information

Article

Interactive Visualisation of Complex Street Network Graphs from OSM in New Zealand

Jun Yi Ng [1], Jing Ma [1,*], Anuradha Singh [1], Edmund M.-K. Lai [1] and Steven Hayman [2]

[1] School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland 1010, New Zealand; gnw9753@autuni.ac.nz (J.Y.N.); anuradha.singh@aut.ac.nz (A.S.); edmund.lai@aut.ac.nz (E.M.-K.L.)
[2] New Zealand Transport Agency, Wellington 6011, New Zealand; steven.hayman@nzta.govt.nz
* Correspondence: jing.ma@aut.ac.nz

Abstract

Street network graphs model interconnected land transport infrastructure, including roads and intersections, enabling traffic analysis, route planning, and network optimization. Directed network graphs (digraphs) add directionality to these connections, reflecting one-way streets and complex traffic flows. While OpenStreetMap (OSM) offers extensive data, visualizing large-scale directed networks with complex junctions remains computationally challenging for browser-based tools. This paper presents an interactive visualization tool integrating OSM data with the New Zealand Transport Agency's National Network Performance (NNP) analysis toolbox using PyDeck and WebGL. We introduce a directional offset algorithm to resolve edge overlaps and a geometry-aware node placement method for complex intersections. Experimental results demonstrate that our PyDeck implementation significantly outperforms existing solutions like Bokeh and OSMnx. On standard datasets, the system achieves up to 238× faster processing speeds and a 93% reduction in output file size compared to Bokeh. Furthermore, it successfully renders metropolitan-scale networks (∼1.3 million elements) where traditional visualisation tools fail to execute. This visualisation approach serves as a critical debugging instrument for NNP, allowing transport modellers to efficiently identify connectivity errors and validate the structural integrity of large-scale transport models.

Keywords: street network graphs; OSM; interactive visualisation; PyDeck; network analysis; urban transportation

Academic Editor: Aneta Poniszewska-Maranda

Received: 31 October 2025
Revised: 30 November 2025
Accepted: 3 December 2025
Published: 7 December 2025

Citation: Ng, J.Y.; Ma, J.; Singh, A.; Lai, E.M.-K.; Hayman, S. Interactive Visualisation of Complex Street Network Graphs from OSM in New Zealand. *Information* **2025**, *16*, 1088. https://doi.org/10.3390/info16121088

1. Introduction

In recent years, the explosion of digital mapping technologies has modernised our understanding and analysis of urban environments and transportation systems. Street network graphs provide a visual representation of road networks, capturing the connectivity, topology, and attributes of streets and intersections. The streets are represented as links between vertices modelled as intersections [1]. While OpenStreetMap (OSM) provides a rich open-source repository of global geospatial data [2,3], its utility for network analysis is constrained by two factors: First, extensive data cleaning is required to extract intersection-centric graph structures [4,5]. Second, conventional visualisation tools struggle to handle digraphs and complex junctions at scale. This paper addresses these challenges by integrating OSM data with New Zealand Transport Agency's National Network Performance (NNP) platform. We utilize PyDeck 0.9.1, a Python implementation of Uber's

Deck.gl framework that enables GPU-accelerated geospatial visualisation. Python-based visualisation libraries such as Bokeh are widely utilised in data science for their interactivity and ease of integration [6]. However, recent performance analyses indicate that traditional client-side rendering technologies often encounter significant latency and rendering bottlenecks when processing vector datasets exceeding 50,000 to 100,000 features [7]. While tools relying on the DOM or Canvas rendering struggle with these loads, WebGL-based approaches demonstrate superior scalability for massive geospatial datasets [7]. Addressing these limitations, our implementation leverages WebGL via PyDeck to maintain stable performance on 400 MB OSM extracts.

The primary objective of this study is to integrate OSM data with the NNP platform to enhance the interactive visualisation and topological quality assessment of street network graphs. Recent frameworks emphasise the necessity of validating OSM data against inconsistencies such as disconnected edges and geometric errors before their use in navigation applications [8]. Consequently, this tool enables users to systematically inspect the network structure and identify connectivity anomalies, addressing quality indicators such as completeness and positional accuracy [9]. By introducing a novel visualisation method that incorporates digraphs while preserving street symmetry, this paper aims to provide a more intuitive and informative representation of urban street networks. A dedicated debugging tool enables the identification of complex intersection vertices in highway geospatial datasets with high vertex density. By supporting interactive features such as tooltips, zooming, and panning, our approach addresses the visual clutter that typically results from representing numerous vertices and edges in street network graph visualisations [10,11]. This allows users to explore complex street networks at various levels of detail while maintaining a clear visual distinction between different road types and directional flows. It's worth noting that the representation of bidirectional roads as duplicate coordinates in our visualisation is a result of NNP transforming OSM's single 'way' data (which contains tags indicating one or two-way streets) into a digraph structure.

The primary contributions of this paper are:

- A PyDeck-based visualisation pipeline processing 400 MB OSM datasets
- A debugging tool for complex intersection identification and network error detection
- An edge-offset algorithm resolving directional ambiguities at junctions
- Comparison with an existing OSMnx model [12]

The rest of this article is structured as follows. Section 2 presents related work in directed network graphs and interactive visualisation tools. Section 3 discusses the proposed methodology and experimental design, including our algorithmic approach to data processing and visualisation. Section 4 presents the experimental results along with performance analysis and comparison with existing methods, highlighting both the strengths and limitations of our work. Section 3.2 details the processing and transformation of OpenStreetMap data for network visualization. Section 5 presents our visualisation results and discussion, including system capabilities, performance analysis, and limitations. Section 6 concludes the paper, summarising our contributions and proposing potential directions for future work.

2. Related Work

2.1. Directed Network Graphs

Directed network graphs, also known as digraphs, are mathematical structures consisting of vertices and digraphs with incoming and outgoing directions that connect pairs of vertices [13]. Digraphs find applications in various fields, including computer networking and transportation, where they model traffic flow, route optimisation, and navigation systems [14]. Significant research has focused on the representation, analytics, and visuali-

sation of road network graphs, emphasising the importance of both urban street network topology and geometric characteristics [15,16].

2.2. Interactive Visualisation Tool Landscape

Python has become a dominant programming language for data analysis, visualisation, and machine learning, with various tools and libraries aiding in the interactive visualisation of directed network graphs. OSMnx as introduced in [17], provides a streamlined method for acquiring, constructing, analysing, and visualising complex street networks. Its aim is to enhance scalability, generalisability, and interpretability in empirical street network research. With just a single line of Python code, OSMnx stands out for constructing non-planar complex street networks, including walking, driving, or biking networks [18]. It utilises NetworkX, Matplotlib, and GeoPandas libraries, equipping it with network analytic capabilities, straightforward visualisations, and R-tree indexing spatial queries.

However, OSMnx might not fully represent or visualise the directed nature of edges in specific situations, such as one-way streets or directed transportation networks [19]. It could encounter difficulties in accurately illustrating complex intersections within digraphs when multiple digraphs converge or overlap. Additionally, its simplified code might lead to a lack of advanced interactive features and tools for exploring digraphs, such as rendering graphs with numerous directed vertices and edges or filtering specific types of digraphs or vertices based on user-defined criteria. As OSMnx primarily concentrates on generating and visualising street network graphs, its compatibility with specialised graph analysis and visualisation tools for digraphs could be limited, potentially impeding in-depth analysis and exploration of directed network structures. Having OSMnx implemented with it up and running was surprisingly simple meaning its barrier of entry was for everyone, the main issue lies in its ability to render large data sets and downloading large data sets as OSMnx automatically downloads datasets within the area specified. OSMnx is great for basic visualisation of roads but struggles with complex analysis of it whether its visually expressing a junction or a one-way road as previously mentioned.

2.3. Network Graph Debugging Tool

National Network Performance (NNP) is an advanced transport network analysis toolbox used by the New Zealand Transport Agency. Our visualisation tool serves as a robust quality assurance instrument that allows users to validate the network topology and identify integration errors for NNP analysis. It creates an interactive map visualisation of transport network graphs using PyDeck and OSM data. The visualisation aims to highlight intricate vertices and intersections while displaying digraphs representing one-way streets with no overlapping edges. Through the integration of NNP, PyDeck, and OSM data, users can interactively explore and analyse the street network graph, focusing on specific road types and intricate intersections, enhancing understanding and decision-making in urban transportation.

Large city street networks consist of thousands to millions of vertices and edges, demanding high processing capabilities from Graphics Processing Units (GPUs) to render these intricate network structures. Achieving interactivity in visualising these complex directed network graphs involves navigation, panning, zooming, clicking, and filtering for enhanced map visualisations. The integration of NNP, PyDeck, and OSM data addresses a gap highlighted in existing literature on the interactive visualisation of directed street network graphs. Prior research has underscored the importance of interactive visualisation tools for effectively exploring and analysing complex network structures. For instance, studies such as [20,21] have emphasised the significance of interactive visualization in understanding maps and spatial data in directed urban transportation systems.

Challenges in studying street network graphs include limitations in processing capacity to handle large volumes of vertices and edges [22]. Initially Bokeh was used as the main visualisation tool in this research, but has been swapped to PyDeck which largely removes the issue with performance that Bokeh encountered. From the literature reviewed, our approach, which incorporates NNP, PyDeck, and OSM data, aligns with the need to provide a debugging tool for interactive visualisation of intricate street network graph intersections. In debugging large network graphs, we utilise directed network visualisation with PyDeck, a Python library that offers both server (If combined with StreamLit) and Jupyter Notebook graph rendering on HTML web pages. PyDeck simplifies the debugging process for NNP by providing visualisation options with filtering, panning, zooming, tooltips, WebGL and layer-tiling interactive features. PyDeck, which is made by Uber, has a higher emphasis on road network visualisation [23].

2.4. Image-to-Road Network Translation

Recent work by Lu et al. [24] introduces a novel approach to road network visualization and extraction through sequence-to-sequence translation. Their key innovation, RoadNet Sequence, provides a unified representation combining geometric features (landmark coordinates and curve shapes) and topological relationships in a single integer series format. This approach particularly excels at visualizing complex junctions and intersections by

- Resolving visual ambiguities at overlapping intersections through a direction-aware offset algorithm;
- Preserving road hierarchy information via colour-coded classifications;
- Maintaining clear representation of one-way streets and turn restrictions.

Their work demonstrates how translation-based approaches can improve both the accuracy and visual clarity of road network representations, particularly for intricate urban environments with multiple layers of infrastructure. The integration of standard definition maps (SD-Maps) as prior knowledge further enhances the system's ability to accurately visualise complex road structures, even in cases with partial occlusion or limited visibility.

3. Methods

The interactive visualisation tool for the street network is developed on a custom built computer with an Ryzen 5800×, 3070 ti, 32 GB ram and also an Intel i5 11th gen laptop. The study utilises two primary datasets provided by the New Zealand Transport Agency (NZTA), extracted on 1 October 2023. These datasets were selected to benchmark performance across different network scales: 'Wellington_highway_vertices.tsv' (34,423 elements) representing a medium-density urban network, and 'barry_akl.csv' (1,327,491 elements) representing a high-density metropolitan network (Auckland). The geospatial data is provided in both WGS84 (EPSG:4326) format for web-based visualisation and New Zealand Transverse Mercator (NZTM, EPSG:2193) for metric analysis. The Python code is created in either a basic script file which outputs a HTML or can be created in a Jupyter notebook which generates a visualisation in the notebook. The Jupyter notebook can avoid the issue with extremely large datasets that exceed most browsers built-in limit of 1 GB.

The Python code is created in either a basic script file which outputs a HTML file or can be created in a Jupyter notebook which generates a visualisation in the notebook. The Jupyter notebook can avoid the issue with extremely large datasets that exceed most browsers' built-in limit of 1GB. The code relies on the following Python libraries: pandas for data manipulation and analysis, numpy for numerical computations, and PyDeck version 0.9.1.

3.1. System Architecture and Data Flow

Our implementation follows a structured pipeline for processing and visualising road network data, as illustrated in Figure 1. The system architecture consists of several interconnected algorithms that work together to transform raw OSM data into an interactive visualisation.

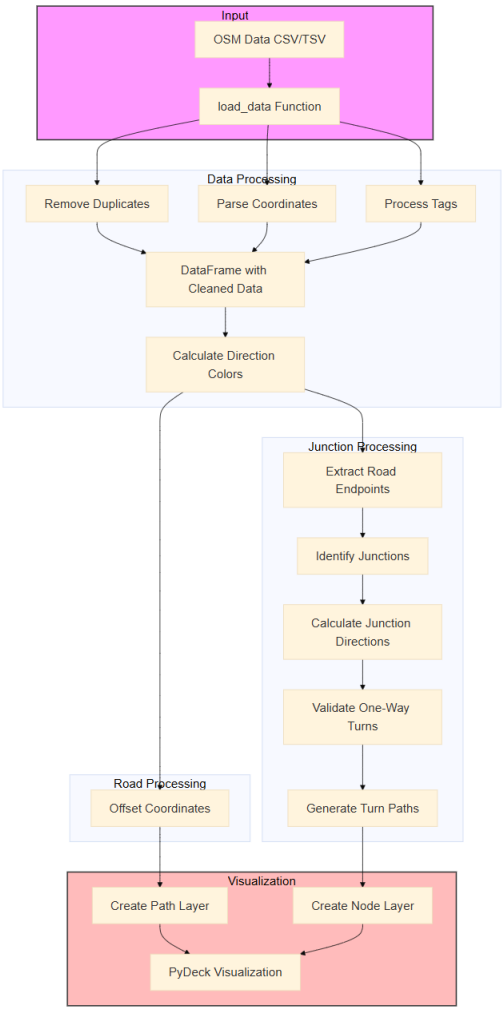

Figure 1. Data processing and visualisation pipeline showing the interconnection between different algorithms and processing stages.

3.2. Data Processing and Visualisation Setup

The pseudo code outlined in Algorithm 1 describes the steps to create an interactive map visualisation showcasing the Wellington Highway Street Network graph using PyDeck. This can be created with any dataset as long as it uses any of the common coordinate systems. It emphasises intricate vertices and intersections as digraphs, representing one-way streets without overlapping edges. The visualisation allows users to explore the road hierarchy by filtering and interactively navigating the highway vertices. Data preprocessing is required to handle the specific export format of the NNP toolbox, where multi-lane roads are often represented as multiple data rows sharing identical centreline geometries (one row per lane). To prevent rendering redundancy and visual artefacts (such as z-fighting or opacity accumulation), the system performs coordinate deduplication. This process identifies and removes rows that share identical 'wkt_wgs84' coordinate sequences and start/end node IDs, retaining a single geometric representation for the physical road segment while preserving the attribute data required for visualisation.

Algorithm 1 Data Processing and Visualisation Setup

Load dataset into DataFrame with specified columns and data types
Remove duplicate entries based on coordinate column
Parse coordinate strings using Algorithm 2 ▷ Coordinate String Parsing
Extract structural features (layer, bridge, tunnel) from tags column
Compute direction colours using Algorithm 3 ▷ Direction-Based Colour
Identify road endpoints and detect junctions (nodes with $\geq$2 connecting roads)
Process junctions using Algorithm 4 ▷ Junction Direction Processing
Validate one-way road turns using Algorithm 5 ▷ One-Way Road Validation
Generate turn path data for junction visualisation using Algorithm 6
Calculate offsets using Algorithm 7 ▷ Street Offset Calculation
Create PyDeck PathLayer for roads and ScatterplotLayer for junctions
Configure interactive view state and tooltip information
Render interactive map visualisation using PyDeck

Algorithm 2 Coordinate String Parsing

procedure PARSECOORDINATES(coord_string)
    if string contains "[array" **then**
        Remove [,], array, and commas from string
        Split cleaned string into numerical tokens
        Convert tokens to floating-point numbers
        Group numbers into pairs, swapping order: (lat, lon)
        return set of (lat, lon) pairs
    else
        Evaluate string as Python literal using `ast.literal_eval`
        Swap coordinate order from (lat, lon) to (lon, lat)
        return list of (lon, lat) pairs
    end if
end procedure

Algorithm 3 Direction-Based Color Calculation

procedure CALCULATECOLOR(coordinates)
    if length(coordinates) < 2 **then**
        return default color ▷ Insufficient points
    end if
    $start \leftarrow$ first coordinate (lon, lat)
    $end \leftarrow$ last coordinate (lon, lat)
    $dx \leftarrow end_{lon} - start_{lon}$
    $dy \leftarrow end_{lat} - start_{lat}$
    $\theta \leftarrow \arctan 2(dy, dx)$ ▷ Calculate bearing angle
    $\theta_{deg} \leftarrow (\theta \times 180/\pi + 360) \mod 360$
    if $0 \leq \theta_{deg} < 180$ **then**
        return blue color ▷ North/East direction
    else
        return red color ▷ South/West direction
    end if
end procedure

Algorithm 4 Junction Direction Processing

```
for all junction points in road network do
    Initialize direction structure with null values
    for all adjacent road segments do
        Calculate bearing angle between junction and adjacent coordinate
        Convert bearing to compass direction:
            0° − 45° → North
            45° − 135° → East
            135° − 225° → South
            225° − 315° → West
        if direction slot empty then
            Store coordinate and road name in direction structure
        end if
    end for
    Generate turn path visualisation nodes using geometric interpolation
end for
```

Algorithm 5 Improved One-Way Turn Validation

```
procedure IsValidOneWayTurn(incoming_dir, turn_dir, turn_data)
    if not turn_data['is_oneway'] then
        return True                          ▷ Allow all turns for bidirectional roads
    end if
    Initialize empty permitted_directions list
    if turn_data['direction_color'] = [0, 0, 255] then          ▷ Blue = north/east
        permitted_directions ← ['north', 'east']
    else if turn_data['direction_color'] = [255, 0, 0] then     ▷ Red = south/west
        permitted_directions ← ['south', 'west']
    else
        return True                          ▷ Default to allowed if direction unclear
    end if
    if turn_dir not in permitted_directions then
        return False          ▷ Invalid if trying to travel against allowed direction
    end if
    Define valid_turn_combinations map:
        ('north', 'east') requires 'east' permission
        ('north', 'west') requires 'west' permission
        ('south', 'east') requires 'east' permission
        ('south', 'west') requires 'west' permission
        ('east', 'north') requires 'north' permission
        ('east', 'south') requires 'south' permission
        ('west', 'north') requires 'north' permission
        ('west', 'south') requires 'south' permission
    required_permission ← valid_turn_combinations[(incoming_dir, turn_dir)]
    if required_permission exists and required_permission not in permitted_directions
then
        return False  ▷ Invalid if turn direction conflicts with road permissions
    end if
    return True                                                  ▷ Turn is valid
end procedure
```

3.3. OpenStreetMap Data Structure

OpenStreetMap (OSM) data [25] presents unique challenges for network visualisation due to its encoding of bidirectional roads. In raw OSM data, a single 'way' with tags indicating one or two-way streets is used to represent roads. However, when this data is transformed by National Network Performance (NNP) into a digraph for network analysis,

bidirectional roads are represented as duplicate coordinate sequences in reverse order. For example, consider the following coordinate sequences from Buller Street in Wellington:

```
[(-41.2934713, 174.7708912), (-41.2934969, 174.7709145),
 (-41.2935152, 174.7709333), (-41.2935389, 174.7709976),
 (-41.2935606, 174.7710564)]

[(-41.2935606, 174.7710564), (-41.2935389, 174.7709976),
 (-41.2935152, 174.7709333), (-41.2934969, 174.7709145),
 (-41.2934713, 174.7708912)]
```

These coordinates represent the same physical road segment but in opposite directions, resulting from NNP's transformation of OSM data into a digraph structure. When visualised directly, these duplicate segments would occupy identical spatial positions, making it impossible to distinguish between opposite travel directions. This transformed data structure, while logical for network analysis and querying, creates significant visualisation challenges when attempting to represent directional traffic flows clearly.

Bidirectional roads represent a majority of urban road networks, and without proper processing, visualisations would appear cluttered and ambiguous, with overlapping lines making it difficult to interpret network structure and traffic flow patterns. This characteristic of NNP-transformed OSM data makes specialised processing not merely advantageous but essential for effective network visualisation and error identification.

Algorithm 6 Turn Path Generation

procedure GENERATETURNPATHS(junction_key, junction_data)
    Get junction type (T-junction, 4-way, etc.)
    Get junction coordinates
    $node\_distance\_ratio \leftarrow 0.2$ ▷ Position nodes 20% from junction
    for all incoming directions with valid data **do**
        Get adjacent point coordinates from road data
        Calculate interpolated position along road segment:
        $node\_position_x \leftarrow junction_x + (adjacent_x - junction_x) \times node\_distance\_ratio$
        $node\_position_y \leftarrow junction_y + (adjacent_y - junction_y) \times node\_distance\_ratio$
        Initialise empty possible_turns list
        for all potential turn directions turn_dir **do**
            if turn_dir = incoming_dir **then**
                continue ▷ Skip the incoming direction itself
            end if
            Validate turn using Algorithm 5 ▷ Check one-way constraints
            if turn is valid **then**
                Determine turn label ('straight', 'left', 'right', etc.)
                Add turn to possible_turns
            end if
        end for
        Create node with position, direction, and valid turns data
    end for
    return list of visualisation nodes
end procedure

Algorithm 7 Street Offset Calculation

procedure OFFSETCOORDINATES(coordinates, is_oneway, offset_distance)
  if is_oneway is True **then**
    return original coordinates unchanged
  end if
  Initialize empty offset_coords list
  for each consecutive point pair $(start, end)$ in coordinates **do**
    Calculate direction vector $\vec{d} = (dx, dy) = (end_x - start_x, end_y - start_y)$
    Compute length $l = \sqrt{dx^2 + dy^2}$
    if $l < 10^{-10}$ **then**
      Add original start point to offset_coords
      continue to next iteration
    end if
    Calculate perpendicular direction:
    $\vec{p} = (-\frac{dy}{l} \cdot \delta, \frac{dx}{l} \cdot \delta)$ where δ=offset_distance
    Create new offset point:
    $new\_start = (start_x + p_x, start_y + p_y)$
    Add new_start to offset_coords
    if processing second-to-last point **then**
      Create offset end point:
      $new\_end = (end_x + p_x, end_y + p_y)$
      Add new_end to offset_coords
    end if
  end for
  return offset_coords
end procedure

3.4. Coordinate String Parsing

The Coordinate String Parsing algorithm (Algorithm 2) addresses the challenge of handling diverse coordinate data formats in transportation network datasets. The algorithm supports two primary input formats: array-style strings and Python literal strings. For array-style inputs, it performs a series of string cleaning operations to remove special characters and delimiters before converting the cleaned string into numerical coordinate pairs. For Python literal strings, it utilizes the `ast.literal_eval` function to safely evaluate the string as a Python expression. In both cases, the algorithm handles coordinate order conversion, ensuring that the output consistently follows the (longitude, latitude) convention required for mapping applications. This standardisation is crucial for maintaining data consistency across different data sources and visualisation platforms.

3.5. Junction Direction Processing

The Junction Direction Processing algorithm (Algorithm 4) implements a crucial component for analysing road network topology by identifying and categorising junction points based on their connecting road segments. This algorithm processes each junction point in the network, determining the cardinal directions (North, East, South, West) of all adjacent road segments. The process begins by initialising a direction structure for each junction point, then iterates through all adjacent road segments to calculate their bearing angles. These angles are converted to compass directions using a quantised approach, where the 360-degree compass is divided into four 90-degree sectors. The algorithm stores the first encountered road segment for each direction, effectively creating a simplified representation of the junction's connectivity. This directional information is essential for generating turn path visualisations and understanding traffic flow patterns at intersections.

The quantisation of bearing angles into four 90° sectors (North, East, South, West) serves a specific theoretical purpose in network debugging. Real-world intersections

rarely exhibit perfect orthogonality; a road may approach a junction at 85° or 95° due to terrain constraints or digitising noise. For the purpose of topological validation and turn-path visualisation, preserving these minor geometric irregularities increases cognitive load without adding semantic value. By binning continuous angles into discrete cardinal directions, the algorithm filters geometric noise, allowing the user to focus on the logical connectivity of the intersection rather than its precise geometric footprint.

3.6. Turn Path Generation

The Turn Path Generation Algorithm 6 has been enhanced to incorporate comprehensive one-way road validation. For each junction, the algorithm identifies all incoming directions and calculates node positions using geometric interpolation. This ensures that visualisation nodes appear correctly positioned along the actual road geometry, regardless of the road's orientation.

For each incoming direction, the algorithm evaluates all potential turn directions using the One-Way Road Validation procedure (Algorithm 5). This validation step filters out turns that would violate one-way restrictions, ensuring that the visualisation accurately represents the navigable network topology. Valid turns are then categorised as 'straight', 'left', or 'right' based on their relationship to the incoming direction, and this information is incorporated into the visualisation node's metadata for display in tooltips.

For bidirectional roads, the algorithm simply allows all turns. However, for one-way roads, the process requires multiple validation steps. First, the algorithm determines the permitted travel directions based on the road's directional colour encoding, where blue indicates north/east travel permission and red indicates south/west permission. It then verifies that the intended turn direction aligns with these permitted directions.

The algorithm's core innovation is its handling of the relationship between incoming direction and turn direction. Through a comprehensive mapping of valid turn combinations, the system verifies that the physical turn movement is consistent with the directional permissions of the target road. For example, a vehicle approaching from the south and turning east must verify that eastbound travel is permitted on the target road. This two-step validation ensures accurate representation of complex urban traffic patterns, particularly at intersections where multiple one-way streets converge.

3.7. Street Offset Calculation

The Street Offset Calculation algorithm (Algorithm 7) implements a sophisticated approach to visualising bidirectional streets by creating parallel offset paths for opposite traffic directions.

The offset distance parameter δ was empirically selected to approximate standard urban lane widths (typically 3.5–3.7 m). This parameter choice balances two competing visualisation constraints: (1) the need for sufficient visual separation to distinguish bidirectional flows at high zoom levels, and (2) the need to maintain spatial coherence at lower zoom levels. An offset derived from physical lane widths ensures that the visualisation remains semantically meaningful to transport engineers, as the rendered separation corresponds to the physical footprint of the road median, thereby avoiding visual artefacts where road segments might appear disconnected from their intersections.

3.8. Direction-Based Colour Calculation

The Direction-Based Colour Calculation algorithm (Algorithm 3) provides a method for visually differentiating road segments based on their overall direction, enhancing the readability of the network visualisation. The algorithm takes a list of coordinates representing a road segment and calculates its predominant direction by examining the bearing between its start and end points. It employs the arctangent function ($\arctan 2$) to

compute the bearing angle, which is then normalised to the range $[0, 360)$ degrees. Based on this angle, the algorithm assigns one of two colors: blue for segments oriented in the North/East directions (0° to 180°) and red for segments oriented in the South/West directions (180° to 360°). This binary colour coding scheme helps users quickly identify the general flow direction of road segments, particularly useful for understanding traffic patterns in one-way street systems.

4. Results

4.1. PyDeck vs. Bokeh Performance Analysis

As shown in Table 1, PyDeck demonstrates remarkable performance advantages over Bokeh across all measured metrics. For the Wellington dataset (12.5 MB), PyDeck achieves a total execution time of 1.59 s compared to Bokeh's 378.87 s, a 238× improvement in processing speed. Memory efficiency is similarly striking, with PyDeck using only 177.05 MB compared to Bokeh's 856.34 MB, representing a 79% reduction in memory consumption. The output file size shows an even more dramatic improvement, with PyDeck generating a 49.05 MB file versus Bokeh's 715.01 MB, a 93% reduction in size. The resulting visualisations for the Wellington dataset are compared in Figures 2 and 3.

Most notably, when scaling to the larger Auckland dataset (392.2 MB) as presented in Table 2, Bokeh failed to complete the visualisation task (represented as DNF—Did Not Finish), while PyDeck successfully processed the data in 26.88 s with manageable memory usage of 1621.29 MB.

Table 1. Bokeh and PyDeck Comparison in Wellington Dataset (∼12.5 MB).

Metric	PyDeck	Bokeh
Total Execution (s)	1.59	378.87
Output Size (MB)	49.05	715.01
Memory Usage (MB)	177.05	856.34
Frame Rate (FPS)	60	15–30

Table 2. Bokeh and PyDeck Comparison in Auckland Dataset (∼392.2 MB).

Metric	PyDeck	Bokeh
Total Execution (s)	26.88	DNF
Output Size (MB)	821.55	DNF
Memory Usage (MB)	1621.29	DNF
Frame Rate (FPS)	60	DNF

4.2. Comparison with Existing Methods

Our performance analysis includes comparison with OSMnx, a widely-used existing method for road network visualisation and analysis. OSMnx, combined with Folium for visualisation as demonstrated in previous work, represents the current state-of-the-art in road network analysis. While OSMnx provides comprehensive network analysis capabilities, our PyDeck-based approach offers significant improvements in processing efficiency and visualisation performance, particularly for large-scale networks, as detailed in Tables 3 and 4.

Table 3. OSMnx and PyDeck Comparison in Wellington Dataset (∼12.5 MB).

Metric	PyDeck	OSMnx + Folium
Total Execution (s)	1.59	42.76
Output Size (MB)	49.05	708.15
Memory Usage (MB)	177.05	899.60
Frame Rate (FPS)	60	60

Table 4. OSMnx and PyDeck Comparison in Auckland Dataset (∼392.2 MB).

Metric	PyDeck	OSMnx + Folium
Total Execution (s)	26.88	175.79
Output Size (MB)	821.55	1785.64
Memory Usage (MB)	1621.29	1977.25
Frame Rate (FPS)	60	60

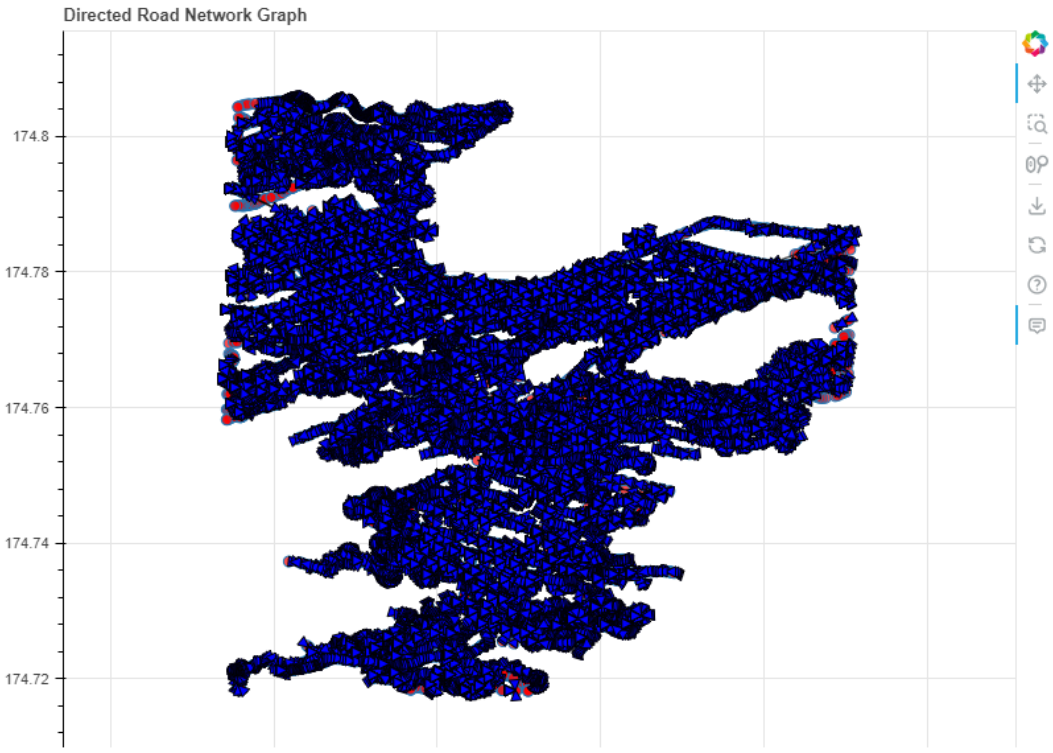

Figure 2. Bokehvisualization of Wellington road network showing limited visual clarity and heavy resource usage (378.87 s execution time, 856.34 MB memory). (Colour Legend: Blue arrows = road direction indicators; Red markers = junction nodes).

4.3. PyDeck vs. OSMnx Performance Analysis

The comparison between PyDeck and OSMnx reveals significant performance differences, particularly in processing efficiency and resource utilisation. As evidenced in Table 3, with the Wellington dataset, PyDeck completes processing in 1.59 s compared to OSMnx's 42.76 s, a 27× improvement in execution speed. PyDeck maintains this efficiency advantage with the larger Auckland dataset (Table 4), completing in 26.88 s versus OSMnx's 175.79 s.

Memory utilisation shows similar improvements, with PyDeck consuming 177.05 MB versus OSMnx's 899.60 MB for the Wellington dataset, and managing the Auckland dataset with 1621.29 MB compared to OSMnx's 1977.25 MB. While both tools achieve 60 FPS in rendering as shown in Tables 3 and 4, PyDeck's significantly smaller output file sizes (49.05 MB vs 708.15 MB for Wellington, 821.55 MB vs 1785.64 MB for Auckland) make it more practical for web deployment and sharing. The visual output of the OSMnx

implementation for Wellington is shown in Figure 4. Comparisons of the larger Auckland dataset are presented in Figure 5 (OSMnx) and Figure 6 (PyDeck).

PyDeck demonstrates superior performance across several key metrics:

- Processing Efficiency: PyDeck processes both small and large datasets significantly faster, with execution times 27–238× faster than competing tools.
- Memory Optimisation: Consistently lower memory usage, requiring 79–80% less memory than Bokeh and 20–80% less than OSMnx.
- Output Size: Generates substantially smaller output files, with reductions of 93% compared to Bokeh and 54–93% compared to OSMnx.
- Scalability: Successfully handles large datasets where Bokeh fails completely, while maintaining better performance than OSMnx.

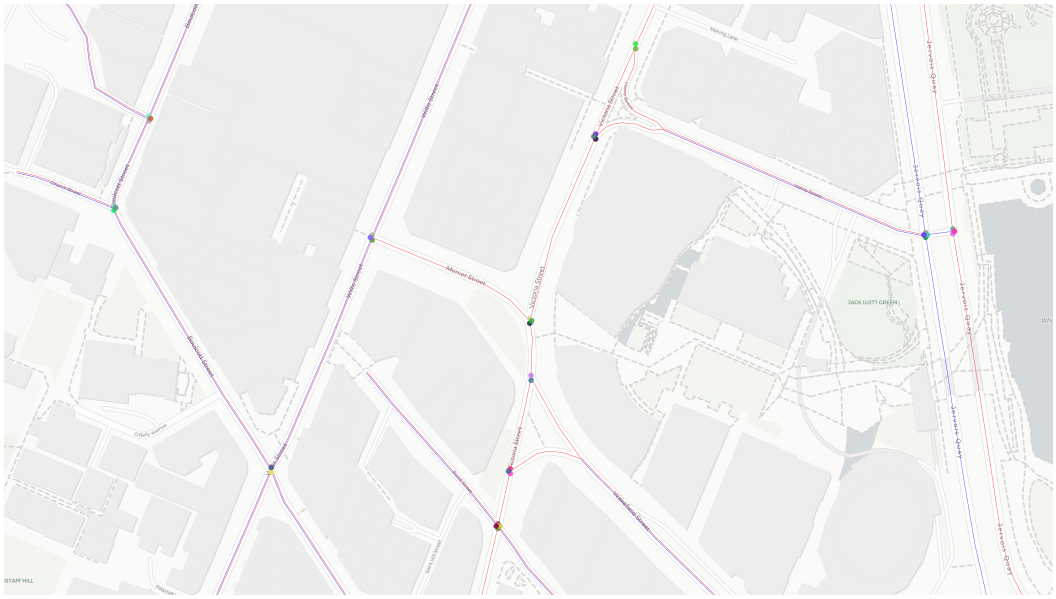

Figure 3. PyDeck visualization of Wellington with improved clarity and efficient resource usage (1.59 s execution time, 177.05 MB memory). The visual density represents the macroscopic scale; specific edge and node interactions are detailed in subsequent figures. (Colour Legend: Blue lines = North/East traffic flow; Red lines = South/West traffic flow. Note: Dense overlapping lines represent high-density urban infrastructure).

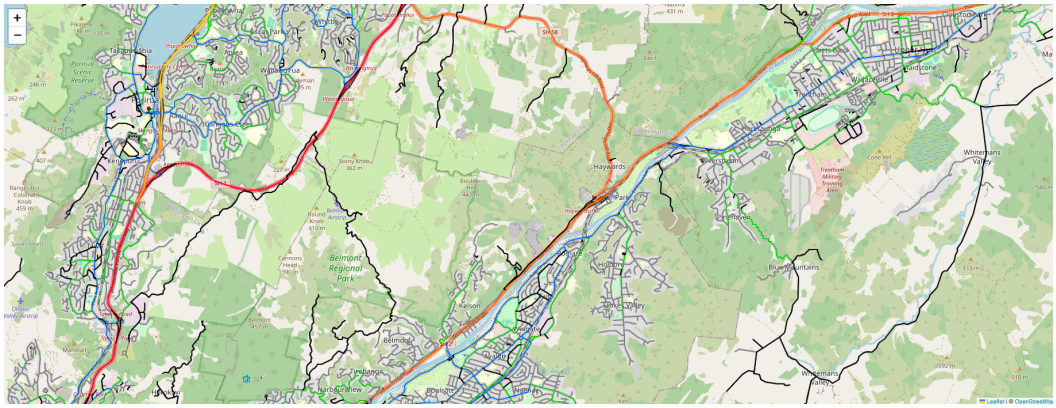

Figure 4. OSMnx visualization of Wellington showing detailed road hierarchy but with significant resource overhead (42.76 s execution time, 708.15 MB memory). The dense overlapping content accurately reflects the complexity of the metropolitan road network at a macro scale. (Colour Legend: Different line colors represent distinct road hierarchy levels, such as motorways, arterial, and residential streets).

Figure 5. OSMnxvisualization of Auckland showing detailed road hierarchy but with significant resource overhead (175.79 s execution time, 1977.25 MB memory). The dense overlapping content accurately reflects the complexity of the metropolitan road network at a macro scale. (Colour Legend: Different line colors represent distinct road hierarchy levels, such as motorways, arterial, and residential streets).

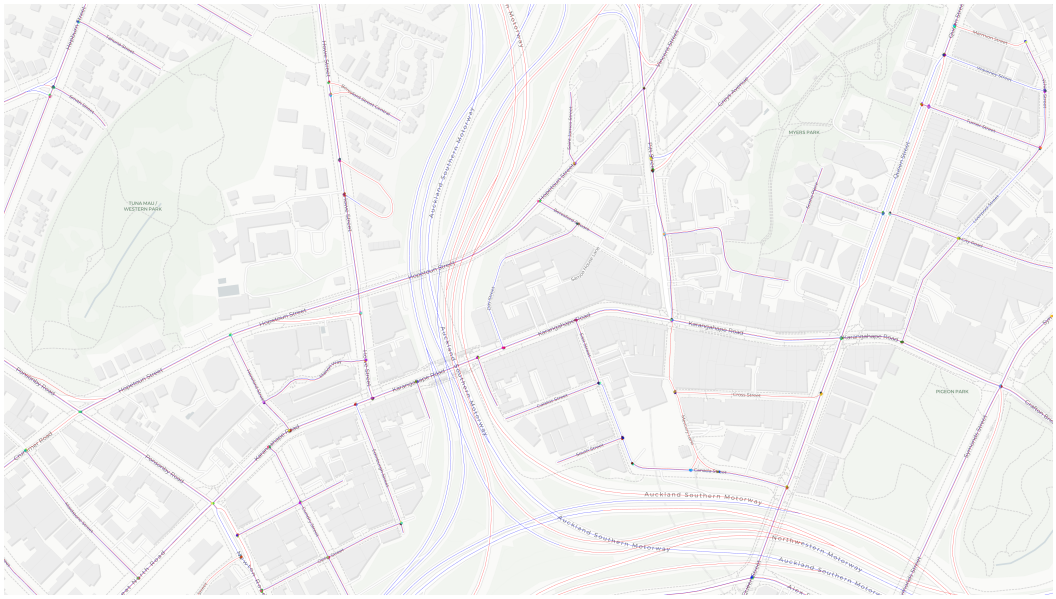

Figure 6. PyDeck visualization of Auckland demonstrating superior performance with large datasets (26.88 s execution time, 1621.29 MB memory). Overlapping lines indicate high-density urban infrastructure. (Color Legend: Blue lines = North/East traffic flow; Red lines = South/West traffic flow; Distinct coloured nodes = Turn possibilities).

4.4. Visualisation Examples

Our visualisation tool effectively demonstrates several key features of Wellington's road network infrastructure through three distinct examples. Figures 7–9 showcase different aspects of the network visualisation capabilities.

Figure 7. Interactive junction visualisation showing turn possibilities at a 4-way intersection. The tooltip displays available turns for Jervois Quay. (Colour Legend: Blue lines = North/East-bound traffic; Red lines = South/West-bound traffic; Distinct coloured nodes = Turn possibilities).

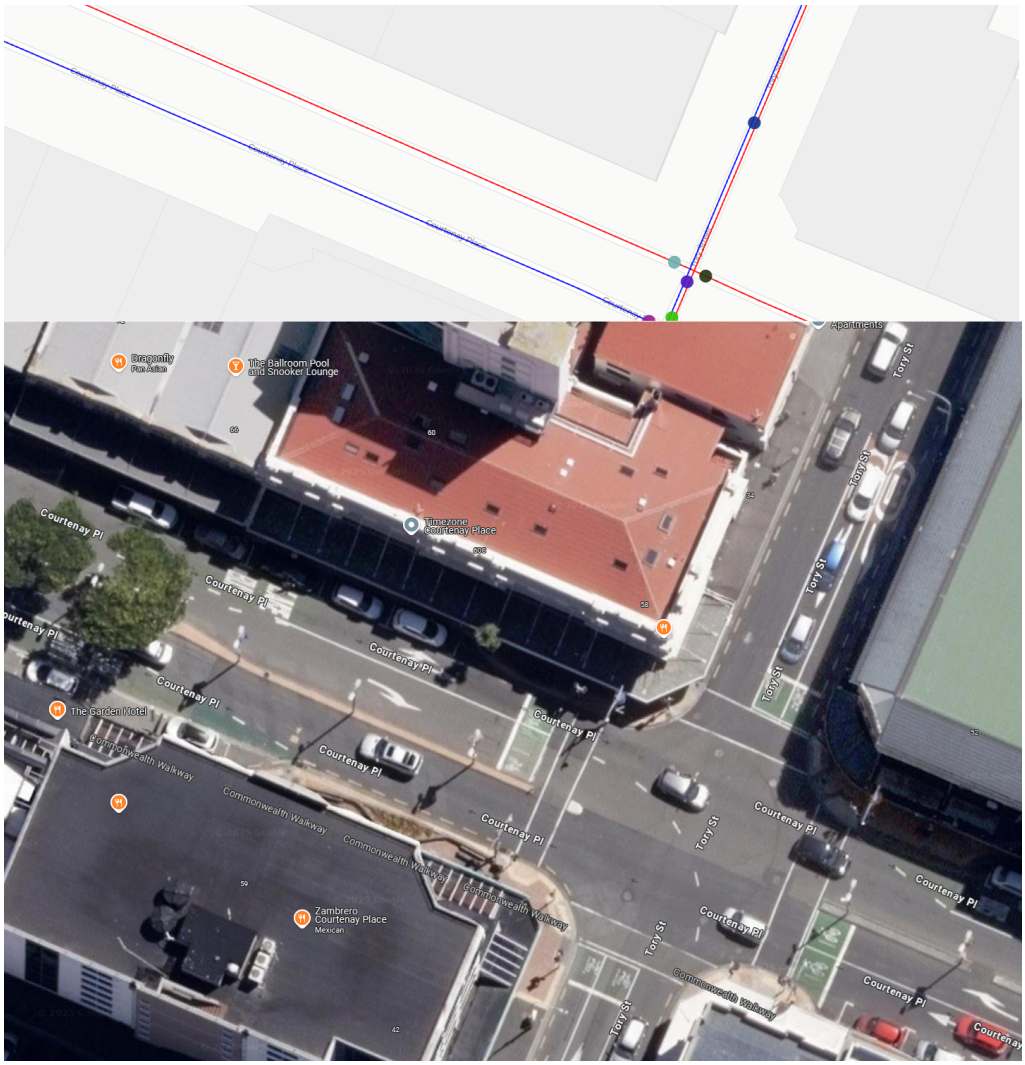

Figure 8. Visualization of one-way roads in Wellington's CBD. The single-colored lines indicate one-way traffic flow on Courtenay Place, verified by satellite imagery. (Colour Legend: Blue = North/East flow; Red = South/West flow. Parallel Red/Blue lines indicate bidirectional roads; Distinct coloured nodes = Turn possibilities).

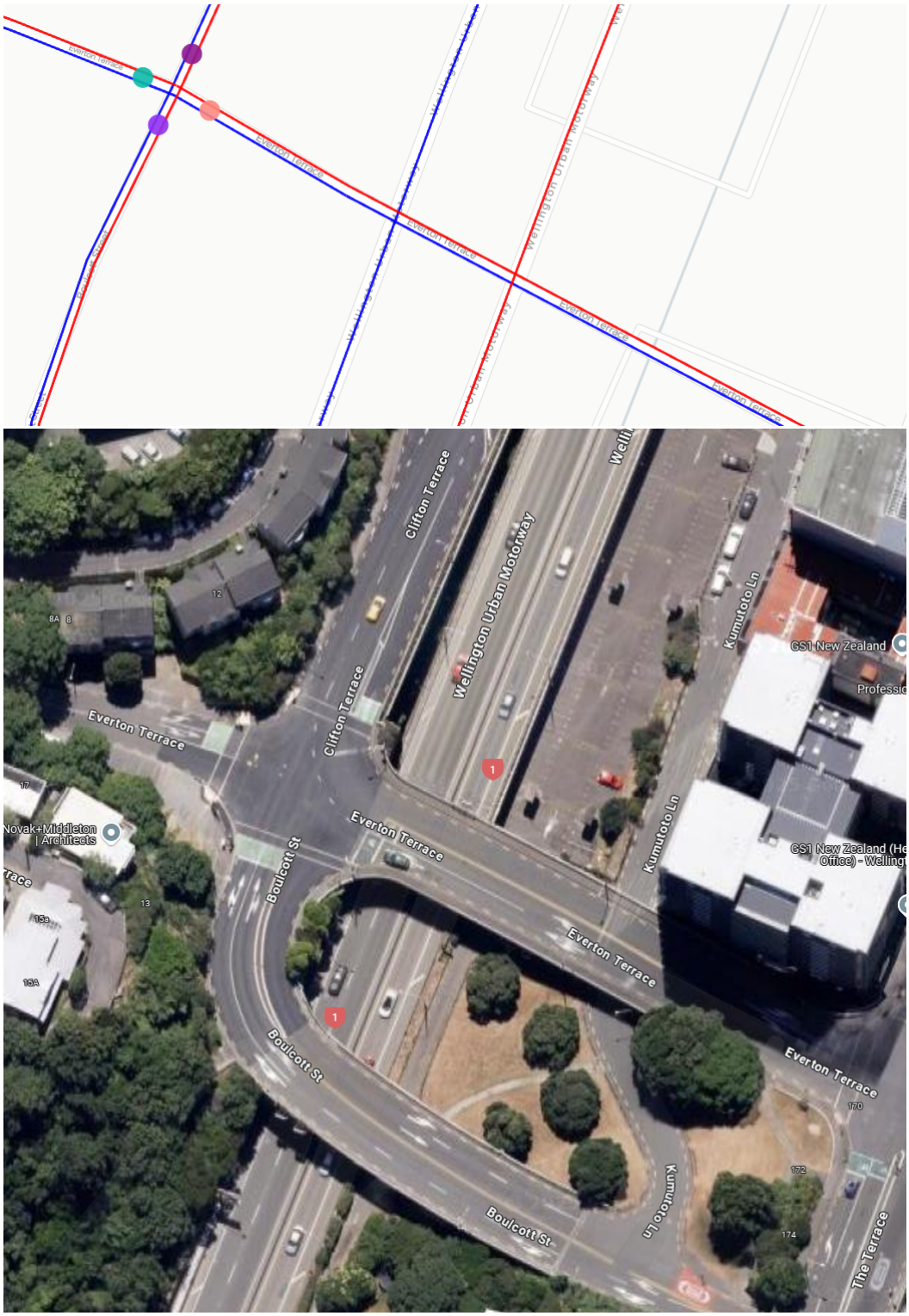

Figure 9. Multi-layer road representation showing the Wellington Urban Motorway passing beneath Salamanca Road. While these roads appear to intersect in 2D coordinates, they exist at different elevations—the motorway in a tunnel (lower layer) and Cliffton Terrace at ground level (base layer). (Colour Legend: Blue/Red lines = traffic flow direction. The absence of a node marker at the crossover point visually confirms the grade separation).

5. Discussion

5.1. Directed Network Graph Implementation

Using algorithms defined in the design section, we create directed network graphs that display vertices as dots and edges as straight lines. Multiple edges describe bidirectional traffic flow, while single-colored lines depict one-way streets. Each directed network graph represents different intricate intersections, where the directionality and one-way streets are outlined as separate edges without overlapping.

5.2. Performance Analysis

The integration of OpenStreetMap data with National Network Performance (NNP) via PyDeck has proven to be an effective approach for visualising and debugging complex street networks. Our system successfully addresses several key challenges in transportation network visualisation that previous approaches struggled with, while also helping to interrogate the network and identify errors for NNP analysis.

The performance analysis clearly demonstrates the superiority of PyDeck over both Bokeh and OSMnx for large-scale network visualisation tasks. The dramatic improvements in processing speed (up to 238× faster than Bokeh), memory efficiency (up to 80% reduction), and output file size (up to 93% smaller) represent significant advancements in making large-scale urban network visualisation practical for real-world applications. These improvements are particularly valuable for urban planning and transportation engineering contexts where responsiveness and interactivity are essential for effective analysis.

5.3. System Capabilities and Features

The representation of bidirectional flows and overlapping edges in node-link diagrams presents a significant challenge in network visualisation, often resulting in visual clutter that obscures topological structures [26]. While recent approaches have explored matrix-based or dynamic visualisations to mitigate this issue [27], geometric displacement remains necessary for preserving spatial context in street networks. Our directional offset algorithm addresses this by creating parallel offset paths for opposite traffic directions and implementing a direction-based colour scheme, our approach significantly enhances the readability of the network structure. This visualisation technique is particularly valuable for understanding traffic flow patterns and identifying potential bottlenecks or design issues in urban transportation networks.

Another key contribution is the system's ability to correctly represent multi-layer infrastructure. As noted by Barranquero et al. [28], standard 2D projections of OSM data often fail to capture vertical relationships, leading to ambiguity in complex urban environments. By utilising the layer attribute from the OSM metadata, our visualisation can accurately depict the 3D nature of modern urban infrastructure within a 2D visual space. This capability is crucial for understanding the true connectivity of transportation networks in complex urban environments.

5.4. Scalability and Limitations

The successful handling of the Auckland dataset (1,327,491 vertices and edges) demonstrates the scalability of our approach, making it viable for analysing metropolitan-scale transportation networks. This represents a significant improvement over existing tools that typically struggle with datasets exceeding 100,000 elements.

While our implementation has shown considerable promise, there are limitations that should be acknowledged. The system currently relies on preprocessed OSM data, which requires a separate conversion step before visualisation. A significant limitation within the context of network interrogation is the reliance on visual adjacency to identify

connections between network elements. Future work should focus on explicitly visualising topological links rather than relying on visual adjacency. Boeing [29] demonstrates that relying on spatial proximity (lines touching) often leads to "intersection miscounts" where grade-separated crossings (like overpasses) are falsely identified as nodes. Explicit link visualisation would mitigate these non-planar errors and improve the identification of true network connectivity. Additionally, the current implementation does not incorporate real-time traffic data or support for multimodal transportation analysis, which would be valuable extensions for comprehensive urban mobility planning.

6. Conclusions

This paper introduces an innovative method for visualising and debugging complex street network graphs by integrating OpenStreetMap data with National Network Performance (NNP) while utilising PyDeck's GPU-accelerated rendering capabilities. Our approach effectively tackles key challenges in transportation network visualisation, including managing large-scale datasets and accurately representing complex intersections, while enabling users to interrogate the network and identify errors for NNP analysis.

The primary accomplishments of this work include the development of an efficient visualisation pipeline capable of handling datasets over 400 MB, encompassing millions of edges and vertices, while maintaining interactive performance. By implementing advanced algorithms for junction direction processing, coordinate parsing, and street offset calculation, we have successfully represented intricate road network features such as multi-layer intersections, one-way streets, turn restrictions, and seamless junction visualisations. Notably, the directional offset algorithm, in particular, has proven effective in resolving visual ambiguities at complex junctions while preserving the network's hierarchical structure.

Testing with datasets from both Wellington and Auckland has validated the tool's scalability and effectiveness across diverse urban environments. The implementation consistently delivers responsive performance during interactive operations such as panning and zooming, even when handling datasets with over 1 million vertices and edges. The incorporation of layer-based visualisation enables clear representation of complex infrastructure elements such as bridges, tunnels, and overlapping roadways, while the interactive tooltip system offers instant access to relevant metadata for debugging.

Future work could explore several promising directions to enhance the tool's capabilities:

- Integration with real-time traffic data to enable dynamic flow visualisation and analysis.
- Implementation of advanced filtering mechanisms for specific network attributes and structural patterns.
- Development of automated anomaly detection for identifying potential data quality issues in the road network.
- Extension of the visualisation system to support additional transportation modes and infrastructure types.

This research contributes to the broader field of transportation network analysis by providing a robust, scalable solution for visualising and debugging complex street networks. The tools and methodologies developed here offer valuable capabilities for urban planners, transportation engineers, and researchers working with large-scale road network data.

Author Contributions: Conceptualization, J.M. and E.M.-K.L.; methodology, J.Y.N., J.M. and A.S.; software, J.Y.N.; validation, S.H.; formal analysis, J.Y.N.; investigation, J.Y.N.; resources, S.H.; data curation, S.H.; writing—original draft preparation, J.Y.N.; writing—review & editing, J.M. and A.S.; visualisation, J.Y.N.; supervision, J.M. and A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Auckland University of Technology, 2024–2025 summer research project.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author(s).

Acknowledgments: This project is supported by New Zealand Transport Agency. The datasets and related assistance offered by NZTA are gratefully acknowledged. We would also like to thank Milcent Mugandane for her initial exploration of alternative visualisation methods which helped inform the direction of this research.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Neira, M.; Murcio, R. Graph representation learning for street networks. *arXiv* **2022**, arXiv:2211.04984. [CrossRef]
2. Weiss, D.J.; Nelson, A.; Gibson, H.; Temperley, W.; Peedell, S.; Lieber, A.; Hancher, M.; Poyart, E.; Belchior, S.; Fullman, N.; et al. A global map of travel time to cities to assess inequalities in accessibility in 2015. *Nature* **2018**, *553*, 333–336. [CrossRef] [PubMed]
3. Grinberger, A.Y.; Minghini, M.; Yeboah, G.; Juhász, L.; Mooney, P. Bridges and barriers: An exploration of engagements of the research community with the OpenStreetMap community. *ISPRS Int. J. Geo-Inf.* **2022**, *11*, 54. [CrossRef]
4. Boeing, G. Street network models and measures for every US City, county, urbanized area, census tract, and zillow-defined neighborhood. *Urban Sci.* **2019**, *3*, 28. [CrossRef]
5. Karduni, A.; Kermanshah, A.; Derrible, S. A protocol to convert spatial polyline data to network formats and applications to world urban road networks. *Sci. Data* **2016**, *3*, 160046. [CrossRef]
6. Addepalli, L.; Lokhande, G.; Sakinam, S.; Hussain, S.; Mookerjee, J.; Vamsi, U.; Waqas, A.; Vidya Sagar, S.D. Assessing the Performance of Python Data Visualization Libraries: A Review. *Int. J. Comput. Eng. Res. Trends* **2023**, *10*, 28–39. [CrossRef]
7. Balla, D.; Gede, M. Vector Data Rendering Performance Analysis of Open-Source Web Mapping Libraries. *ISPRS Int. J. Geo-Inf.* **2025**, *14*, 336. [CrossRef]
8. Hosseini, R.; Tong, D.; Lim, S.; Sohn, G.; Gidófalvi, G. A framework for performance analysis of OpenStreetMap data in navigation applications: The case of a well-developed road network in Australia. *Ann. GIS* **2025**, *31*, 233–250. [CrossRef]
9. Moradi, M.; Roche, S.; Mostafavi, M.A. Exploring five indicators for the quality of OpenStreetMap road networks: A case study of Québec, Canada. *Geomatica* **2022**, *75*, 178–208. [CrossRef]
10. Zhao, Y.; She, Y.; Chen, W.; Lu, Y.; Xia, J.; Chen, W.; Liu, J.; Zhou, F. Eod edge sampling for visualizing dynamic network via massive sequence view. *IEEE Access* **2018**, *6*, 53006–53018. [CrossRef]
11. Zhao, X.; Xu, J.; Yang, J.; Duan, J. A global urban road network self-adaptive simplification workflow from traffic to spatial representation. *Sci. Data* **2025**, *12*, 883. [CrossRef] [PubMed]
12. Khamis, A.; Wang, Y. From Road Network to Graph. 2021. Available online: https://smartmobilityalgorithms.github.io/book/content/GraphSearchAlgorithms/RoadGraph.html (accessed on 10 November 2025).
13. Riihimäki, H. Simplicial-Connectivity of Directed Graphs with Applications to Network Analysis. *SIAM J. Math. Data Sci.* **2023**, *5*, 800–828. [CrossRef]
14. Shen, G.; Han, X.; Chin, K.; Kong, X. An attention-based digraph convolution network enabled framework for congestion recognition in three-dimensional road networks. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 14413–14426. [CrossRef]
15. Barthélemy, M.; Flammini, A. Modeling urban street patterns. *Phys. Rev. Lett.* **2008**, *100*, 138702. [CrossRef]
16. Xie, F.; Levinson, D. Measuring the structure of road networks. *Geogr. Anal.* **2007**, *39*, 336–356. [CrossRef]
17. Boeing, G. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Comput. Environ. Urban Syst.* **2017**, *65*, 126–139. [CrossRef]
18. Garcia-Robledo, A.; Zangiabady, M. Dash Sylvereye: A Python Library for Dashboard-Driven Visualization of Large Street Networks. *IEEE Access* **2023**, *11*, 121142–121161. [CrossRef]
19. Alawadi, K.; Nguyen, N.H.; Alkaabi, M. The edge and the center in neighborhood planning units: Assessing permeability and edge attractiveness in Abu Dhabi. *Transportation* **2023**, *50*, 677–705. [CrossRef]
20. Sobral, T.; Galvão, T.; Borges, J. Visualization of urban mobility data from intelligent transportation systems. *Sensors* **2019**, *19*, 332. [CrossRef]

21. Conroy, M. Networks, Maps, and Time: Visualizing Historical Networks Using Palladio. *DHQ Digit. Humanit. Q.* **2021**, *15*. [CrossRef]
22. Goss, Q.; Akbaş, M.İ.; Jaimes, L.G.; Sanchez-Arias, R. Street network generation with adjustable complexity using k-means clustering. In Proceedings of the 2019 SoutheastCon, Huntsville, AL, USA, 11–14 April 2019; IEEE: Washington, DC, USA, 2019; pp. 1–6.
23. Uber Technologies, Inc. PyDeck Documentation. 2024. Available online: https://pydeck.gl/ (accessed on 10 November 2025).
24. Lu, J.; Peng, R.; Cai, X.; Xu, H.; Wen, F.; Zhang, W.; Zhang, L. Translating Images to Road Network: A Sequence-to-Sequence Perspective. *arXiv* **2024**, arXiv:2402.08207. [CrossRef]
25. OpenStreetMap Contributors. OpenStreetMap Wiki: Tags, Layers, and Junctions Documentation. 2025. Available online: https://wiki.openstreetmap.org/wiki/Map_Features (accessed on 31 October 2025).
26. Filipov, V.; Arleo, A.; Miksch, S. Are We There Yet? A Roadmap of Network Visualization from Surveys to Task Taxonomies. *Comput. Graph. Forum* **2023**, *42*, e14794. [CrossRef]
27. Burch, M.; Ten Brinke, K.B.; Castella, A.; Karray, G.; Peters, S.; Shteriyanov, V.; Vlasvinkel, R. Dynamic graph exploration by interactively linked node-link diagrams and matrix visualizations. *Vis. Comput. Ind. Biomed. Art* **2021**, *4*, 23. [CrossRef]
28. Barranquero, M.; Olmedo, A.; Gómez, J.; Tayebi, A.; Hellín, C.J.; Saez de Adana, F. Automatic 3D Building Reconstruction from OpenStreetMap and LiDAR Using Convolutional Neural Networks. *Sensors* **2023**, *23*, 2444. [CrossRef]
29. Boeing, G. Topological Graph Simplification Solutions to the Street Intersection Miscount Problem. *Trans. GIS* **2025**, *29*, e70037. [CrossRef]