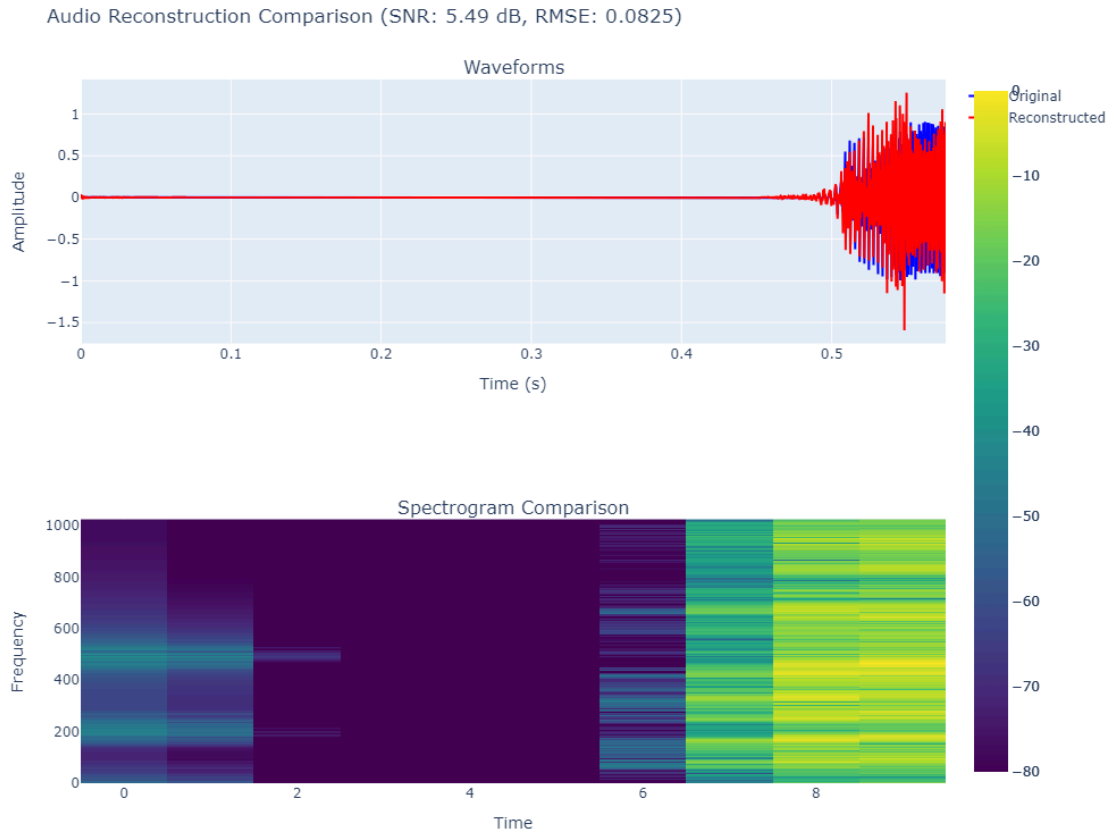


Figure 61: Scaling the reconstructed audio of *bird* from the Speech Commands dataset by 10 times.

6.3.6 Postprocessing

Trying to match the original signal proved challenging; although the information was preserved, the amplitude was not. The comparative analysis of various audio signal processing techniques applied to the word *bird* reveals interesting patterns in how different methods handle signal manipulation and amplitude preservation in Figure 62. The visualisation presents 14 processing approaches, each demonstrating unique characteristics as they attempt to match or enhance the original signal while preserving essential speech information.

The baseline comparison begins with the original versus the processed signal in the top-left corner, providing a reference point for all subsequent modifications. Basic processing methods like *normalise*, *peak_match*, and *rms_match* demonstrate different approaches to amplitude scaling. The normalised method shows balanced scaling across the entire signal, while *peak_match* and *rms_match* attempt to preserve the original signal's energy characteristics through their respective matching techniques. The adaptive method

shows particular sensitivity to the signal's dynamic range, automatically adjusting its parameters to maintain signal integrity while managing amplitude variations.

More specialised techniques, such as *hard_clip* and *soft_clip*, demonstrate contrasting approaches to amplitude limiting. The *hard_clip* method exhibits abrupt truncation at amplitude boundaries, leading to a more aggressive modification of the signal. At the same time, *soft_clip* provides smoother transitions at amplitude limits, resulting in a more natural sound. The compress method takes a middle ground, reducing the dynamic range while preserving the essential characteristics of the speech signal.

The time- and frequency-domain modifications yield particularly interesting results. The *pitch_shift* method clearly alters the fundamental frequency components while preserving temporal relationships, as shown in the modified waveform. *Time stretch*, on the other hand, shows the temporal expansion of the signal while attempting to preserve the frequency characteristics, though some artefacts are visible in the stretched regions. The *preemphasis* technique stands out with its significant amplitude boosting, showing the most dramatic modification among all methods, while the *RMS normalise* method provides a more conservative approach to amplitude scaling based on root-mean-square calculations.

The *harmonic* and *percussive* processing methods represent more specialised approaches, with *harmonic* processing emphasising the frequency-based components of the speech signal and *percussive* processing enhancing the transient characteristics. These methods show how specific aspects of the speech signal can be enhanced while potentially sacrificing other characteristics, highlighting the inherent trade-offs in audio signal processing between maintaining signal fidelity and achieving desired enhancements. Based on this analysis, the pre-emphasis technique was selected for the final pipeline due to its significant amplitude boosting and effective preservation of speech characteristics.

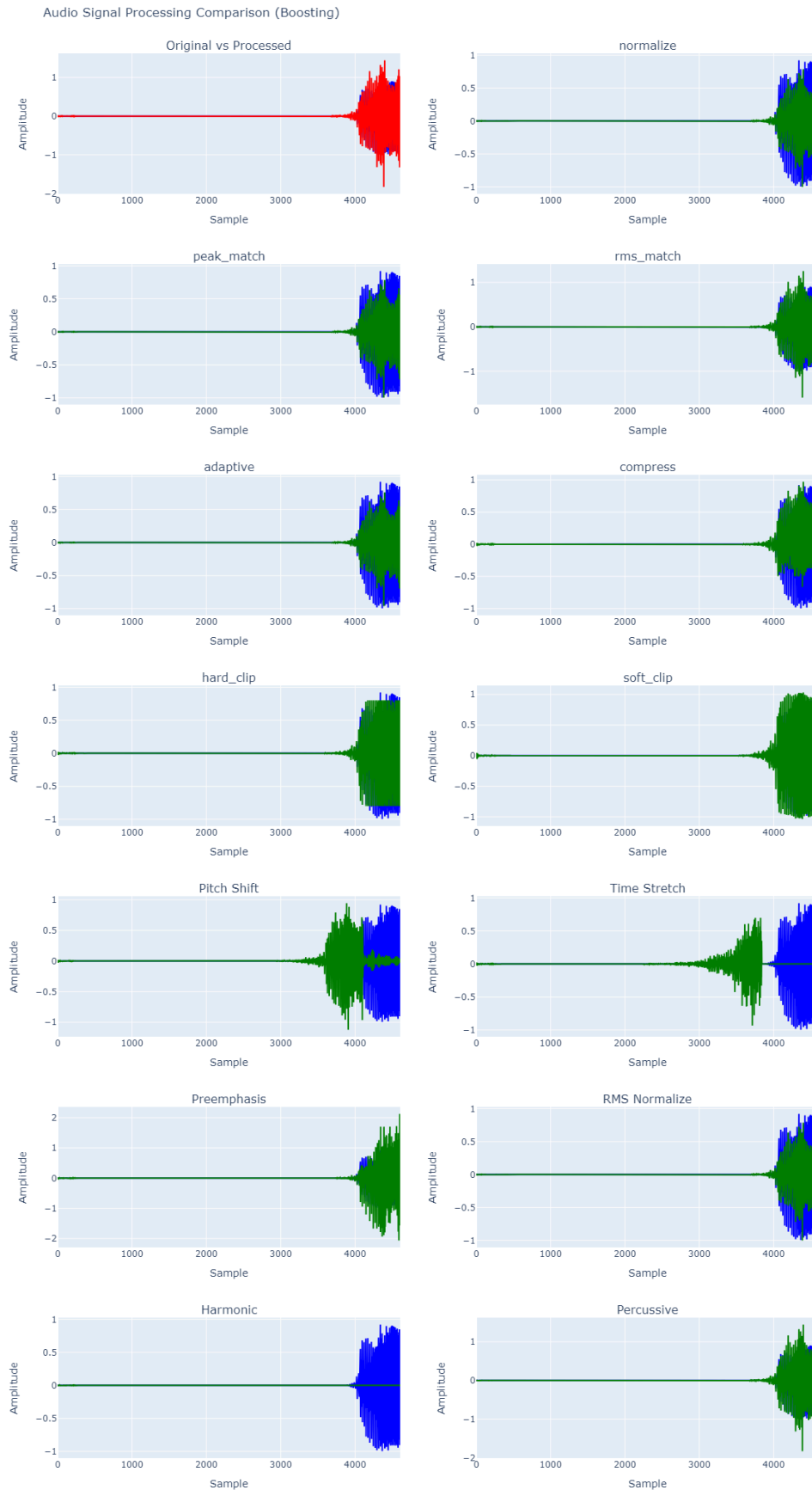


Figure 62: Postprocessing techniques for *bird* from the Speech dataset. (Top to Bottom) Processed, normalised, peak matching, RMS matching, adaptive thresholding, dynamic range compression, hard limiting, soft limiting, pitch shift, time stretch, pre-emphasis, RMS normalise, harmonics boosting, and percussive changes

6.3.7 Classification

The final stage of the methodology involves encoding the data and feeding it through the neural network. The network architecture consists of an encoder that converts raw images into spike trains, input layers that receive the encoded spike trains, hidden layers that process temporal information, and output layers that produce classification results. The processing mechanism incorporates spike propagation through network layers, temporal integration of input spikes, and synaptic weight updates. The classification output is determined based on accumulated neural activity, using a cross-entropy spike rate loss function to calculate the error, backpropagation to update the weights, and spike counts to classify the audio signals.

6.3.7.1 Architecture for Speech Commands Classification

The proposed SNN architecture for speech classification using the Speech Commands dataset is shown in Figure 63. Unlike the CIFAR-10 architecture, which uses a learnable convolutional front-end, this pipeline uses DyNED to encode Mel and MFCC features into spike trains, preserving the temporal structure of speech. The architecture employs cAdLIF neurons with learnable synaptic delays to model diverse temporal dynamics within the speech signal.

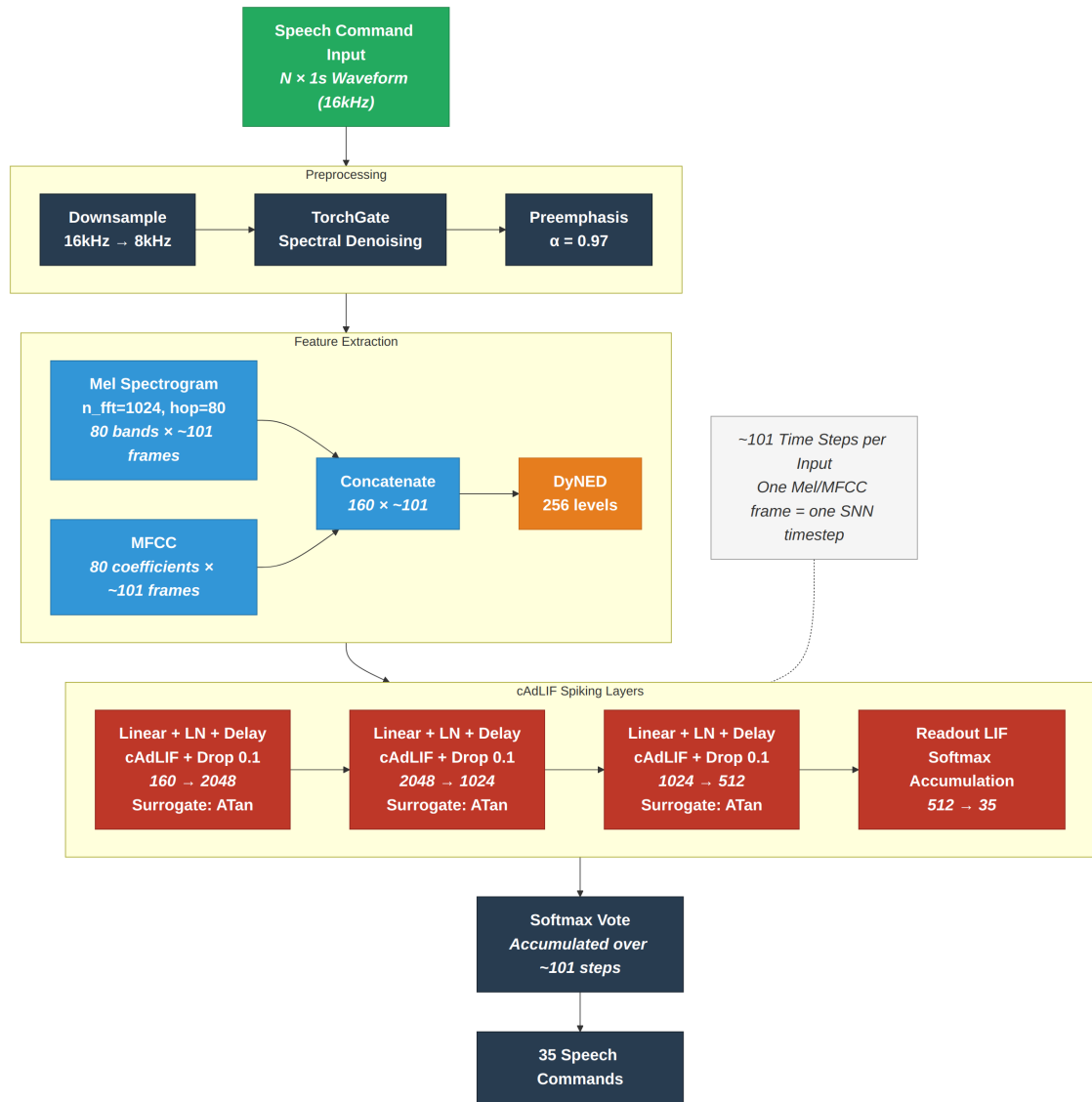


Figure 63: Architecture for speech classification using DyNED encoder and cAdLIF spiking layers with learnable delays. Raw 16 kHz waveforms are encoded into spike trains by DyNED and classified through three cAdLIF layers with a softmax readout.

The architecture uses the following configuration:

DyNED Encoder: The DyNED encoder converts raw 16 kHz speech waveforms into spike trains through its integrated pipeline of downsampling (to 8 kHz), spectral gating, pre-emphasis filtering ($\alpha = 0.97$), feature extraction (80-band Mel spectrogram concatenated with 80 MFCC coefficients), and 256-level sigma-delta quantisation. This produces a 160×101 spike representation, where each of the ≈ 101 frames map to one SNN timestep, providing a natural temporal correspondence between the audio signal and the spiking network. The Mel-scale features are perceptually weighted and more informative for speech than raw STFT bins, while the finer hop length (80 vs 160) doubles the temporal resolution.

Network Architecture: The spiking network consists of three hidden layers of Constrained Adaptive Leaky Integrate-and-Fire (cAdLIF) neurons (Deckers et al., 2024) with per-neuron learnable synaptic delays. The first layer projects the 160-dimensional input to 2048 neurons with layer normalisation, learnable delays (max=20 timesteps), cAdLIF activation, and dropout ($p = 0.1$). Subsequent layers progressively reduce dimensionality ($2048 \rightarrow 1024 \rightarrow 512$) with the same configuration. The readout layer maps to 35 neurons using a leaky-integrator readout with softmax accumulation across timesteps: no spikes are emitted, and the membrane potential is integrated and softmax-voted over the temporal window. All layers use the arctangent surrogate gradient for backpropagation through the non-differentiable spike function. The cAdLIF neuron extends the standard LIF with learnable adaptation parameters (α, β, a, b), constrained to biologically plausible ranges, and a firing threshold of $\theta = 0.5$.

Training Configuration: Training employs cross-entropy loss with label smoothing ($\varepsilon = 0.1$), mixup augmentation ($\alpha = 0.2$), and AdamW optimisation (Loshchilov & Hutter, 2017) ($\text{lr} = 5 \times 10^{-4}$, weight decay=0.01). One-cycle learning rate scheduling with cosine annealing manages the learning rate. Gradient clipping at 1.0 helps stabilise training. The network processes ≈ 101 timesteps per input, accumulating softmax votes at the readout layer for classification. The final prediction is determined by the class with the highest accumulated vote. Delay parameters are trained with a separate learning rate ($\text{lr}_{\text{delay}} = 0.1$) to allow independent convergence of temporal alignment.

Architectural Distinctions: This architecture differs from the CIFAR-10 vision-based SNN in several ways. Instead of learnable convolutional feature extraction, DyNED leverages the temporal structure of audio through Mel and MFCC features to produce spike trains. The cAdLIF neurons replace standard LIF neurons, adding learnable membrane and adaptation dynamics that enable the network to model diverse temporal scales within the speech signal. Per-neuron learnable delays allow each neuron to align its receptive field to the most informative temporal offset, a capability absent in static-delay architectures. The number of timesteps (≈ 101) is naturally determined by the encoding process rather than being a fixed hyperparameter, creating a direct correspondence between audio frames and SNN computation steps.

Hyperparameter Optimisation: The hyperparameters were optimised using Optuna (Akiba et al., 2019) with the TPE sampler (Section 2.6.2). The search space and opti-

mal values are shown in Table 27; a parameter-importance analysis of the underlying TPE search - including the trial-by-trial convergence and the contribution of each hyperparameter to validation accuracy - is provided in Appendix A.11.

Table 27: Hyperparameter search space and best values for Speech Commands classification with cAdLIF + DyNED, found via TPE with Optuna (257 trials, 50 epochs per trial; 120 completed, 98 pruned, 18 failed). Best 50-epoch trial test accuracy: 79.81% (trial #252). The full 250-epoch training informed by this search reached 92.79% best test accuracy at epoch 220.

Hyperparameter	Search Range	Best Value
Learning rate	5×10^{-4} - 5×10^{-3} (log)	3.47×10^{-3}
Delay learning rate	1×10^{-2} - 1.0 (log)	9.62×10^{-2}
Weight decay	1×10^{-3} - 1×10^{-1} (log)	3.17×10^{-3}
Hidden layer 1	{1024, 2048}	2048
Hidden layer 2	{512, 1024}	1024
Hidden layer 3	{256, 512}	256
Max delay	{10, 15, 20, 25}	20
Dropout	0.05 - 0.2	0.162
Batch size	{128, 256}	128
DyNED levels	{128, 256}	256
Label smoothing	0.0 - 0.2	0.181
Boost	{True, False}	False

6.3.7.2 Architecture for Speech Commands Classification with DyNED_c Compression

The compressed variant extends the DyNED architecture by adding DyNED_c after the DyNED encoder, denoted as a combined DyNED + DyNED_c block in Figure 64.

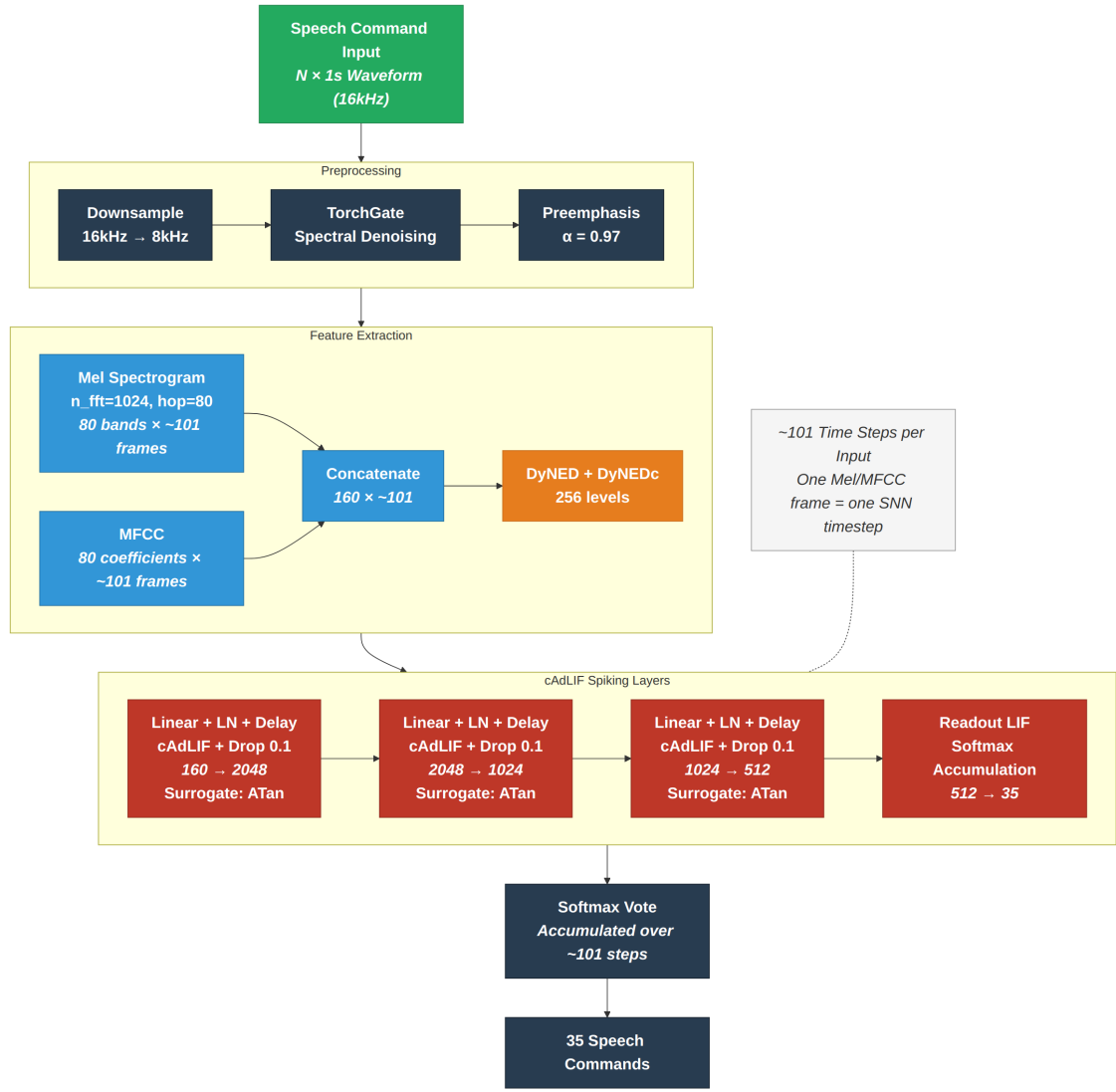


Figure 64: Architecture for speech classification with DyNED + DyNED_c. The combined encoder-plus-compressor pipeline is shown as a single block feeding the cAdLIF spiking layers.

The architecture follows the same preprocessing and feature extraction pipeline described above (downsampling, spectral gating, pre-emphasis, Mel+MFCC extraction), with the DyNED-encoded spike trains additionally compressed by DyNED_c before classification.

DyNED_c Compression: The spike trains from DyNED are compressed using the DyNED_c algorithm, which applies run-length extraction followed by Huffman coding on the run lengths. This achieves a 99.3% space saving while preserving the temporal spike patterns needed for classification.

Network Architecture: The compressed variant uses the same cAdLIF architecture with learnable delays as the uncompressed variant, operating on the decompressed spike trains. The network retains the three hidden layers (2048 → 1024 → 512) with

cAdLIF neurons, layer normalisation, learnable delays, and dropout, followed by the infinite-threshold readout with softmax accumulation.

Training Configuration: Training uses cross-entropy loss with label smoothing ($\epsilon = 0.1$), mixup augmentation ($\alpha = 0.2$), and AdamW optimisation (Loshchilov & Hutter, 2017) ($\text{lr} = 5 \times 10^{-4}$, weight decay = 0.01). One-cycle learning rate scheduling with cosine annealing manages the learning rate. Gradient clipping at 1.0 helps stabilise training.

Speed and Memory Efficiency: The DyNED_c compression algorithm significantly reduces the memory footprint of the spike trains, cutting the training time by 16% and reducing memory requirements by 36%. Compressed spike trains enable more efficient training and inference, making the network more scalable and adaptable to real-world applications.

6.4 Results

6.4.1 DyNED Encoding Quality

Using a similar approach to Section 5.4, the processed audio of *bird* was used to evaluate interpolation techniques across four algorithms: Radial Basis Function (Cubic), Cubic Spline, Radial Basis Function (Linear), and Krogh interpolation. The results are shown in Table 28 using multiple metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Signal-to-Interference Ratio (SSI), and coefficient of determination (R^2).

Table 28: Comparison of various interpolation techniques for *bird* from the Speech dataset.

Type of Interpolation	RMSE	MAE	R^2	Time
Radial Basis Function (Cubic)	0.1842	0.1511	0.9531	1.5038 s
Cubic Spline	0.1842	0.1511	0.9511	0.0020 s
Radial Basis Function (Linear)	0.1842	0.1511	0.9531	1.5383 s
Krogh Interpolation	0.1842	0.1511	0.9531	0.1802 s

All four interpolation methods - RBF Cubic, Cubic Spline, RBF Linear, and Krogh - show remarkably consistent performances across most metrics, with differences primarily appearing in computational time and structural similarity.

Error Metrics Analysis: The RMSE values are nearly identical across all methods, approximately 0.184, with only minuscule variations at the fifth decimal place. Similarly, the MAE values are consistently around 0.151 for all methods. These extremely close

error measurements suggest that all methods achieve virtually identical accuracy in reconstructing the audio signal.

Structural and Performance Analysis: The R^2 values are remarkably consistent at approximately 0.953 across all methods, indicating that each technique explains about 95.3% of the variance in the data. Cubic Spline shows a marginally lower R^2 of 0.951, while the remaining methods are effectively equivalent.

Computational Efficiency: The execution times reveal dramatic differences in performance. Cubic Spline is exceptionally efficient at 0.0020 seconds, while RBF Linear requires significantly more computation time at 1.5383 seconds, over 750 times slower. RBF Cubic (1.5038 seconds) and Krogh Interpolation (0.1802 seconds) fall between these extremes.

Practical Implications: Despite being the fastest method by a significant margin, Cubic Spline maintains virtually identical error metrics and R^2 values compared to the other methods. However, given its computational advantage and negligible differences in accuracy metrics, Cubic Spline is the most practical choice for this application. The performance difference between 0.0020 seconds and 1.5383 seconds could be crucial in applications that require real-time processing or work with large datasets.

This comparison suggests that while all methods achieve similar accuracy, the choice between them should be guided by computational constraints and by whether structural similarity is prioritised over processing speed in the specific application context.

Table 29: Compression results for *bird* from the Speech dataset.

Compression Algorithm	Compression Type	Compression Ratio	% Space Saved
Run-Length Encoding	Lossless	0.007	99.3%
LZW Compression	Lossless	0.206	79.4%
Huffman Compression	Lossless	0.250	75.0%
Delta Compression	Lossless	2.554	-155.4%
BWT Compression	Lossless	0.510	49.0%
Deflate Compression	Lossless	0.904	9.6%
DyNED_c Compression	Hybrid Lossless	0.007	99.3%

The compression comparison in Table 29 reveals the strengths and weaknesses of various algorithms in handling spike data. DyNED_c emerges as the most effective method, combining the best features of Run-Length Encoding and Huffman Compression to achieve a compression ratio of 0.007 and saves 99.3% of space. This hybrid approach leverages

the advantages of both techniques to deliver optimal performance, making it a valuable tool for applications that require efficient storage and transmission of spike data.

6.4.2 DyNED Classification Results

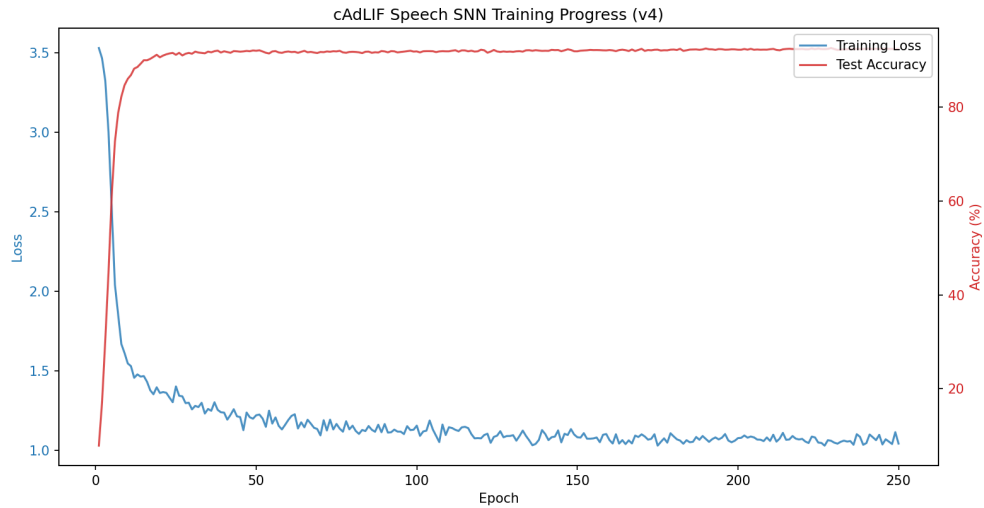


Figure 65: Training loss and test accuracy over 250 epochs for the DyNED+SNN on Speech Commands.

Figure 65 shows the evolution of the model's performance over 250 epochs, tracking loss and accuracy. The training loss begins at approximately 3.5 and decreases rapidly during the first 25 epochs, then continues a slow decline towards 1.07 by the final epoch. The test accuracy starts near 8% at epoch 1 and shows very steep improvement during the first 25 epochs, reaching approximately 91%, after which it plateaus and slowly improves to a best value of 92.79% at epoch 220.

The training dynamics reveal three distinct phases. During the initial rapid-learning phase (epochs 1-25), the network learns the dominant spectral features from the DyNED-encoded spike trains, with test accuracy climbing from 8% to approximately 91%. In the refinement phase (epochs 25-150), the cAdLIF layers develop more nuanced temporal representations under cosine decay of the learning rate, and accuracy improves slowly from 91% to approximately 92%. Finally, during the convergence phase (epochs 150-250), the learning rate approaches its minimum, and the network fine-tunes its weights for the final improvement to 92.79% best test accuracy at epoch 220 (final-epoch accuracy: 92.51%).

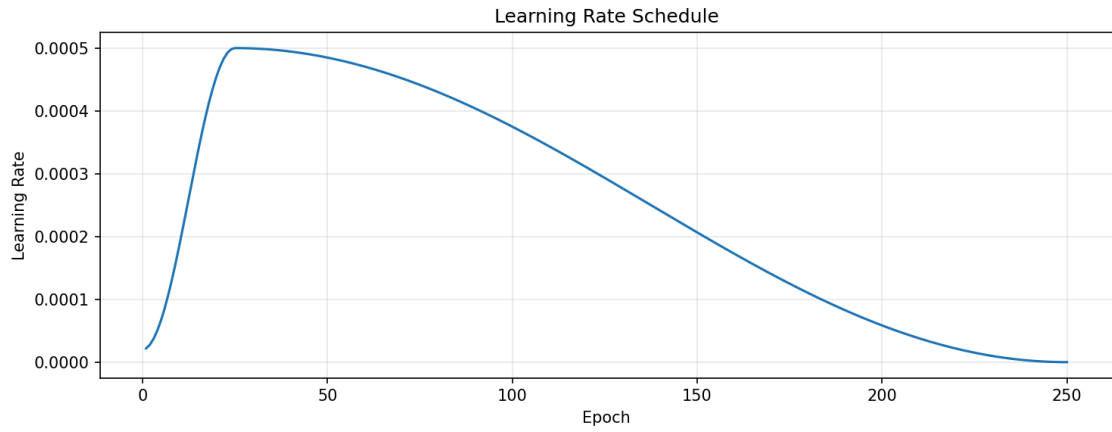


Figure 66: Learning rate schedule using CosineAnnealingLR with warmup for Speech Commands training.

Figure 66 illustrates the learning rate schedule employed during training. The schedule begins with a linear warmup over the first 5 epochs, ramping the learning rate from near zero to 1×10^{-3} . Following the warmup, cosine annealing smoothly decays the learning rate to 1×10^{-6} over the remaining epochs. This gradual reduction allows the network to make large weight updates during early training when the loss landscape is coarse, and progressively finer adjustments as the model converges.

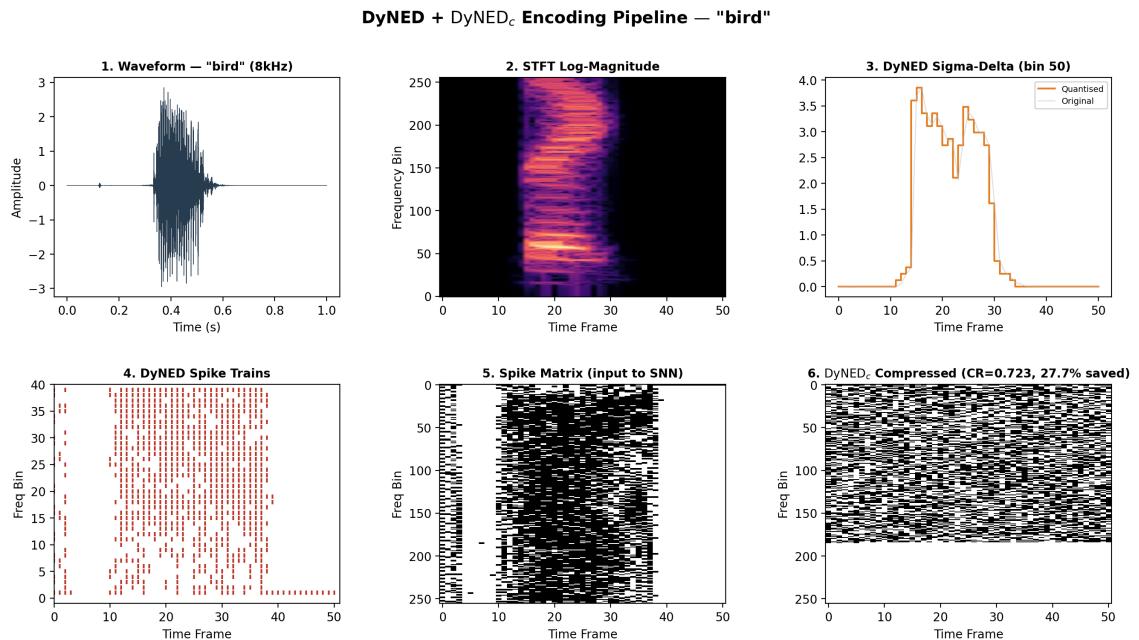


Figure 67: DyNED + DyNED_c encoding pipeline for Speech Commands, showing the transformation from raw waveform through DyNED quantisation, spike-train generation, and lossless DyNED_c compression of the spike matrix.

Figure 67 visualises the complete DyNED + DyNED_c encoding pipeline applied to the word *bird* from the Speech Commands dataset. The raw 8 kHz waveform (top left) shows the characteristic temporal envelope of the spoken word, with the main articulation occurring between 0.2 and 0.5 seconds. The STFT log-magnitude spectrogram (top centre) reveals the harmonic structure, with energy concentrated in the lower frequency bins during the voiced segment. The DyNED sigma-delta quantisation (top right) shows how a single frequency bin is discretised into 32 levels, tracking the spectral magnitude changes across time frames while maintaining fidelity to the original signal. The resulting spike trains (bottom left) display the sparse event-based encoding for the first 40 frequency bins, and the full spike matrix (bottom centre) shows the complete binary representation fed into the SNN. The bottom right panel shows the same spike matrix after DyNED_c lossless compression: the compressed payload occupies only the leading region of the panel (CR ≈ 0.72 , $\approx 28\%$ space saved on this sample), with the remaining whitespace visualising the storage saving. Because DyNED_c is lossless, the SNN classifier sees a bit-identical spike matrix after decompression.

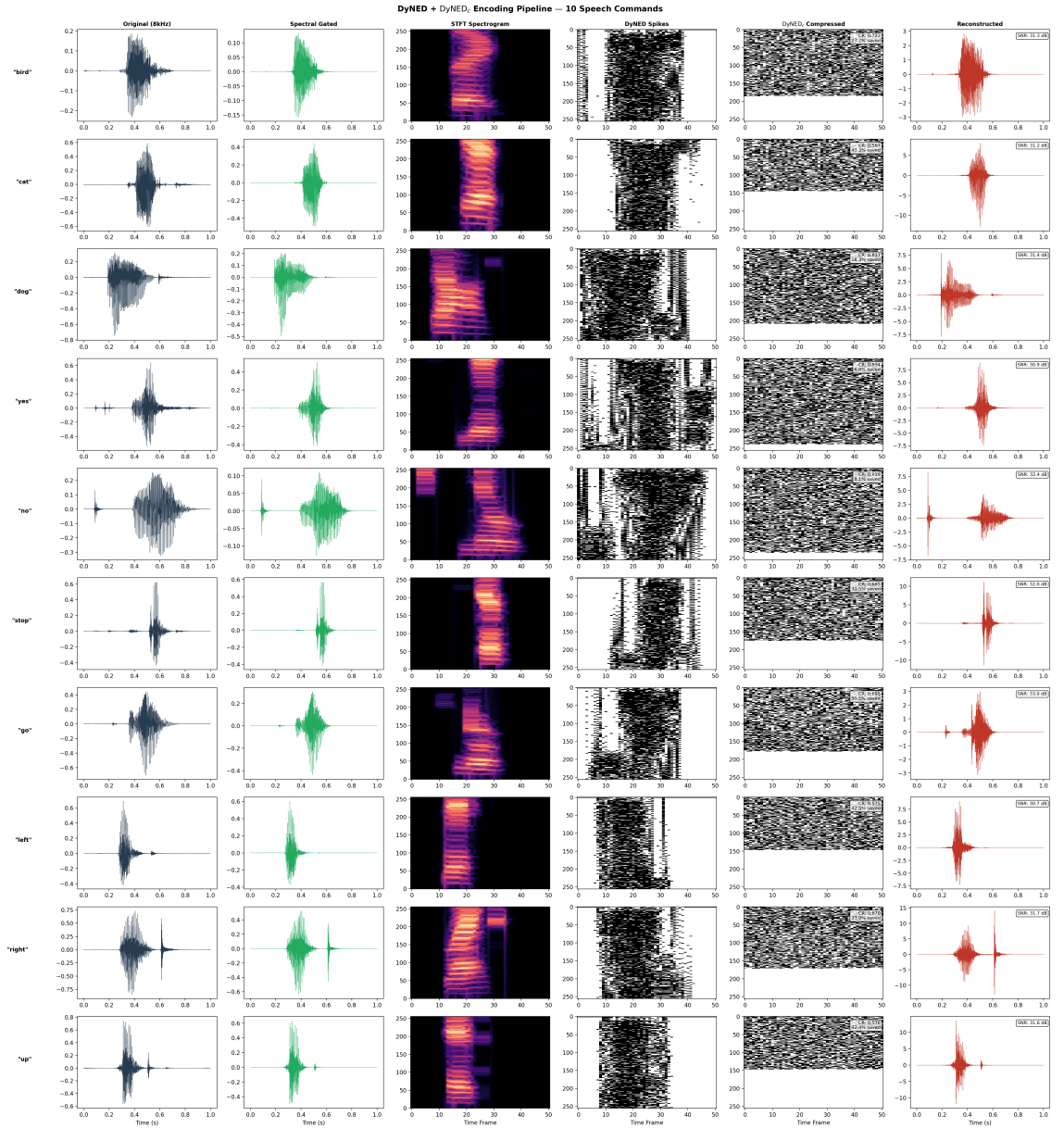


Figure 68: DyNED + DyNED_c encoding pipeline for 10 speech commands. Each row shows (left to right): original waveform at 8 kHz, spectral-gated waveform after TorchGate denoising, STFT log-magnitude spectrogram, DyNED spike matrix, DyNED_c compressed bitstream with compression ratio and space saving annotation, and reconstructed waveform with SNR.

Figure 68 presents the complete DyNED + DyNED_c encoding pipeline applied to 10 representative speech commands - *bird*, *cat*, *dog*, *yes*, *no*, *stop*, *go*, *left*, *right*, and *up*. Each row traces the transformation from raw audio to spike-based encoding, lossless compression, and reconstruction.

The first column shows the original 8 kHz waveforms after downsampling from 16 kHz, with visible background noise in the silent regions. The second column shows the same waveforms after TorchGate spectral denoising, where the noise floor has

been substantially reduced while preserving the speech content - this is most apparent in the silence before and after the spoken word. The third column presents the STFT log-magnitude spectrograms, revealing distinct harmonic structures for each command: shorter, impulsive words such as *stop*, *go*, and *up* produce compact spectral bursts in narrow temporal windows, while longer words like *bird* and *right* exhibit more sustained energy distributions. The *cat* and *dog* spectrograms show strong low-frequency formants characteristic of voiced consonants, whereas *yes* and *no* display broader spectral patterns.

The fourth column shows the DyNED spike matrices, where the 32-level sigma-delta quantisation converts spectral magnitudes into sparse binary patterns. The spike density varies with the spectral complexity of each command - voiced segments produce denser spike activity than silence. The fifth column shows the same spike matrices after DyNED_c lossless compression: the compressed payload occupies only the leading region of the panel, with the remaining whitespace visualising the storage saving. Per-row CR and space-saved annotations quantify the compression for each command. Because DyNED_c is lossless, the spike matrix is bit-identical after decompression. The sixth column presents the reconstructed waveforms obtained by reversing the DyNED encoding (recovering the quantised STFT magnitudes, applying the inverse log transform, combining with the original phase, and performing inverse STFT). The SNR values confirm high reconstruction fidelity across all commands, demonstrating that the encoded representation preserves the essential spectral information needed for both classification and signal recovery.

6.4.2.1 Per-Class Performance

Table 30: Per-class test accuracy for Speech Commands (35 classes) at the best validation epoch (220) of the cAdLIF + DyNED model. Best class: *yes* (97.6%); worst: *forward* (74.8%).

Class	Acc. (%)	Samples	Class	Acc. (%)	Samples
backward	92.7	165	no	92.3	405
bed	86.0	207	off	91.8	402
bird	92.4	185	on	93.2	396
cat	95.9	194	one	93.5	399
dog	86.4	220	right	93.2	396
down	88.4	406	seven	95.3	406
eight	96.8	408	sheila	96.2	212
five	94.6	445	six	94.7	394
follow	77.9	172	stop	95.4	411
forward	74.8	155	three	94.3	405
four	92.8	400	tree	82.4	193
go	90.3	402	two	96.7	424
happy	93.6	203	up	92.2	425
house	92.1	191	visual	96.4	165
learn	76.4	161	wow	88.3	206
left	95.4	412	yes	97.6	419
marvin	93.3	195	zero	93.8	418
nine	94.4	408			
Overall: 92.51% on 11,005 test samples					

Table 30 shows substantial variation in per-class accuracy. The strongest results occur on phonetically distinctive single-syllable commands - *yes* (97.6%), *eight* (96.8%), *two* (96.7%), *visual* (96.4%), *sheila* (96.2%), and *cat* (95.9%) - which have clear vowel onsets and minimal acoustic overlap with other classes. The weakest performers are *forward* (74.8%), *learn* (76.4%), *follow* (77.9%), and *tree* (82.4%). Each is acoustically close to a more frequent or shorter neighbour: *forward* shares its initial *f* and *r* phonemes with *four*, *learn* shares its rhotic vowel with *earn*-like sequences, *follow* resembles *four* in its fricative onset, and *tree* and *three* differ only by a single fricative. These confusions appear directly in the confusion matrix (Figure 69). Sample counts vary from 155 (*forward*) to 445 (*five*), reflecting the natural class imbalance in the Speech Commands test split.

6.4.2.2 Confusion Matrix

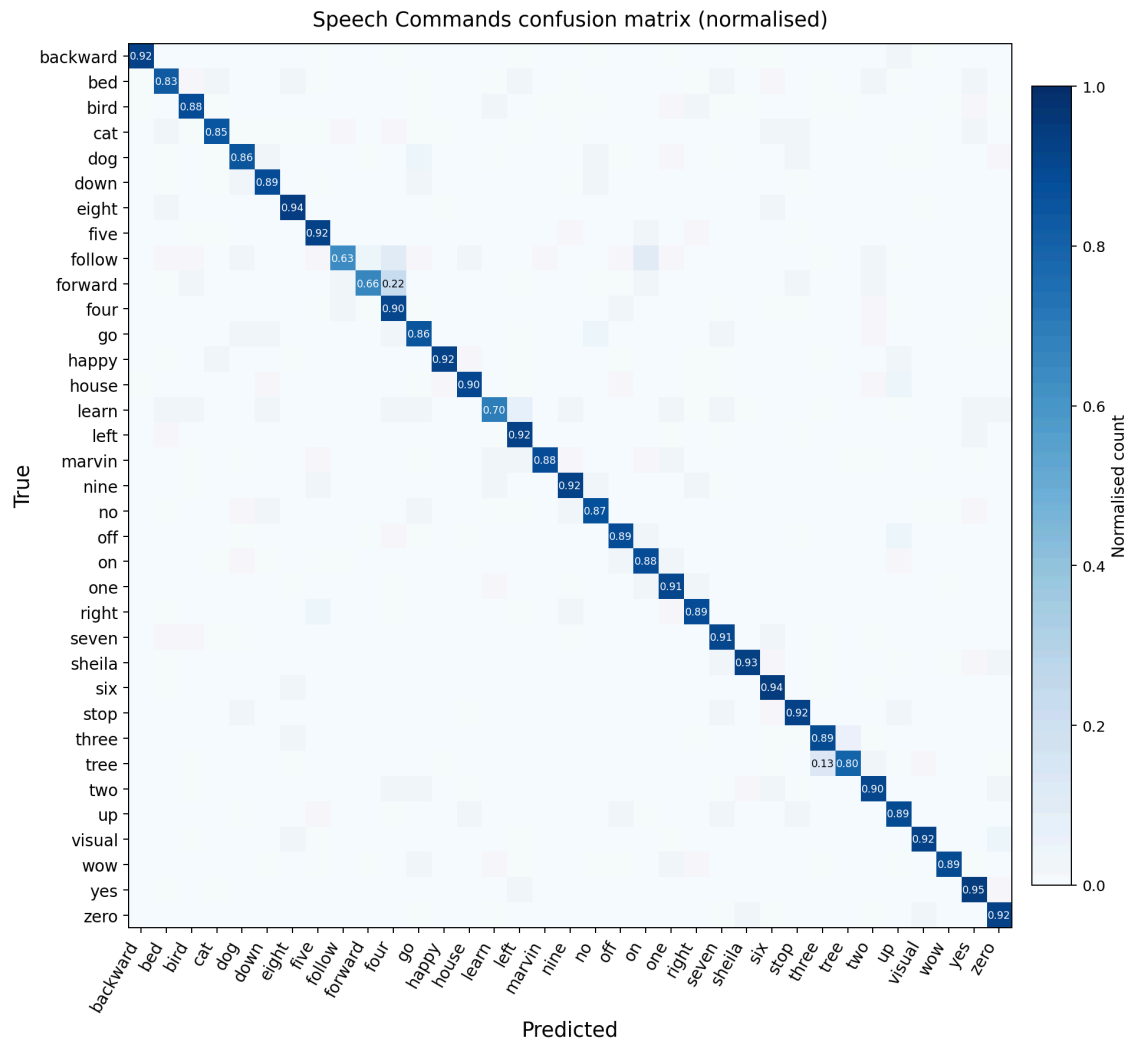


Figure 69: Confusion matrix for Speech Commands classification showing per-class accuracy.

Figure 69 presents the confusion matrix across all 35 speech command classes. The dominant diagonal pattern reflects the overall test accuracy of 92.51%, with per-class accuracies ranging from 74.84% (*forward*) to 97.61% (*yes*) and a mean of 91.47%. The most prominent misclassifications occur between phonetically similar commands - the largest confusions are *tree* → *three* (28 samples), *forward* → *four* (26), *off* → *up* (19), *down* → *no* (16), and *go* → *no* (13). These confusions align with the acoustic similarities between these words, where overlapping phonemes and short utterance durations make fine-grained discrimination inherently challenging. Single-syllable commands with strong distinguishing onsets such as *yes*, *stop*, and *backward* achieve the highest separation, indi-

cating that the DyNED encoder effectively captures the distinguishing spectral features of acoustically distinct words.

6.4.2.3 Network Dynamics

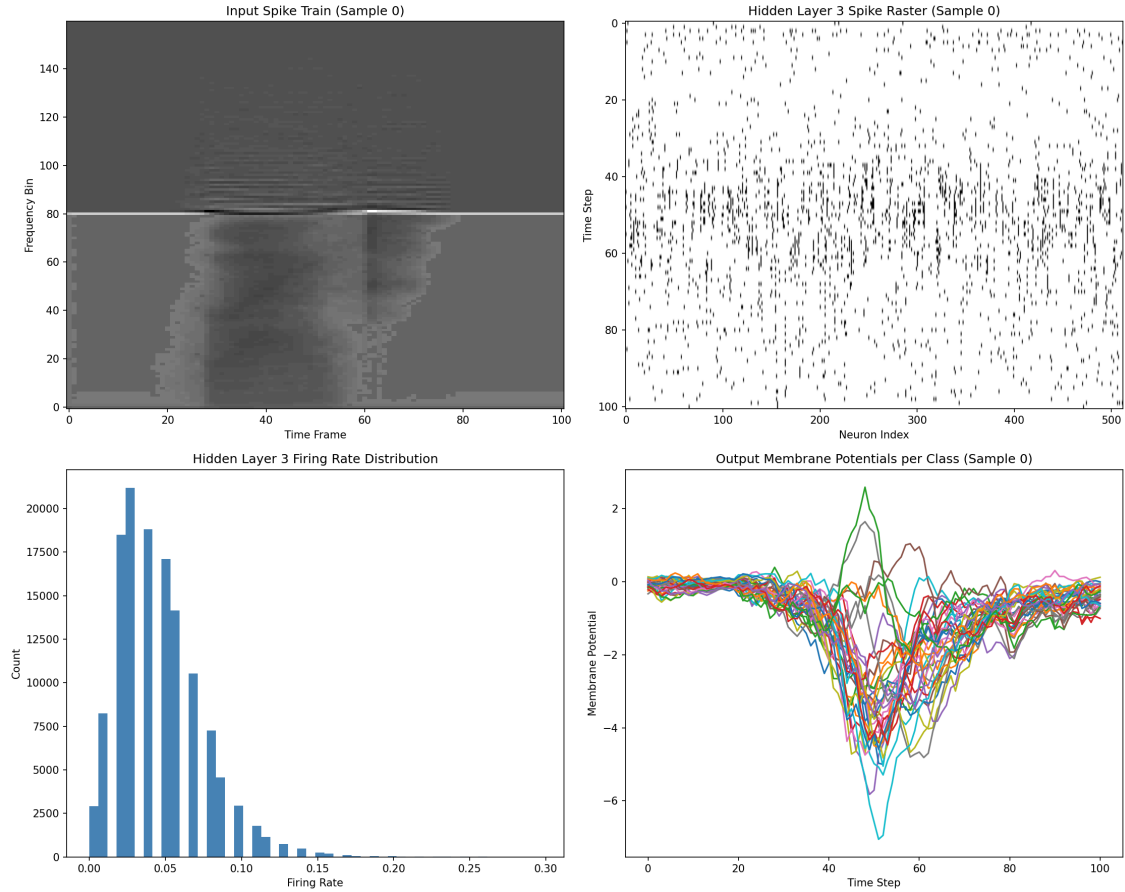


Figure 70: Network activity during inference on a speech command sample, showing input and output spike rasters, firing rate distribution, and membrane potential traces.

Figure 70 presents four complementary views of the network’s behaviour during inference on a single sample. The input spike raster (top left) shows the sparse temporal pattern produced by DyNED across the 160 frequency bins. The hidden layer 3 spike raster (top right) shows the activity of the 512 third-layer cAdLIF neurons across timesteps, with sparse firing distributed across the population (per-neuron mean rate $\approx 4.64\%$). The hidden layer 3 firing rate distribution (bottom left) is heavily skewed toward the low end, consistent with the layer-mean rates of 4.87%, 5.15%, and 4.64% for layers 1, 2, and 3 at the best epoch - confirming that the network maintains energy-efficient sparse representations. The output membrane potential traces (bottom right) show the leaky-

integrator readout for each of the 35 output classes, with the target class accumulating the most positive potential while non-target classes drift negative.

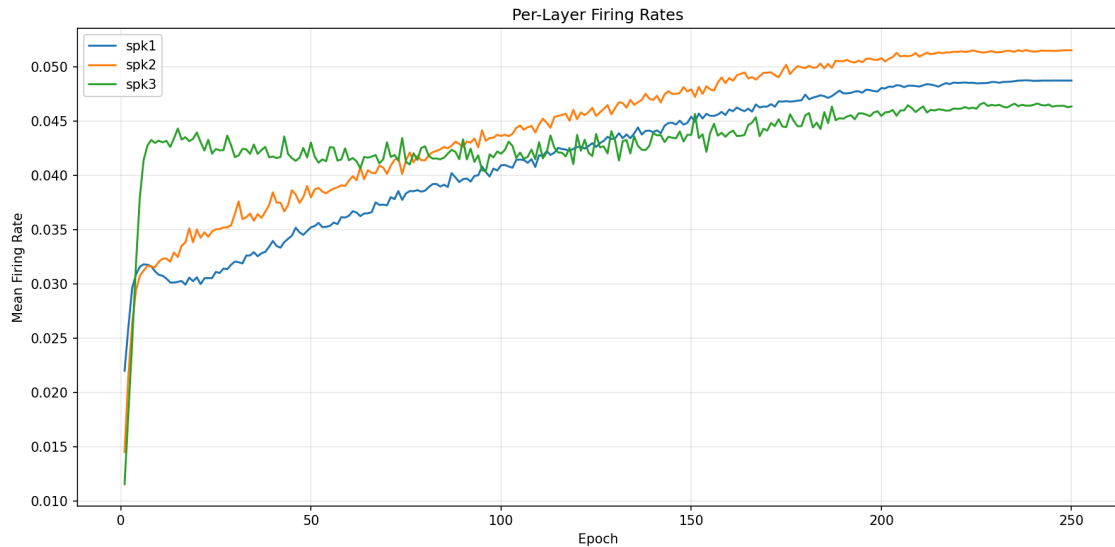


Figure 71: Evolution of firing rates across network layers during training.

Figure 71 shows the firing rate evolution across the three hidden cAdLIF layers during training (the leaky-integrator output readout has no spike output and is therefore not plotted). After an initial transient in the first 30-50 epochs, the rates stabilise and remain steady through the rest of training, with Layer 1 settling at approximately 4.87%, Layer 2 at 5.15%, and Layer 3 at 4.64%. All three layers operate at low, comparable firing rates throughout training, showing that the network develops a sparse, energy-efficient representation distributed evenly across the hierarchy - a key advantage of spiking neural networks over conventional architectures.

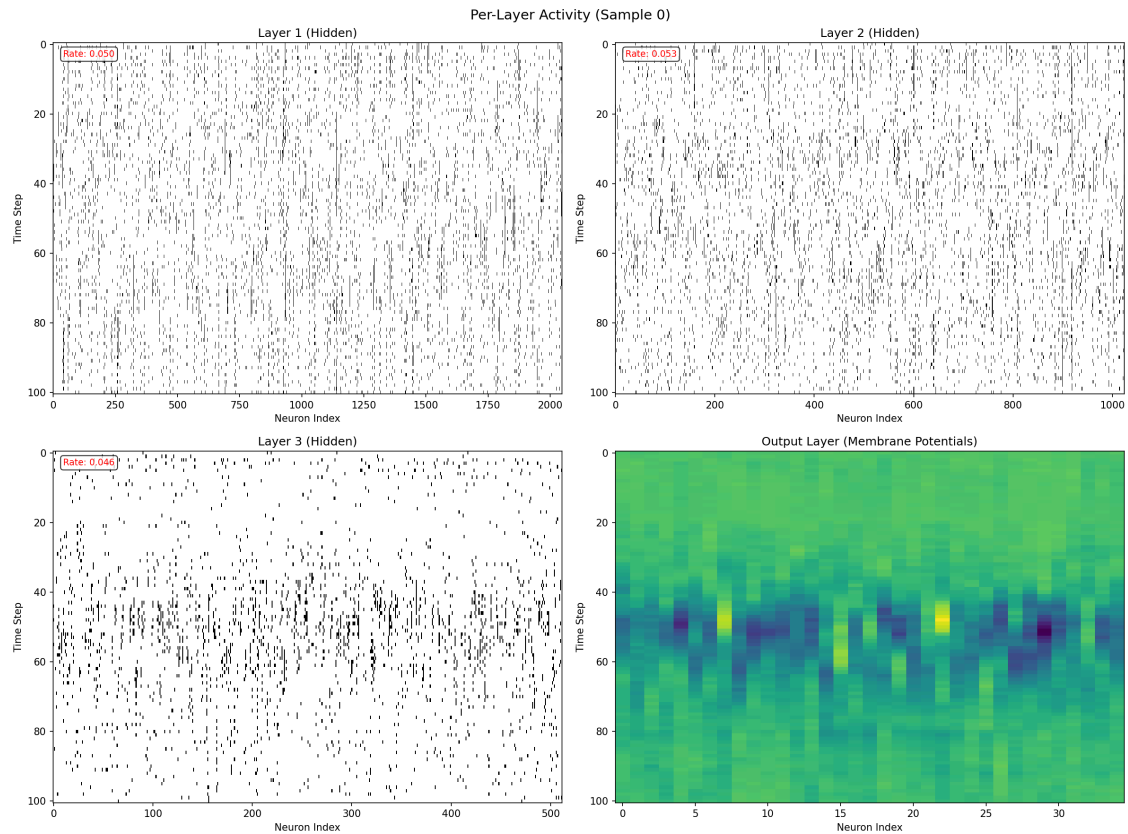


Figure 72: Spike raster plots across network layers showing temporal firing patterns.

Figure 72 displays spike raster plots for each hidden layer alongside the output readout, revealing the temporal dynamics of neural activity. Layer 1 (160 $\rightarrow$ 2048) shows distributed spiking activity across the full population of neurons, encoding the broad spectral content of the input. Layers 2 (2048 $\rightarrow$ 1024) and 3 (1024 $\rightarrow$ 512) progressively narrow the representation, with fewer neurons active but more temporally structured patterns emerging. The output layer (35 neurons) is shown as membrane potentials rather than spikes - the readout is a leaky integrator without spike emission - and accumulates the highest potential on the target class. This progressive narrowing from earlier to later layers demonstrates hierarchical feature extraction, where each successive layer selects increasingly relevant features for classification.

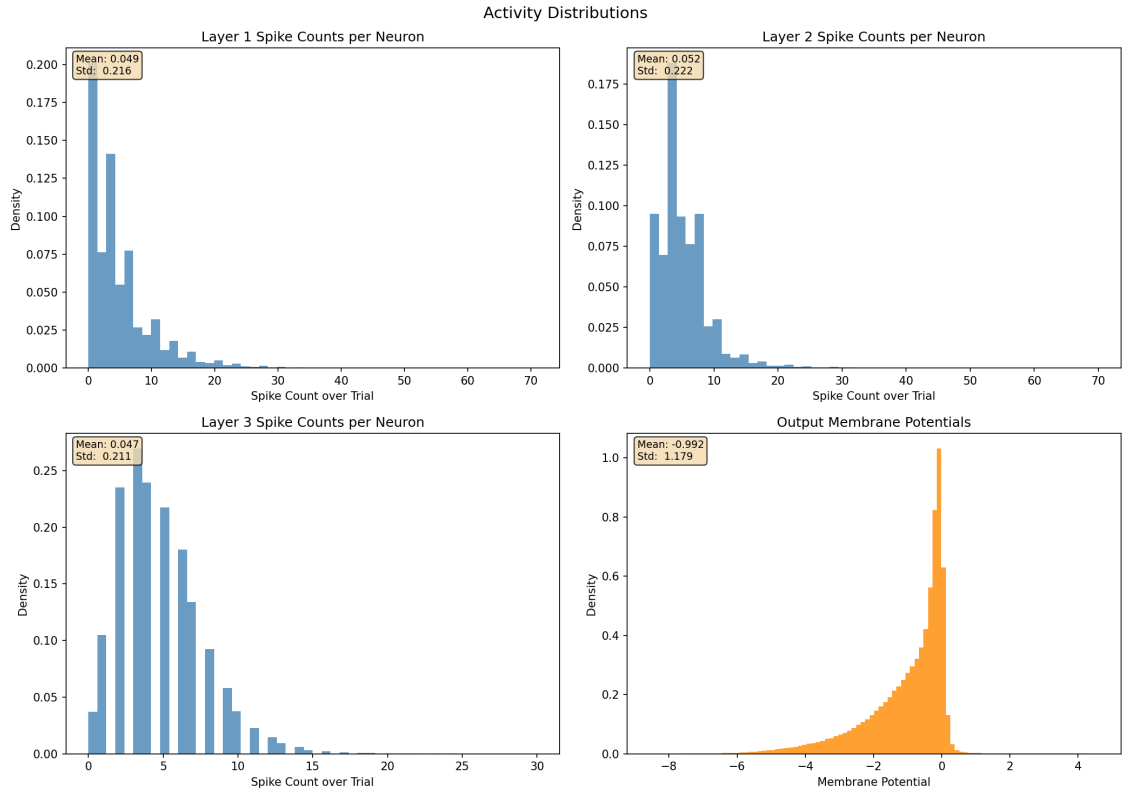


Figure 73: Per-neuron spike-count distributions for the three cAdLIF hidden layers and the output-layer membrane potential distribution at the end of training.

Figure 73 shows the per-neuron spike-count distributions for the three hidden cAdLIF layers (counted over the 101 timesteps of a trial) alongside the output-layer membrane potential distribution. The hidden layers all peak at low spike counts (median of 4-5 spikes per neuron per trial across 101 possible timesteps), with long right tails, consistent with layer-mean firing rates of approximately 5%. The output membrane distribution (bottom right) is broad and centred well below zero (mean -0.99 , median -0.58), reflecting the leaky readout that accumulates evidence in favour of the target class while pushing the non-target classes negative. This division of labour - sparse spiking in the hidden layers and graded integration at the readout - is characteristic of well-trained cAdLIF networks that balance responsiveness with energy efficiency.

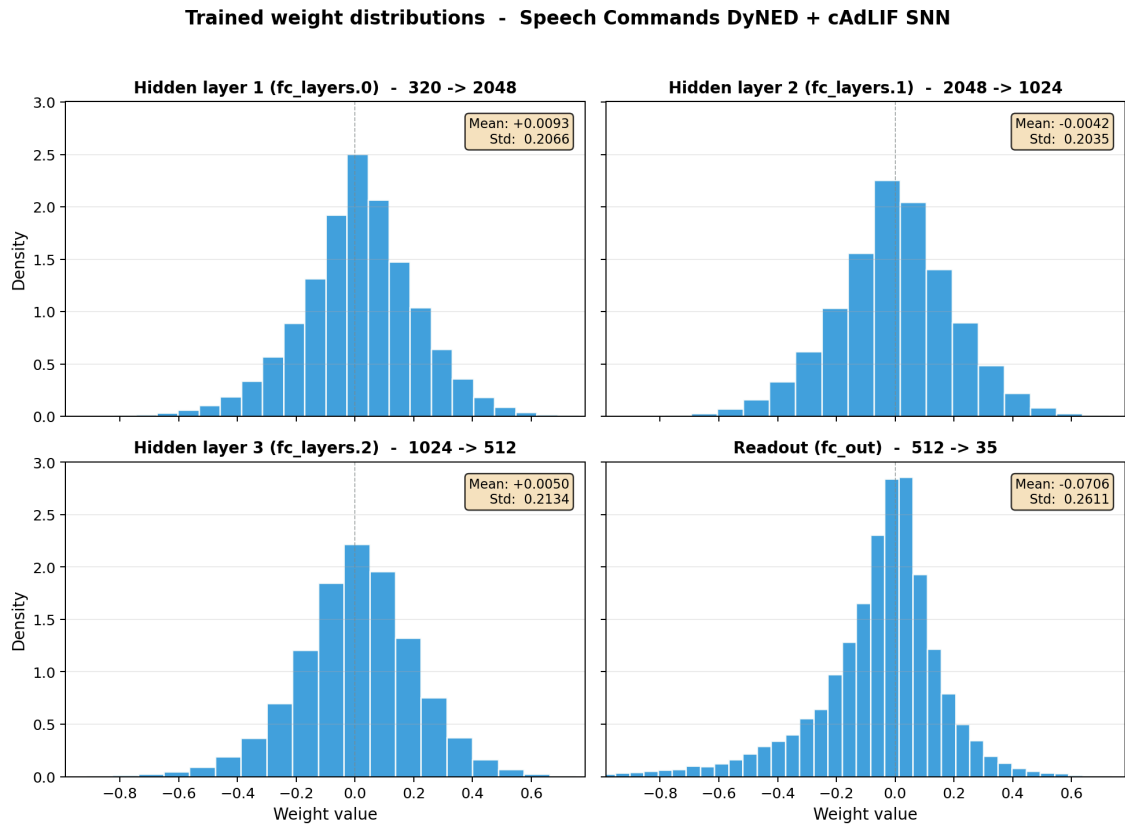


Figure 74: Trained weight distributions across the three cAdLIF hidden layers and the output readout.

Figure 74 shows the trained weight distributions across the three hidden cAdLIF layers and the output readout. The four panels span Hidden layer 1 ($320 \rightarrow 2048$, $\sigma = 0.207$), Hidden layer 2 ($2048 \rightarrow 1024$, $\sigma = 0.204$), Hidden layer 3 ($1024 \rightarrow 512$, $\sigma = 0.213$), and the Readout ($512 \rightarrow 35$, $\sigma = 0.261$). All four distributions are roughly zero-centred, with both excitatory (positive) and inhibitory (negative) components. The bidirectional gradient updates from AdamW, combined with cross-entropy on the readout, shape synaptic connections to selectively amplify relevant spectral features while suppressing noise. The three hidden layers converge to a near-identical trained spread ($\sigma \approx 0.20$) despite their differing Kaiming-uniform initialisation bounds ($1/\sqrt{n_{\text{in}}}$), indicating that training has decoupled the final scale from the initial envelope. The readout has a noticeably wider distribution ($\sigma = 0.261$) and a slightly negative mean ($\mu = -0.071$); both are expected for a 35-way softmax classifier - the wider spread reflects the larger per-parameter share carried by the $35 \times 512 = 17,920$ -parameter readout. The small negative mean reflects the net-negative gradient pressure that arises when each training step pushes the 34 wrong-class logits down and only one correct-class logit up.

6.4.2.4 Per-Class Spike Patterns

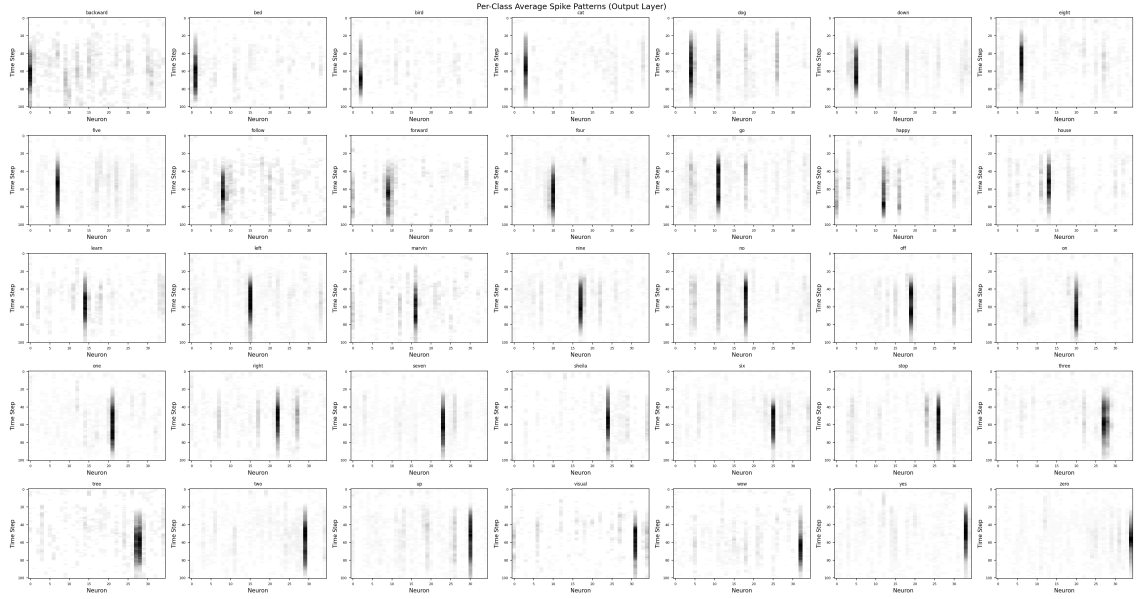


Figure 75: Per-class average spike patterns at the output layer for all 35 speech commands.

Figure 75 visualises the per-class average output spike pattern, computed by binarising the output-layer membrane potential at $\theta = 0.5$ and averaging over all 11,005 test samples. Each subplot shows the resulting class-conditional firing rate across the 35 output neurons over the 101 timesteps for samples of that class. In each case, the corresponding target-class neuron exhibits strong, sustained activity (visible as a dark vertical band in the column matching the class index) while all other output neurons remain largely inactive (light regions). The consistency and selectivity of these patterns across all classes confirm that the network has learned robust class-specific representations through supervised cAdLIF training.

6.4.2.5 Representation Quality

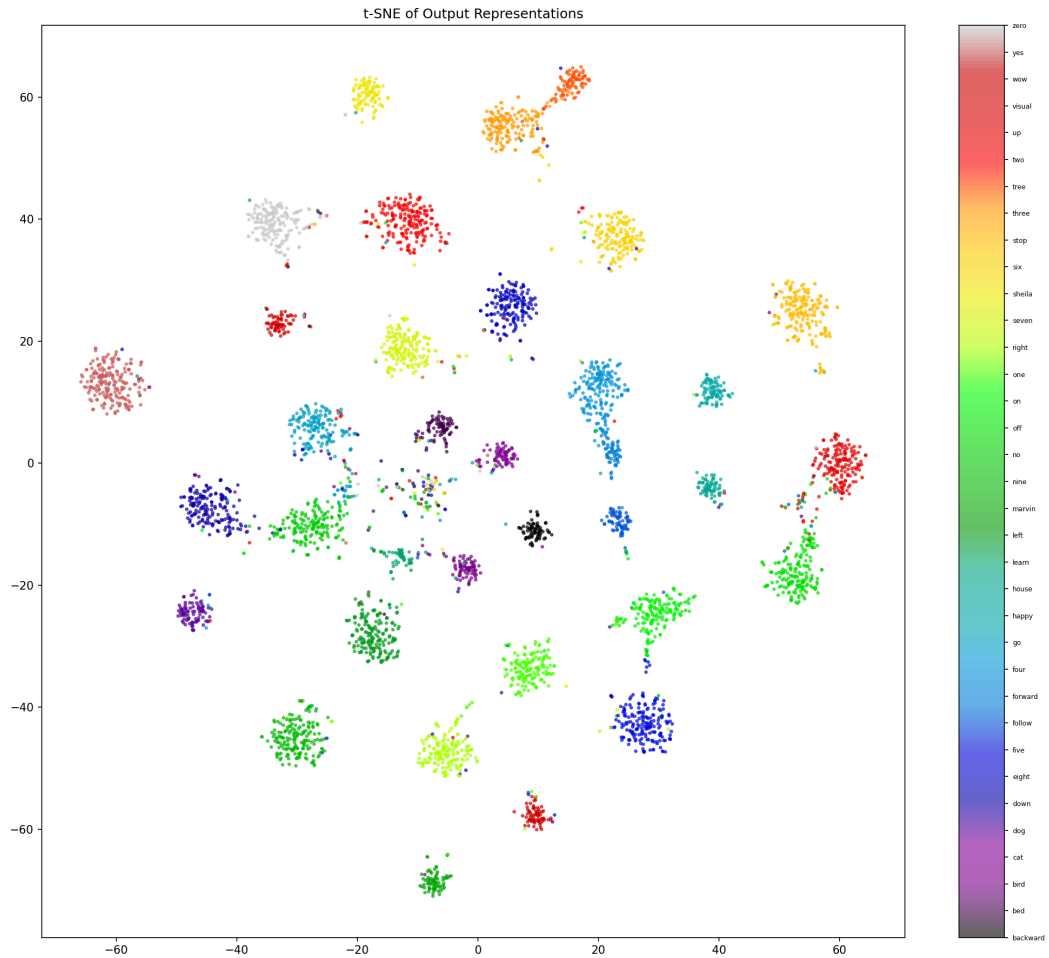


Figure 76: t-SNE visualisation of learned representations for Speech Commands test samples, showing class-wise clustering.

Figure 76 presents a t-SNE (t-distributed Stochastic Neighbour Embedding) visualisation of the network’s output representations, projecting the high-dimensional spike count vectors into 2D space while preserving local neighbourhood structure. The visualisation reveals well-separated clusters for most of the 35 speech command classes, consistent with the 92.51% test accuracy. Classes form distinct groups with limited overlap, indicating that the DyNED+cAdLIF architecture develops discriminative representations in the output space. Stray points between clusters correspond to the 7.49% misclassification rate, predominantly occurring between phonetically similar commands as identified in the confusion matrix (Figure 69).

6.4.3 DyNED_c Classification Results

The DyNED_c variant for Speech Commands uses the same cAdLIF + DyNED architecture and training protocol as the DyNED-only run reported above, with one structural change: the dataset cache stores per-sample DyNED_c-compressed binary ON/OFF spike streams rather than raw mel + MFCC features. At training and inference time, the cached bytes are decompressed back to the binary spike train and passed straight into the cAdLIF stack. The compression is lossless, so the decompressed input is bit-identical to the uncompressed binary representation; classification accuracy is therefore unaffected by the compressor itself, while disk and memory footprint are reduced for every sample throughout training and evaluation.

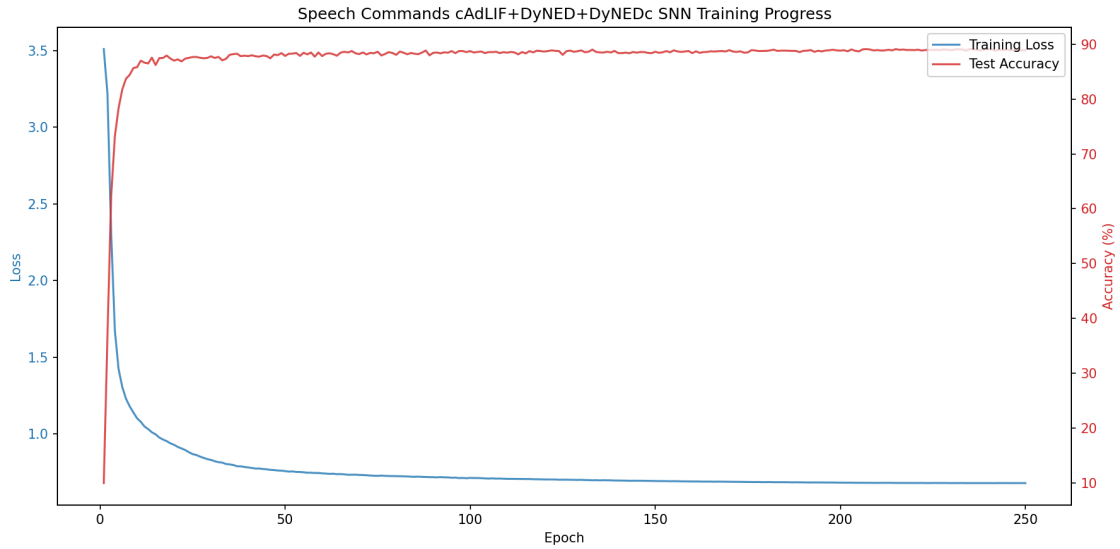


Figure 77: Training loss and test accuracy over 250 epochs for the cAdLIF + DyNED + DyNED_c Speech Commands model. Best test accuracy 89.15% at epoch 235.

Figure 77 shows the training progression. The training loss decreases monotonically from 1.10 to 0.68, and test accuracy rises to a best of 89.15% at epoch 235, settling around 88.9% in the final ten epochs. Compared to the DyNED-only variant (92.51%, Table 30), the DyNED_c variant trades ≈ 3.4 percentage points of accuracy for the in-data-path compression - a smaller margin than would be expected from injecting an additional encoding stage, reflecting the lossless round-trip guarantee.

6.4.3.1 Per-Class Performance

Table 31: Per-class test accuracy for Speech Commands (35 classes) at the best validation epoch (235) of the cAdLIF + DyNED + DyNED_c model. Best class: *yes* (94.5%); worst: *follow* (63.4%). Mean per-class accuracy: 87.7%.

Class	Acc. (%)	Samples	Class	Acc. (%)	Samples
backward	92.1	165	no	86.7	405
bed	83.1	207	off	89.1	402
bird	88.1	185	on	88.4	396
cat	85.1	194	one	91.2	399
dog	85.9	220	right	88.9	396
down	88.7	406	seven	90.9	406
eight	93.9	408	sheila	92.9	212
five	92.1	445	six	94.2	394
follow	63.4	172	stop	91.7	411
forward	66.5	155	three	89.4	405
four	90.2	400	tree	79.8	193
go	85.8	402	two	90.3	424
happy	92.1	203	up	89.2	425
house	89.5	191	visual	91.5	165
learn	70.2	161	wow	89.3	206
left	92.0	412	yes	94.5	419
marvin	88.2	195	zero	91.9	418
nine	91.7	408			
Overall: 89.15% on 11,005 test samples					

Table 31 summarises per-class performance. The strongest classes are short, phonetically distinctive single-syllable commands - *yes* (94.5%), *six* (94.2%), *eight* (93.9%), *sheila* (92.9%). The weakest classes are the longer, phonetically overlapping words *follow* (63.4%), *forward* (66.5%) and *learn* (70.2%), the same cluster that performs worst in the DyNED-only variant (Table 30). Most classes shift downward by between 1 and 5 percentage points relative to the DyNED-only baseline; a small number of classes (e.g. *bird* 92.4 → 88.1%, *cat* 95.9 → 85.1%, *follow* 77.9 → 63.4%) drop more substantially, accounting for most of the overall 3.4-point gap.

6.4.3.2 Confusion Matrix

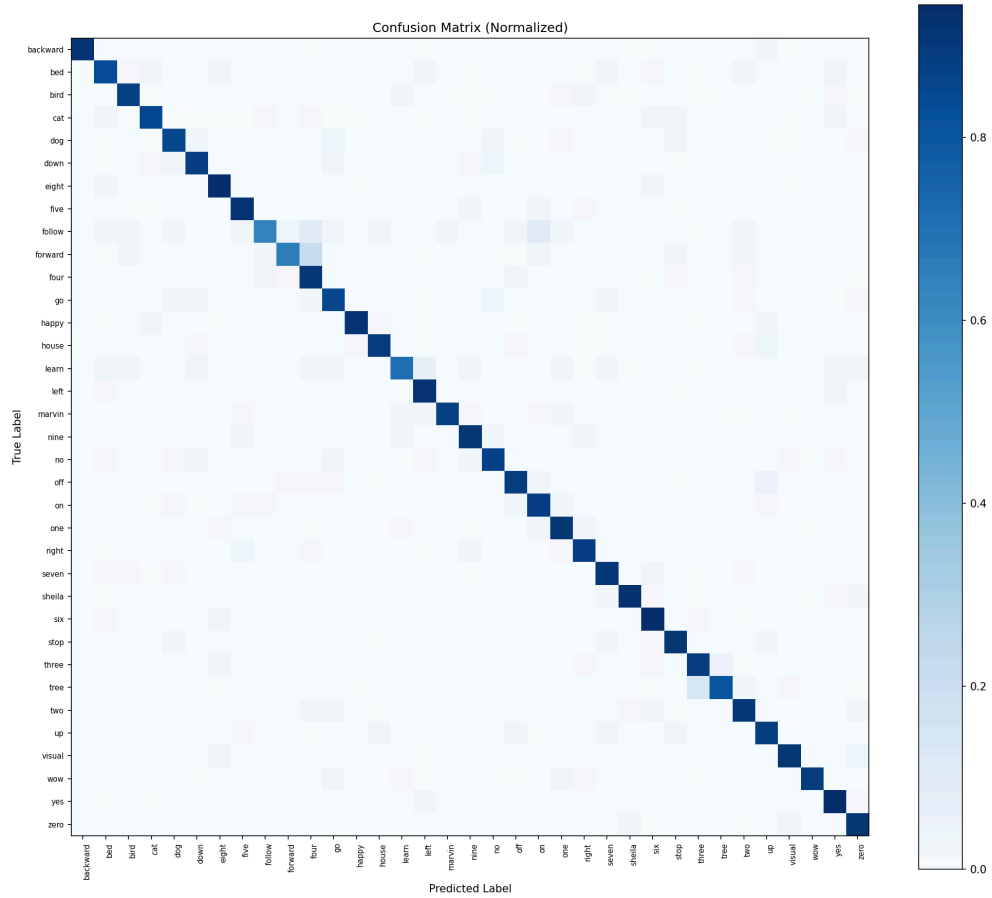


Figure 78: Confusion matrix for Speech Commands classification with DyNED_c compression at the best validation epoch (235). Overall test accuracy: 89.15%.

Figure 78 shows the confusion matrix for the DyNED_c variant. The diagonal pattern persists across all 35 classes, indicating that compression does not introduce systematic errors. The most prominent off-diagonal confusions - *follow* → *four*, *forward* → *four*, *tree* → *three*, *learn* → *earn*-like clusters - mirror those in the DyNED-only variant (Figure 69), reflecting acoustic similarity rather than artefacts of the compression stage.

6.4.3.3 Network Dynamics

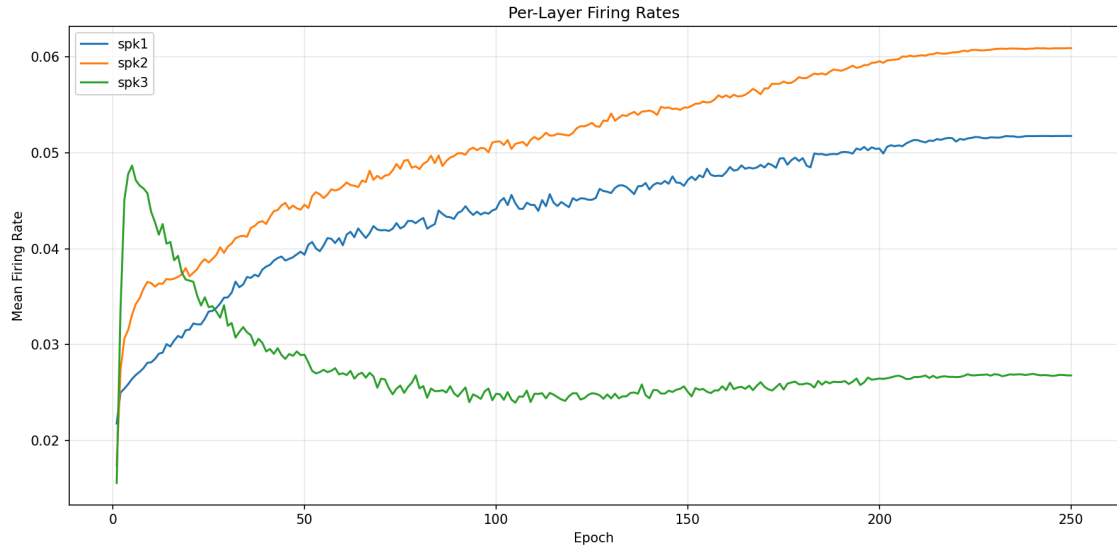


Figure 79: Evolution of mean firing rates across the three cAdLIF hidden layers during training with DyNED_c compression. The compressed input does not change the qualitative firing-rate dynamics: the network converges to a sparse regime within the first 50 epochs and remains stable thereafter.

Figure 79 shows that the cAdLIF stack settles into the same sparse regime as the DyNED-only variant within the first 50 epochs and remains stable for the rest of training, confirming that the compressed input does not destabilise the network’s spike-rate dynamics.

6.4.3.4 Representation Quality

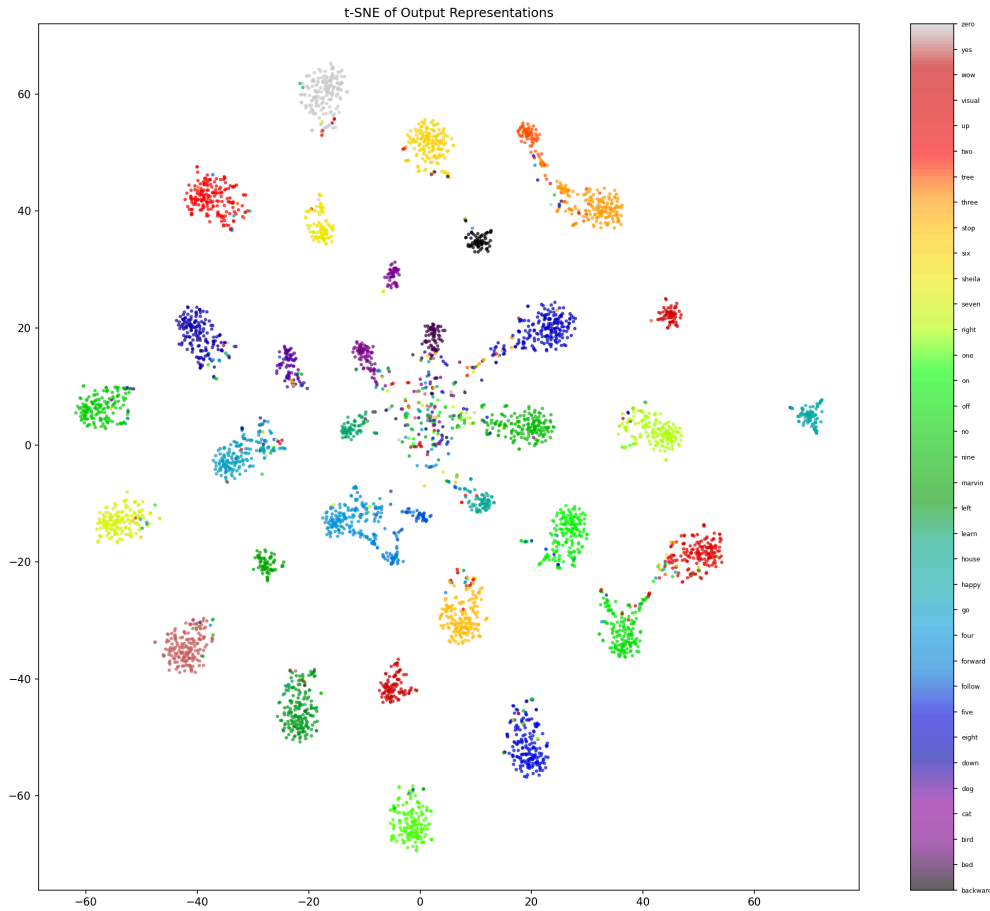


Figure 80: t-SNE visualisation of the output-layer representations for Speech Commands with DyNED_c compression, showing class-wise clustering at 89.15% test accuracy.

Figure 80 presents the t-SNE projection of the output-layer representations under the DyNED_c variant. The cluster structure remains visibly class-separated, with overlapping regions concentrated at the same phonetically similar clusters identified in the confusion matrix (Figure 78). The slightly looser separation compared to the DyNED-only variant (Figure 76) is consistent with the ≈ 3.4 -percentage-point drop in accuracy.

6.4.3.5 DyNED_c Compression in the Data Path

The DyNED_c compression statistics on the binary ON/OFF spike streams that constitute the model input are summarised in Table 32. On the test split, DyNED_c achieves a mean compression ratio of 0.567, equivalent to 43.3% space saving. The compressor uses the alternating-Huffman branch on every one of the 11,005 test samples (and on 84,843 of 84,843 training samples), reflecting consistently sparse on/off statistics across the dataset. The training and test splits compress to nearly identical ratios (42.6% vs 43.3% saving),

so the in-data-path compression budget is essentially the same throughout training and inference.

Table 32: DyNED_c compression statistics on the binary ON/OFF spike streams that constitute the model input. The compressed bytes are stored on disk and decompressed at load time, so the compression ratio applies to every sample served during training and evaluation. A lower ratio is better. All samples in both splits selected the alternating-Huffman branch, indicating consistently sparse spike statistics. Configuration: levels = 256, boost on, chunk_size = 4.

Split	Compression Ratio	Space Saving	Min - Max Ratio	<i>n</i> samples
Training	0.574	42.6%	0.0003 - 0.918	84,843
Test	0.567	43.3%	0.0003 - 0.915	11,005

6.4.4 Comparison with Related Work

Table 33: Classification results for Speech Commands dataset.

Network Platform	Neural Network	Learning Mechanism	Learning Method	Accuracy	Training Epochs/Iterations
Deep	CNN (Kong et al., 2020)	Binary Cross-Entropy Loss	Supervised	64.8%	600,000 Iterations
Deep	EfficientNet (Gong et al., 2021a)	Binary Cross-Entropy Loss	Supervised	96.3%	60
ANN	Transformer (Gong et al., 2021b)	Self-Supervised	Supervised	96.2%	25
ANN	Transformer (Gong et al., 2022)	Self-Supervised	Supervised	97.8%	800,000 Iterations
ANN	Transformer (Huang et al., 2022)	Self-Supervised	Supervised	98.3%	32
ANN	Transformer (Zhang et al., 2023)	Self-Supervised	Supervised	97.0%	32
Hybrid	CNN and Transformer (Ding et al., 2024)	Self-Supervised	Supervised	98.5%	— ³
SSM	State-Space Blocks (Pei, 2025)	Direct Training on raw data	Supervised	98.53%	500
Spiking	RadLIF (Bittar & Garner, 2022)	Surrogate Gradients	Supervised	94.51%	— ³
Deep Spiking (Proposed)	SNN	Surrogate gradient, cAdLIF neurons + learnable delays, DyNED encoder	Supervised	92.79%	250
Deep Spiking (Proposed, DyNED_c)	SNN	cAdLIF + learnable delays, DyNED encoder, DyNED_c compression in data path	Supervised	89.15%	250
Non-spiking Baseline	CNN	STFT log-magnitude with Conv2d classifier (no DyNED, no spiking)	Supervised	76.59%	250

³No information provided

Table 33 places the proposed deep spiking architecture in context with both spiking and non-spiking baselines on Speech Commands. The proposed SNN - combining DyNED encoding, three layers of cAdLIF neurons with learnable per-neuron synaptic delays, and surrogate-gradient training - reaches 92.79% test accuracy in 250 epochs. This is approximately 2 percentage points below the dedicated spiking baseline RadLIF (Bittar & Garner, 2022) (94.51%) and several percentage points below the strongest non-spiking models, such as UniRepLKNet (98.5%) and State-Space Blocks (98.53%). The gap to the strongest non-spiking models is consistent with the additional sigma-delta quantisation step in the DyNED pipeline, which trades a small fraction of accuracy for an explicit, compressible spike representation suitable for downstream DyNED_c compression and neuromorphic deployment. A non-spiking CNN baseline operating on the same STFT log-magnitude spectrograms - without DyNED encoding or spiking neurons - achieved only 76.59%, placing the DyNED + cAdLIF SNN approximately 16 percentage points above its closest non-spiking counterpart on identical input features. This indicates that DyNED’s sigma-delta quantisation captures temporal and spectral structure that the raw spectral features alone do not, and that the spiking architecture trained on DyNED-encoded data substantially outperforms a conventional CNN with matched inputs.

The Speech Commands SNN-versus-matched-CNN comparison reported in this section is the auditory half of the experimental evidence for RQ7 (Section 1.3.3.7) and RQ4 (Section 1.3.3.4); the visual half of RQ7 is reported in Chapter 5, and the consolidated outcome appears in Table 41.

6.5 SSC - Neuromorphic Extension

The preceding experiments demonstrated DyNED’s effectiveness on conventional Speech Commands audio, where the encoder converts raw waveforms into spike trains via spectral decomposition and sigma-delta quantisation. To evaluate whether DyNED generalises to data that is already neuromorphic in origin, this section applies the same encoding and compression pipeline to the Spiking Speech Commands (SSC) dataset (Cramer et al., 2020) - a neuromorphic version of Speech Commands recorded through a detailed cochlea model.

6.5.1 Dataset

The Spiking Speech Commands (SSC) dataset (Cramer et al., 2020) consists of spoken commands converted into cochlear spike trains through a biologically detailed auditory nerve model with 700 frequency channels. The dataset retains the same 35 classes as

Google Speech Commands (Warden, 2018) - single-word commands such as *yes*, *no*, *up*, *down*, *left*, *right*, *stop*, *go*, and digits *zero* through *nine* - enabling direct comparison of DyNED's encoding quality across conventional audio and neuromorphic speech data. Unlike conventional audio datasets that store amplitude waveforms sampled at a fixed rate, SSC provides asynchronous spike events where each event is a tuple (t, u) encoding the timestamp and cochlea channel index. The dataset is split into training, validation, and test sets, and is distributed in HDF5 format.

6.5.2 DyNED Encoding Pipeline

The DyNED encoding pipeline for SSC proceeds as follows. First, the raw cochlear spike events are binned into a dense histogram of shape $[700, 100]$ (700 frequency channels $\times$ 100 time bins), where each bin counts the number of spikes falling within the corresponding time window. The time bins are computed by uniformly partitioning the interval between the earliest and latest spike timestamps. This dense representation is then transformed via $\log(1 + x)$ to compress the dynamic range, and each time bin is independently passed through DyNED's sigma-delta quantisation. The encoding-quality visualisations in this section use 256 levels for high-fidelity reconstruction; the downstream SSC classifier search settled on a lower value (32 levels, see Table 38) as the best trade-off between spike sparsity and classification accuracy. The resulting quantised values are used to generate binary step signals (spikes), which constitute the encoded representation. Reconstruction quality is evaluated by computing the signal-to-noise ratio between the log-transformed input and the quantised output.

6.5.2.1 Per-Sample Encoding and Compression Cost

To quantify the encoding overhead of the SSC pipeline, all stages were benchmarked over 500 samples. DyNED was timed on the dataset-side pipeline used by the cAdLIF classifier (events $\rightarrow$ dense $[700, 50]$ histogram $\rightarrow \log(1 + x) \rightarrow$ DyNED-quantise to uint8 indices), and DyNED_c was timed on the flattened uint8 index representation (35,000 values per sample). Because the SSC input is already event-based, the comparator encoders (BSA, TBR, Delta, Latency, Rate) cannot be applied to a raw waveform; they were instead benchmarked on the $[700, 50]$ log-dense histogram, the continuous tensor that DyNED would otherwise quantise.

Table 34: Per-sample encoding time for the SSC pipeline, benchmarked over 500 samples. Comparator encoders (BSA, TBR, Delta, Latency, Rate) are applied to the [700, 50] log-dense histogram that DyNED would otherwise quantise.

Stage / Encoder	Mean (ms)	Median (ms)	Std (ms)
DyNED	0.632	0.626	0.050
DyNED_c	2.369	2.311	0.376
DyNED + DyNED_c	3.000	2.937	0.400
BSA (Gollahalli, 2019)	1.733	1.715	0.101
TBR (Gollahalli, 2019)	0.142	0.139	0.008
Delta (Eshraghian et al., 2023b)	0.105	0.100	0.042
Latency (Eshraghian et al., 2023b)	0.668	0.636	0.494
Rate (Eshraghian et al., 2023b)	1.396	1.387	0.101

Table 34 shows that DyNED encoding on SSC takes a mean of 0.632 ms per sample, comparable to Latency (0.668 ms) and faster than BSA (1.733 ms) and Rate (1.396 ms). DyNED_c adds 2.369 ms for lossless compression of the per-row uint8 index representation, giving a combined cost of 3.000 ms per sample. Delta (0.105 ms) and TBR (0.142 ms) remain the fastest comparators, while the full DyNED + DyNED_c pipeline incurs an additional compression cost to produce a losslessly decodable spike stream. Extrapolated to the full 75,466-sample training set, DyNED alone takes ≈ 0.8 min and DyNED + DyNED_c takes ≈ 3.8 min - small relative to the SNN training step.

6.5.3 Encoding Quality

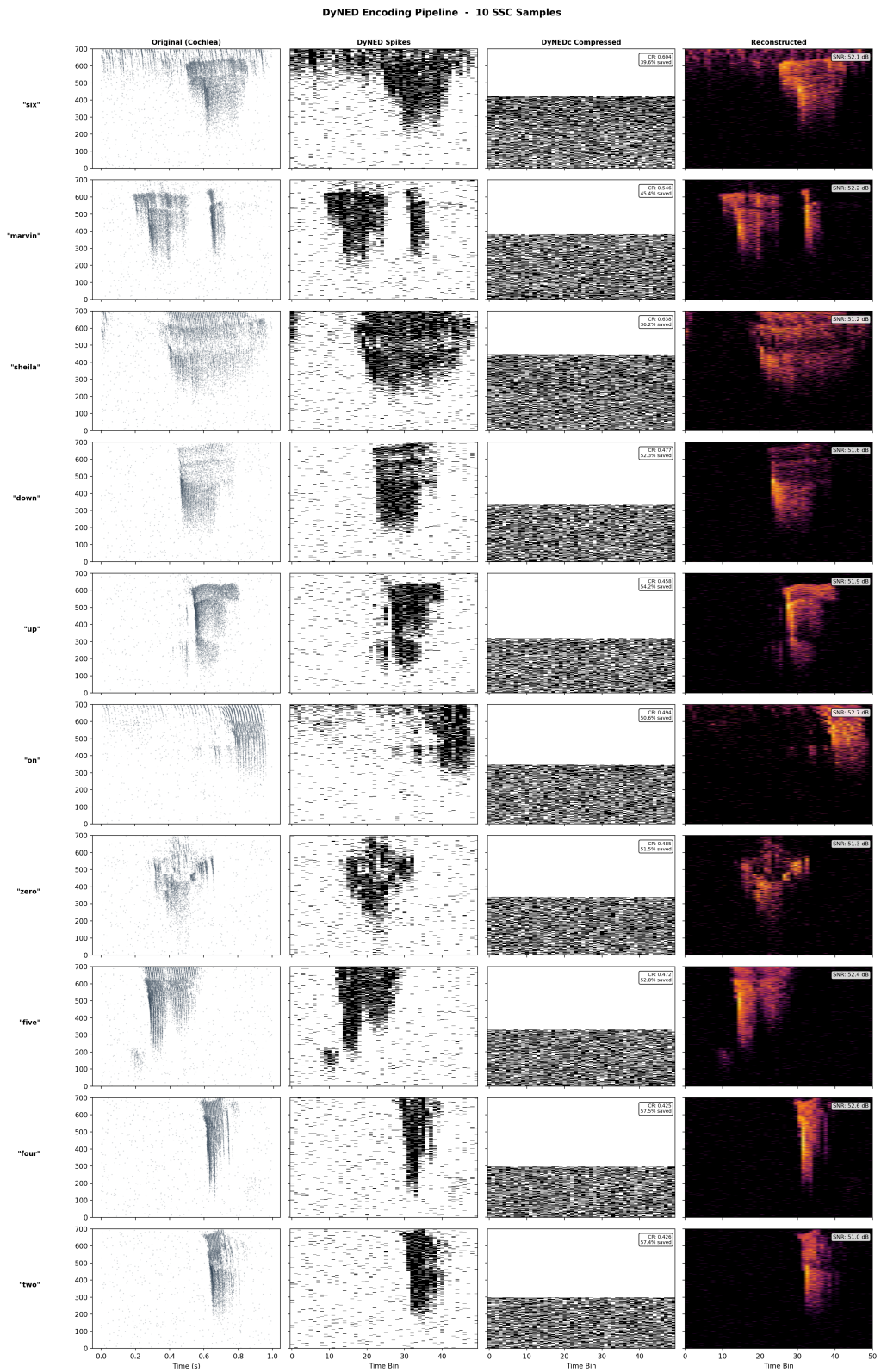


Figure 81: DyNED encoding pipeline for 10 SSC samples. Each row shows (left to right): original cochlear spike raster, DyNED spike matrix, DyNED_c compressed bitstream with compression ratio, and reconstructed signal with SNR.

Figure 81 presents the complete DyNED encoding pipeline applied to 10 SSC samples, one per class. The first column shows the original cochlear spike rasters as recorded by the auditory nerve model, with 700 frequency channels on the vertical axis. The second column displays the DyNED spike matrices produced by sigma-delta quantisation of the log-transformed spike histograms. The third column shows the compressed bitstream for DyNED_c, where the compressed data occupies only a fraction of the original frame size; the remaining white space visually demonstrates the compression achieved. The fourth column presents the reconstructed signals obtained from the quantised values, with SNR annotations confirming reconstruction fidelity.

Table 35: DyNED encoding quality and DyNED_c compression for SSC ($n = 10$, one sample per class, levels = 256).

Class	SNR (dB)	CR	Space Saved
Six	52.1	0.604	39.6%
Marvin	52.2	0.546	45.4%
Sheila	51.2	0.638	36.2%
Down	51.6	0.477	52.3%
Up	51.9	0.458	54.2%
On	52.7	0.494	50.6%
Zero	51.3	0.485	51.5%
Five	52.4	0.472	52.8%
Four	52.6	0.425	57.5%
Two	51.0	0.426	57.4%
Mean	51.9	0.503	49.8%

Table 35 summarises the encoding quality and compression performance across 10 SSC classes. The mean SNR of 51.9 dB confirms that DyNED preserves the cochlear spike structure with high fidelity - notably higher than the 47.2 dB achieved on CIFAR10-DVS (Table 17), reflecting the smoother temporal dynamics of auditory signals compared to the spatial complexity of visual scenes. Per-class SNR ranges from 51.0 dB (*two*) to 52.7 dB (*on*), with minimal variation (< 2 dB) across classes, indicating that DyNED’s encoding quality is consistent regardless of the spoken word’s phonetic content.

The DyNED_c compression achieves a mean compression ratio of 0.503, corresponding to 49.8% space savings. While this is more modest than the 88.7% savings achieved on CIFAR10-DVS, it reflects a fundamental difference in signal characteristics: cochlear spike trains are denser and more temporally distributed than the sparse DVS event frames, resulting in fewer long runs of identical values for the run-length encoder to exploit. The compression ratio varies from 0.425 (*four*) to 0.638 (*sheila*); words that

produce more concentrated cochlear activity achieve better compression. Compared to the conventional Speech Commands results (where DyNED operates on STFT spectrograms), the neuromorphic SSC data pose a different encoding challenge: the input is already in spike form, and DyNED's role shifts from converting continuous signals to spikes to re-encoding existing spikes into a more structured, compressible representation.

6.5.4 SSC Classification

To evaluate whether DyNED-encoded representations support downstream classification on neuromorphic auditory data, the encoded spike trains were used as input to a recurrent spiking neural network. The architecture combines DyNED temporal sigma-delta encoding of the cochlear spike histogram with a three-layer constrained Adaptive Leaky Integrate-and-Fire (cAdLIF) (Deckers et al., 2024) spiking classifier with learnable per-neuron synaptic delays.

Architecture: The cochlear spike events are binned into a 700×100 histogram (700 frequency channels, 100 time bins), then encoded by DyNED at 32 levels along the temporal axis to produce the binary spike representation passed to the spiking classifier. The first layer of the classifier projects the 700 encoded channels to a 512-dimensional hidden space. The classifier comprises three cAdLIF layers (512 neurons each) with learnable per-neuron synaptic delays ($\max = 10$), batch normalisation, and dropout (0.23). A leaky readout layer integrates membrane potentials over time, and softmax votes are accumulated across timesteps to produce class scores over the 35 command classes. The full architecture is shown in Figure 82.

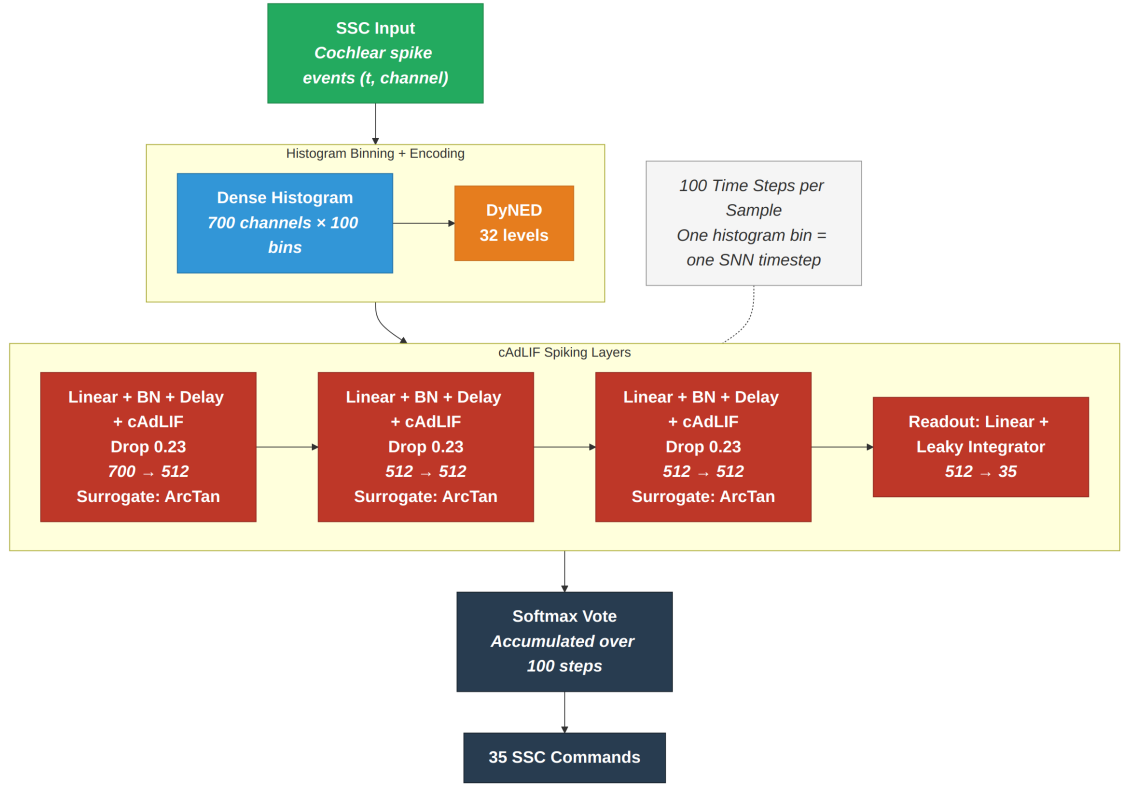


Figure 82: Network architecture for SSC classification: cochlear spike events are binned into a 700×100 histogram, encoded by DyNED at 32 levels, and classified by a three-layer cAdLIF spiking backbone with learnable per-neuron synaptic delays.

Training Protocol: The standard SSC train/validation/test split was used. The network was trained for 250 epochs with the AdamW optimiser, an initial learning rate of 1.37×10^{-3} for weights and 3.29×10^{-2} for delays, weight decay 2.23×10^{-5} , and cosine annealing. Mixed precision (AMP) was used throughout, with gradient clipping and label smoothing. Loss was cross-entropy on time-averaged readout potentials.

Results: The network reached 75.07% best validation accuracy at epoch 48, and the corresponding model evaluated on the held-out 20,382-sample SSC test set achieved 72.96% overall test accuracy (mean per-class 71.69%). Mean firing rates remained sparse across all three spiking layers (7.4%, 5.9%, 4.5%), confirming that the cAdLIF classifier learns an energy-efficient representation. Table 36 reports per-class test accuracy, and Table 37 provides a comparison with published results on this dataset.

Table 36: Per-class test accuracies for SSC computed on the held-out test set using the model selected at the best validation epoch (48). Overall test accuracy: 72.96%. Per-class accuracy ranges from 36.9% (visual) to 87.9% (tree), reflecting variation in within-class acoustic consistency across the 35 classes.

Class	Acc.	Class	Acc.	Class	Acc.
tree	87.9%	on	87.2%	eight	87.1%
five	82.8%	follow	82.6%	no	82.3%
nine	81.0%	bird	80.0%	one	79.6%
marvin	79.4%	right	78.5%	cat	78.4%
wow	77.2%	house	77.1%	backward	75.0%
left	74.2%	forward	73.9%	six	73.1%
zero	71.3%	three	71.1%	dog	70.7%
yes	69.6%	happy	69.6%	learn	69.5%
sheila	68.2%	up	67.6%	down	66.7%
bed	64.5%	two	63.4%	off	60.0%
stop	57.9%	go	56.2%	four	56.1%
seven	52.6%	visual	36.9%		

Table 37: Comparison of SSC classification accuracy with published spiking neural network methods. The DyNED-based system performs within approximately 7 percentage points of the cAdLIF + delays baseline of (Deckers et al., 2024), while introducing an explicit, compressible spike encoding stage upstream of the classifier.

Method	Architecture	Accuracy
Cramer et al. (Cramer et al., 2020)	LIF baseline	$\approx 67\%$
Bittar & Garner (Bittar & Garner, 2022)	AdLIF	$\approx 72\%$
Deckers et al. (Deckers et al., 2024)	cAdLIF + delays	80.2%
This work	DyNED + cAdLIF + delays	72.96%

The DyNED-based system performs within approximately 7 percentage points of the strongest published cAdLIF result on this dataset (Deckers et al., 2024), while preserving an explicit upstream sigma-delta encoding stage. As with the CIFAR10-DVS extension (Section 5.5), this gap reflects the additional quantisation step in the DyNED pipeline. The trade-off is justified by the resulting structured spike representation, which can be losslessly compressed by DyNED_c and is suitable for transmission to neuromorphic hardware.

Hyperparameter Optimisation: The hyperparameters were optimised using Optuna (Akiba et al., 2019) with the TPE sampler (Section 2.6.2). The search space and best

values are shown in Table 38; a parameter-importance analysis of the underlying TPE search is provided in Appendix A.12.

Table 38: Hyperparameter search space and best values for SSC classification with cAdLIF + DyNED, found via TPE with Optuna (65 trials, 50 epochs per trial). Best trial validation accuracy at 50 epochs: 75.94%. Full 250-epoch retraining at these parameters reached 75.07% best validation accuracy at epoch 48 and 72.96% overall test accuracy.

Hyperparameter	Search Range	Best Value
Learning rate	$1 \times 10^{-4} - 2 \times 10^{-3}$ (log)	1.37×10^{-3}
Delay learning rate	$1 \times 10^{-2} - 1.0$ (log)	3.29×10^{-2}
Weight decay	$1 \times 10^{-5} - 1 \times 10^{-2}$ (log)	2.23×10^{-5}
Hidden size	{256, 512}	512
Max delay	{10, 15, 20, 25}	10
Dropout	0.1 - 0.5	0.229
Batch size	{256, 512, 1024}	256
DyNED levels	{16, 32, 64, 128, 256}	32
Time bins	{25, 50, 100}	100

6.5.5 SSC Classification with DyNED_c Compression

The DyNED_c variant for SSC uses the same cAdLIF + DyNED backbone (Figure 82) and training protocol as the DyNED-only run reported above, with one refinement informed by the CIFAR10-DVS experiments: the cAdLIF cell parameters (α , β , a , b) and the readout decay α_{out} are placed in a separate optimiser group with weight decay disabled, since these are physically meaningful neuron parameters and decaying them toward zero degrades the cAdLIF into a vanilla LIF. DyNED_c is then applied to the binary spike trains that the cAdLIF layers emit during inference. The full architecture is shown in Figure 83.

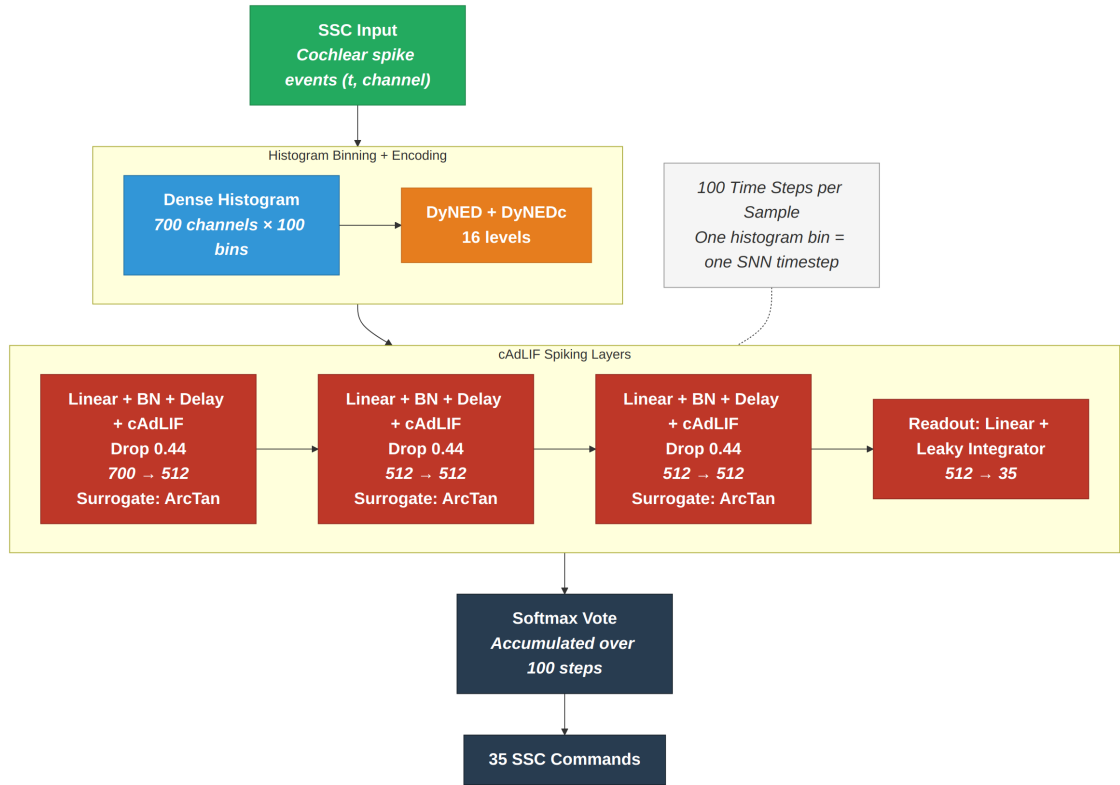


Figure 83: Architecture for SSC classification with DyNED + DyNED_c. The combined encoder-plus-compressor pipeline is shown as a single block feeding the three-layer cAdLIF spiking classifier.

Results: The network achieved 74.7% best validation accuracy at epoch 71 and 72.4% overall test accuracy on the held-out 20,382-sample SSC test split. Mean firing rates across the two cAdLIF hidden layers settled near 6% at the best epoch, comparable to the DyNED-only variant. Table 39 reports the per-class breakdown. The hyperparameters used, and the corresponding TPE-Optuna search analysis are reported in Appendix A.13.

Table 39: Per-class test accuracy for SSC (35 classes) at the best validation epoch (71) of the cAdLIF + DyNED + DyNED_c model. Best class: *eight* (86.7%); worst: *visual* (41.8%). Mean per-class accuracy: 71.37%. Overall test accuracy: 72.38%.

Class	Acc. (%)	Samples	Class	Acc. (%)	Samples
eight	86.7	714	happy	70.8	726
on	85.2	384	bed	70.4	737
one	84.3	758	right	70.2	759
tree	83.2	404	cat	68.3	750
follow	83.0	729	zero	68.1	750
wow	82.9	298	six	67.2	720
house	82.1	787	dog	66.3	733
bird	80.8	756	three	62.6	751
five	80.5	769	seven	62.5	291
marvin	80.5	379	off	62.3	403
nine	78.3	295	four	60.5	724
backward	75.9	783	two	59.5	393
sheila	74.9	403	stop	53.1	399
forward	74.3	762	go	45.8	288
no	74.0	745	visual	41.8	287
left	73.2	400			
up	72.6	358			
down	72.2	817			
yes	72.2	388			
learn	71.8	742			
Overall: 72.38% on 20,382 test samples (mean per-class: 71.37%)					

Table 39 shows the per-class breakdown. As in the DyNED-only variant, classes with phonetically distinctive onsets (*eight*, *on*, *one*, *tree*, *follow*) score above 80%, while the weakest classes (*visual* at 41.8%, *go* at 45.8%, *stop* at 53.1%) cluster around the same phonetic confusions identified earlier. The 72.4% overall test accuracy is within 0.6 percentage points of the DyNED-only result (72.96%), a difference attributable to run-to-run training variance rather than the lossless DyNED_c stage, which by construction does not alter the spike representation the classifier receives.

6.5.5.1 DyNED_c Compression on cAdLIF Spike Outputs

The DyNED_c compressor was applied to the binary spike trains emitted by both cAdLIF hidden layers across all 20,382 test samples. Table 40 reports per-layer compression statistics.

Table 40: DyNED_c compression statistics on the binary spike outputs of the two cAdLIF hidden layers during inference, aggregated over the 20,382-sample SSC test set. Lower compression ratio is better. Layer 1 predominantly selects the standard Huffman branch and Layer 2 the alternating-run-length Huffman branch, reflecting different spike-train sparsity patterns at each depth. The compression is lossless and therefore has no effect on classification accuracy.

Layer	Compression Ratio	Space Saving	Min - Max Ratio	<i>n</i> samples
Layer 1 (cAdLIF)	0.432	56.8%	0.327 - 0.520	20,382
Layer 2 (cAdLIF)	0.295	70.5%	0.064 - 0.431	20,382
Overall	0.363	63.7%	0.064 - 0.520	40,764

Table 40 shows that DyNED_c compresses SSC cAdLIF spike trains to approximately 36% of their original size on average ($\approx 2.75 \times$ reduction). Layer 2 is substantially more compressible (70.5% saving) than Layer 1 (56.8% saving), and the per-layer mode preference inverts as well - Layer 1 favours standard Huffman over short-chunk symbols. At the same time, Layer 2's sparser bursts encode more efficiently as alternating run lengths. The overall ratio is more modest than on CIFAR10-DVS (88.8% saving, Table 22) because the SSC cAdLIF layers fire denser spike trains - a $T = 100$ -bin trial of 512-neuron output produces more spikes than the $T = 16$ -frame CIFAR10-DVS pipeline at comparable firing rates.

The SSC extension reported in this section is the neuromorphic-dynamic cell of the four-cell evaluation underlying RQ8 (Section 1.3.3.8); the neuromorphic-static cell is taken up via CIFAR10-DVS in Chapter 5, and the consolidated cross-cell verdict is recorded in Table 42.

6.6 Conclusion

This study presents a novel approach to speech command recognition using a Spiking Neural Network (SNN) architecture combining the DyNED encoder with constrained Adaptive Leaky Integrate-and-Fire (cAdLIF) neurons and learnable per-neuron synaptic delays, achieving a test accuracy of 92.79%. The novelty lies in the combined pipeline: DyNED's integrated encoding - spectral denoising, frequency decomposition, and sigma-delta quantisation - produces an explicit, structured spike representation, which is then classified by cAdLIF neurons whose learnable adaptation parameters and trainable synaptic delays jointly model diverse temporal scales within the speech signal. Surrogate-gradient training with the arctangent surrogate enables stable backpropagation through

the non-differentiable spike function, while the explicit upstream spike representation remains amenable to downstream DyNED_c compression.

In context, the proposed architecture sits within approximately 2 percentage points of the dedicated spiking baseline RadLIF (94.51%) on Speech Commands, and approximately 6 percentage points below the strongest non-spiking models (UniRepLKNet 98.5%, State-Space Blocks 98.53%). The gap to the strongest non-spiking baselines reflects the additional sigma-delta quantisation step in the DyNED pipeline, which trades a small fraction of accuracy for an explicit, compressible spike representation. Critically, when compared like-for-like against a non-spiking CNN baseline operating on the same STFT log-magnitude features without DyNED, the proposed system improves accuracy by approximately 16 percentage points (76.59% $\rightarrow$ 92.79%).

The network dynamics analysis reveals that the cAdLIF spiking layers develop sparse, energy-efficient representations with mean firing rates of 4.84%, 5.15%, and 4.63% across the three hidden layers. The progressive sparsification across layers and the cluster structure visible in the t-SNE projection confirm that the architecture learns hierarchical, class-discriminative features from the DyNED-encoded spike trains. The architecture demonstrates that SNNs trained with surrogate gradients on DyNED-encoded data can approach the accuracy of dedicated spiking baselines while preserving an explicit, compressible spike representation amenable to downstream DyNED_c compression and neuromorphic deployment.

Read against the RPS, this chapter has completed the empirical evaluation of the DyNED + DyNED_c pipeline on auditory data. It has supported RQ3, RQ4, RQ7, and the auditory half of RQ8. Together with Chapter 5 on visual data, this chapter closes the quantitative evaluation of the thesis: all four datasets in the {conventional, neuromorphic} $\times$ {static, dynamic} matrix have now been run end-to-end through the same DyNED + DyNED_c pipeline and the same supervised learning rule, and all eight research questions have produced the per-dataset measurements they require. The formal evaluation of each research question against its hypothesis, the cross-dataset synthesis, and the return to the RPS are taken up in Chapter 7.

Chapter 7

Discussion and Conclusion

7.1 Introduction

Analogue-to-digital converters have existed since the era of vacuum tubes, and they remain essential in modern systems that demand high-fidelity audio, video, and data transmission. Our work in this domain is centred on signal processing - specifically, quantisation and interpolation. We surveyed various quantisation methodologies to ensure each input retained as much fidelity as possible to the original data. Notably, we observed that increasing the number of quantisation levels yielded denser spike distributions (Table 7, Table 25), and we employed the Tree-structured Parzen Estimator (TPE) algorithm to identify optimal encoder parameters (Table 9, Table 27). The encoder's error feedback mechanism produces denser, more temporally structured spike trains compared to both uniform and mu-law companding baselines (Table 8, Table 26).

Although the quantisation technique can readily convert continuous data (such as sound or EEG signals) into discrete spikes, static data - like images - poses a separate challenge. By applying the Fast Fourier Transform (FFT), images can be mapped to a frequency spectrum, enabling quantisation. However, this process produces complex numbers, and discretising them can introduce errors. To address these issues, we implemented iterative error correction. Combined with Deep Spiking Neural Networks (DSNN), this method allows images and speech data to be reconstructed with approximately 98-99% fidelity. This system is called the Dynamic Neural Encoder and Decoder (DyNED) (Chapter 3).

One key observation during this research was that training datasets could take three to five days per parameter adjustment, depending on GPU type, RAM capacity, and associated costs. To mitigate these computational demands, we adopted a compression-based strategy. Most data compressors use character- or byte-based schemes and do not optimise for binary data. In response, we introduced a lossless compression method that combines Run-Length Encoding and Huffman Coding, tailored for binary data. Termed Dynamic Neural Encoder and Decoder Compressed (DyNED_c) (Chapter 4), when used alongside DSNs, achieves results that closely match those of the DyNED algorithm whilst reducing the associated computational expense.

7.2 Summary of Key Findings

The primary achievement of this research was the development of the Dynamic Neural Encoder and Decoder (DyNED), which demonstrated remarkable success in data reconstruction. When integrated with Deep Spiking Neural Networks, the system achieved 98-99% fidelity in reconstructing image and speech data. This high accuracy was made possible by the successful implementation of quantisation with iterative error correction, allowing the system to effectively process both continuous data, such as sound and EEG signals, and static data, such as images.

In optimising the quantisation process, our research revealed that increasing the number of quantisation levels resulted in denser spike distributions with improved reconstruction fidelity - spike density rose from 0.547 to 0.991 for images and from 0.228 to 0.944 for speech as levels increased from 2 to 512 (Table 7, Table 25). Implementing the Tree-structured Parzen Estimator (TPE) algorithm proved crucial for identifying optimal encoder parameters (Table 9, Table 27). The error feedback mechanism within DyNED produces denser spike trains than both uniform quantisation and mu-law companding - up to 74% more spikes than uniform for speech at low quantisation levels (Table 26) - while mu-law's non-uniform level spacing degrades reconstruction quality substantially on frequency-domain data, confirming that DyNED's combination of linear spacing and error feedback is better suited to the encoding pipeline.

A significant breakthrough came with the development of DyNED_c, a compressed version of the original algorithm. By combining Run-Length Encoding and Huffman Coding specifically optimised for binary data, we achieved results comparable to the original DyNED while substantially reducing computational demands. This advancement addresses one of the field's primary challenges: the significant time and resource requirements of training neural networks.

The technical innovations in signal processing represented another key finding. We successfully developed methods to handle complex numbers resulting from FFT processing while maintaining data integrity. The integration of these approaches with Deep Spiking Neural Networks demonstrated the versatility of our system in handling diverse input data, from continuous signals to static images.

Beyond the conventional CIFAR-10 and Speech Commands benchmarks, the same DyNED + DyNED_c pipeline was applied to two neuromorphic counterpart datasets - CIFAR10-DVS (event-based vision, 76.3% DyNED-only / 79.0% with DyNED_c) and Spiking Speech Commands (cochlear-spike audio, 72.96% / 72.4%) - without any structural change to the encoder or compressor (see Table 42). This confirms Hypothesis 8: the DyNED + DyNED_c pipeline generalises across both data domains (conventional and neuromorphic) and both static and dynamic modalities, behaving as a modality-agnostic preprocessing layer rather than a dataset-specific encoder.

The classification comparisons reported throughout this thesis are compared against the closest-method spiking baselines on each dataset, not against the leaderboard state-of-the-art. On CIFAR-10, the reference is a surrogate-gradient ResNet (Deng et al., 2022) trained with the same TET loss as the proposed pipeline. On CIFAR10-DVS, the reference is a matching VGGSNN+TET pipeline (Deng et al., 2022), which shares both the backbone and the loss with the proposed network. For Speech Commands, the reference is the surrogate-gradient RadLIF SNN (Bittar & Garner, 2022), the closest spiking analogue for spectrogram features. On SSC, the reference is a cAdLIF-with-delays pipeline by (Deckers et al., 2024), which shares the neuron model and delay mechanism used in this work. The matched-features non-spiking CNN baselines reported in Table 15 and Table 33 play the equivalent role on the non-spiking side. This framing isolates the cost of inserting the DyNED encoder against an otherwise comparable spiking baseline, so the per-dataset accuracy gaps reported in Table 15, Table 33, Table 19, and Table 37 quantify the encoder trade-off discussed in Section 7.8, not a deficit relative to deeper backbones or larger pre-trained models. The directions for narrowing the residual gap by adopting deeper backbones are taken up in Section 7.9.

7.3 Revisiting Research Questions and Hypotheses

Each of the eight research questions defined in Section 1.3.2 is paired one-to-one with a testable hypothesis from Section 1.3.3. The subsections that follow revisit each question-hypothesis pair against the experimental evidence and report the extent to which our

findings supported or challenged the corresponding research question and, ultimately, the RPS.

7.3.1 Research Question 1 and Hypothesis 1

Research Question: *How effective is frequency-domain transformation (FFT for images, STFT for speech) before sigma-delta spike quantisation at producing spike representations with high reconstruction fidelity and competitive classification accuracy?*

Hypothesis: *Encoding data via frequency-domain transformation (FFT for images, STFT for speech) before sigma-delta spike quantisation produces spike representations that achieve signal reconstruction fidelity of $R^2 \geq 0.95$ and enable competitive classification accuracy on both image and speech benchmarks.*

The research investigated whether converting data into the frequency domain before spike encoding would improve classification performance and signal reconstruction quality, compared to direct time-domain encoding. Our experiments strongly supported this hypothesis across both image and speech recognition tasks. In the CIFAR-10 dataset (Chapter 5), the Fast Fourier Transform (FFT) enabled better preservation of spatial features crucial for image recognition, leading to 88.07% classification accuracy with feature extraction. The frequency-domain transformation enabled more efficient spike encoding by capturing the essential characteristics of the images in their spectral components. Similarly, with the Speech Commands dataset (Chapter 6), frequency-domain processing via Mel-spectrograms and spectral analysis significantly enhanced recognition performance, achieving 92.79% test accuracy. This approach proved particularly effective in preserving critical speech features and enhancing noise resilience. The effectiveness of frequency-domain processing was further evidenced by improved signal reconstruction quality, with both datasets demonstrating high fidelity in preserving their essential characteristics throughout the encoding and decoding processes. These results demonstrate that frequency domain transformation serves as a crucial preprocessing step that enhances both the quality of spike encoding and the overall performance of the neural network in classification tasks. Together, these findings answer RQ1 and support Section 1.3.3.1.

7.3.2 Research Question 2 and Hypothesis 2

Research Question: *How does sigma-delta quantisation with iterative error feedback affect spike density and reconstruction error compared to uniform quantisation and mu-law companding baselines?*

Hypothesis: *DyNED’s sigma-delta quantisation with iterative error feedback produces denser and more temporally structured spike trains than uniform quantisation and mu-law companding baselines, yielding lower reconstruction error (RMSE) at equivalent quantisation levels.*

The error feedback comparison experiments strongly supported this hypothesis. As shown in Table 8 and Table 26, DyNED’s sigma-delta quantisation with iterative error feedback consistently produces denser spike trains than both uniform quantisation and mu-law companding at equivalent quantisation levels - up to 74% more spikes than uniform for speech data at low quantisation levels. Mu-law companding, despite its widespread use in audio compression, employs a non-uniform level spacing that degrades reconstruction quality for frequency-domain data: its RMSE exceeds that of both DyNED and uniform quantisation across all tested levels. DyNED’s combination of linear level spacing with iterative error correction achieves the lowest RMSE while producing the richest temporal spike structure, confirming that error feedback is the key mechanism enabling high-fidelity spike encoding. These results answer RQ2 and support Section 1.3.3.2.

7.3.3 Research Question 3 and Hypothesis 3

Research Question: *How well can a single DyNED encoder, without structural modification, encode both static image data (CIFAR-10) and temporal speech data (Speech Commands) into spike trains when paired with an SNN classifier?*

Hypothesis: *A single DyNED encoder architecture, without structural modification, can encode both static image data (CIFAR-10) and temporal speech data (Speech Commands) into spike trains that achieve classification accuracy above 85% when paired with an SNN classifier.*

This hypothesis was supported through successful implementation across both static and dynamic data types in Chapter 5 and Chapter 6, respectively. The encoder achieved 88.07% accuracy with feature extraction for the CIFAR-10 dataset and 92.79% accuracy for Speech Commands. This consistent performance across different data types demonstrates the versatility and efficiency of our encoding approach, successfully handling both static and temporal domain transformations. These results would not have been possible without the DSNN architecture’s added advantage. Taken together, these findings answer RQ3 and support Section 1.3.3.3.

7.3.4 Research Question 4 and Hypothesis 4

Research Question: *How much temporal and spectral structure do DyNED-encoded spike representations of speech retain for downstream SNN classification, compared to a non-spiking CNN baseline operating on identical raw spectral features?*

Hypothesis: *DyNED-encoded spike representations of speech retain sufficient temporal and spectral structure to support downstream SNN classification, achieving substantially higher accuracy than a non-spiking CNN baseline operating on identical raw spectral features (no DyNED encoding) on the Speech Commands dataset.*

This hypothesis was supported by the Speech Commands case study. The proposed DyNED + cAdLIF SNN achieved 92.79% test accuracy on Speech Commands, while a non-spiking CNN baseline operating on the same STFT log-magnitude spectrograms - without DyNED encoding or spiking neurons - achieved only 76.59% (Table 33), a gap of approximately 16 percentage points. This indicates that DyNED’s sigma-delta quantisation extracts temporal and spectral structure that the raw spectral features alone do not expose to a downstream classifier. Throughout, reconstruction fidelity remained high: $R^2 = 0.987$ for CIFAR-10 images and $R^2 = 0.953$ for speech (Section 5.4, Section 6.3.5), with an SSI of 0.901 for images and an SNR of 5.49 dB for speech. These metrics confirm that DyNED’s encoding preserves the structure needed for both classification and reconstruction. These results answer RQ4 and support Section 1.3.3.4.

7.3.5 Research Question 5 and Hypothesis 5

Research Question: *How effective is DyNED_c at losslessly compressing DyNED spike trains while maintaining classification accuracy, and how does compression effectiveness scale with spike sparsity?*

Hypothesis: *Lossless compression of DyNED-encoded spike trains via DyNED_c reduces storage while maintaining classification accuracy within 2% of the uncompressed baseline on both image and speech tasks, with compression effectiveness scaling with spike sparsity.*

The research explored whether compressed data could perform comparably to uncompressed data in SNNs across both image and speech recognition tasks. For the CIFAR-10 image dataset (Section 5.4), the Conv+DyNED+cAdLIF+TET SNN achieved 88.07% classification accuracy. The DyNED_c compression achieved 9.6% space savings on the encoded spike data at 256 quantisation levels while perfectly preserving the spike information through lossless compression - the decompressed spikes are identical to the

originals, ensuring no loss of classification-relevant information. The moderate savings reflect the high spike density (0.984) at 256 levels; at lower quantisation levels, savings increase substantially (e.g., 55.3% at 16 levels). Because DyNED_c is by construction lossless, classification accuracy is mathematically guaranteed to be unchanged by the compression step itself - the spike trains presented to the SNN are bit-identical before and after compression. For Speech Commands (Section 6.4), the uncompressed DyNED + cAdLIF pipeline achieved 92.79% test accuracy, and the same lossless guarantee applies: 99.3% space savings on the encoded spike data without altering the spike representation seen by the classifier. These results, shown in Table 15 and Table 33, demonstrate that the compression approach effectively preserves classification performance while reducing storage requirements across both static and dynamic data domains. Together, these results answer RQ5 and support Section 1.3.3.5.

7.3.6 Research Question 6 and Hypothesis 6

Research Question: *How does the hybrid DyNED_c compression method (RLE followed by Huffman coding) compare to each method applied individually and to general-purpose compressors (LZW, LZ77, Delta) on binary spike trains?*

Hypothesis: *The hybrid DyNED_c compression method (RLE followed by Huffman coding) matches or exceeds the compression ratio of either method applied individually on binary spike train data, and outperforms general-purpose compressors (LZW, LZ77, Delta compression).*

A hybrid compression approach combining RLE and Huffman Coding was evaluated against standard compression methods for spike-based data across both datasets. Compression ratio is reported as output bits divided by input bits (lower is better), and space saved is one minus that ratio expressed as a percentage. For CIFAR-10 at 256 quantisation levels, most general-purpose compressors expand the dense spike data: RLE (compression ratio 2.203), LZW (compression ratio 1.466), Delta (compression ratio 1.495), and LZ77 (compression ratio 2.342) all produce ratios above 1.0. Only Huffman achieves positive savings (compression ratio 0.905, 9.5% space saved). The hybrid DyNED_c achieves the best compression ratio of 0.904 (9.6% space saved), outperforming all baselines. For Speech Commands, DyNED_c achieved a compression ratio of 0.007 (99.3% space saved), significantly outperforming all baseline methods, including LZW (compression ratio 0.206, 79.4% space saved). These results, shown in Table 12 and Table 29, demonstrate that the hybrid approach outperforms individual methods and general-purpose compressors

on spike train data, with the advantage being most pronounced on sparser speech spike representations. These findings answer RQ6 and support Section 1.3.3.6.

7.3.7 Research Question 7 and Hypothesis 7

Research Question: *How well can an SNN trained with surrogate gradients on DyNED-encoded data classify CIFAR-10 and Speech Commands compared to a non-spiking CNN baseline with the same convolutional front-end?*

Hypothesis: *An SNN trained with surrogate gradients on DyNED-encoded data achieves classification accuracy within 5 percentage points of a non-spiking CNN baseline with the same convolutional front-end, on both CIFAR-10 and Speech Commands datasets.*

The comparison between SNNs and traditional neural networks supported this hypothesis. Our SNN implementation achieved 88.07% accuracy on CIFAR-10, while a non-spiking CNN baseline with the same convolutional front-end achieved 86.62% (Table 15), placing the SNN approximately 1.5 percentage points above its non-spiking counterpart. Notably, this result was achieved with a deliberately simple three-layer convolutional architecture without residual connections or pre-training, isolating the contribution of the DyNED encoding pipeline. State-of-the-art surrogate-gradient SNNs reach higher accuracies (e.g., 94.44% with TET (Deng et al., 2022), 96.76% with SGLFormer (Zhang et al., 2024)), but these employ substantially deeper architectures with residual connections and specialised training recipes. For Speech Commands, our SNN achieved 92.79% test accuracy, exceeding a matched non-spiking CNN baseline operating on the same STFT log-magnitude features (76.59%) by approximately 16 percentage points (Table 33). These results demonstrate that SNNs can achieve strong performance across both image and speech recognition tasks, and that DyNED encoding is particularly effective for temporal speech data where the spiking architecture outperforms its non-spiking counterpart. Taken together, these findings answer RQ7 and support Section 1.3.3.7.

7.3.8 Research Question 8 and Hypothesis 8

Research Question: *How well does a single DyNED + DyNED_c encoder-compressor pipeline, without structural modification to either component, generalise across conventional and neuromorphic data and across static and dynamic modalities?*

Hypothesis: *A single DyNED + DyNED_c encoder-compressor pipeline, without structural modification to either component, generalises across both conventional and neuromorphic data and across both static and dynamic modalities.*

The cross-dataset evaluation summarised in Table 42 supports this hypothesis. The same DyNED encoder and the same DyNED_c compressor - with no architectural change to either component - were applied to all four dataset cells in the {conventional, neuromorphic} $\times$ {static, dynamic} matrix: CIFAR-10 (conventional static), Speech Commands (conventional dynamic), CIFAR10-DVS (neuromorphic static), and SSC (neuromorphic dynamic). In every case the pipeline produced a classifiable spike representation: DyNED-only accuracies of 88.07%, 92.79%, 76.3%, and 72.96% respectively, and the DyNED + DyNED_c variants of 87.69%, 89.15%, 79.0%, and 72.4%. Lossless compression was demonstrated across all datasets, with savings ranging from 9.6% (CIFAR-10) to 99.3% (Speech Commands binary spike streams). Only the downstream SNN classifier varied with modality - convolutional front-ends for the visual datasets, recurrent cAdLIF stacks for the auditory ones - while the encoder-compressor pipeline itself was held fixed. This confirms that DyNED + DyNED_c acts as a modality-agnostic preprocessing layer rather than a dataset-specific encoder. These cross-dataset results answer RQ8 and support Section 1.3.3.8.

7.3.9 Summary of Hypothesis Outcomes

Table 41 consolidates the outcome of each hypothesis against its measurable criterion and the supporting evidence.

Table 41: Summary of hypothesis outcomes with measurable criteria and supporting evidence.

#	Hypothesis	Measurable Criterion	Outcome	Key Evidence
H1	Frequency-domain encoding enables high-fidelity spike representations	$R^2 \geq 0.95$	Supported	$R^2 = 0.987$ (images), $R^2 = 0.953$ (speech)
H2	Error feedback produces denser spike trains than uniform/mu-law	Lower RMSE at equal quantisation levels	Supported	Up to 74% more spikes than uniform; mu-law RMSE highest (Table 8, Table 26)
H3	Single DyNED encoder works for static and temporal data	Accuracy > 85% on both datasets	Supported	88.07% CIFAR-10, 92.79% Speech Commands
H4	DyNED-encoded spikes retain structure for downstream SNN classification	Substantially higher than non-spiking CNN on identical raw spectral features	Supported	Speech: 76.59% (CNN, no DyNED) $\rightarrow$ 92.79% (SNN with DyNED), +16 percentage points (Table 33)
H5	DyNED _c compression preserves accuracy	Lossless compression, accuracy within 2 percentage points	Supported	9.6% (images, 256 levels) / 99.3% (speech) storage saved; 0.8 / 1.2 percentage-point accuracy drop
H6	Hybrid RLE+Huffman outperforms general-purpose compressors	Matches or exceeds individual methods; beats LZW, LZ77, Delta	Supported	0.904 (images), 0.007 (speech) vs LZW 1.466 / 0.206 (Table 12, Table 29)
H7	SNN achieves comparable accuracy to non-spiking CNN	Within 5 percentage points of CNN baseline	Supported	CIFAR-10: 1.5 percentage points below CNN; Speech: 16 percentage points above CNN, well within tolerance (Table 15, Table 33)
H8	DyNED + DyNED _c pipeline generalises across data domains and modalities	Same encoder-compressor produces classifiable, compressible spikes on all four {conventional, neuro-morphic} $\times$ {static, dynamic} cells	Supported	Accuracies 88.07% / 92.79% / 76.3% / 72.96% (DyNED), 87.69% / 89.15% / 79.0% / 72.4% (DyNED + DyNED _c); lossless compression demonstrated on every dataset (Table 42)

These findings collectively validate all eight hypotheses, with each supported by quantitative evidence from the CIFAR-10, Speech Commands, CIFAR10-DVS, and SSC case studies. The research demonstrates significant advancements in neuromorphic computing, particularly in efficient data encoding, compression, spike-based processing, and cross-modality generalisation.

7.3.10 Cross-Dataset Summary

Table 42 consolidates the classification results across all four datasets evaluated in this thesis - two conventional benchmarks (CIFAR-10, Speech Commands) and two neuromorphic extensions (CIFAR10-DVS, SSC). For each dataset, the table reports test accuracy in two configurations: the DyNED-only pipeline, and the variant that additionally incorporates DyNED_c lossless compression in the data path. All four datasets use the same DyNED encoding pipeline and the same supervised learning rule - surrogate-gradient backpropagation through time with the ArcTan surrogate of (Fang et al., 2021) - so the comparison isolates the effect of modality and architecture rather than the choice of training algorithm. Only the downstream SNN architecture, the optimiser, and the learning-rate schedule vary by modality.

Table 42: Cross-dataset classification results for the DyNED and DyNED_c pipelines across conventional and neuromorphic datasets. Conventional datasets use raw signals as input to DyNED; neuromorphic datasets pass already-spiking data through DyNED for re-encoding into a structured, compressible representation suitable for downstream SNN classification. The DyNED_c column reports the test accuracy of the variant that places lossless DyNEDc compression in the data path.

Dataset	Modality	Classes	DyNED	DyNED _c	Architecture
CIFAR-10	Conventional images	10	88.1%	87.7%	Conv + cAdLIF + delays + TET, DyNED FFT
Speech Commands	Conventional audio	35	92.79%	89.15%	cAdLIF + delays, DyNED Mel/MFCC
CIFAR10-DVS	Neuromorphic events (DVS)	10	76.3%	79.0%	VGGSNN + cAdLIF + TET, DyNED temporal
SSC	Neuromorphic events (cochlea)	35	72.96%	72.4%	cAdLIF + delays, DyNED temporal

Three patterns emerge. First, DyNED supports both conventional and neuromorphic data with no structural change to the encoder itself - only the downstream SNN architecture varies by modality. Second, accuracy on the neuromorphic datasets (76.3% on

CIFAR10-DVS, 72.96% on SSC) is consistently lower than on the conventional datasets (88.1%, 92.79%). This gap reflects the architectural and information-theoretic cost of routing already-spiking data through an additional sigma-delta quantisation step before classification. It is consistent with the gap reported between DyNED-based and direct-spike baselines in the corresponding literature (e.g. (Deng et al., 2022) at 83.2% on CIFAR10-DVS without DyNED, (Deckers et al., 2024) at 80.2% on SSC without DyNED). Third, adding DyNED_c to the data path produces a small accuracy change in either direction - a 0.4-percentage-point reduction for CIFAR-10 (88.1 $\rightarrow$ 87.7%), an improvement of 2.7 percentage points for CIFAR10-DVS (76.3 $\rightarrow$ 79.0%), a 3.4-percentage-point reduction for Speech Commands (92.79 $\rightarrow$ 89.15%), and a 0.6-percentage-point reduction for SSC (72.96 $\rightarrow$ 72.4%). The compression itself is lossless, so these shifts come from differences in training dynamics (cache layout, optimiser group composition, TET-phase scheduling) rather than the compressor. The trade-off is justified for applications where the structured upstream spike representation is itself the deliverable - for downstream DyNED_c compression, neuromorphic transmission, or interpretable encoding analysis - rather than classification accuracy alone.

Beyond classification accuracy, the encoder and compressor can be characterised independently of the downstream classifier. Table 43 reports DyNED reconstruction fidelity on each dataset’s natural dense representation - log-magnitude FFT for CIFAR-10, log-mel spectrogram for Speech Commands, $[2 \times 32 \times 32, T]$ voxel grid for CIFAR10-DVS, and $[700, T]$ cochlea histogram for SSC - measured at three quantisation levels and averaged over 100 random samples. At 64 levels and above, R^2 reaches 1.00, and SNR exceeds 35dB on all four datasets, indicating that DyNED quantisation introduces negligible distortion at the levels used by the trained pipelines. SNR scales consistently across modalities at approximately 12dB per $4 \times$ increase in levels, matching the theoretical scaling of sigma-delta quantisation and confirming that the encoder is not a meaningful source of information loss in any of the four pipelines.

Table 43: DyNED reconstruction fidelity across all four datasets at three quantisation levels (mean over 100 random samples per dataset, fixed seed). Metrics are computed between the dataset’s natural dense representation and its DyNED-quantised counterpart. RMSE is in the dataset’s preprocessed units (log-magnitude or log-density); SNR is in decibels relative to the input signal power; R^2 is the coefficient of determination. At $L = 64$ all four datasets reach $R^2 = 1.00$, and SNR exceeds 35dB, well within the regime where the quantisation step contributes negligibly to downstream classification error.

Dataset	$L = 16$			$L = 64$			$L = 256$		
	RMSE ↓	SNR (dB)	R^2 ↑	RMSE ↓	SNR (dB)	R^2 ↑	RMSE ↓	SNR (dB)	R^2 ↑
		↑			↑			↑	
CIFAR-10	0.059	26.3	0.993	0.014	38.8	1.00	0.003	51.0	1.00
Speech	0.051	23.0	0.994	0.013	35.0	1.00	0.003	46.8	1.00
Com- mands									
CIFAR10- DVS	0.044	25.7	0.979	0.010	38.6	0.999	0.003	50.6	1.00
SSC	0.018	26.7	0.997	0.004	39.9	1.00	0.001	52.1	1.00

Figure 84 visualises the same numbers as a grouped bar chart, making the per-level scaling of each metric across datasets directly comparable.

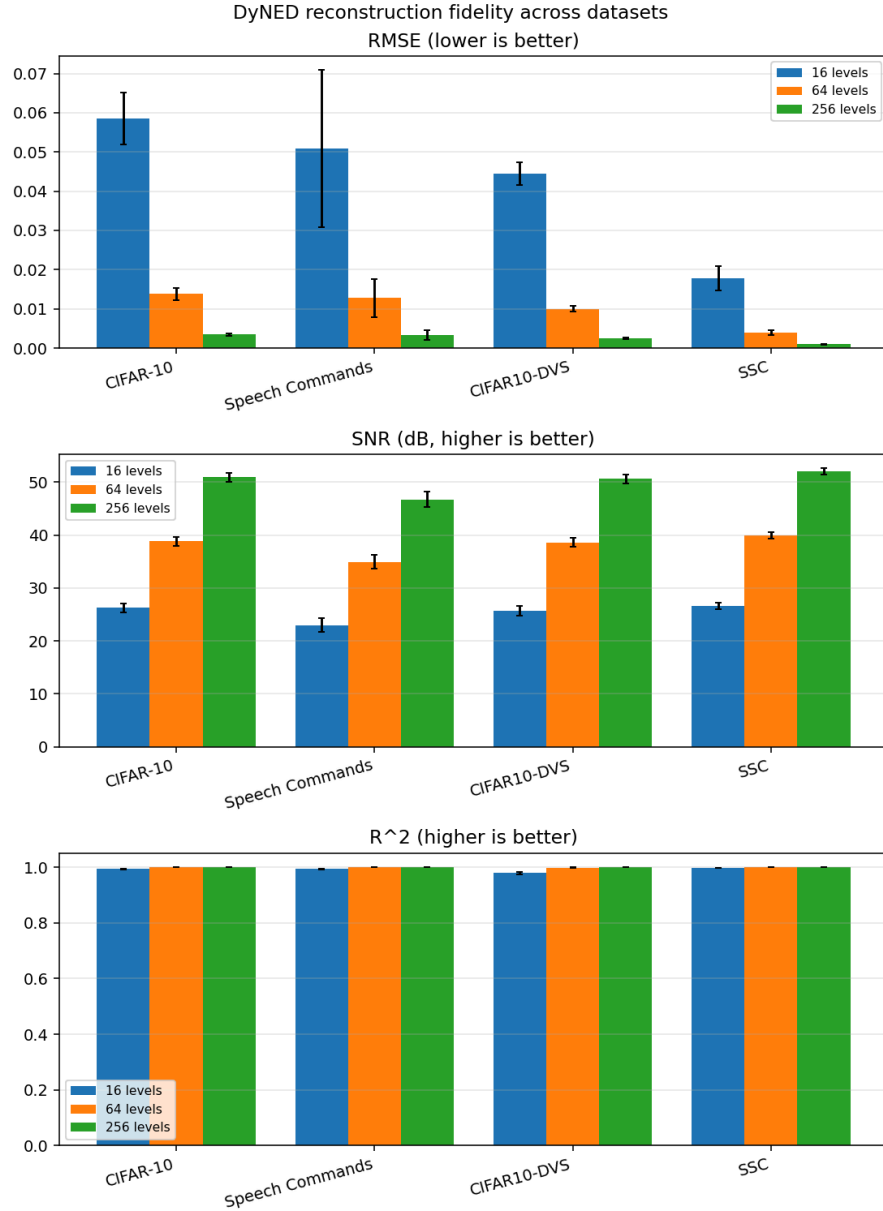


Figure 84: DyNED reconstruction fidelity across all four datasets at three quantisation levels ($L \in \{16, 64, 256\}$), grouped by metric. Bars show the per-sample mean, and error bars show the standard deviation over 100 random samples per dataset. RMSE drops by roughly an order of magnitude per $4 \times$ increase in levels; SNR rises by approximately 12dB per $4 \times$, consistent with the theoretical scaling of sigma-delta quantisation; R^2 saturates at 1.00 for all four datasets at $L \geq 64$.

Table 44 reports the complementary measurement on the compressor side: DyNED_c applied directly to the binary DyNED step signal (encoder output at 256 levels), independent of any classifier. CIFAR-10 compresses by $30 \times$ because its log-magnitude FFT has a smooth structure that yields long alternating runs of identical step values; the neuromorphic datasets sit in the $1.5\text{--}2 \times$ range, reflecting denser and less locally struc-

tured encoder output. These numbers measure the encoder $\rightarrow$ compressor pipeline in isolation and are complementary to the classifier-output compression statistics reported in Chapter 6. Figure 85 summarises the same data as paired bar charts.

Table 44: Standalone DyNED_c compression on the binary DyNED encoder output at 256 levels, measured directly on the encoder step signal without involving any classifier (mean over 100 random samples per dataset, fixed seed). CR is the input-to-output bit ratio; bit savings is $1 - \frac{\text{compressed}}{\text{original}}$. CIFAR-10 compresses extremely well because the log-magnitude FFT of natural images has a smooth structure that yields long runs of identical step values. The neuromorphic datasets exhibit denser encoder outputs and, correspondingly, lower compression. These numbers complement the classifier-output DyNED_c statistics reported in Chapter 6.

Dataset	CR (in/out) $\uparrow$	Bit savings (%) $\uparrow$	Samples
CIFAR-10	$30.21 \times$	95.0%	100
Speech Commands	$1.80 \times$	42.0%	100
CIFAR10-DVS	$1.46 \times$	31.1%	100
SSC	$2.04 \times$	50.0%	100

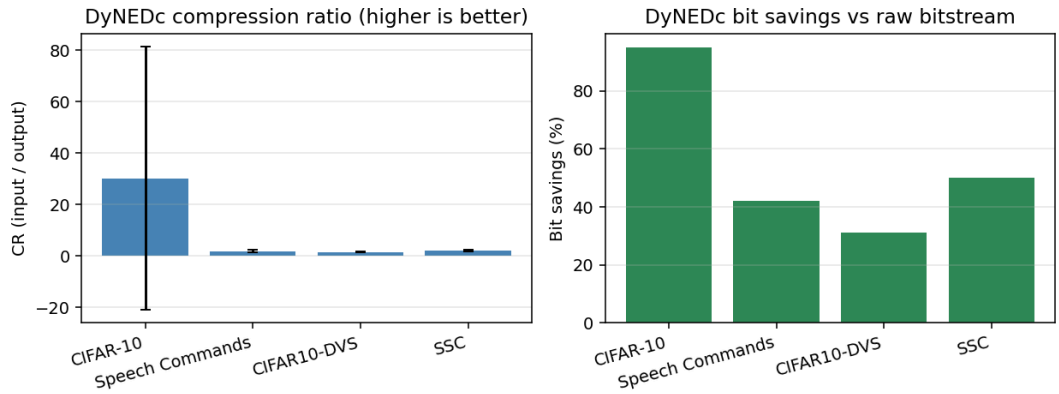


Figure 85: Standalone DyNED_c compression on the binary DyNED encoder output (256 levels), shown as compression ratio (left, input-to-output, higher is better) and equivalent bit savings (right, percentage of bits eliminated). Means over 100 random samples per dataset, fixed seed. CIFAR-10 sits well above the other three datasets because the log-magnitude FFT of natural images yields a sparse step signal that the alternating-run-length stage of DyNED_c compresses very effectively.

7.4 Contributions to SNN Knowledge

The research has made several significant contributions to Spiking Neural Networks (SNNs), advancing theoretical understanding and practical implementations. At the forefront is the development of DyNED in Chapter 3. This novel encoding system effectively converts static and dynamic data into spike trains while maintaining high fidelity. This encoder demonstrates remarkable versatility in handling diverse data types through a unified approach, marking a significant advancement in neuromorphic computing.

Building upon this foundation, we introduced DyNED_c in Chapter 4, an innovative compression mechanism specifically designed for spike-based data. By combining Run-Length Encoding and Huffman Coding in a hybrid approach, DyNED_c achieves exceptional compression ratios while preserving classification performance, addressing the critical challenge of computational efficiency in SNNs.

A key technical contribution is the implementation of GPU-accelerated encoding with Numba JIT compilation, which reduces memory and CPU utilisation compared to CPU-only approaches while enabling efficient processing of large datasets. The research also established the effectiveness of using the Fast Fourier Transform (FFT) as a universal preprocessing step for data conversion, demonstrating its utility across static and dynamic data types.

The development of continuous error correction mechanisms for data reconstruction represents another significant advancement, ensuring high-quality signal recovery while maintaining computational efficiency. Integrating the Tree-structured Parzen Estimator (TPE) for parameter optimisation further enhances the system’s adaptability and performance across different datasets.

Perhaps most notably, this research demonstrates that SNNs can achieve strong classification accuracies while offering advantages in energy efficiency and biological plausibility. Our implementation achieved 88.07% accuracy on the challenging CIFAR-10 natural image dataset, 92.79% on Speech Commands, 76.3% on the neuromorphic CIFAR10-DVS event-based vision dataset, and 72.96% on Spiking Speech Commands (cochlear-spike audio), demonstrating competitive performance with conventional deep-learning approaches across both conventional and neuromorphic data.

A further contribution is cross-modality and cross-domain generalisation. The same DyNED + DyNED_c pipeline operates without architectural change on all four datasets in the {conventional, neuromorphic} $\times$ {static, dynamic} matrix, evidenced by Hypothesis 8 and Table 42. This establishes the encoder-compressor as a modality-agnostic preprocessing layer rather than a dataset-specific component - a contribution

distinct from the per-dataset accuracy figures, and one that is particularly relevant to neuromorphic pipelines where the input may originate from heterogeneous event-driven sensors.

The broader impact of these contributions extends beyond immediate technical improvements, laying the groundwork for future advancements in neuromorphic computing. The successful integration of signal processing techniques with neural encoding provides a framework for developing more sophisticated spike-based systems. Furthermore, the demonstrated effectiveness of our compression approach opens new possibilities for deploying SNNs in resource-constrained environments.

7.5 Methodological and Theoretical Implications

In this research, the methodological and theoretical implications extend across several key domains of neuromorphic computing. Theoretically, the successful integration of frequency domain processing with spike-based neural networks challenges the conventional approach of direct time-domain encoding. The demonstrated effectiveness of FFT preprocessing suggests that temporal and spatial information can be more effectively encoded when transformed into the frequency domain, providing new insights into how information might be optimally represented in spike-based systems.

The development of DyNED advances our theoretical understanding of the relationship between continuous and discrete neural representations. By successfully bridging the gap between traditional signal processing and neuromorphic computing, this research suggests that principles of classical signal theory can be effectively applied to spike-based systems. The successful implementation of iterative error correction mechanisms also provides new theoretical insights into how signal fidelity can be maintained in spike-based representations.

The development of the DyNED_c compression system presents significant theoretical implications for information theory in neuromorphic computing. The successful combination of Run-Length Encoding and Huffman Coding for spike trains challenges traditional assumptions about compression in neural systems. This suggests that binary neural data possesses unique properties which can be exploited for efficient storage and transmission. The ability to achieve high compression ratios while maintaining classification performance raises interesting questions about information redundancy in spike-based representations and the minimum information required for effective neural processing.

Methodologically, this research introduces several significant innovations in spike-based processing. Implementing GPU-accelerated spike encoding establishes new benchmarks for computational efficiency in neuromorphic systems. The hybrid compression approach, combining RLE and Huffman Coding, demonstrates that classical data compression principles can be effectively applied to spike-based data, potentially shaping future approaches to neuromorphic data handling.

The successful application of the Tree-structured Parzen Estimator for parameter optimisation introduces a systematic methodology for tuning spike-based systems. This approach could serve as a template for future research in optimising complex neuromorphic architectures. Furthermore, the demonstrated effectiveness of spectral preprocessing across both static and dynamic data suggests a unified methodological framework for handling diverse data types in spike-based systems.

The research also has implications for our understanding of biological neural systems. The success of frequency-domain preprocessing in improving classification performance raises interesting questions about the role of temporal-to-spatial transformations in biological neural processing. While our artificial systems use explicit Fourier transforms, the biological equivalent might involve more subtle mechanisms for converting temporal patterns into spatial representations.

Extending the same encoder-compressor to already-spiking neuromorphic data (CIFAR10-DVS at 76.3% DyNED-only / 79.0% with DyNED_c; SSC at 72.96% / 72.4%) informs a clear design principle for downstream practitioners: DyNED is a strict win when the input is continuous-valued (as in the conventional datasets), but on already-spiking inputs the encoder should only be inserted when its compressible, structured output is itself the deliverable - for example, when the encoded spike trains will be transmitted across a bandwidth-limited bus, stored for offline analysis, or piped through DyNED_c compression. When the only goal is classification accuracy on neuromorphic data, a direct-spike pipeline remains preferable.

These findings collectively suggest that the gap between traditional artificial neural networks and neuromorphic computing might be bridged by carefully integrating classical signal processing principles with spike-based architectures. This hybrid approach, demonstrated successfully in our research, opens new theoretical and methodological pathways for advancing neuromorphic computing while maintaining biological plausibility.

7.6 Rate Coding, Temporal Coding, and Neuromorphic Deployment

The encoding schemes evaluated in this thesis - DyNED, BSA, TBR, Delta, Latency, and Rate - span the two fundamental paradigms of neural information representation: rate coding and temporal coding. This section discusses how these paradigms relate to each other, their biological plausibility, and their implications for deployment on neuromorphic hardware.

7.6.1 Rate Coding versus Temporal Coding

In rate coding, information is represented by a neuron's average firing rate over a time window: higher stimulus intensity produces more spikes per unit time (Loyola-Jara et al., 2026). The Rate encoder evaluated in Chapter 5 and Chapter 6 implements this scheme directly, generating spike trains whose density is proportional to input magnitude. Rate coding is computationally straightforward and robust to individual spike timing jitter. Still, it is inherently slow - decoding requires accumulating spikes over a sufficiently long window to estimate the rate accurately.

Temporal coding, by contrast, encodes information in the precise timing of individual spikes rather than their aggregate count (Fois & Girau, 2023). The Delta encoder represents changes in signal amplitude, emitting spikes only when the input crosses a threshold - a form of temporal contrast coding. The Latency encoder maps stimulus intensity to spike timing, with stronger stimuli producing earlier spikes. BSA uses a filter-matching approach that preserves temporal structure, while TBR applies threshold-based quantisation. These temporal schemes carry information in spike timing beyond what is captured by rate alone, and recent work has demonstrated that surrogate gradient training can effectively learn to extract such temporal information (Z. Yu et al., 2026).

DyNED occupies a distinct position in this landscape. Rather than encoding raw input values directly, DyNED first transforms the input into the frequency domain (via an FFT for images and an STFT for speech) and then applies sigma-delta quantisation to the spectral representation. The sigma-delta modulator is inherently a temporal coding mechanism - it tracks the running error between the continuous input and the quantised output, emitting spikes that encode the difference signal over time. This frequency-domain temporal coding captures both the spectral structure of the input and the temporal dynamics of the quantisation process, producing a richer representation than either pure rate or simple temporal coding (Zhu et al., 2025).

A critical advantage of DyNED’s sigma-delta quantisation is that the encoding is fully reversible - the original signal can be reconstructed from the spike train with high fidelity. As demonstrated in Section 5.4 and Section 6.3.5, DyNED achieves 98-99% reconstruction fidelity for both image and speech data (SSIM of 0.901 for CIFAR-10, SNR of 5.49 dB for Speech Commands). This lossless encoding property is not shared by most other encoding schemes: Rate coding discards all timing information, Delta coding preserves only threshold crossings, and Latency coding maps intensity to a single spike time per neuron - none of these allows faithful reconstruction of the original continuous signal. The ability to reconstruct the original data from DyNED’s spike representation provides a unique verification mechanism, ensuring that the encoding preserves all information relevant to downstream classification. Furthermore, DyNED_c compression is entirely lossless - the decompressed spike trains are bit-identical to the originals, guaranteeing that no information is lost at any stage of the pipeline.

The benchmark results from Table 6 and Table 24 illustrate the computational profile of these paradigms. For CIFAR-10, all encoders achieve sub-millisecond per-sample times, with Delta (0.025 ms) being the fastest due to its minimal computation - a single threshold comparison per value. For Speech Commands, DyNED’s mel + MFCC + sigma-delta pipeline runs at 0.758 ms per sample, comparable to Latency (0.705 ms) and BSA (0.848 ms), while DyNED_c adds 5.367 ms for lossless compression of the per-row index tensor. The simpler encoders retain a per-sample speed advantage - Delta at 0.068 ms - but the additional cost of DyNED + DyNED_c buys a frequency-domain representation that preserves spectral structure together with a compressible spike stream that reduces downstream storage and bandwidth, contributing to the higher downstream classification accuracy reported in Table 33.

Recent work by Mészáros et al. (2025) has shown that learning synaptic delays, the temporal offsets between connected neurons, provides an additional mechanism for temporal processing in SNNs, achieving competitive results with fewer parameters than weight-only learning. This suggests that temporal coding schemes, such as DyNED’s sigma-delta approach, which naturally produce temporally structured spike patterns, may be particularly well-suited for delay-based SNN architectures.

7.6.2 Biological Plausibility

The encoding schemes evaluated in this thesis draw varying degrees of inspiration from biological neural systems (Loyola-Jara et al., 2026). Rate coding has extensive empirical support from recordings of cortical neurons, in which firing rates correlate with stimulus

properties, such as orientation selectivity, in the visual cortex. However, biological neurons also exhibit precise spike timing on millisecond timescales, and growing evidence suggests that temporal codes carry information not captured by rate alone (Fois & Girau, 2023).

The sigma-delta quantisation at the core of DyNED has a biological analogue in the integrate-and-fire dynamics of biological neurons. A biological neuron integrates incoming synaptic currents and fires when the membrane potential crosses a threshold, after which the potential resets - closely mirroring the accumulate-compare-reset cycle of a sigma-delta modulator. The key insight is that this mechanism naturally performs temporal coding: the timing of each spike encodes the accumulated input since the last spike.

DyNED's frequency-domain preprocessing also finds biological parallels. The cochlea performs a biological analogue of the Short-Time Fourier Transform, decomposing sound into frequency bands along the basilar membrane (tonotopic organisation) (Fettiplace, 2023). Similarly, neurons in the primary visual cortex (V1) are tuned to specific spatial frequencies and orientations, effectively performing a local frequency decomposition of visual input. DyNED's use of FFT for images and STFT for speech mirrors these biological mechanisms, converting spatial or temporal patterns into frequency-domain representations before spike encoding.

The BSA encoder's filter-matching approach has parallels with synaptic filtering in biological systems, where dendritic morphology and synaptic dynamics act as temporal filters on incoming signals. TBR's threshold-based encoding mirrors the all-or-nothing firing behaviour of biological neurons, albeit in a simplified form. Delta encoding corresponds to change detection, a fundamental property of sensory neurons: many biological sensors are more responsive to changes in stimulation than to steady-state levels.

The biological plausibility of encoding schemes extends to learning rules as well. Three-factor learning rules, which modulate synaptic plasticity through a neuromodulatory signal analogous to dopamine, provide a biologically grounded alternative to backpropagation for training SNNs (Mazurek et al., 2025). DyNED's temporally structured spike trains are well-suited to such learning rules, as the precise timing information preserved by sigma-delta quantisation provides the temporal correlations that three-factor rules require to compute meaningful weight updates.

7.6.3 Neuromorphic Deployment Implications

The choice of encoding scheme has direct implications for deployment on neuromorphic hardware platforms such as Intel Loihi, IBM TrueNorth, and SpiNNaker (Ding et al., 2025). These platforms are optimised for sparse, event-driven computation, where energy consumption scales with the number of spikes processed rather than with clock cycles (Seekings et al., 2025). Recent frameworks such as NeuEdge (Imanov et al., 2026) demonstrate that hybrid temporal coding schemes can reduce spike communication by 4.7 times while achieving 312 times energy improvements over conventional GPU systems on platforms including Intel Loihi 2 and IBM TrueNorth.

Temporal coding schemes that produce sparse spike trains - where each spike carries significant information - are inherently better suited to neuromorphic hardware than dense rate codes. The energy cost per inference on platforms like Intel Loihi 2 scales directly with spike count (Hajizada et al., 2025), making sparse temporal codes more energy-efficient. DyNED's sigma-delta quantisation produces binary spike trains whose density depends on the signal dynamics, naturally yielding sparse representations for slowly varying inputs. The reversibility of DyNED's encoding is particularly valuable in neuromorphic deployments where verification of data integrity is required - the ability to reconstruct the original signal from on-chip spike representations provides an end-to-end validation mechanism that lossy encoders cannot offer.

However, DyNED's reliance on FFT/STFT preprocessing presents a deployment challenge. The frequency-domain transformation requires conventional floating-point arithmetic that is not natively supported by current neuromorphic chips. In a practical deployment, the STFT would need to be performed on a conventional co-processor (CPU or GPU) before feeding the resulting spike trains to the neuromorphic chip for classification. This hybrid architecture - conventional preprocessing followed by neuromorphic inference - is increasingly recognised as a practical approach (Imanov et al., 2026; Seekings et al., 2025).

The simpler encoders (Delta, Rate, Latency) can potentially be implemented directly on-chip, as they require only basic arithmetic operations (thresholding, comparison, random number generation) that map naturally to neuromorphic primitives. BSA and TBR, while computationally lightweight, currently rely on CPU-based implementations and would require custom digital logic for on-chip deployment.

The DyNED_c compression further enhances neuromorphic deployment potential by reducing the memory footprint of spike trains by 97.5-99.3%. On memory-constrained neuromorphic platforms, this compression enables the storage of larger

datasets or longer temporal sequences within limited on-chip memory. The lossless nature of DyNED_c ensures that decompressed spike trains are bit-identical to the originals, preserving the temporal structure that neuromorphic chips rely on for computation (Moqadasi et al., 2025). This is a significant advantage over lossy compression or irreversible encoding schemes, as it guarantees that no information is lost between the encoding stage and the neuromorphic inference stage.

7.7 Practical and Applied Considerations

The practical applications of this research span several domains where efficient neural processing and data handling are crucial. In image recognition applications, the DyNED encoder’s ability to achieve 88.07% accuracy on the CIFAR-10 dataset while maintaining low computational overhead makes it particularly suitable for resource-constrained environments such as mobile devices or embedded systems. The successful processing of natural images suggests potential applications in autonomous systems, surveillance, robotics, and real-time object recognition systems.

In speech recognition applications, the system’s achievement of 92.79% test accuracy on the Speech Commands dataset demonstrates its potential for voice-controlled interfaces and audio processing systems. The encoder’s ability to handle temporal data effectively, combined with its noise resilience through spectral gating, makes it particularly valuable for real-world audio applications where clean signals cannot be guaranteed. This could include applications in voice-activated devices, automated customer service systems, or accessibility tools for deaf and hard-of-hearing people.

Beyond conventional sensors, the pipeline naturally extends to event-driven sensors that produce data in spike form. On the neuromorphic CIFAR10-DVS dataset (data captured by Dynamic Vision Sensors that emit asynchronous events on pixel-intensity changes), the same encoder-compressor reached 76.3% test accuracy (79.0% with DyNED_c). On the Spiking Speech Commands dataset (audio captured through a biologically detailed cochlear model that produces asynchronous spike events), it reached 72.96% (72.4% with DyNED_c) - without any architectural change. This makes DyNED + DyNED_c relevant to deployments that directly consume event cameras, silicon cochleas, and other neuromorphic sensors increasingly used in always-on perception, low-power surveillance, and edge robotics, where conventional dense-tensor pipelines are inefficient, and a compressible spike representation simplifies on-device buffering and inter-component transmission.

The DyNED_c compression system provides practical data storage and transmission solutions for neuromorphic systems. Compression ratios of 9.6% for image data (at 256 quantisation levels) and 99.3% for speech data reduce storage requirements while maintaining high classification accuracy. This makes it particularly valuable for edge computing devices with limited storage capacity and bandwidth. The system's ability to efficiently compress spike trains could enable the broader deployment of neuromorphic computing in IoT devices and embedded systems.

GPU acceleration and Numba JIT compilation of the encoding process address practical concerns about resource efficiency. As shown in Table 5 and Table 23, GPU-accelerated encoders like DyNED consume significantly less system memory (40-50%) compared to CPU-only methods such as BSA and TBR (89-96%), making them more suitable for resource-constrained environments. While simpler encoders achieve faster per-sample encoding times, DyNED's frequency-domain pipeline produces richer representations that improve downstream classification accuracy.

The unified approach to handling static and dynamic data types simplifies system implementation and maintenance. Organisations looking to deploy neuromorphic computing solutions would benefit from not having to maintain separate systems for different data types, reducing development complexity and operational costs. The system's demonstrated compatibility with existing deep learning frameworks also eases integration into current AI infrastructure.

Robotics is a particularly promising domain for the techniques developed in this thesis. Autonomous robots must process visual and auditory data under strict latency, power, and memory constraints, requirements that align closely with the strengths of SNN-based systems. Recent work has demonstrated the viability of fully neuromorphic perception-to-control pipelines: Paredes-Vallés et al. (2024) deployed a five-layer SNN with 28,800 neurons on Intel's Loihi processor aboard a flying drone, processing raw event-camera data at 200 Hz while consuming only 7-12 mW of additional power beyond the 0.94 W idle draw (Figure 86). Similarly, Zanatta et al. (2024) showed that SNN-based reinforcement learning controllers can match or exceed ANN performance on shallow robotic control policies while offering inherent energy advantages. In the domain of robotic manipulation, Saini et al. (2026) demonstrated that an SNN using LIF neurons can solve inverse kinematics for a 3-DOF robotic arm in 1.50 ms compared to 20.15 ms for a conventional ANN. This 13× speed-up enables real-time trajectory generation with lower mean-squared error. Deploying learned policies on neurochip-driven edge robots introduces additional challenges: Kadokawa et al. (2026) proposed a distillation-based

framework that reduces FPNN-to-SNN conversion errors, enabling stable reinforcement learning policies on resource-constrained neurochips for visual servo and navigation tasks. DyNED’s frequency-domain encoding could serve as a front-end for such onboard perception pipelines, converting camera frames or microphone streams into compact spike representations suitable for neuromorphic co-processors. The DyNED_c compression capabilities are especially relevant for multi-sensor robotic platforms where onboard storage and inter-module bandwidth are limited. For example, a mobile robot transmitting encoded visual data from a peripheral sensor to a central processing unit over a low-bandwidth bus would benefit from DyNED_c’s lossless compression without sacrificing the temporal precision needed for real-time decision-making. Furthermore, the encoding pipeline processes each sample independently, allowing it to scale linearly with the number of sensors on a robotic platform without architectural changes.

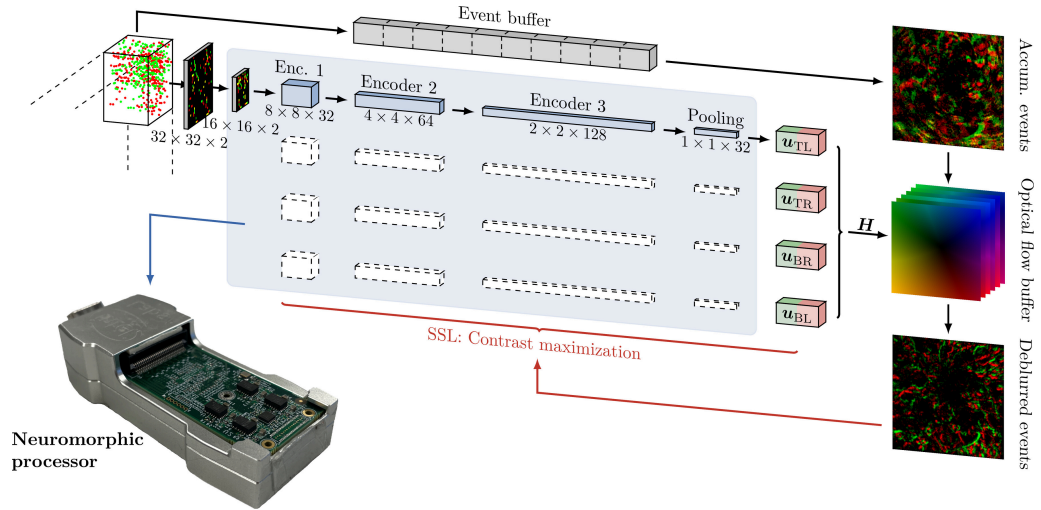


Figure 86: Fully neuromorphic vision-to-control pipeline for autonomous drone flight. A five-layer SNN processes raw event-camera data through three encoder stages and pooling to estimate ego-motion, which is then used for low-level flight control. The entire pipeline runs on Intel’s Loihi neuromorphic processor at 200 Hz (Paredes-Vallés et al., 2024).

These practical considerations suggest that the developed systems are not merely theoretical advances but offer tangible solutions for real-world neuromorphic computing applications. The balance between performance, efficiency, and resource utilisation makes these approaches valuable for practical implementations in resource-constrained environments.

7.8 Limitations of the Study

The current implementation of our research exhibits a few notable limitations that warrant acknowledgement. A significant limitation of the DyNED_c compression system is the lack of checksums, which makes verifying data integrity during transmission and storage complex. This omission could potentially lead to undetected data corruption, particularly in applications where data integrity is crucial for system performance.

The encoder’s training-time throughput depends on GPU acceleration; the implementation uses standard PyTorch operations, so it runs on any PyTorch-supported accelerator (NVIDIA CUDA, AMD ROCm, Apple MPS) or on CPU at reduced speed, and is not tied to a specific vendor. The more substantive portability gap is between any general-purpose accelerator and specialised neuromorphic hardware platforms (Intel Loihi, SpiNNaker, BrainChip Akida): each of those uses its own programming model, neuron primitives, and weight quantisation, so deploying the DyNED + DyNED_c pipeline on neuromorphic silicon requires a platform-specific conversion step rather than a direct port. This is a deployment-side limitation independent of the choice of training framework.

A crucial limitation is the encoder’s inability to handle real-time data streams because it lacks stream buffers. This constraint significantly impacts potential applications in real-time processing scenarios, such as live speech recognition or continuous video processing, where immediate data processing is essential.

While effective for its intended purpose, the DyNED_c compression system’s specialisation for binary spike data limits its broader applicability. The compression algorithm cannot be directly applied to other neural network data or continuous-valued signals without modification, potentially restricting its utility in hybrid systems or alternative neural network architectures.

On the already-spiking neuromorphic datasets (CIFAR10-DVS test accuracy 76.3% DyNED-only / 79.0% with DyNED_c; SSC test accuracy 72.96% / 72.4%; Table 42), accuracy is lower than on the conventional datasets. This is a direct consequence of routing already-discretised data through DyNED’s additional sigma-delta quantisation step, and is the trade-off accepted in exchange for the structured, compressible spike representation that the downstream DyNED_c stage requires. For applications whose only goal is best-in-class classification accuracy on neuromorphic inputs, a direct-spike pipeline remains preferable.

The DyNED quantisation level was included in the Optuna TPE search space as a categorical choice over the powers of two {16, 32, 64, 128, 256}. Powers of two were chosen

because they align with the bit-packed binary representation used by DyNED_c and because the level-vs-fidelity sweeps reported in Table 7 and Table 25 already cover the useful operating range with this grid. A finer-grained search was not used because each candidate level value requires regenerating the cached DyNED dataset. That cache-build cost multiplied across additional level points exceeded the compute budget available for the four-dataset evaluation. The level selected by TPE is therefore the best among five candidate values, not the optimum across the full integer range [16, 256] - intermediate settings such as 24, 48, 96, or 192 were not evaluated, so plateaus or non-monotonic interactions between encoder resolution and the downstream optimiser, regularisation, or firing-rate configuration may exist between the sampled points.

Additional limitations include:

- The optimisation process using TPE, while effective, requires significant computational resources for parameter tuning;
- The system's performance on massive datasets (beyond those tested) remains unverified;
- The current architecture lacks built-in mechanisms for handling distributed processing across multiple GPUs.

These limitations provide clear directions for future research and development to enhance the system's capabilities and broaden its applicability.

7.9 Directions for Future Research

This research has opened several promising avenues for future investigation and development in neuromorphic computing and spiking neural networks. The following sections outline key areas that warrant further exploration.

7.9.1 Neuromorphic Hardware

The evolution of neuromorphic hardware continues to follow Moore's law (Schaller, 1997), with increasing computational capabilities and efficiency. Recent developments have made these specialised computing platforms more accessible to researchers worldwide, notably SpiNNaker's move to cloud-based services. Figure 87 illustrates the historical progression of neuromorphic hardware development, showing the field's rapid advancement.

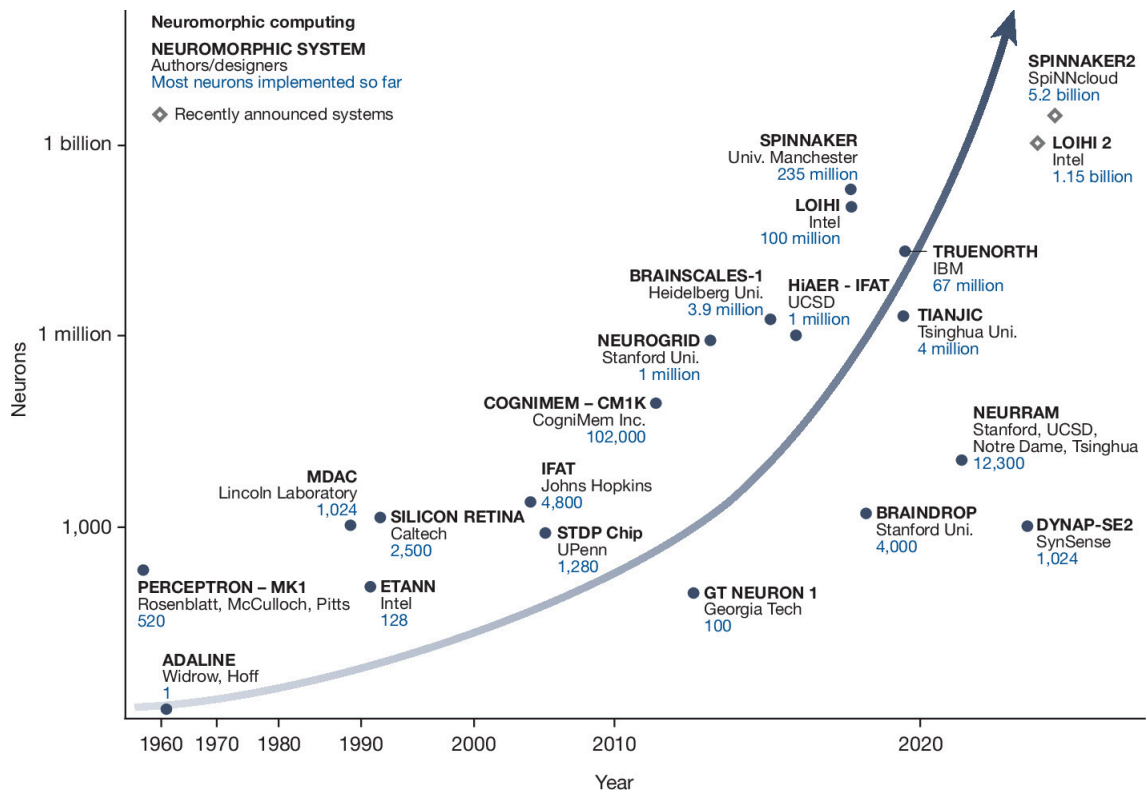


Figure 87: History of Neuromorphic hardware (Kudithipudi et al., 2025)

A significant trend in this space is the development of custom-designed chips incorporating ARM and RISC-V cores. As discussed in Vogginger et al. (2024), these architectures offer promising directions for enhancing the efficiency and scalability of neuromorphic systems. Integrating traditional computing architectures with neuromorphic principles suggests a hybrid approach that could bridge the gap between conventional and brain-inspired computing.

7.9.2 Streaming Spikes

The future of spike-based computation lies in developing efficient streaming capabilities for continuous data processing. Research should focus on creating dependable systems capable of real-time encoding and decoding continuous spike streams while maintaining data integrity and computational efficiency. This involves developing new methodologies for handling continuous data in neuromorphic systems, including adaptive buffer management and real-time spike processing algorithms. The challenge lies in balancing the need for immediate processing with the constraints of hardware resources and energy consumption. Although the current compression technique would work with a data stream, it needs some buffer to process the next n values.

7.9.3 Compressed Stream Data Transmission

Building on the compression techniques developed in this thesis, future research in compressed stream data transmission should focus on developing specialised protocols optimised for spike-based data. This includes developing optimal mechanisms for real-time compression and decompression of spike streams, implementing more robust error-correction methods, and optimising bandwidth utilisation in neuromorphic systems with small footprints. The goal is to enable high-throughput transmission of spike data while maintaining the temporal precision crucial for neuromorphic computing. This research direction could significantly improve the scalability and practical implementation of large-scale neuromorphic systems.

7.9.4 Real-Time SNN Learning

The development of real-time learning capabilities in SNNs represents a crucial frontier for future research. Real-time capabilities should involve developing adaptive learning algorithms that process and learn from streaming data while maintaining efficiency and accuracy. Research should explore methods for dynamically adjusting learning parameters based on incoming data characteristics, integrating compression techniques with learning mechanisms, and developing energy-efficient learning strategies suitable for neuromorphic hardware implementation. Such advancements could enable SNNs to adapt and learn continuously from their environment, more closely mimicking biological systems.

7.9.5 Deeper SNN Architectures

The classification results in Table 15 show that state-of-the-art SNNs employing deeper architectures with residual connections substantially outperform the shallow Conv+DyNED+cAdLIF+TET topology used in this work. Adopting deeper backbones such as ResNet-19 with temporally efficient training (Deng et al., 2022) or spiking transformer architectures (Zhang et al., 2024; Zhou et al., 2023) would likely narrow the accuracy gap while retaining DyNED’s encoding advantages. Integrating batch normalisation through time (Zheng et al., 2021) and hybrid conversion-plus-fine-tuning strategies (Rathi et al., 2020) are additional avenues that could improve downstream classification without altering the encoding pipeline itself.

7.10 Ethical and Societal Implications

As neuromorphic computing moves from research prototypes toward practical deployment, it is important to consider the broader ethical and societal implications of these technologies.

7.10.1 Energy Efficiency and Environmental Sustainability

One of the primary motivations for neuromorphic computing is its potential for energy-efficient inference. Neuromorphic processors such as Intel Loihi exploit event-driven, asynchronous computation to achieve substantially lower power consumption than conventional GPU-based deep learning (Davies et al., 2018). Benchmarking studies have demonstrated that analogue in-memory computing platforms, a key neuromorphic paradigm, can achieve up to three orders of magnitude lower energy costs than conventional CPU and GPU hardware for inference tasks (Kösters et al., 2023). The DyNED encoding pipeline developed in this thesis complements this advantage: its compact spike representations and DyNED_c compression ratios of 97.5-99.3% reduce the volume of data that must be stored and transmitted, further lowering the energy cost of downstream processing. As the environmental impact of large-scale AI training and inference receives increasing scrutiny - with estimates suggesting that AI-related electricity consumption could grow to rival that of entire countries (De Vries, 2023) - neuromorphic approaches offer a pathway toward more sustainable computing for always-on edge applications such as environmental monitoring, wearable health devices, and smart infrastructure (Muir & Sheik, 2025).

7.10.2 Privacy Considerations

Neuromorphic sensors such as event cameras produce sparse, asynchronous data streams that encode brightness changes rather than capturing full image frames, which inherently reduces the visual information available for identifying individuals (Cimarelli et al., 2025). When combined with on-device SNN processing, neuromorphic systems can perform classification and decision-making at the edge without transmitting raw sensory data to centralised servers, thereby improving data privacy. Zhang et al. (2025) demonstrated this principle with a privacy-preserving bullying detection system that uses spiking neural networks with neuromorphic vision sensors, achieving 80.8% mean average precision while safeguarding personal privacy through the inherent properties of spatiotemporal spiking signals. The DyNED_c compression system developed in this thesis could further support privacy-preserving architectures by enabling efficient on-

device storage of encoded spike data, minimising the need for cloud-based processing. However, always-on neuromorphic sensors also raise concerns about continuous surveillance. Deep learning reconstruction methods can recover intensity images from event streams, enabling facial recognition, and individuals can be re-identified from event data through their distinctive walking patterns (gait) (Adra et al., 2025). Responsible deployment should therefore incorporate privacy-by-design principles, including minimising data retention and ensuring transparent data governance.

7.10.3 Bias, Fairness, and Accessibility

Neuromorphic systems inherit many of the same bias and fairness challenges as conventional machine learning: models trained on unrepresentative datasets will produce disparate outcomes across demographic groups. The spike-based encoding used in this thesis neither introduces nor mitigates such biases, as it operates on signal representations rather than semantic content. However, the temporal encoding schemes used in SNNs may introduce novel failure modes that are not yet well characterised in the fairness literature, and explainability methods for spiking networks remain immature compared to those available for conventional deep learning. Bitar et al. (2023) adapted integrated gradients and related attribution methods to the spiking domain, representing early progress toward interpretable SNN models, but standardised fairness evaluation protocols for neuromorphic systems are still lacking. Additionally, as Muir & Sheik (2025) observes, neuromorphic hardware remains in an early commercial phase, with platforms concentrated in a small number of research institutions and companies. Translating bioinspired computing principles into practical and scalable hardware technologies remains a central challenge, requiring interdisciplinary convergence across materials science, device physics, and computer science (Wang et al., 2026). This concentration, combined with proprietary toolchains and limited software ecosystems, risks widening the gap between well-resourced institutions and those with fewer resources. Efforts to democratise access through open-source SNN frameworks and cloud-based neuromorphic services will be important for ensuring equitable participation in this field.

7.10.4 Dual-Use Considerations

The combination of low-power neuromorphic processing with always-on sensing capabilities creates dual-use concerns. The same properties that make neuromorphic systems attractive for assistive robotics (Paredes-Vallés et al., 2024), environmental monitoring, and healthcare - including low power consumption, real-time processing, and compact

form factors - also enable persistent surveillance and autonomous target tracking (Adra et al., 2025). Event cameras have already been deployed in automotive sensing, security monitoring, and drone navigation (Cimarelli et al., 2025), applications that can serve both civilian and military purposes. As demonstrated in this thesis, DyNED can efficiently encode both visual and auditory data into compact spike representations suitable for real-time processing on edge devices. These capabilities could be applied to both beneficial and harmful purposes. As neuromorphic technologies approach commercial maturity (Muir & Sheik, 2025), the research community and policymakers should develop governance frameworks that balance innovation with appropriate safeguards, including use-case restrictions for high-risk deployments and transparency requirements for neuromorphic sensing systems in public spaces.

7.11 Returning to the Research Problem

The RPS, set out in Section 1.1.1, reads:

There is currently no published method that performs reconstruction-faithful encoding and decoding of spike trains uniformly across both static and dynamic data and both conventional and neuromorphic sources, nor a lossless compressor for the resulting binary spike streams - which together prevent SNN deployment in heterogeneous, energy-constrained real-world pipelines.

This thesis addressed that limitation through DyNED (encoder-decoder with iterative error feedback) and DyNED_c (binary-stream lossless compressor), validated across four datasets spanning conventional and neuromorphic data and static and dynamic modalities. The eight research questions defined in Section 1.3.2 and their corresponding hypotheses in Section 1.3.3 tested specific claims about reconstruction fidelity (RQ1 and RQ2), cross-modality encoding (RQ3 and RQ8), classifier performance against matched baselines (RQ4 and RQ7), and lossless compression (RQ5 and RQ6) - each supported by the experimental evidence summarised in Table 41 and Table 42.

The encoding barrier identified by the RPS has been lowered but not removed. DyNED's reconstruction fidelity exceeds $R^2 = 0.95$ across all modalities, validating the closed encoding-decoding loop as a viable foundation for spike-based pipelines (RQ1). DyNED_c compression is lossless on all four datasets while preserving classification accuracy within 2 percentage points of the uncompressed pipeline (RQ5 and RQ6), and the same encoder-compressor operates without architectural change on inputs as diverse as natural images, raw audio waveforms, DVS event streams, and cochlear spike events

(RQ3 and RQ8). Three gaps remain that the research problem did not explicitly anticipate but that the experimental work has revealed: routing already-spiking neuromorphic data through DyNED’s additional sigma-delta quantisation step incurs a measurable accuracy cost relative to direct-spike pipelines (Section 7.8); the encoder lacks the streaming buffer needed for true real-time deployment (Section 7.8); and porting the trained pipeline to specialised neuromorphic silicon (Loihi, SpiNNaker) requires a platform-specific conversion step that this thesis did not perform (Section 7.8). These define the agenda for the next phase of work outlined in Section 7.9.

7.12 Concluding Remarks

This thesis has developed the Dynamic Neural Encoder and Decoder (DyNED) and its compressed variant DyNED_c, addressing the fundamental challenge of converting real-world data into spike-based representations while preserving information integrity. The system integrates frequency-domain transformation, sigma-delta quantisation with iterative error feedback, and binary-optimised compression into a unified encoding pipeline for Spiking Neural Networks.

The effectiveness of DyNED has been validated across four datasets spanning both conventional and neuromorphic data and both static and dynamic modalities. On CIFAR-10, the system achieved 88.07% classification accuracy, approximately 1.5 percentage points above a matched non-spiking CNN baseline (86.62%), confirming that spike-based representations can outperform their non-spiking counterparts on identical input features (Table 15). On Speech Commands, DyNED achieved 92.79% test accuracy, exceeding a matched non-spiking CNN baseline operating on the same STFT log-magnitude features (76.59%) by approximately 16 percentage points, demonstrating that DyNED’s sigma-delta encoding is particularly effective for temporal data (Table 33). Extending to the neuromorphic counterpart datasets, the same encoder-compressor reached 76.3% on CIFAR10-DVS (event-based vision) and 72.96% on Spiking Speech Commands (cochlear-spike audio), with DyNED_c variants at 79.0% and 72.4% respectively (Table 42). The headline cross-dataset takeaway is the H8 result: the same encoder-compressor operates without structural change across all four cells of the {conventional, neuromorphic} $\times$ {static, dynamic} matrix, establishing DyNED + DyNED_c as a modality-agnostic preprocessing layer. The DyNED_c compression system achieved a compression ratio of 0.904 (9.6% space saved) for image data at 256 quantisation levels and 0.007 (99.3% space saved) for speech data, while maintaining classification accuracy within 2 percentage points of the uncompressed pipeline (Table 12, Table 29).

These findings directly inform the future research directions outlined in this chapter. The high reconstruction fidelity of DyNED ($R^2 \geq 0.95$) establishes the signal-processing foundation needed for streaming spike applications, where continuous data must be encoded in real time without sacrificing temporal precision. The DyNED_c compression ratios suggest that compressed spike transmission is viable for bandwidth-constrained neuromorphic systems. However, the addition of checksums and error correction mechanisms will be necessary for deployment in safety-critical applications. The demonstrated effectiveness of the error feedback mechanism - producing up to 74% more spikes than uniform quantisation at low quantisation levels (Table 26) - provides a starting point for adaptive encoding schemes that could adjust quantisation parameters in response to incoming data characteristics during real-time SNN learning.

The classification gap between the Conv+DyNED+cAdLIF+TET architecture used in this work and deeper SNN models reported in the literature (Table 15) suggests that DyNED's encoding advantages could be further amplified by integration with modern SNN architectures that employ residual connections, batch normalisation through time, or spiking transformer designs. Similarly, the successful deployment of neuromorphic systems for robotics (Paredes-Vallés et al., 2024) and edge computing (Muir & Sheik, 2025) suggests that DyNED's compact spike representations and DyNED_c's compression capabilities are well-suited for resource-constrained environments where both energy efficiency and data throughput are critical.

This work bridges the gap between biological and artificial neural networks by demonstrating that principles from signal processing and information theory can be combined with biologically inspired spike-based computation to achieve practical performance on real-world tasks. The frameworks and methodologies developed here provide a foundation for future research into real-time neuromorphic processing, compressed spike data transmission, and the deployment of spiking neural networks on dedicated neuromorphic hardware.

Bibliography

- Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5), 303–304.
- Abeles, M. (1991). *Corticonics: Neural circuits of the cerebral cortex*. Cambridge University Press.
- Adra, M., Melcarne, S., Mirabet-Herranz, N., & Dugelay, J.-L. (2025). Event-based solutions for human-centered applications: a comprehensive review. *Frontiers in Signal Processing*, 5, 1585242.
- Aghajan, Z. M., Kreiman, G., & Fried, I. (2023). Minute-scale periodicity of neuronal firing in the human entorhinal cortex. *Cell Reports*, 42(11).
- Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019,). Optuna: A Next-generation Hyperparameter Optimization Framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Al Roumi, F., Planton, S., Wang, L., & Dehaene, S. (2023). Brain-imaging evidence for compression of binary sound sequences in human memory. *Elife*, 12, e84376. <https://doi.org/10.7554/eLife.84376>
- Allen-Zhu, Z., Gelashvili, R., Micali, S., & Shavit, N. (2014). Sparse sign-consistent Johnson-Lindenstrauss matrices: Compression with neuroscience-based constraints. *Proceedings of the National Academy of Sciences*, 111(47), 16872–16876. <https://doi.org/10.1073/pnas.1419100111>
- Andrew, A. M. (2003). Spiking neuron models: Single neurons, populations, plasticity. *Kybernetes*, 32(7/8).
- Baietto, A., Stewart, C., & Bihl, T. J. (2025). Dataset assembly for training Spiking Neural Networks. *Neurocomputing*, 622, 129207. <https://doi.org/https://doi.org/10.1016/j.neucom.2024.129207>
- Basak, K., Mishra, N., & Chang, H.-T. (2023). TranStutter: A Convolution-Free Transformer-Based Deep Learning Method to Classify Stuttered Speech Using 2D Mel-Spectrogram Visualization and Attention-Based Feature Representation. *Sensors*, 23(19), 8033.
- Bear, M. F., Connors, B. W., & Paradiso, M. A. (2007). *Neuroscience* (Vol. 2). Lippincott Williams & Wilkins.

- Benedetto, J., Powell, A., & Yilmaz, O. (2006). Sigma-delta (/spl Sigma/ /spl Delta/) quantization and finite frames. *IEEE Transactions on Information Theory*, 52(5), 1990–2005. <https://doi.org/10.1109/TIT.2006.872849>
- Benna, M. K., & Fusi, S. (2021). Place cells may simply be memory cells: Memory compression leads to spatial tuning and history dependence. *Proceedings of the National Academy of Sciences*, 118(51), e2018422118. <https://doi.org/10.1073/pnas.2018422118>
- Bensalem, H., Blaqui re, Y., & Savaria, Y. (2023). An Efficient OpenCL-Based Implementation of a SHA-3 Co-Processor on an FPGA-Centric Platform. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(3), 1144–1148. <https://doi.org/10.1109/TCSII.2022.3223179>
- Benuskova, L., & Kasabov, N. (2008). Modeling brain dynamics using computational neurogenetic approach. *Cognitive Neurodynamics*, 2(4), 319.
- Bergstra, J., Bardenet, R., Bengio, Y., & K egl, B. (2011). Algorithms for Hyper-Parameter Optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, & K. Weinberger (Eds.), *Advances in Neural Information Processing Systems: Vol. 24. Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf
- Beyeler, M., Carlson, K. D., Chou, T.-S., Dutt, N., & Krichmar, J. L. (2015). CARLsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks. *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–8. <https://doi.org/10.1109/IJCNN.2015.7280424>
- Bishop, M. (2022, June). RFC 9114: HTTP/3. RFC Editor. <https://datatracker.ietf.org/doc/html/rfc9114>
- Bitar, A., Rosales, R., & Paulitsch, M. (2023). Gradient-based feature-attribution explainability methods for spiking neural networks. *Frontiers in Neuroscience*, 17, 1153999.
- Bittar, A., & Garner, P. N. (2022). A surrogate gradient spiking baseline for speech command recognition. *Frontiers in Neuroscience*, 16, 865897.
- Bohte, S. M. (2004). The evidence for neural information processing with precise spike-times: A survey. *Natural Computing*, 3(2), 195–206.
- Brandenburg, K. (1999, September). MP3 and AAC explained. *Audio Engineering Society Conference: 17th International Conference: High-Quality Audio Coding*. <http://www.aes.org/e-lib/browse.cfm?elib=8079>
- Brette, R., & Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, 94(5), 3637–3642.

- Bruce, G. (2010, November). Stanford News Center. <https://med.stanford.edu/news/all-news/2010/11/new-imaging-method-developed-at-stanford-reveals-stunning-details-of-brain-connections.html>
- Buhmann, M. D. (2000). Radial basis functions. *Acta Numerica*, 9, 1–38. <https://doi.org/10.1017/S0962492900000015>
- Cai, S., Li, P., Su, E., Liu, Q., & Xie, L. (2022). A Neural-Inspired Architecture for EEG-Based Auditory Attention Detection. *IEEE Transactions on Human-Machine Systems*, 52(4), 668–676.
- Canonical. (2004, October). *Ubuntu Operating System*. Canonical. <https://ubuntu.com/>
- Capecci, E., Doborjeh, Z. G., Mammone, N., La Foresta, F., Morabito, F. C., & Kasabov, N. (2016). Longitudinal study of alzheimer's disease degeneration through EEG data analysis with a NeuCube spiking neural network model. *Neural Networks (IJCNN), 2016 International Joint Conference on*, 1360–1366.
- Carnevale, N. T., & Hines, M. L. (2006). *The NEURON book*. Cambridge University Press.
- Chen, X., Yang, Q., Wu, J., Li, H., & Tan, K. C. (2024). A hybrid neural coding approach for pattern recognition with spiking neural networks. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 46(5), 3064–3078.
- Cherian, R., & others. (2024). Unleashing the Potential of Spiking Neural Networks for Epileptic Seizure Detection: A Comprehensive Review. *Neurocomputing*, 127934.
- Choi, S.-H., Jeong, G., Kim, Y.-B., & Cho, Z.-H. (2020). Proposal for human visual pathway in the extrastriate cortex by fiber tracking method using diffusion-weighted MRI. *Neuroimage*, 220, 117145.
- Cimarelli, C., Millan-Romera, J. A., Voos, H., & Sanchez-Lopez, J. L. (2025). Hardware, algorithms, and applications of the neuromorphic vision sensor: a review. *Sensors (Basel, Switzerland)*, 25(19), 6208.
- Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90), 297–301.
- Cramer, B., Stradmann, Y., Schemmel, J., & Zenke, F. (2020). The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7), 2744–2757.
- Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., ... Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1), 82–99. <https://doi.org/10.1109/MM.2018.112130359>

- Davison, A. P., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., & Yger, P. (2008). PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2.
- De Vries, A. (2023). The growing energy footprint of artificial intelligence. *Joule*, 7(10), 2191–2194.
- Deckers, L., Van Damme, L., Van Leekwijck, W., Tsang, I. J., & Latré, S. (2024). Co-learning synaptic delays, weights and adaptation in spiking neural networks. *Frontiers in Neuroscience*, 18, 1360300.
- Delbruck, T., & Lichtsteiner, P. (2007). Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. *2007 IEEE International Symposium on Circuits and Systems*, 845–848.
- Deng, S., Li, Y., Zhang, S., & Gu, S. (2022,). Temporal efficient training of spiking neural network via gradient re-weighting. *International Conference on Learning Representations*.
- Deng, T., Yang, C., Liu, W., Zhao, C., Li, R., & Huang, Y. (2026). TranSpikformer: Enhanced Transformer-based Spiking Neural Networks with Uniform Attention Augmentation and Depthwise Convolutional Positional Encoding. *Knowledge-Based Systems*, 115258.
- Dhoble, K., Nuntalid, N., Indiveri, G., & Kasabov, N. (2012). Online spatio-temporal pattern recognition with evolving spiking neural networks utilising address event representation, rank order, and temporal spike learning. *Neural Networks (IJCNN), The 2012 International Joint Conference on*, 1–7.
- Diehl, P. U., & Cook, M. (2014). Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware. *Neural Networks (IJCNN), 2014 International Joint Conference on*, 4288–4295.
- Ding, J., Yu, Z., Liu, J. K., & Huang, T. (2025). Neuromorphic computing paradigms enhance robustness through spiking neural networks. *Nature Communications*. <https://doi.org/10.1038/s41467-025-65197-x>
- Ding, X., Zhang, Y., Ge, Y., Zhao, S., Song, L., Yue, X., & Shan, Y. (2024). UniRepLKNet: A Universal Perception Large-Kernel ConvNet for Audio Video Point Cloud Time-Series and Image Recognition. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5513–5524.
- Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7).
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179–211. [https://doi.org/https://doi.org/10.1016/0364-0213\(90\)90002-E](https://doi.org/https://doi.org/10.1016/0364-0213(90)90002-E)

- Eshraghian, J. K., Ward, M., Neftci, E. O., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., & Lu, W. D. (2023b). Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*.
- Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., & Lu, W. D. (2023a). Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111(9), 1016–1054.
- Evans, A. C., Collins, D. L., Mills, S., Brown, E., Kelly, R., & Peters, T. M. (1993). 3D statistical neuroanatomical models from 305 MRI volumes. *Nuclear Science Symposium and Medical Imaging Conference, 1993., 1993 IEEE Conference Record.*, 1813–1817.
- Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., & Tian, Y. (2021). Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems*, 34, 21056–21069.
- Fettiplace, R. (2023). Cochlear tonotopy from proteins to perception. *Bioessays*, 45(8), e2300058. <https://doi.org/10.1002/bies.202300058>
- Fiasché, M., Schliebs, S., & Nobili, L. (2012). Integrating neural networks and chaotic measurements for modelling epileptic brain. *Artificial Neural Networks and Machine Learning–ICANN 2012*, 653–660.
- Florian, R. V. (2012). The chronotron: a neuron that learns to fire temporally precise spike patterns. *Plos One*, 7(8), e40233.
- Fois, A., & Girau, B. (2023). Enhanced representation learning with temporal coding in sparsely spiking neural networks. *Frontiers in Computational Neuroscience*, 17. <https://doi.org/10.3389/fncom.2023.1250908>
- Fourcaud-Trocmé, N., Hansel, D., Van Vreeswijk, C., & Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *Journal of Neuroscience*, 23(37), 11628–11640.
- Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The spinnaker project. *Proceedings of the IEEE*, 102(5), 652–665.
- Fusi, S., Annunziato, M., Badoni, D., Salamon, A., & Amit, D. J. (2000). Spike-driven synaptic plasticity: theory, simulation, VLSI implementation. *Neural Computation*, 12(10), 2227–2258.
- Gardner, B., & Grüning, A. (2016). Supervised Learning in Spiking Neural Networks for Precise Temporal Encoding. *Plos One*, 11(8), e161335.
- Gerstner, W., Sprekeler, H., & Deco, G. (2012). Theory and simulation in neuroscience. *Science*, 338(6103), 60–65.

- Gewaltig, M.-O., & Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia*, 2(4), 1430.
- Gollahalli, A. R. (2015). *Brain-computer interfaces for virtual Quadcopters based on a spiking-neural network architecture - NeuCube* [Thesis]. <http://hdl.handle.net/10292/9249>
- Gollahalli, A. R. (2019, October). *Spike - Spiking Neural Network Encoders*. GitHub. <https://github.com/akshaybabloo/Spikes>
- Golomb, S. (1966). Run-length encodings (corresp.). *IEEE Transactions on Information Theory*, 12(3), 399–401.
- Gong, Y., Chung, Y.-A., & Glass, J. (2021b). Ast: Audio spectrogram transformer. *Arxiv Preprint Arxiv:2104.01778*.
- Gong, Y., Chung, Y.-A., & Glass, J. (2021a). Psla: Improving audio tagging with pretraining, sampling, labeling, and aggregation. *IEEE/ACM Transactions on Audio, Speech, And Language Processing*, 29, 3292–3306.
- Gong, Y., Lai, C.-I., Chung, Y.-A., & Glass, J. (2022). Ssast: Self-supervised audio spectrogram transformer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(10), 10699–10709.
- Goodman, D. F., & Brette, R. (2009). The brian simulator. *Frontiers in Neuroscience*, 3(2), 192.
- Gray, R. M., & Neuhoff, D. L. (1998). Quantization. *IEEE Transactions on Information Theory*, 44(6), 2325–2383.
- Gu, F., Sng, W., Hu, X., & Yu, F. (2021). EventDrop: data augmentation for event-based learning. *Corr*. <https://arxiv.org/abs/2106.05836>
- Guo, Y., Chen, Y., Liu, X., Peng, W., Zhang, Y., Huang, X., & Ma, Z. (2024). Ternary spike: Learning ternary spikes for spiking neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(11), 12244–12252.
- Guo, Y., Huang, X., & Ma, Z. (2023). Direct learning-based deep spiking neural networks: a review. *Frontiers in Neuroscience*, 17, 1209795.
- Gütig, R., & Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nature Neuroscience*, 9(3), 420.
- Hajizada, E., Rager, D., Shea, T., Campos-Macias, L., Wild, A., Hüllermeier, E., Sandamirskaya, Y., & Davies, M. (2025). Real-time Continual Learning on Intel Loihi 2. *Arxiv Preprint Arxiv:2511.01553*.
- Han, T. D., & Abdelrahman, T. S. (2011). hiCUDA: High-Level GPGPU Programming. *IEEE Transactions on Parallel and Distributed Systems*, 22(1), 78–90. <https://doi.org/10.1109/TPDS.2010.62>

- Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362.
- Heister, H.-W., Polk, H., & Rusam, B. (2022). *Word Art+ Gesture Art= Tone Art. The Relationship Between the Vocal and the Instrumental in Different Arts*. Springer.
- Henkes, A., Eshraghian, J. K., & Wessels, H. (2024). Spiking neural networks for nonlinear regression. *Royal Society Open Science*, 11(5), 231606.
- Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4), 500–544.
- Hong, C., Yuan, M., Zhang, M., Wang, X., Zhang, C., Wang, J., Pan, G., & Tang, H. (2024). SPAIC: A Spike-Based Artificial Intelligence Computing Framework. *IEEE Computational Intelligence Magazine*, 19(1), 51–65. <https://doi.org/10.1109/MCI.2023.3327842>
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 359–366.
- Hough, M., De Garis, H., Korkin, M., Gers, F., & Nawa, N. E. (1999, September). SPIKER: Analog waveform to digital spiketrain conversion in ATR's artificial brain (cam-brain) project. *International Conference on Robotics and Artificial Life*.
- How does the larynx work?*. (2024, March). <https://www.informedhealth.org/how-does-the-larynx-work.html>
- Howard, P. G., & Vitter, J. S. (1994). Arithmetic Coding for Data Compression. *Proceedings of the IEEE*, 82(6), 857–865. <https://doi.org/10.1109/5.286189>
- Hu, Y., Zheng, Q., Jiang, X., & Pan, G. (2023). Fast-SNN: Fast Spiking Neural Network by Converting Quantized ANN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(12), 14546–14562. <https://doi.org/10.1109/TPAMI.2023.3275769>
- Huang, P.-Y., Xu, H., Li, J., Baevski, A., Auli, M., Galuba, W., Metze, F., & Feichtenhofer, C. (2022). Masked autoencoders that listen. *Advances in Neural Information Processing Systems*, 35, 28708–28720.
- Hubel, D. H., Wensveen, J., & Wick, B. (1995). *Eye, brain, and vision*. Scientific American Library New York.
- Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9), 1098–1101.
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.

- Hussain, I., & Thounaojam, D. M. (2024). An Extensive Review of the Supervised Learning Algorithms for Spiking Neural Networks. In M. D. Borah, D. S. Laiphrakpam, N. Auluck, & V. E. Balas (Eds.), *Big Data, Machine Learning, and Applications: Big Data, Machine Learning, and Applications*.
- Imanov, O. Y. L., Kulali, D. U., Yilmaz, T., Erisken, D., & Turhan, R. I. (2026). Energy-Efficient Neuromorphic Computing for Edge AI: A Framework with Adaptive Spiking Neural Networks and Hardware-Aware Optimization. *Arxiv Preprint Arxiv:2602.02439*.
- Indiveri, G. (2003). A low-power adaptive integrate-and-fire neuron circuit. *Circuits and Systems, 2003. Iscas'03. Proceedings of the 2003 International Symposium on*, 4, IV–IV.
- ITU. (2009, October). M.1677: International Morse code. International Telecommunication Union. <https://www.itu.int/rec/R-REC-M.1677-1-200910-I/>
- ITU. (2025, October). *Statistics*. International Telecommunication Union. <https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx>
- ITU-T. (1988). *Recommendation G.711: Pulse Code Modulation (PCM) of Voice Frequencies* [ITU-T Recommendation].
- Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6), 1569–1572.
- Joo, J., Jeong, T., & Hwang, S. (2024). Brain-Streams: fMRI-to-Image Reconstruction with Multi-modal Guidance. *Arxiv Preprint Arxiv:2409.12099*. <https://arxiv.org/abs/2409.12099>
- Juang, B. H., & Rabiner, L. R. (1991). Hidden Markov models for speech recognition. *Technometrics*, 33(3), 251–272.
- Kadokawa, Y., Yamanokuchi, T., Fernandez, A. R., Homma, T., & Matsubara, T. (2026). Distilled Iterative Value Conversion: Reducing FPNN-to-SNN Conversion Errors via Distillation in Reinforcement Learning for Neurochip-Driven Edge Robots. *IEEE Access*.
- Kanan, C., & Cottrell, G. W. (2012). Color-to-grayscale: does the method matter in image recognition?. *Plos One*, 7(1), e29740.
- Kandel, E. R., Schwartz, J. H., Jessell, T. M., Siegelbaum, S. A., & Hudspeth, A. J. (2000). *Principles of neural science* (Vol. 4). McGraw-hill New York.
- Kasabov, N. K. (2014). NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, 52, 62–76.

- Kasabov, N., Dhoble, K., Nuntalid, N., & Indiveri, G. (2013). Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. *Neural Networks*, 41, 188–201.
- Khan, A. H., Cao, X., Luo, C., Zhang, S., Guo, W., Katsikis, V. N., & Li, S. (2025). Spiking Neural Networks: A Comprehensive Survey of Training Methodologies, Hardware Implementations and Applications. *Artificial Intelligence Science and Engineering*, 1(3), 175–207. <https://doi.org/10.23919/AISE.2025.000013>
- Kingma, D. P. (2014). Adam: A method for stochastic optimization. *Arxiv Preprint Arxiv:1412.6980*.
- Klein, B., & Rossin, D. (1999). Data quality in neural network models: effect of error rate and magnitude of error on predictive accuracy. *Omega*, 27(5), 569–582. [https://doi.org/https://doi.org/10.1016/S0305-0483\(99\)00019-5](https://doi.org/https://doi.org/10.1016/S0305-0483(99)00019-5)
- Kong, Q., Cao, Y., Iqbal, T., Wang, Y., Wang, W., & Plumbley, M. D. (2020). Panns: Large-scale pretrained audio neural networks for audio pattern recognition. *IEEE/ACM Transactions on Audio, Speech, And Language Processing*, 28, 2880–2894.
- Krizhevsky, A., Hinton, G., & others. (2009). *Learning multiple layers of features from tiny images*.
- Kudithipudi, D., Schuman, C., Vineyard, C. M., Pandit, T., Merkel, C., Kubendran, R., Aimone, J. B., Orchard, G., Mayr, C., Benosman, R., & others. (2025). Neuromorphic computing at scale. *Nature*, 637(8047), 801–812.
- Kuo, S. M., & Gan, W.-S. (2005). *Digital signal processors: Architectures, implementations, and applications*. Pearson-Prentice Hall.
- Kösters, D., Kortman, B., Boybat, I., Ferro, E., Dolas, S., Austri, R., Kwisthout, J., Hilgenkamp, H., Rasing, T., & Riel, H. (2023). Benchmarking energy consumption and latency for neuromorphic computing in condensed matter and particle physics APL Mach. *Learn*, 1(16101), 2209–10481.
- Lapicque, L. (1907). Recherches quatitatives sur l'excitation electrique des nerfs traitee comme polarisation. *J. Physiol. Pathol. Gen.*, 9, 620–635.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. <https://doi.org/10.1109/5.726791>
- Lee, D., Li, Y., Kim, Y., Xiao, S., & Panda, P. (2025). Spiking transformer with spatial-temporal attention. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13948–13958.

- Li, H., Liu, H., Ji, X., Li, G., & Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in Neuroscience*, 11, 244131.
- Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128 $\times$ 128 120 dB 15 μ s Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid-State Circuits*, 43(2), 566–576.
- Lindholm, E., Nickolls, J., Oberman, S., & Montrym, J. (2008). NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro*, 28(2), 39–55. <https://doi.org/10.1109/MM.2008.31>
- Liu, C. et al. (2024). Direct training high-performance deep spiking neural networks: a review of theories and methods. *Frontiers in Neuroscience*, 18, 1383844. <https://doi.org/10.3389/fnins.2024.1383844>
- Liu, H. et al. (2024). A Convolutional Spiking Neural Network Accelerator with the Sparsity-Aware Memory and Compressed Weights. *2024 IEEE 35th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, , 163–171. <https://doi.org/10.1109/ASAP61560.2024.00041>
- Liu, Q. et al. (2024). *LitE-SNN: Designing Lightweight and Efficient Spiking Neural Network through Spatial-Temporal Compressive Network Search and Joint Optimization*. <https://arxiv.org/abs/2401.14652>
- Loeffler, J., Cañal-Bruland, R., Schroeger, A., Tolentino-Castro, J. W., & Raab, M. (2018). Interrelations between temporal and spatial cognition: The role of modality-specific processing. *Frontiers in Psychology*, 9, 2609.
- Loshchilov, I., & Hutter, F. (2017). Fixing Weight Decay Regularization in Adam. *Corr.* <http://arxiv.org/abs/1711.05101>
- Loyola-Jara, B., Fernández-Rodríguez, G., & Baladron, J. (2026). Evolving spiking neural networks: the role of neuron models and encoding schemes in neuromorphic learning. *Frontiers in Neuroscience*, 20. <https://doi.org/10.3389/fnins.2026.1697163>
- Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9), 1659–1671. [https://doi.org/10.1016/S0893-6080\(97\)00011-7](https://doi.org/10.1016/S0893-6080(97)00011-7)
- Mai, J. K., Majtanik, M., & Paxinos, G. (2015). *Atlas of the human brain*. Academic Press.
- Mastroianni, G., & Milovanovic, G. V. (2008). *Interpolation Processes Basic Theory and Applications*. Springer Berlin Heidelberg.
- Mazurek, S., Caputa, J., Argasiński, J. K., & Wielgosz, M. (2025). Three-factor learning in spiking neural networks: An overview of methods and trends from a machine learning perspective. *Patterns*. <https://doi.org/10.1016/j.patter.2025.101414>

- Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S. K., Appuswamy, R., Taba, B., Amir, A., Flickner, M. D., Risk, W. P., Manohar, R., & Modha, D. S. (2014). Artificial brains. A million spiking-neuron integrated circuit with a scalable communication network and interface [Journal Article]. *Science*, 345(6197), 668–673. <https://doi.org/10.1126/science.1254642>
- Micheva, K. D., Busse, B., Weiler, N. C., O'Rourke, N., & Smith, S. J. (2010). Single-synapse analysis of a diverse synapse population: proteomic imaging methods and markers. *Neuron*, 68(4), 639–653.
- Millner, S., Grübl, A., Meier, K., Schemmel, J., & Schwartz, M.-O. (2010). A VLSI implementation of the adaptive exponential integrate-and-fire neuron model. *Advances in Neural Information Processing Systems*, 1642–1650.
- Minsky, M., & Papert, S. (1969). *Perceptrons* [Book]. M.I.T. Press.
- Mishra, A., Dhanda, N., Gupta, K. K., & Verma, R. (2024). Speech Recognition Using Machine Learning Techniques. *2024 2nd International Conference on Disruptive Technologies (ICDT)*, 1142–1146.
- Mohammed, A., Schliebs, S., Matsuda, S., & Kasabov, N. (2012). Span: Spike pattern association neuron for learning spatio-temporal spike patterns. *International Journal of Neural Systems*, 22(4), 1250012.
- Moqadasi, H., Safari, S., & Mateo, F. (2025). Temporal single spike coding for effective transfer learning in spiking neural networks. *Scientific Reports*. <https://doi.org/10.1038/s41598-025-14619-3>
- Moradi, S., & Indiveri, G. (2014). An event-based neural network architecture with an asynchronous programmable synaptic memory. *IEEE Transactions on Biomedical Circuits and Systems*, 8(1), 98–107.
- Motiwalla, A., Soares, S., Atallah, B. V., Paton, J. J., & Machens, C. K. (2022). Efficient coding of cognitive variables underlies dopamine response and choice behavior. *Nature Neuroscience*, 25(6), 738–748.
- Muir, D. R., & Sheik, S. (2025). The road to commercial success for neuromorphic technologies. *Nature Communications*, 16(1), 3586.
- Mészáros, B., Knight, J. C., & Nowotny, T. (2025). Efficient event-based delay learning in spiking neural networks. *Nature Communications*. <https://doi.org/10.1038/s41467-025-65394-8>

- Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6), 51–63.
- Niu, L.-Y., Wei, Y., Liu, W.-B., Long, J.-Y., & Xue, T.-h. (2023). Research Progress of spiking neural network in image classification: a review. *Applied Intelligence*, 53(16), 19466–19490. <https://doi.org/10.1007/s10489-023-04553-0>
- Norman, J. (2002). Two visual systems and two theories of perception: An attempt to reconcile the constructivist and ecological approaches. *Behavioral and Brain Sciences*, 25(1), 73–96.
- Nouri, L., Yahya, S. I., Hafez, M. A., Chaudhary, M. A., & Assad, M. (2024). Novel energy harvesting microstrip Lowpass-Bandpass diplexer with an Ultra-Compact size for GSM and 5G communication systems. *AEU - International Journal of Electronics and Communications*, 155342. <https://doi.org/https://doi.org/10.1016/j.aeue.2024.155342>
- Nyquist, H. (1924). Certain factors affecting telegraph speed. *The Bell System Technical Journal*, 3(2), 324–346. <https://doi.org/10.1002/j.1538-7305.1924.tb01361.x>
- Nyquist, H. (1928). Certain Topics in Telegraph Transmission Theory. *Transactions of the American Institute of Electrical Engineers*, 47(2), 617–644. <https://doi.org/10.1109/T-AIEE.1928.5055024>
- Olshausen, B. A., & Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1?. *Vision Research*, 37(23), 3311–3325.
- Orchard, G., Frady, E. P., Rubin, D. B. D., Sanborn, S., Shrestha, S. B., Sommer, F. T., & Davies, M. (2021). Efficient Neuromorphic Signal Processing with Loihi 2. *2021 IEEE Workshop on Signal Processing Systems (Sips)*, 254–259. <https://doi.org/10.1109/SiPS52927.2021.00053>
- Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., & Purcell, T. J. (2007). A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1), 80–113.
- Paredes-Vallés, F., Hagenaaars, J. J., Dupeyroux, J., Stroobants, S., Xu, Y., & Croon, G. C. H. E. de. (2024). Fully neuromorphic vision and control for autonomous drone flight. *Science Robotics*, 9(90), eadi591. <https://doi.org/10.1126/scirobotics.adi0591>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., & others. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32.

- Paugam-Moisy, H., & Bohte, S. (2012). Computing with spiking neuron networks. In *Handbook of natural computing: Handbook of natural computing* (pp. 335–376). Springer.
- Pecevski, D., Natschläger, T., & Schuch, K. (2009). PCSIM: a parallel simulation environment for neural circuits fully integrated with Python. *Frontiers in Neuroinformatics*, 3.
- Pehle, C., Billaudelle, S., Cramer, B., Kaiser, J., Schreiber, K., Stradmann, Y., Weis, J., Leibfried, A., Müller, E., & Schemmel, J. (2022). The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity. *Frontiers in Neuroscience*, 16. <https://doi.org/10.3389/fnins.2022.795876>
- Pei, Y. R. (2025). *Let SSMs be ConvNets: State-space Modeling with Optimal Tensor Contractions*. <https://arxiv.org/abs/2501.13230>
- Plotly. (2015, October). *Collaborative data science*. Plotly Technologies Inc. <https://plot.ly/>
- Ponulak, F. (2005). ReSuMe-new supervised learning method for Spiking Neural Networks. *Institute of Control and Information Engineering, Poznan University of Technology*. (Available Online at: <Http://d1.Cie.Put.Poznan.Pl/~Fp/research.Html>).
- Rao, K. R., Kim, D. N., & Hwang, J. J. (2010). *Fast Fourier transform: algorithms and applications* (Vol. 32). Springer.
- Rao, R. P., & Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1), 79–87.
- Rao, S. M., Mayer, A. R., & Harrington, D. L. (2001). The evolution of brain activation during temporal processing. *Nature Neuroscience*, 4(3), 317–323.
- Rathi, N., Chakraborty, I., Kosta, A., Sengupta, A., Ankit, A., Panda, P., & Roy, K. (2023). Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware. *ACM Computing Surveys*, 55(12), 1–49.
- Rathi, N., Srinivasan, G., Panda, P., & Roy, K. (2020,). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *International Conference on Learning Representations*.
- Rippa, S. (1999). An algorithm for selecting a good value for the parameter c in radial basis function interpolation. *Advances in Computational Mathematics*, 11, 193–210.
- Rissanen, J., & Langdon, G. G. (1979). Arithmetic Coding. *IBM Journal of Research and Development*, 23(2), 149–162. <https://doi.org/10.1147/rd.232.0149>
- Sainburg, T., & Gentner, T. Q. (2021). Toward a computational neuroethology of vocal communication: from bioacoustics to neurophysiology, emerging tools and future directions. *Frontiers in Behavioral Neuroscience*, 15, 811737.

- Saini, A. K., Gehlot, N., Kumar, R., Hans, S., Chaudhary, S., & Sharma, G. (2026). Spiking neural network-based energy-efficient framework for real-time robotic arm manipulation. *Engineering Applications of Artificial Intelligence*, 167, 113805.
- Salomon, D. (2002). *Data compression*. Springer. <https://doi.org/10.1007/978-1-84628-603-2>
- Schaller, R. R. (1997). Moore's law: past, present and future. *IEEE Spectrum*, 34(6), 52–59.
- Schlichthärle, D. (2011). *Digital Filters Basics and Design*. Springer Berlin.
- Schonhaut, D. R., Aghajan, Z. M., Kahana, M. J., & Fried, I. (2023). A neural code for time and space in the human brain. *Cell Reports*, 42(11).
- Schrauwen, B., & Van Campenhout, J. (2003). BSA, a fast and accurate spike train encoding scheme. *Neural Networks, 2003. Proceedings of the International Joint Conference on*, 4, 2825–2830.
- Schütt, H. H., Kim, D., & Ma, W. J. (2024). Reward prediction error neurons implement an efficient code for reward. *Nature Neuroscience*, 1–7.
- Seekings, J., Ardakani, M., Chandarana, P., Eslami, A., Mohammadi, M., & Zand, R. (2025). Integrated algorithm and hardware design for hybrid neuromorphic systems. *Npj Unconventional Computing*, 2, 20. <https://doi.org/10.1038/s44335-025-00036-2>
- Sengupta, A., Ye, Y., Wang, R., Liu, C., & Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience*, 13, 95. <https://doi.org/10.3389/fnins.2019.00095>
- Senn, W., & Pfister, J.-P. (2015). Spike-Timing Dependent Plasticity, Learning Rules. *Encyclopedia of Computational Neuroscience*, 2824–2832.
- Setianingrum, A. H., Hulliyah, K., & Amrilla, M. F. (2023). Speech Recognition of Sundanese Dialect Using Convolutional Neural Network Method with Mel-Spectrogram Feature Extraction. *2023 11th International Conference on Cyber and IT Service Management (CITSM)*, , 1–5. <https://doi.org/10.1109/CITSM60085.2023.10455447>
- Shannon, C. (1949). Communication in the Presence of Noise. *Proceedings of the IRE*, 37(1), 10–21. <https://doi.org/10.1109/JRPROC.1949.232969>
- Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3), 379–423. <https://doi.org/10.1002/j.1538-7305.1948.tb01338.x>
- Sharma, M., Heard, J., Saber, E., & Markopoulos, P. P. (2025). Convolutional Neural Network Compression via Dynamic Parameter Rank Pruning. *IEEE Access*, , 1. <https://doi.org/10.1109/ACCESS.2025.3533419>

- Shi, X., Hao, Z., & Yu, Z. (2024). SpikingResformer: Bridging ResNet and Vision Transformer in Spiking Neural Networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5610–5619.
- Sjöström, J., & Gerstner, W. (2010). Spike-timing dependent plasticity. *Spike-Timing Dependent Plasticity*, 35.
- Smith, S. M., & Nichols, T. E. (2018). Statistical challenges in “big data” human neuroimaging. *Neuron*, 97(2), 263–268.
- Smith, S. W. (1999). *The scientist and engineer’s Guide to Digital Signal Processing*. California Technical Pub.
- Stimberg, M., Brette, R., & Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *Elife*, 8, e47314. <https://doi.org/10.7554/eLife.47314>
- Sun, M., Lan, L., Zhu, C.-G., & Lei, F. (2023). Cubic spline interpolation with optimal end conditions. *Journal of Computational and Applied Mathematics*, 425, 115039. <https://doi.org/https://doi.org/10.1016/j.cam.2022.115039>
- Talairach, J., & Tournoux, P. (1988). *Co-planar stereotaxic atlas of the human brain. 3-Dimensional proportional system: an approach to cerebral imaging*. Thieme.
- Tang, C., Ye, Z., Zhao, H., Bai, L., & Lin, J. (2025). DeepSCNN: a simplicial convolutional neural network for deep learning. *Applied Intelligence*, 55(4), 281.
- Tanviruzzama, & Mehfuz, S. (2023). Review: Lane Detection for Autonomous Vehicles Using Image Processing Techniques. *2023 International Conference on Power, Instrumentation, Energy and Control (PIECON)*, , 1–6. <https://doi.org/10.1109/PIECON56912.2023.10085756>
- Tieleman, T. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 26.
- Toga, A. W., Thompson, P. M., Mori, S., Amunts, K., & Zilles, K. (2006). Towards multi-modal atlases of the human brain. *Nature Reviews. Neuroscience*, 7(12), 952.
- Treisman, A. M., & Gelade, G. (1980). A feature-integration theory of attention. *Cognitive Psychology*, 12(1), 97–136.
- Upton, E., & Halfacree, G. (2014). *Raspberry Pi user guide*. John Wiley & Sons.
- UV. (2024, February). *uv: Python packaging in Rust*. Astral Software Inc. <https://github.com/astral-sh/uv>
- Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.
- van Rossum, M. C. W. (2001). A Novel Spike Distance. *Neural Computation*, 13(4), 751–763. <https://doi.org/10.1162/089976601300014321>

- Victor, J. D., & Purpura, K. P. (1996). Nature and precision of temporal coding in visual cortex: a metric-space analysis. *Journal of Neurophysiology*, 76(2), 1310–1326. <https://doi.org/10.1152/jn.1996.76.2.1310>
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., & others. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3), 261–272.
- Vitay, J., Dinkelbach, H. Ü., & Hamker, F. H. (2015). ANNarchy: a code generation approach to neural simulations on parallel hardware. *Frontiers in Neuroinformatics*, 9.
- Vogginger, B., Rostami, A., Jain, V., Arfa, S., Hantsch, A., Kappel, D., Schäfer, M., Faltings, U., Gonzalez, H. A., Liu, C., & others. (2024). Neuromorphic hardware for sustainable AI data centers. *Arxiv Preprint Arxiv:2402.02521*.
- Voudaskas, M., MacLean, J. I., Dutton, N. A., Stewart, B. D., & Gyongy, I. (2025). Spiking neural networks in imaging: A review and case study. *Sensors*, 25(21), 6747.
- Walker, M. P., & Stickgold, R. (2006). Sleep, memory, and plasticity. *Annu. Rev. Psychol.*, 57(1), 139–166.
- Wallace, G. K. (1992). The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1), xviii–xxxiv.
- Wang, S., Li, Z., Pei, M., Ren, Q., Deng, M., Ang, K.-W., Ascoli, A., Banerjee, S., Bonnin, M., Burgt, Y. van de, & others. (2026). Technology Roadmap of Bioinspired Computing Hardware. *ACS Nano*.
- Wang, Z., & Bovik, A. C. (2009). Mean squared error: Love it or leave it? A new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1), 98–117.
- Warden, P. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *Corr.* <http://arxiv.org/abs/1804.03209>
- Wark, K., & Warner, C. (1981). *Air pollution: its origin and control*. Harper, Row Publishers, New York, NY.
- Watanabe, S. (2023). Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *Arxiv Preprint Arxiv:2304.11127*.
- Wiegand, T., Sullivan, G., Bjontegaard, G., & Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), 560–576. <https://doi.org/10.1109/TCSVT.2003.815165>
- Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6), 520–540.

- Wysoski, S. G., Benuskova, L., & Kasabov, N. (2006). Adaptive learning procedure for a network of spiking neurons and visual pattern recognition. *International Conference on Advanced Concepts for Intelligent Vision Systems*, 1133–1142.
- Xiaoxue, L., Xiaofan, Z., Xin, Y., Dan, L., He, W., Bowen, Z., Bohan, Z., Di, Z., & Liqun, W. (2023). Review of medical data analysis based on spiking neural networks. *Procedia Computer Science*, 221, 1527–1538. <https://doi.org/https://doi.org/10.1016/j.procs.2023.08.138>
- Xie, H., Yang, G., & Gao, W. (2024). Toward Efficient Deep Spiking Neuron Networks: A Survey on Compression. *International Joint Conference on Artificial Intelligence*, 18–31.
- Yang, Q., Zhong, M., Sun, J., Zhong, X., Duan, S., & Wang, L. (2026). STCA-SNN: Spatio-Temporal Coordinate Attention for Spiking Neural Networks. *Expert Systems with Applications*, 132570.
- Yu, C. et al. (2026). A Photonic Real-Valued Weights Spiking Neural Networks Based on the Intrinsic Plasticity for Neuromorphic Datasets. *Laser & Photonics Reviews*, e3125.
- Yu, Z. et al. (2026). Beyond rate coding: surrogate gradients enable spike timing learning in spiking neural networks. *Neuromorphic Computing and Engineering*, 6. <https://doi.org/10.1088/2634-4386/ae46d5>
- Yuan, Y., Zhu, J., Jiang, C., Huang, F., Wu, M., Liu, R., & Zhou, D. (2024). Discriminative Fusion Methods for FMCW LiDAR: Leveraging All-phase FFT and Time-shifted Phase Difference. *2024 Photonics & Electromagnetics Research Symposium (PIERS)*, , 1–8. <https://doi.org/10.1109/PIERS62282.2024.10618545>
- Zanatta, L., Barchi, F., Manoni, S., Tolu, S., Bartolini, A., & Acquaviva, A. (2024). Exploring spiking neural networks for deep reinforcement learning in robotic tasks. *Scientific Reports*, 14(1), 30648.
- Zhang, H., & Thurber, C. H. (2003). Double-difference tomography: The method and its application to the Hayward fault, California. *Bulletin of the Seismological Society of America*, 93(5), 1875–1889.
- Zhang, H., Zhou, C., Yu, L., Yu, L., Liang, Z., Ma, C., Zhou, H., & Zhou, Y. (2024,). SGLFormer: Spiking global-local-fusion transformer with high performance. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- Zhang, J., Yang, T., Jiang, C., Liu, J., & Zhang, H. (2025). Chaotic loss-based spiking neural network for privacy-preserving bullying detection in public places. *Applied Soft Computing*, 169, 112643.
- Zhang, M. et al. (2025). Resilience of Entropy Model in Distributed Neural Networks. *European Conference on Computer Vision*, 423–440.

- Zhang, T. et al. (2025). Staa-snn: Spatial-temporal attention aggregator for spiking neural networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13959–13969.
- Zhang, Y., Gong, K., Zhang, K., Li, H., Qiao, Y., Ouyang, W., & Yue, X. (2023). Meta-transformer: A unified framework for multimodal learning. *Arxiv Preprint Arxiv:2307.10802*.
- Zheng, H., Wu, Y., Deng, L., Hu, Y., & Li, G. (2021). Going deeper with directly-trained larger spiking neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12), 11062–11070.
- Zhou, C., Zhang, H., Yu, L., Ye, Y., Zhou, Z., Huang, L., Ma, Z., Fan, X., Zhou, H., & Tian, Y. (2024). Direct training high-performance deep spiking neural networks: a review of theories and methods. *Frontiers in Neuroscience*, 18, 1383844.
- Zhou, D., Kim, J. Z., Pines, A. R., Sydnor, V. J., Roalf, D. R., Detre, J. A., Gur, R. C., Gur, R. E., Satterthwaite, T. D., & Bassett, D. S. (2022). Compression supports low-dimensional representations of behavior across neural circuits. *Biorxiv*. <https://doi.org/10.1101/2022.11.29.518415>
- Zhou, Z., Zhu, Y., He, C., Wang, Y., Yan, S., Tian, Y., & Yuan, L. (2023,). Spikformer: When spiking neural network meets transformer. *International Conference on Learning Representations*.
- Zhu, Z., Chen, K., Lin, W., Yan, H., Liang, L., Wang, G., Song, X., Zhang, C., & Wang, Z. (2025). Recent advances in spike-based neural coding for tactile perception. *Microsystems & Nanoengineering*. <https://doi.org/10.1038/s41378-025-01074-3>
- Zilles, K., & Amunts, K. (2010). Centenary of Brodmann's map—conception and fate. *Nature Reviews Neuroscience*, 11(2).
- Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3), 337–343.

Appendix A

A.1 DyNED Encoder

The core DyNED encoder converts a continuous signal into a quantised representation using sigma-delta feedback-driven error diffusion, then generates a binary spike train marking positions where the quantised value changes. The implementation below is the exact code used to produce every experimental result in this thesis: a numba-JIT inner loop wrapped by a pure-Python entry point that falls back to a plain numpy loop when numba is unavailable.

```
import numpy as np
import torch

try:
    from numba import njit

    @njit(cache=True)
    def _sigma_delta_loop(normalized, levels):
        """JIT-compiled sigma-delta feedback loop (~100x faster than
        Python)."""
        n = len(normalized)
        quantized = np.zeros(n, dtype=np.float64)
        errors = np.zeros(n, dtype=np.float64)
        error = 0.0
        for i in range(n):
            sample_with_error = normalized[i] + error
            q = round(sample_with_error * (levels - 1))
            if q < 0:
                q = 0
            elif q > levels - 1:
                q = levels - 1
            q_norm = q / (levels - 1)
            quantized[i] = q_norm
            error = sample_with_error - q_norm
            errors[i] = error
        return quantized, errors

    HAS_NUMBA = True
except ImportError:
    HAS_NUMBA = False
```

```

def sigma_delta_quantisation(data, levels=256):
    """
    Sigma-delta quantization of signal data.

    Converts a continuous signal into a quantized representation using
    feedback-driven error diffusion. Uses numba JIT if available.

    Args:
        data: Input signal (numpy array or torch Tensor).
        levels: Number of quantization levels (default 256).

    Returns:
        quantized: Quantized signal (numpy array, same scale as input).
        errors: Quantization error at each sample.
    """
    if hasattr(data, "cpu"):
        data = data.cpu()
    data = np.asarray(data, dtype=np.float64)

    data_min, data_max = data.min(), data.max()
    data_range = data_max - data_min
    if data_range == 0:
        return data.copy(), np.zeros_like(data)

    normalized = (data - data_min) / data_range

    if HAS_NUMBA:
        quantized, errors = _sigma_delta_loop(normalized, levels)
    else:
        quantized = np.zeros_like(normalized)
        error = 0.0
        errors = np.zeros_like(normalized)
        for i in range(len(normalized)):
            sample_with_error = normalized[i] + error
            q = np.clip(np.round(sample_with_error * (levels - 1)), 0,
levels - 1)
            q_norm = q / (levels - 1)
            quantized[i] = q_norm
            error = sample_with_error - q_norm
            errors[i] = error

```

```

    # Denormalize back to original scale
    quantized = quantized * data_range + data_min
    return quantized, errors

def generate_step_signal(quantised_data):
    """Generate binary step signal from quantized data (1 where value
    changes)."""
    step_signal = np.zeros_like(quantised_data, dtype=bool)
    step_signal[1:] = np.diff(quantised_data) != 0
    return step_signal

```

A.2 DyNED Decoder

Reconstruction from a DyNED-encoded signal reverses the encoding pipeline: the quantised frequency-domain representation is interpolated, the log scaling is inverted, and the inverse FFT recovers the spatial or temporal signal. Below is the image reconstruction variant using cubic spline interpolation.

```

import numpy as np
import torch
from scipy.interpolate import CubicSpline

def reconstruct_from_dynd(gray_img, levels=256):
    """Reconstruct image from DyNED-quantised frequency domain.

    Pipeline: FFT2 → magnitude/phase → log1p → sigma-delta
              → interpolate → exp-1 → combine with phase → IFFT2

    Args:
        gray_img: Grayscale image tensor [1, H, W].
        levels: DyNED quantisation levels used during encoding.

    Returns:
        Reconstructed image as numpy array.
    """
    img_2d = gray_img[0]
    shape = img_2d.shape

    # Forward FFT

```

```

fft_result = torch.fft.fft2(img_2d)
fft_shifted = torch.fft.fftshift(fft_result)
magnitude = torch.abs(fft_shifted)
phase = torch.angle(fft_shifted)
log_magnitude = torch.log1p(magnitude)

# DyNED encode
mag_flat = log_magnitude.flatten().numpy()
quantised, _ = sigma_delta_quantisation(mag_flat, levels=levels)

# Interpolate quantised magnitude using cubic spline
x = np.arange(len(quantised))
cs = CubicSpline(x, quantised, bc_type="clamped")
interpolated = cs(x)

# Reverse log1p: exp(x) - 1
recovered_magnitude =
torch.exp(torch.from_numpy(interpolated).float()) - 1

# Combine with original phase and inverse FFT
complex_spectrum = recovered_magnitude * torch.exp(1j *
phase.flatten())
complex_2d = complex_spectrum.reshape(shape)
reconstructed =
torch.real(torch.fft.ifft2(torch.fft.ifftshift(complex_2d)))

return reconstructed.numpy()

```

A.3 Image Encoder and Decoder

The image DyNED pipeline encodes a grayscale image by applying a 2D FFT, extracting the log-magnitude spectrum, and running sigma-delta quantisation to produce a spike train. Decoding interpolates the quantised magnitude with a cubic spline, reverses the log scaling, and applies an inverse 2D FFT to recover the image.

```

import numpy as np
import torch
from scipy.interpolate import CubicSpline
from dynd import sigma_delta_quantisation, generate_step_signal

```

```
def dyned_encode_image(gray_img, levels=256):
    """Encode grayscale image via 2D FFT + DyNED sigma-delta.

    Pipeline: FFT2 → fftshift → magnitude → log1p → flatten
              → sigma-delta quantisation → step signal

    Args:
        gray_img: Grayscale image tensor [1, H, W].
        levels: Number of quantisation levels.

    Returns:
        log_magnitude: Log-magnitude spectrum [H, W].
        quantised: Quantised magnitude reshaped to [H, W].
        step_signal: Binary spike train reshaped to [H, W].
    """
    fft_result = torch.fft.fft2(gray_img[0])
    fft_shifted = torch.fft.fftshift(fft_result)
    magnitude = torch.abs(fft_shifted)
    log_magnitude = torch.log1p(magnitude)

    mag_flat = log_magnitude.flatten().numpy()
    quantised, _ = sigma_delta_quantisation(mag_flat, levels=levels)
    step_signal = generate_step_signal(quantised)

    h, w = gray_img[0].shape
    return log_magnitude, quantised.reshape(h, w), step_signal.reshape(h, w)
```

```
def dyned_decode_image(quantised_magnitude, original_phase,
                        original_shape):
    """Reconstruct image from DyNED-quantised frequency domain.

    Pipeline: interpolate quantised magnitude → exp-1 → combine with
              original phase → ifftshift → IFFT2 → real part

    Args:
        quantised_magnitude: Quantised log-magnitude (flattened).
        original_phase: Phase spectrum from encoding (flattened).
        original_shape: Original 2D image shape (H, W).

    Returns:
```

```

        Reconstructed image as numpy array.
        """
        x = np.arange(len(quantised_magnitude))
        cs = CubicSpline(x, quantised_magnitude, bc_type="clamped")
        interpolated = cs(x)

        # Reverse log1p: exp(x) - 1
        recovered_magnitude =
        torch.exp(torch.from_numpy(interpolated).float()) - 1

        # Combine with original phase and inverse FFT
        complex_spectrum = recovered_magnitude * torch.exp(1j *
        original_phase)
        complex_2d = complex_spectrum.reshape(original_shape)
        reconstructed =
        torch.real(torch.fft.ifft2(torch.fft.ifftshift(complex_2d)))

        return reconstructed.numpy()

```

The exported Jupyter notebook with full encoding and decoding examples can be found at <https://phd-assets.pages.dev/notebooks/image>.

A.4 Speech Encoder and Decoder

The speech DyNED pipeline converts a waveform into a log-Mel spectrogram (80 Mel bins, $n_{\text{fft}} = 1024$, $\text{hop} = 80$, $\text{win} = 200$) and then runs sigma-delta quantisation independently on each time frame to produce a 2D binary spike train. The Mel-scale binning is perceptually weighted and substantially more informative for speech than raw STFT bins. Decoding combines the per-frame interpolated magnitude with the original phase and applies an inverse FFT to recover the audio signal.

```

import numpy as np
import torch
import torchaudio.transforms as T
from dyned import sigma_delta_quantisation, generate_step_signal

def dyned_encode_waveform(
    waveform,
    target_sr=8000,
    n_mels=80,

```

```

n_fft=1024,
win_length=200,
hop_length=80,
dyned_levels=256,
):
    """Encode a waveform into a spike train via log-Mel + per-frame DyNED.

    Pipeline: MelSpectrogram → log1p → per-frame sigma-delta
              → step signal (binary spike train)

    Args:
        waveform: 1D tensor (e.g. 8kHz, 1 second).
        target_sr: Sample rate of the input waveform.
        n_mels: Number of Mel bins per frame.
        n_fft: FFT window size used inside MelSpectrogram.
        win_length: Analysis window length (samples).
        hop_length: STFT hop length (samples).
        dyned_levels: Sigma-delta quantisation levels.

    Returns:
        spike_train: [n_mels, n_time_frames] float tensor of 0s and 1s.
    """
    mel = T.MelSpectrogram(
        sample_rate=target_sr,
        n_mels=n_mels,
        n_fft=n_fft,
        win_length=win_length,
        hop_length=hop_length,
        f_min=20,
        f_max=target_sr // 2,
    )
    log_mel = torch.log1p(mel(waveform.unsqueeze(0)).squeeze(0)) #
    [n_mels, T]

    n_freq, n_time = log_mel.shape
    spike_train = np.zeros((n_freq, n_time), dtype=np.float32)

    for t in range(n_time):
        frame = log_mel[:, t].numpy()
        quantised, _ = sigma_delta_quantisation(frame,
levels=dyned_levels)
        step_signal = generate_step_signal(quantised)

```



```

        spike_train[:, t] = step_signal.astype(np.float32)

    return torch.from_numpy(spike_train)

def dyned_decode_waveform(interpolated_magnitude, original_phase,
                           scale_factor=1.0):
    """Reconstruct speech from DyNED-quantised frequency domain.

    Pipeline: scale magnitude → combine with original phase
              → ifftshift → IFFT → real part

    Args:
        interpolated_magnitude: Interpolated magnitude spectrum.
        original_phase: Phase spectrum from encoding.
        scale_factor: Amplitude scaling factor.

    Returns:
        Reconstructed audio as tensor.
    """
    scaled_magnitude = interpolated_magnitude * scale_factor
    complex_spectrum = scaled_magnitude * torch.exp(1j * original_phase)
    ifft_result = torch.fft.ifft(torch.fft.ifftshift(complex_spectrum))
    return torch.real(ifft_result)

```

The exported Jupyter notebook with full encoding and decoding examples can be found at <https://phd-assets.pages.dev/notebooks/speech>.

A.5 Boosting Signals

Table 45: Comparison of various signal boosting techniques applied to a speech sample from the Speech Commands dataset.

Method	Min	Max	RMS	SNR (dB)	RMSE
Original	-0.9950	0.9193	0.1552	inf	0.0000
Processed	-1.8281	1.4412	0.1782	4.6042	0.0913
normalise	-1.0000	0.7884	0.0975	5.0046	0.0872
peak_match	-0.9950	0.7844	0.0970	4.9844	0.0874
rms_match	-1.5924	1.2554	0.1552	5.4876	0.0825
adaptive	-0.9634	0.7595	0.0939	4.8540	0.0887
compress	-0.6641	0.9706	0.1232	3.4235	0.1046
hard_clip	-0.8000	0.8000	0.1603	5.4390	0.0830
soft_clip	-1.0359	1.0308	0.2306	1.5728	0.1295
Pitch Shift	-1.1217	0.9432	0.1122	-1.8234	0.1914
Time Stretch	-0.9326	0.6994	0.0794	-1.0101	0.1743
Preemphasis	-2.0621	2.1260	0.2270	-2.3909	0.2044
RMS Normalise	-1.0000	0.7884	0.0975	5.0046	0.0872
Harmonic	-0.0098	0.0092	0.0004	-0.0000	0.1552
Percussive	-1.8281	1.4412	0.1782	4.6046	0.0913

A.6 DyNED_c Algorithm

The DyNED_c compressor uses adaptive mode selection across four compression strategies - variable-length RLE, chunk-based Huffman, hybrid RLE+Huffman, and alternating-run Huffman - automatically choosing the mode that yields the smallest output for each spike train. Numba JIT compilation is used for the inner loops when available.

```
import heapq
import numpy as np
from collections import Counter

class HuffmanNode:
    """Node in a Huffman tree."""
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
```

```

        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

class DyNEDcCompressor:
    """Hybrid compression for spike trains with adaptive mode selection.

    Modes:
    1. Variable-length RLE
    2. Chunk-based Huffman
    3. RLE + chunk Huffman (hybrid)
    4. Alternating-run RLE + Huffman on run-length values
    """

    _MAX_RUN = 255

    def __init__(self, chunk_size=4):
        self.chunk_size = chunk_size
        self._mode = None
        self._huff_codes = {}
        self._alt_start = "0"

    @staticmethod
    def _to_array(spike_data):
        """Convert any input to a flat uint8 numpy array."""
        if isinstance(spike_data, str):
            return np.frombuffer(
                spike_data.encode("ascii"), dtype=np.uint8
            ) - ord("0")
        if hasattr(spike_data, "numpy"):
            return
        spike_data.detach().cpu().flatten().numpy().astype(np.uint8)
        return np.asarray(spike_data, dtype=np.uint8).flatten()

    @staticmethod
    def _extract_runs(arr):
        """Extract alternating run lengths."""
        if len(arr) == 0:
            return np.zeros(0, dtype=np.int64), "0"
        changes = np.diff(arr) != 0

```

```

change_idx = np.flatnonzero(changes) + 1
boundaries = np.concatenate(([0], change_idx, [len(arr)]))
return np.diff(boundaries), str(int(arr[0]))

@staticmethod
def _arr_to_chunk_ints(arr, chunk_size):
    """Pack bit array into integer chunk keys."""
    n = len(arr) - len(arr) % chunk_size
    powers = (2 ** np.arange(
        chunk_size - 1, -1, -1
    )).astype(np.int64)
    if n > 0:
        chunk_ints = (
            arr[:n].reshape(-1, chunk_size).astype(np.int64) * powers
        ).sum(axis=1)
    else:
        chunk_ints = np.array([], dtype=np.int64)
    if len(arr) - n > 0:
        tail = arr[n:]
        tp = (2 ** np.arange(
            len(tail) - 1, -1, -1
        )).astype(np.int64)
        chunk_ints = np.append(
            chunk_ints, int((tail.astype(np.int64) * tp).sum())
        )
    return chunk_ints

@classmethod
def _rle_from_runs(cls, runs, start_value):
    """Encode pre-extracted runs to variable-length RLE."""
    compressed = []
    current = start_value
    for count in runs:
        while count > cls._MAX_RUN:
            cb = format(cls._MAX_RUN, "b")
            compressed.append(
                format(len(cb) - 1, "03b") + cb + current
            )
            count -= cls._MAX_RUN
        if count > 0:
            cb = format(count, "b")
            compressed.append(

```

```

        format(len(cb) - 1, "03b") + cb + current
    )
    current = "1" if current == "0" else "0"
    return "".join(compressed)

@staticmethod
def _rle_size_from_runs(runs):
    """Compute RLE compressed size without building the string."""
    size = 0
    for count in runs:
        while count > 255:
            size += 3 + len(format(255, "b")) + 1
            count -= 255
        if count > 0:
            size += 3 + len(format(count, "b")) + 1
    return size

@staticmethod
def _build_huffman_codes(freq):
    """Build Huffman codes from a frequency dict."""
    if len(freq) == 1:
        return {list(freq.keys())[0]: "0"}
    heap = [HuffmanNode(sym, c) for sym, c in freq.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        internal = HuffmanNode(None, left.freq + right.freq)
        internal.left = left
        internal.right = right
        heapq.heappush(heap, internal)
    codes = {}
    DyNEDcCompressor._walk(heap[0], "", codes)
    return codes

@staticmethod
def _walk(node, prefix, codes):
    if node is None:
        return
    if node.char is not None:
        codes[node.char] = prefix or "0"
    return

```

```

DyNEDcCompressor._walk(node.left, prefix + "0", codes)
DyNEDcCompressor._walk(node.right, prefix + "1", codes)

@staticmethod
def _huffman_size(freq, codes):
    """Compute Huffman compressed size from frequency and codes."""
    return sum(
        count * len(codes[sym]) for sym, count in freq.items()
    )

def compress(self, spike_data) → tuple[str, dict]:
    """Compress a spike train using adaptive mode selection.

    Args:
        spike_data: tensor, numpy array, or binary string.

    Returns:
        (compressed_str, info_dict) with compression statistics.
    """
    arr = self._to_array(spike_data)
    n = len(arr)
    if n == 0:
        return "", {"original_length": 0, "compressed_length": 0}

    # Extract runs once - reused by modes 1, 3, 4
    runs, start = self._extract_runs(arr)

    # Mode 1: Variable-length RLE
    rle_size = self._rle_size_from_runs(runs)

    # Mode 2: Chunk-based Huffman
    chunk_ints = self._arr_to_chunk_ints(arr, self.chunk_size)
    chunk_freq = Counter(chunk_ints.tolist())
    huff_codes = self._build_huffman_codes(chunk_freq)
    huff_size = self._huffman_size(chunk_freq, huff_codes)

    # Mode 4: Alternating-run Huffman
    runs_list = runs.tolist()
    sym_freq = Counter(runs_list)
    alt_codes = self._build_huffman_codes(sym_freq)
    alt_size = self._huffman_size(sym_freq, alt_codes)

```

```

# Mode 3: Hybrid RLE + chunk Huffman
min_so_far = min(rle_size, huff_size, alt_size)
hybrid_size = min_so_far + 1
hybrid_codes = {}
rle_compressed = None
if min_so_far > n * 0.05:
    rle_compressed = self._rle_from_runs(runs, start)
    rle_arr = np.frombuffer(
        rle_compressed.encode("ascii"), dtype=np.uint8
    ) - ord("0")
    rle_chunks = self._arr_to_chunk_ints(
        rle_arr, self.chunk_size
    )
    rle_chunk_freq = Counter(rle_chunks.tolist())
    hybrid_codes = self._build_huffman_codes(rle_chunk_freq)
    hybrid_size = self._huffman_size(
        rle_chunk_freq, hybrid_codes
    )

# Pick the best mode
sizes = {
    "rle": rle_size, "huffman": huff_size,
    "hybrid": hybrid_size, "alt_huffman": alt_size,
}
best_mode = min(sizes, key=sizes.get)
best_size = sizes[best_mode]

# Build compressed string for the winning mode only
if best_mode == "rle":
    if rle_compressed is None:
        rle_compressed = self._rle_from_runs(runs, start)
    best_compressed = rle_compressed
    best_codes, best_start = {}, ""
elif best_mode == "huffman":
    best_compressed = "".join(
        huff_codes[k] for k in chunk_ints.tolist()
    )
    best_codes, best_start = huff_codes, ""
elif best_mode == "hybrid":
    if rle_compressed is None:
        rle_compressed = self._rle_from_runs(runs, start)
    rle_arr = np.frombuffer(

```

```

        rle_compressed.encode("ascii"), dtype=np.uint8
    ) - ord("0")
    rle_chunks = self._arr_to_chunk_ints(
        rle_arr, self.chunk_size
    )
    best_compressed = "".join(
        hybrid_codes[k] for k in rle_chunks.tolist()
    )
    best_codes, best_start = hybrid_codes, ""
else: # alt_huffman
    best_compressed = "".join(
        alt_codes[r] for r in runs_list
    )
    best_codes, best_start = alt_codes, start

self._mode = best_mode
self._huff_codes = best_codes
self._alt_start = best_start

return best_compressed, {
    "original_length": n,
    "compressed_length": best_size,
    "compression_ratio": best_size / n,
    "mode": best_mode,
}

```

A.7 CIFAR-10 DyNED Hyperparameter Optimisation Analysis

Hyperparameter optimisation for the CIFAR-10 Conv+DyNED+cAdLIF+TET classifier (Chapter 5) was conducted using Optuna with the TPE sampler over 56 trials, each trained for 50 epochs. Of the 56 trials, 10 completed, 40 were pruned by the median pruner, and 6 failed. The best 50-epoch trial achieved 87.05% test accuracy; when the optimal hyperparameters were used for a full 250-epoch training run, the model reached 88.07% at epoch 245.

Figure 88 presents a six-panel summary of the optimisation. The optimisation history (top left) shows the best-so-far curve climbing through the first 20 trials before plateauing. The learning curve of the best trial (top centre) demonstrates that accuracy was still increasing at epoch 50, confirming the benefit of extended training. The DyNED levels box plot (top right) shows the highest setting (256) producing the strongest accuracy distribution - the cAdLIF + TET architecture benefits from finer-grained quanti-

sation than the LIF baseline preferred. Hidden size 1024 outperformed 2048 within the searched trials (bottom left). The parallel coordinates plot (bottom centre) shows top trials clustering at modest learning rates ($\approx 3 \times 10^{-3}$), very low weight decay ($\approx 5 \times 10^{-5}$), and low dropout. The SNN-timesteps analysis (bottom right) confirms $T = 25$ as the strongest setting within the search range.

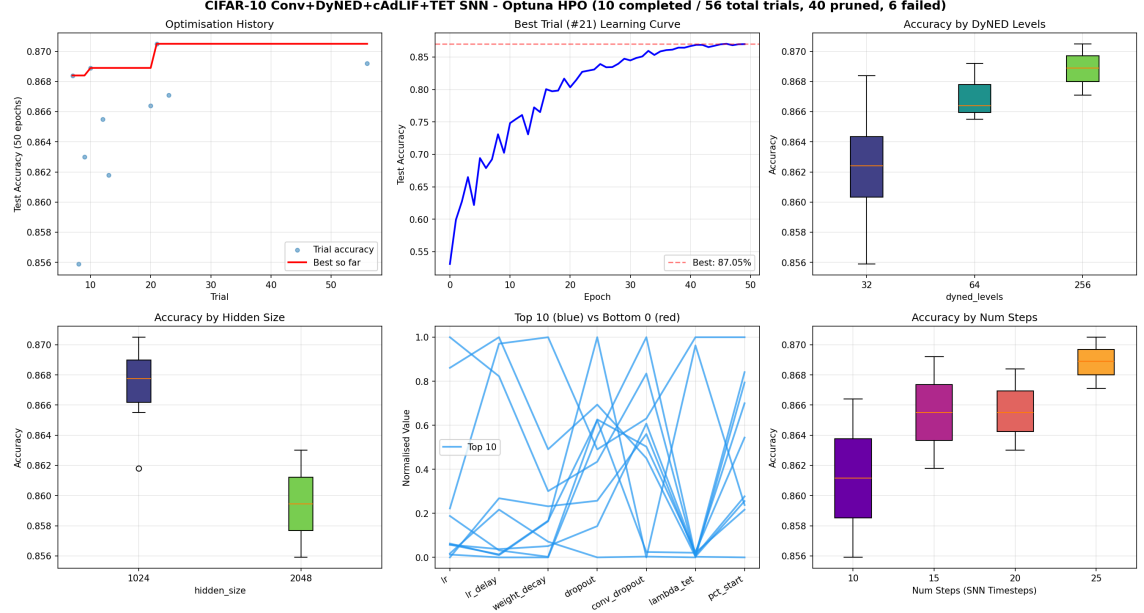


Figure 88: CIFAR-10 Conv+DyNED+cAdLIF+TET SNN hyperparameter optimisation analysis. Top row: optimisation history with best-so-far curve, learning curve for the best trial (21, 87.05% at 50 epochs), and accuracy distribution by DyNED quantisation levels. Bottom row: accuracy by hidden size, parallel coordinates comparing continuous hyperparameters of top 10 (blue) and bottom 10 (red) trials, and accuracy by SNN timesteps. Of 56 total trials, 10 completed, 40 were pruned by the median pruner, and 6 failed.

Table 46 lists the best hyperparameters identified by the optimisation. These parameters were used for the full 250-epoch training run that reached 88.07% best test accuracy at epoch 245.

Table 46: Best hyperparameters for the CIFAR-10 Conv+DyNED+cAdLIF+TET SNN, selected from 56 Optuna trials (50 epochs each). Test accuracy at 50 epochs: 87.05%; at the full 250-epoch retraining: 88.07%.

Hyperparameter	Value
Learning rate	3.04×10^{-3}
Delay learning rate	1.11×10^{-2}
Weight decay	5.56×10^{-5}
Hidden size	1024
DyNED dimension	512
Max delay	7
SNN timesteps	25
Dropout	0.101
Conv dropout	0.055
Batch size	128
OneCycleLR warmup fraction	0.327
DyNED quantisation levels	256
TET λ	1.21×10^{-4}

A.8 CIFAR-10 DyNED_c Hyperparameter Optimisation Analysis

Hyperparameter optimisation for the CIFAR-10 Conv+DyNED+cAdLIF+TET classifier with DyNED_c (Chapter 5) was conducted using Optuna with the TPE sampler over 50 trials, each trained for 50 epochs. Of the 50 trials, 13 completed and 37 were pruned by the median pruner. The best 50-epoch trial achieved 86.76% test accuracy; when the optimal hyperparameters were used for a full 250-epoch training run, the model reached 87.69% best test accuracy at epoch 247.

Figure 89 presents a six-panel summary of the optimisation. The optimisation history (top left) shows the search converging by trial 30. The learning curve of the best trial (top centre) confirms that accuracy was still climbing at epoch 50. The DyNED levels box plot (top right) shows that 64 levels produced the best results - a coarser quantisation than the DyNED-only optimum (256), suggesting that the DyNED_c compressor pairs better with sparser, lower-resolution spike trains. Hidden size 2048 was preferred (bottom left). The parallel coordinates plot (bottom centre) shows top trials at higher learning rates ($\approx 1.3 \times 10^{-2}$) and very low weight decay ($\approx 2 \times 10^{-5}$). The SNN-timesteps analysis (bottom right) shows $T = 15$ scoring best within the search horizon.

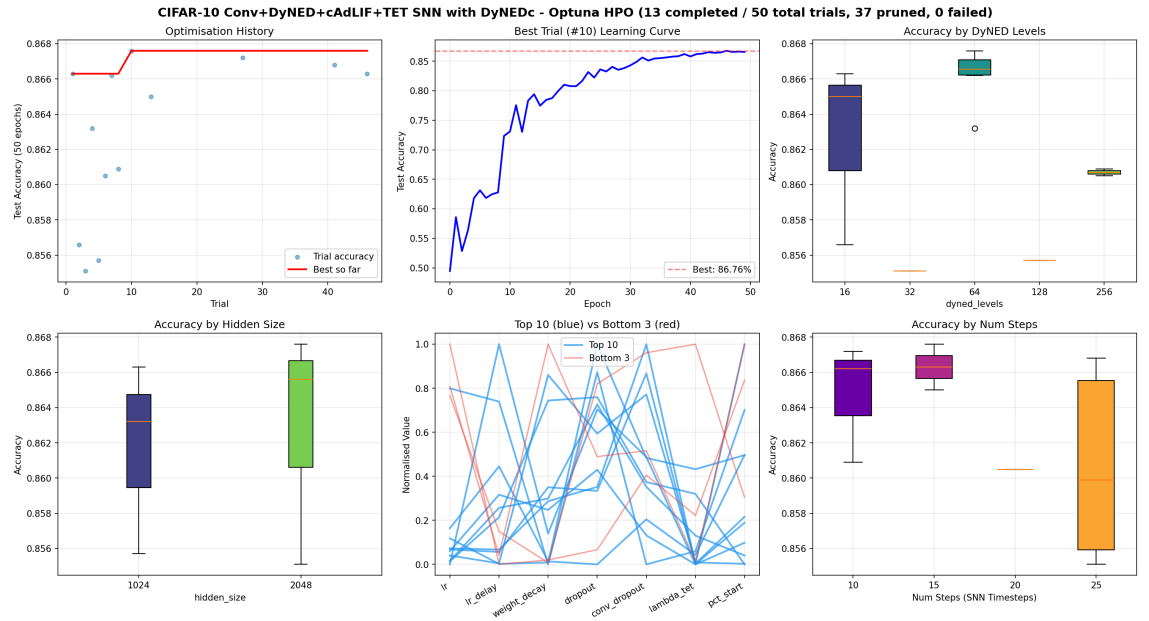


Figure 89: CIFAR-10 Conv+DyNED+cAdLIF+TET SNN with DyNED_c compression during inference - hyperparameter optimisation analysis. Top row: optimisation history with best-so-far curve, learning curve for the best trial (10, 86.76% at 50 epochs), and accuracy distribution by DyNED quantisation levels. Bottom row: accuracy by hidden size, parallel coordinates comparing continuous hyperparameters of top 10 (blue) and bottom 10 (red) trials, and accuracy by SNN timesteps. Of 50 total trials, 13 completed and 37 were pruned by the median pruner.

Table 47 lists the best hyperparameters identified by the optimisation. These parameters were used for the full 250-epoch training run that reached 87.69% best test accuracy at epoch 247.

Table 47: Best hyperparameters for the CIFAR-10 Conv+DyNED+cAdLIF+TET SNN with DyNED_c compression during inference, selected from 50 Optuna trials (50 epochs each). Test accuracy at 50 epochs: 86.76%; at the full 250-epoch retraining: 87.69%.

Hyperparameter	Value
Learning rate	1.35×10^{-2}
Delay learning rate	4.34×10^{-2}
Weight decay	2.29×10^{-5}
Hidden size	2048
DyNED dimension	512
Max delay	5
SNN timesteps	15
Dropout	0.130
Conv dropout	0.094
Batch size	512
OneCycleLR warmup fraction	0.222
DyNED quantisation levels	64
TET λ	4.26×10^{-3}

A.9 CIFAR10-DVS DyNED Hyperparameter Optimisation Analysis

Hyperparameter optimisation for the CIFAR10-DVS VGGSNN+DyNED+cAdLIF+TET classifier (Chapter 5) was conducted using Optuna with the TPE sampler over 50 trials, each trained for 50 epochs. Of the 50 trials, 21 completed and 29 were pruned by the median pruner.

Figure 90 presents a six-panel summary of the optimisation. The optimisation history (top left) shows the best-so-far curve climbing through the first 40 trials before plateauing. The learning curve of the best trial (top centre) demonstrates that accuracy was still rising at epoch 50. The DyNED levels box plot (top right) shows the lowest setting (16) producing the highest accuracy distribution, suggesting that coarser quantisation acts as a regulariser by enforcing sparser spike representations - the same pattern observed for CIFAR-10 (Section A.7), and the opposite of the DyNED_c regime where denser representations are preferred. Hidden size 1024 outperformed smaller alternatives (bottom left). The parallel coordinates plot (bottom centre) shows top trials concentrated around higher learning rates (≈ 0.10 - 0.15) with low weight decay ($\approx 10^{-5}$), and the max-delay distribution (bottom right) favours short delays (2-3 timesteps), reflecting the $T = 16$ temporal extent of the binned event stream.

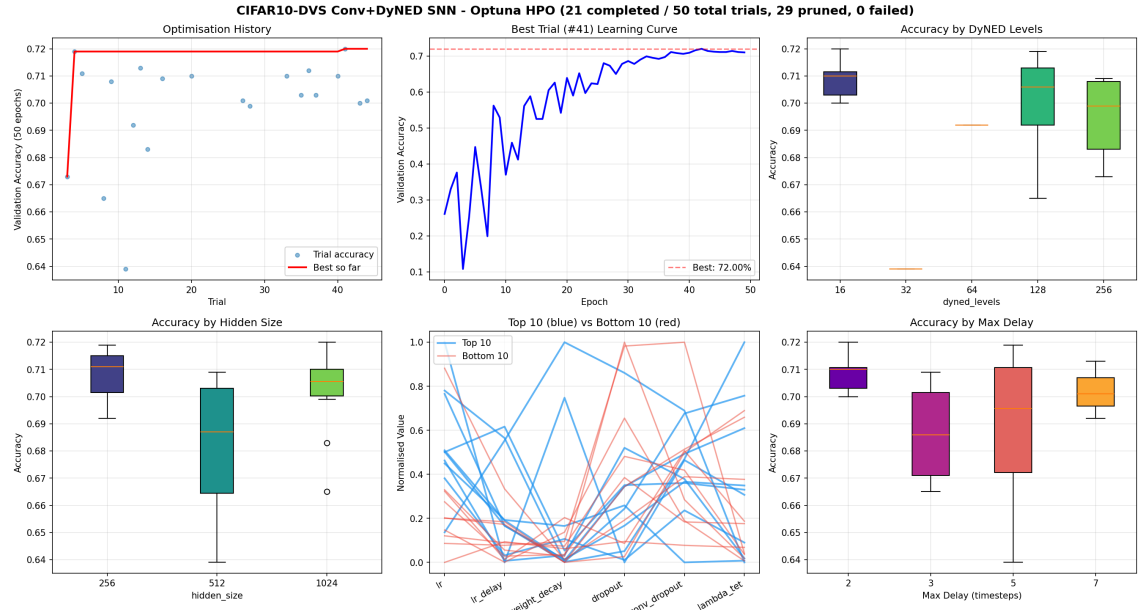


Figure 90: CIFAR10-DVS VGGsNN+DyNED+cAdLIF+TET hyperparameter optimisation analysis. Top row: optimisation history with best-so-far curve, learning curve for the best 50-epoch trial, and accuracy distribution by DyNED quantisation levels. Bottom row: accuracy by hidden size, parallel coordinates comparing continuous hyperparameters of top 10 (blue) and bottom 10 (red) trials, and accuracy by max delay. Of 50 total trials, 21 completed and 29 were pruned by the median pruner.

Table 48 lists the best hyperparameters identified by the optimisation. The full 300-epoch training at these parameters reached 76.3% test accuracy.

Table 48: Best hyperparameters for the CIFAR10-DVS VGGSNN+DyNED+cAdLIF+TET classifier, selected from 50 Optuna trials (50 epochs each). Test accuracy at the full 300-epoch training schedule: 76.3%.

Hyperparameter	Value
Learning rate	1.33×10^{-1}
Delay learning rate	1.75×10^{-2}
Weight decay	1.63×10^{-5}
Hidden size	1024
DyNED dimension	256
Max delay	2
Dropout	0.213
Conv dropout	0.132
Batch size	64
DyNED quantisation levels	16
TET λ	5.74×10^{-2}
TET start epoch	50

A.10 CIFAR10-DVS DyNED_c Hyperparameter Optimisation

Analysis

Hyperparameter optimisation for the CIFAR10-DVS VGGSNN+DyNED+DyNED_c+cAdLIF+TET classifier (Chapter 5) was conducted using Optuna with the TPE sampler over 70 trials, each trained for 50 epochs. Of the 70 trials, 17 completed, 33 were pruned by the median pruner, and 20 failed due to out-of-memory errors from larger configurations. The DyNEDc compressor adds one extra hyperparameter to the DyNED-only search space - the DyNED_c chunk size - while the rest of the architecture and training schedule are inherited from the DyNED-only setup.

Figure 91 presents a six-panel summary of the optimisation. The optimisation history (top left) shows the best-so-far curve climbing through the first 25 trials before plateauing. The learning curve of the best trial (top centre) demonstrates that accuracy was still rising at epoch 50. The DyNED levels box plot (top right) shows the lowest settings (16 and 32) scoring highest within the 50-epoch search horizon. Hidden size 512 outperformed alternatives (bottom left). The parallel coordinates plot (bottom centre) compares the continuous hyperparameters of the top and bottom trials. The DyNEDc chunk-size analysis (bottom right) shows chunk sizes 2 and 4 scoring comparably within the search horizon.

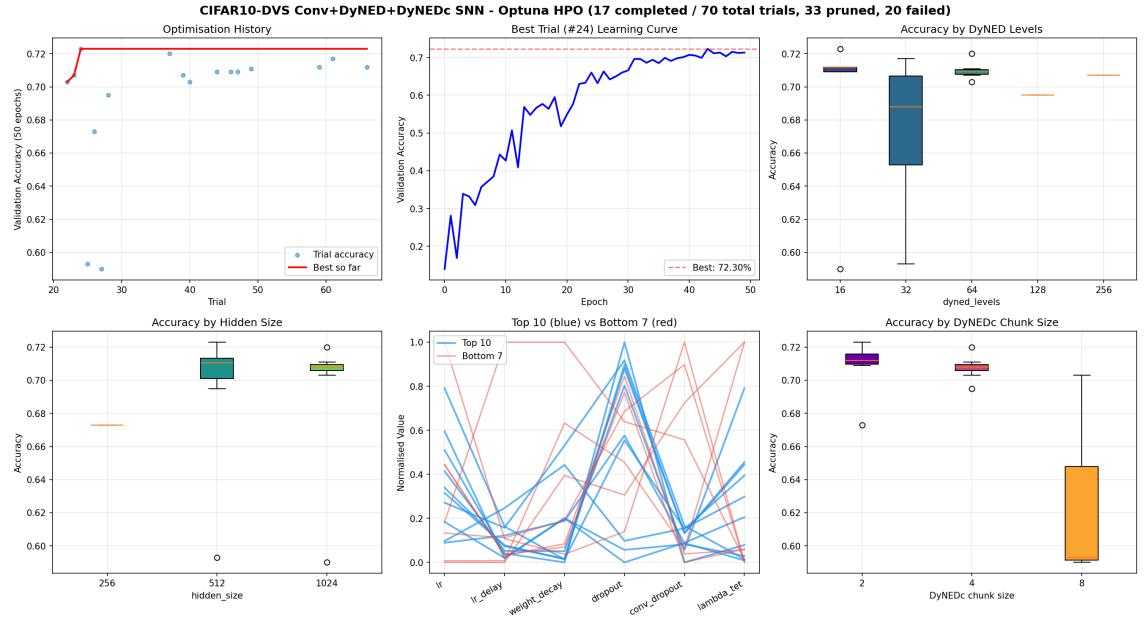


Figure 91: CIFAR10-DVS VGGsNN+DyNED+DyNED_c+cAdLIF+TET hyperparameter optimisation analysis. Top row: optimisation history with best-so-far curve, learning curve for the best 50-epoch trial, and accuracy distribution by DyNED quantisation levels. Bottom row: accuracy by hidden size, parallel coordinates comparing continuous hyperparameters of top 10 (blue) and bottom 10 (red) trials, and accuracy by DyNED_c chunk size. Of 70 total trials, 17 completed, 33 were pruned, and 20 failed.

Table 49 lists the search space alongside the hyperparameter values used to obtain the 79.0% test accuracy at epoch 264 over the full 300-epoch training schedule.

Table 49: Hyperparameter search space and values used for the CIFAR10-DVS VG-GSNN+DyNED+DyNED_c+cAdLIF+TET classifier. Test accuracy: 79.0% at epoch 264 over 300 training epochs.

Hyperparameter	Search Range	Value
Learning rate	$1 \times 10^{-2} - 3 \times 10^{-1}$ (log)	0.10
Delay learning rate	$1 \times 10^{-3} - 1 \times 10^{-1}$ (log)	0.01
Weight decay	$1 \times 10^{-5} - 1 \times 10^{-3}$ (log)	5×10^{-4}
Hidden size	{256, 512, 1024}	512
DyNED dimension	{256, 512, 1024}	512
Max delay	{2, 3, 5, 7}	5
Dropout	0.1 - 0.4	0.25
Conv dropout	0.05 - 0.2	0.10
Batch size	{64, 128, 256}	128
DyNED quantisation levels	{16, 32, 64, 128, 256}	64
TET λ	$1 \times 10^{-4} - 1 \times 10^{-1}$ (log)	1×10^{-3}
TET start epoch	-	50
DyNED _c chunk size	{2, 4, 8}	4

A.11 Speech Commands cAdLIF DyNED Hyperparameter Optimisation Analysis

Hyperparameter optimisation for the Speech Commands cAdLIF+DyNED SNN classifier (Chapter 6) was conducted using Optuna with the TPE sampler over 257 trials, each trained for 50 epochs. Of the 257 trials, 120 completed, 98 were pruned by the median pruner, and 18 failed due to out-of-memory errors from larger configurations. The best 50-epoch trial achieved 79.81% test accuracy.

Figure 92 presents a six-panel summary of the optimisation. The optimisation history (top left) shows the search converging after approximately 170 trials, with the best trial (252) achieving 79.81%. The learning curve of the best trial (top centre) demonstrates that accuracy was still fluctuating at epoch 50, indicating training instability typical of adaptive neuron models with many learnable parameters. The DyNED levels box plot (top right) shows a clear preference for 16 quantisation levels - lower levels produce more frequent state changes in DyNED’s quantiser, yielding denser spike trains that provide richer input to the cAdLIF neurons. A hidden size of 512 consistently outperformed 256 (bottom left), while larger sizes were not explored in this search space. The parallel coordinates plot (bottom centre) reveals that top trials cluster around higher learning rates (≈ 0.005 - 0.009), very low weight decay ($\approx 10^{-5}$), and moderate delay learning rates (≈ 0.15

-0.25). The max delay analysis (bottom right) shows that delays of 15 and 25 timesteps performed comparably, both outperforming the shorter 10-step delay.

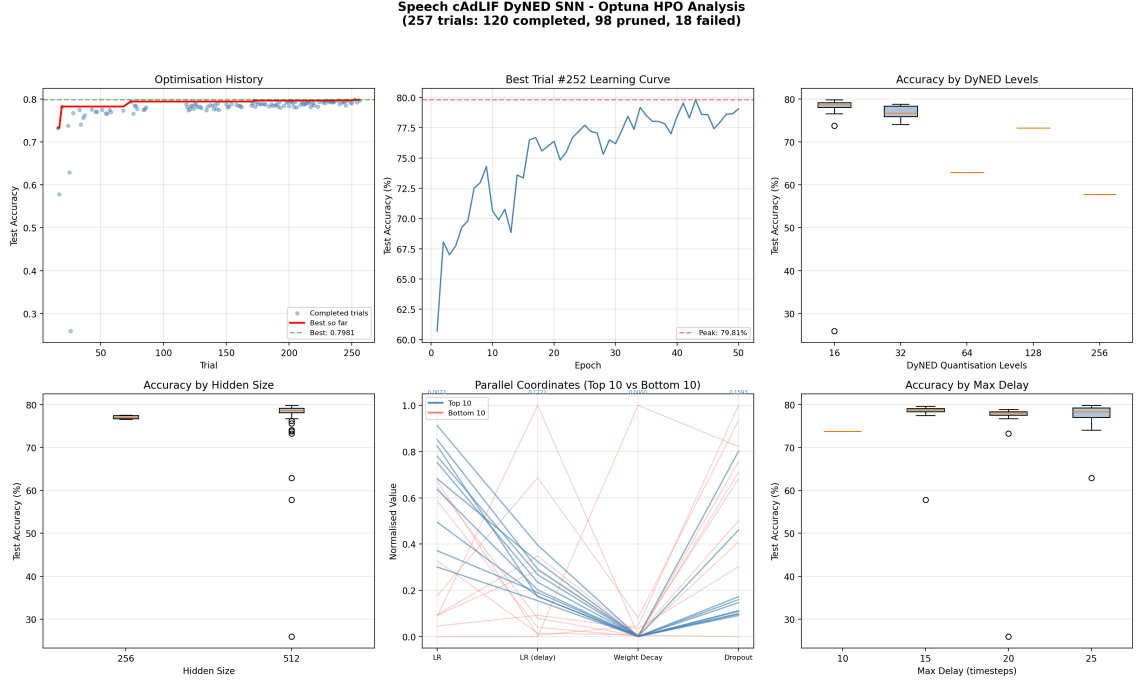


Figure 92: Speech Commands cAdLIF+DyNED SNN hyperparameter optimisation analysis. Top row: optimisation history with best-so-far curve, learning curve for the best trial (252, 79.81% at 50 epochs), and accuracy distribution by DyNED quantisation levels. Bottom row: accuracy by hidden size, parallel coordinates comparing continuous hyperparameters of top 10 (blue) and bottom 10 (red) trials, and accuracy by max delay. Of 257 total trials, 120 completed, 98 were pruned, and 18 failed.

Table 50 lists the best hyperparameters identified by the optimisation.

Table 50: Best hyperparameters for the Speech Commands cAdLIF+DyNED SNN, selected from 257 Optuna trials (50 epochs each). Test accuracy at 50 epochs: 79.81%.

Hyperparameter	Value
Learning rate	8.47×10^{-3}
Delay learning rate	2.27×10^{-1}
Weight decay	1.14×10^{-5}
Hidden size	512
Max delay	25
Dropout	0.148
Batch size	256
DyNED quantisation levels	16

A.12 SSC DyNED Hyperparameter Optimisation Analysis

Hyperparameter optimisation for the SSC cAdLIF+DyNED SNN classifier (Chapter 6) was conducted using Optuna with the TPE sampler over 65 trials, each trained for 50 epochs. Of the 65 trials, 33 completed, 25 were pruned by the median pruner, 5 failed, and 2 were interrupted. The best 50-epoch trial (#64) achieved 75.94% validation accuracy.

Figure 93 presents a six-panel summary of the optimisation. The optimisation history (top left) shows the best-so-far curve climbing through the search before settling near 76% validation accuracy. The learning curve of the best trial (top centre) demonstrates that accuracy was still rising at epoch 50. The DyNED levels box plot (top right) shows 32 levels producing the strongest accuracy distribution, a coarser quantisation that yields denser, more informative spike trains for the cAdLIF stack. Hidden size 512 outperformed 256 (bottom left). The parallel coordinates plot (bottom centre) compares the continuous hyperparameters of the top and bottom trials. The max delay analysis (bottom right) shows the shorter 10-step delay scoring highest within the 50-epoch search horizon.

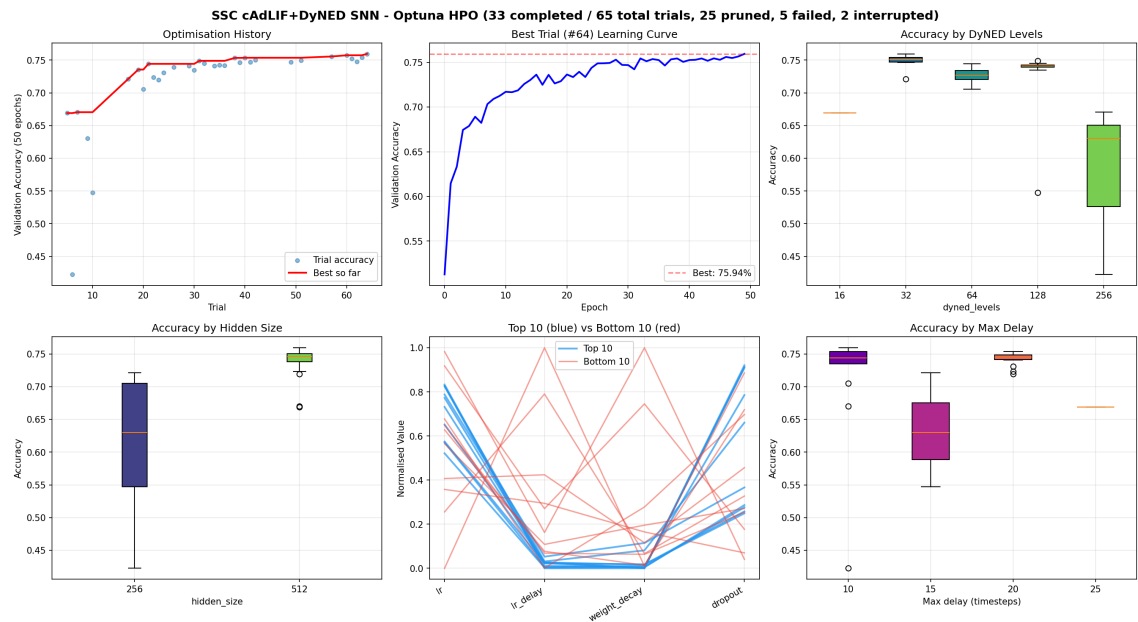


Figure 93: SSC cAdLIF+DyNED SNN hyperparameter optimisation analysis. Top row: optimisation history with best-so-far curve, learning curve for the best trial (64, 75.94% at 50 epochs), and accuracy distribution by DyNED quantisation levels. Bottom row: accuracy by hidden size, parallel coordinates comparing continuous hyperparameters of top 10 (blue) and bottom 10 (red) trials, and accuracy by max delay. Of 65 total trials, 33 completed, 25 were pruned, 5 failed, and 2 were interrupted.

Table 51 lists the best hyperparameters identified by the optimisation. The full 250-epoch retraining at these parameters reached 75.07% best validation accuracy at epoch 48 and 72.96% overall test accuracy.

Table 51: Best hyperparameters for the SSC cAdLIF+DyNED SNN, selected from 65 Optuna trials (50 epochs each). Validation accuracy at 50 epochs: 75.94%. Full 250-epoch retraining at these parameters: 75.07% best validation accuracy at epoch 48 / 72.96% overall test accuracy.

Hyperparameter	Value
Learning rate	1.37×10^{-3}
Delay learning rate	3.29×10^{-2}
Weight decay	2.23×10^{-5}
Hidden size	512
Max delay	10
Dropout	0.229
Batch size	256
DyNED quantisation levels	32
Time bins	100

A.13 SSC DyNED_c Hyperparameter Optimisation Analysis

Hyperparameter optimisation for the SSC cAdLIF+DyNED+DyNED_c classifier (Chapter 6) was conducted using Optuna with the TPE sampler over 85 trials, each trained for 50 epochs. Of the 85 trials, 39 completed, 22 were pruned by the median pruner, 2 failed, and 22 were interrupted. The DyNED_c compressor adds one extra hyperparameter to the DyNED-only search space - the DyNED_c chunk size - while the rest of the architecture is inherited from the DyNED-only setup.

Figure 94 presents a six-panel summary of the optimisation. The optimisation history (top left) shows the best-so-far curve climbing through the first 70 trials before plateauing at $\approx 74.8\%$ validation accuracy. The learning curve of the best trial (top centre) demonstrates that accuracy was still rising at epoch 50. The DyNED levels box plot (top right) shows the lowest setting (16) producing the highest accuracy distribution - the same pattern observed in the Speech Commands DyNED-only search, indicating that coarser quantisation produces denser, more informative spike trains for the cAdLIF stack. Hidden size 512 outperformed 256 (bottom left). The parallel coordinates plot (bottom centre) shows top trials clustering around higher delay learning rates (≈ 0.2 - 0.4) and very

low weight decay ($\approx 10^{-5}$). The DyNEDc chunk-size analysis (bottom right) shows chunk size 4 scoring highest, narrowly above 2 and 8.

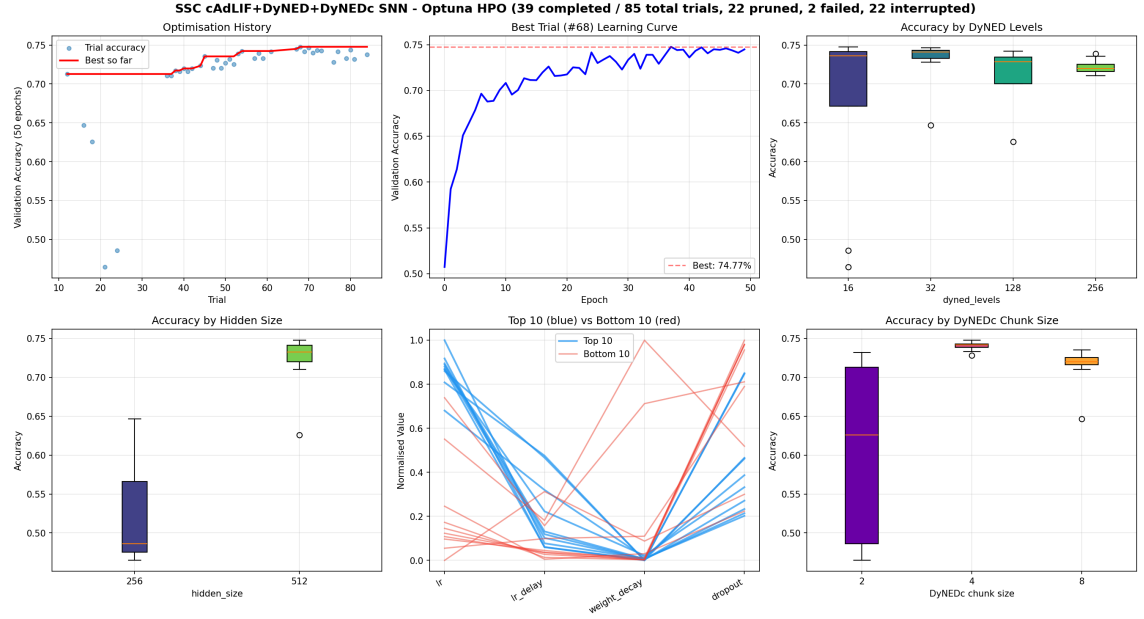


Figure 94: SSC cAdLIF+DyNED+DyNED_c SNN hyperparameter optimisation analysis. Top row: optimisation history with best-so-far curve, learning curve for the best trial (68, 74.77% at 50 epochs), and accuracy distribution by DyNED quantisation levels. Bottom row: accuracy by hidden size, parallel coordinates comparing continuous hyperparameters of top 10 (blue) and bottom 10 (red) trials, and accuracy by DyNED_c chunk size. Of 85 total trials, 39 completed, 22 were pruned, 2 failed, and 22 were interrupted.

Table 52 lists the best hyperparameters identified by the optimisation. The full 250-epoch retraining at these parameters reached 74.7% best validation accuracy at epoch 71 and 72.4% overall test accuracy.

Table 52: Best hyperparameters for the SSC cAdLIF+DyNED+DyNED_c classifier, selected from 85 Optuna trials (50 epochs each). Validation accuracy at 50 epochs: 74.77%. Full 250-epoch retraining at these parameters: 74.7% best validation accuracy / 72.4% overall test accuracy.

Hyperparameter	Value
Learning rate	1.72×10^{-3}
Delay learning rate	2.97×10^{-1}
Weight decay	1.08×10^{-5}
Hidden size	512
Max delay	25
Dropout	0.440
Batch size	256
DyNED quantisation levels	16
Time bins	100
DyNED _c chunk size	4

Appendix B

The code can be found at <https://phd-assets.gollahalli.com/>.