UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

THALES ELERO CERVI

A COMPARATIVE ANALYSIS OF GRAPH NEURAL NETWORKS FOR ESTIMATING FREE-FLOW TRAVEL TIMES IN URBAN ROAD NETWORKS

CURITIBA

2024

THALES ELERO CERVI

A COMPARATIVE ANALYSIS OF GRAPH NEURAL NETWORKS FOR ESTIMATING FREE-FLOW TRAVEL TIMES IN URBAN ROAD NETWORKS

Análise comparativa de redes neurais de grafos na estimação de tempo de percurso livre em redes de ruas urbanas

Dissertação apresentada como requisito para obtenção do título de Mestre em Ciências do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná.

Orientador: Profa. Dra. Myriam Regattieri De Biase da Silva Delgado

Coorientador: Prof. Dr. Ricardo Lüders

CURITIBA

2024

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

THALES ELERO CERVI

A COMPARATIVE ANALYSIS OF GRAPH NEURAL NETWORKS FOR ESTIMATING FREE-FLOW TRAVEL TIMES IN URBAN ROAD NETWORKS

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Computação.

Data de aprovação: 29 de Maio de 2024

Dra. Myriam Regattieri De Biase Da Silva Delgado, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Flavio Vinicius Diniz De Figueiredo, Doutorado - Universidade Federal de Minas Gerais (Ufmg)

Dr. Thiago Henrique Silva, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 29/05/2024.

Aos meus pais, que conduziram e incentivaram minha educação, ao longo de toda minha juventude, com mais exemplos do que palavras. Não fossem eles e seu incentivo, esse trabalho certamente não existiria.

À minha noiva, que compreendeu momentos de ausência e me deu suporte em momentos de incerteza. Seu apoio foi fundamental durante o tempo de execução desse trabalho e a sua companhia eu escolhi para toda uma vida, ao longo da qual pretendo retribuir a compreensão, a paciência, o apoio e o carinho.

Aos meus amigos e à minha família, pelos momentos de ausência durante o período que originou esse trabalho. Seu apoio também foi importante para esse trabalho acontecer.

ACKNOWLEDGEMENTS

Agradeço aos meus orientadores, Profa. Dra. Myriam Regattieri De Biase Da Silva Delgado e Prof. Dr. Ricardo Lüders, pela paciência e sabedoria com as quais me apoiaram durante toda esta trajetória. Seus conselhos foram fundamentais para o direcionamento (incluindo um re-direcionamento) da minha pesquisa. Pude aproveitar muito do convívio com ambos.

Agradeço aos professores do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI-CT), com os quais muito aprendi durante esse período. Em especial, ao Prof. Dr. Thiago Henrique Silva, com quem muito aprendi e tive o prazer de trabalhar junto em pesquisas relacionadas a esse trabalho. Tenho certeza que os conhecimentos adquiridos foram aplicados direta ou indiretamente neste trabalho e serão levados por mim daqui em diante.

Agradeço também à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), ao Projeto Smart City Concepts in Curitiba, IPPUC, e à Prefeitura de Curitiba. Parte dessa pesqueisa teve o apoio do projeto SocialNet (processo 2023/00148-0 da Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP) e do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (processos 313122/2023-7, 314603/2023-9 e 441444/2023-7).

Gostaria de deixar registrado também, o meu reconhecimento à minha família, pois acredito que sem o apoio deles seria muito difícil vencer esse desafio. Enfim, a todos os que por algum motivo contribuíram para a realização deste trabalho.

"The Machine is only a tool after all, which can help humanity progress faster by taking some of the burdens of calculations and interpretations off its back. The task of the human brain remains what it has always been, that of discovering new data to be analyzed, and of devising new concepts to be tested."
(ASIMOV, Isaac, 1950).

RESUMO

O planejamento do trânsito urbano em áreas densamente povoadas é um desafio que constantemente requer soluções eficazes. Redes viárias urbanas podem ser representadas por grafos, ajudando agentes decisores a visualizar e incorporar informações extraídas da estrutura da cidade. Redes Neurais de Grafos (GNNs do inglês *Graph Neural Networks*) são redes neurais artificiais projetadas para aprender com as informações e estrutura dos Grafos. No entanto, suposições implícitas importantes que baseiam as GNNs de uso geral não se aplicam aos grafos de redes viárias e o aprendizado com esse tipo de grafo pode exigir GNNs especializadas. Este estudo investiga o desempenho de GNNs especializadas e de uso geral na estimativa de tempos de viagem de fluxo livre em redes viárias urbanas de doze cidades em todo o mundo. Há ênfase nas Redes Relacionais de Fusão (RFNs do inglês *Relational Fusion Networks*), que são GNNs especializadas e com resultados competitivos na literatura. Os resultados de desempenho das GNNs selecionadas para a tarefa de regressão em arestas são obtidos para comparação. Eles confirmam o melhor desempenho das RFNs e destacam o papel das diferentes funções de fusão no desempenho da regressão. Além disso, este estudo explora a capacidade das RFNs de extrapolar o conhecimento aprendido para diferentes cidades. A investigação analisa ainda as características estruturais das cidades, visando compreender a influência destas nos mecanismos de aprendizagem da RFN e na sua capacidade de extrapolar conhecimento.

Palavras-chave: rede neural para grafo; rede viária urbana; regressão; predição de tempo de viagem para fluxo livre; cidade inteligente.

ABSTRACT

Urban transit planning in densely populated areas is an urban design challenge that constantly requires effective solutions. Urban road networks can be represented by graphs, helping decision-makers visualize and further incorporate information learned from the cities' structure. Graph Neural Networks (GNNs) are artificial neural networks designed to learn from graphs' information and structure. However, important implicit assumptions of general-purpose GNNs do not hold for urban road network graphs and learning with this type of graph may require specialized GNNs. This study investigates the performance of both general-purpose and specialized GNNs to estimate free-flow travel times in urban road networks of twelve cities worldwide. Emphasis is given to Relational Fusion Networks (RFNs), specialized GNNs with competitive results in the literature. Edge regression results of the selected GNNs are obtained for comparison. They show that RFNs perform better and highlight the role of different fusion functions on the regression performance. Additionally, this study explores RFNs' capacity to extrapolate learned knowledge to different cities. The investigation also searches through cities' structural characteristics, aiming to comprehend their influence on RFN's learning mechanisms and its capacity to extrapolate knowledge.

Keywords: graph neural network; urban road network; regression; free-flow travel time prediction; smart city.

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Abbreviations

Avg.	Average
Col.	Column
Concat.	Concatenation
Cont.	Continued
Max.	Maximum
Min.	Minimum
Pool.	Pooling

Pseudo-Acronyms

A	Attention Mechanism
AI	Artificial Intelligence
AMR	Arterial Main Roads
BGD	Batch Gradient Descent
BR	Brazil
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DK	Denmark
ELU	Exponential Linear Unit
FUSE	Fusion Function
GAIN	Graph Attetion Isomorphism Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GCRU	Graph Convolutional Recurrent Unit

GIN	Graph Isomorphism Network
GNN	Graph Neural Network
GPS	Global Positioning System
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GraphSAGE	Graph Sample and Aggregate
H	Attention Header
HPR	Highest Performance Roads
HWY	Highway
IID	Independent and Identical Distribution
JK	Jumping Knowledge
LOR	Link Only Roads
LSTM	Long Short Term Memory
LTR	Local Traffic Roads
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MB	Mini-Batch
MHCR	Major Highway Class Rate
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MLP	Multi-Layer Perceptron
MPR	Minor Public Roads
ODbL	Open Data Commons Open Database License
OSM	OpenStreetMap
OSMF	OpenStreetMap Foundation
OSMnx	Python library used to interact with OpenStreetMap

PC	Personal Computer
PCC	Pearson Correlation Coefficient
PyG	PyTorch Geometric Python library
RAM	Random-Access Memory
RF	Relational Fusion
RFN	Relation Fusion Network
RFN-AA	Relation Fusion Network with Additive fusion and Attentional aggregation
RFN-AI	Relation Fusion Network with Interactional fusion and Attentional aggregation
RFN-NAA	Relation Fusion Network with Additive fusion and Non-Attentional aggregation
RFN-NAI	Relation Fusion Network with Interactional fusion and Non-Attentional aggregation
RMSE	Root Mean Squared Error
RNG	Road Netwok Graph
RNN	Recurrent Neural Networks
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
STGCN	Spatio-Temporal Graph Convolutional Networks
T-GCN	Temporal Graph Convolutional Network
USA	United States of America
UTFPR	Universidade Tecnológica Federal do Paraná
WL	Weisfeiler-Lehman

CONTENTS

1 INTRODUCTION

The social organization of the world population has changed throughout history, especially regarding its relationship with inhabited spaces. Industrialization made urban centers more attractive to big concentrations of economically active people. Several social and environmental issues, resulting from geographic migrations that often do not occur in a sustainable way, add up to challenges of allocation and movement of people in urban areas with constant increase of population density. This scenario demands innovative solutions, especially for public transport and logistics inside cities (ALBINO; BERARDI; DANGELICO, 2015)

Over the last decades, technological advancements have brought new tools to enhance or propose sustainable solutions for urban issues. Technology can instrument and connect urban infrastructure, leveraging collective development and originating *Smart Cities* (HARRISON *et al.*, 2010) (ALBINO; BERARDI; DANGELICO, 2015). In this context, collective knowledge and creativity improve urban design and planning for the community benefit (EGER, 2009).

Urban design is the technical process of shaping the physical features of cities to provide services to people. It deals with a city considering the big picture of buildings, infrastructure, public spaces and streets, aiming to design equitable and sustainable urban environments (KARIMI, 2012). Commonly, street systems are represented by networks when investigating urban design and planning problems (BOEING, 2017). In fact, urban street analysis has been central to network science since the 18^{th} century, when Leonhard Euler published *"Seven Bridges of Königsberg"*, launching the foundations of graph theory and prefiguring the idea of topology (SHIELDS, 2012). Classic literature on urban design has later focused on urban form (STRANO *et al.*, 2013), transportation (PARTHASARATHI; LEVINSON; HOCHMAIR, 2013), and the topology, complexity, and resilience of urban networks (PORTA; CRUCITTI; LATORA, 2006).

Computation techniques, like machine learning, can help with urban design challenges. The success of deep learning techniques has already revolutionized several research fields and is recently being extended to cover problems posed on irregular domains, in which data is usually represented through graphs or networks. Graph representation learning models explicitly leverage latent representations of connections between nodes and are highly based on the concepts of *homophily*[1] and *structural equivalence*[2]. Important works on graph representation learning address node classification tasks in social (YING *et al.*, 2018; SILVA; SILVER, 2024), citation (KIPF; WELLING, 2016), and chemical or biological networks (GILMER *et al.*, 2017) (ZITNIK; AGRAWAL; LESKOVEC, 2018).

Learning with urban street networks though, differs substantially from such tasks and might require specialized architectures and techniques (JEPSEN *et al.*, 2018). Therefore, in addition to general purpose Graph Neural Networks (GNNs), the present work explores Relational Fusion Networks (RFNs). Designed for machine learning on road networks, RFN's basic

[1] When nodes tend to share attributes with their neighbors (ZHOU *et al.*, 2004)

[2] When nodes with similar neighborhood structure will have similar attributes (DONNAT *et al.*, 2017)

premise is to learn representations based on two distinct, but interdependent views: the node-relational and edge-relational views (JEPSEN; JENSEN; NIELSEN, 2020). Whatever the addressed model, generic or specialized, the graph representation of a city can be consumed by deep learning algorithms to evaluate existing transportation systems and network flow characteristics.

1.1 Problem description

The majority of the literature on road network graph learning is interested in spatiotemporal models applied to time series prediction in urban environments. Despite the possible contributions to urban design challenges, not many works have explored road network static characteristics' representation and regression. This work does not explore any temporal traffic data. It aims to solve a regression task for a specific edge characteristic: the free-flow travel time for a driving vehicle, a static measure dependent on structural information like street type, length and maximum legal driving speed.

The present work addresses the task of estimating the free-flow travel time of street segments based on their neighborhoods in the graph. Solving this task can enhance the evaluation of new infrastructure proposals and alternatives to current urban street layouts, enriching urban design decisions with structural and topological data, collectively collected from cities' street networks. A city street network can also be called an urban street network graph (BOEING, 2017) or, simply, a Road Network Graph[3] (JEPSEN *et al.*, 2018).

The central theme of this work is, therefore, the exploration of graph representation learning techniques for the edge (i.e., street segments) regression problem on Road Network Graphs, a task significantly related to spatial and structural information extracted from cities. General purpose and road network-oriented graph learning techniques will be presented and evaluated on their respective regression performance.

1.2 Objective and research questions

The main objective of the present work is to apply graph neural networks (GNNs), relational fusion networks (RFNs) in particular, to the edge free-flow travel time regression on Road Network Graphs, comparing models, exploring training techniques and investigating their representation and extrapolation capacities.

This work adds perspective on using GNNs to edge regression problems on Road Network Graphs by:

[3] The term *road network graph* will be used in this work, even though it is perfectly interchangeable with urban network graph and similar variants.

- exploring different graph representation learning models applied to a Road Network Graph (Aalborg, DK), by testing different loss functions when training for the free-flow travel time regression task;
- exploring mini-batching training with different batch sizes and compiling information on each model: training computation time and final test performance;
- comparing general-purpose GNNs with models specifically designed for road network tasks (i.e, RFNs) to understand which one has the best expressiveness power;
- testing each model on different urban drivable networks from twelve cities worldwide, whose Road Network Graphs have been made publicly available;
- understanding the extrapolation capacity of models trained with a given city road network by testing it on different city networks;
- investigating which global features of Road Network Graphs can enhance or diminish the model extrapolation power.

The following research questions should be addressed:

1. Are GNNs a good fit for Road Network Graph edge regression tasks?
2. How does mini-batch in gradient descent algorithm influence GNNs' performance?
3. How well RFNs extrapolate to new Road Network Graphs?

1.3 Contribution

The main outcomes of this work are i) the extension of the edge regression task analysis to eleven new cities worldwide; ii) the exploration of mini-batching effects on GNNs training for this specific task and, especially, iii) the RFN's extrapolation capacity analysis. Additionally, a dataset with twelve different cities worldwide has been made publicly available[4]. It provides support to the free-flow travel time regression task benchmark on Road Network Graphs. Moreover, the dataset can be easily expanded by including other city data.

The following conference paper has been accepted for publication in a national workshop (CoUrb 2024) with comparisons of RFN variants for the regression task, including an analysis of their representation and extrapolation capacities (CERVI *et al.*, 2024):

CERVI, T. E. et al. Regression performance of relational fusion networks on urban road networks. In: Anais do VIII Workshop de Computação Urbana. Porto Alegre, RS, Brasil: SBC, 2024.

[4] https://github.com/tcervi/cpgei-rng-dataset

1.4 Organization

Chapter 2 presents a literature review on machine learning techniques for solving the problem of representation learning on graphs. Chapter 3 introduces five models selected to be tested and explored along this work, detailing previous work on machine learning for road networks, including original results and the gaps, part of them addressed in the present work. Chapter 4 presents the proposed methodology, encompassing the data representation model, the assembled Road Network Graph dataset, the select machine learning tools, including software and hardware, and the schedule of planned experiments. Chapter 5 presents the experimental results with conclusions and future work in Chapter 6.

2 MACHINE LEARNING ON GRAPHS

Historical advances in computational processing power, both in hardware and software, enabled the development of algorithmic implementation of statistical models that established the foundations for machine learning algorithms. Learning, in this context, is the statistical fine-tuning of parameters intrinsic to such models, i.e., weights and biases (SEUNG; SOMPOLINSKY; TISHBY, 1992; ANTHONY; BARTLETT, 1999; VAPNIK, 2000).

Modern machine learning operates with large datasets to design rich function spaces with the capacity to interpolate over data points. Such datasets usually store strictly defined tensor-like[1] data and models are adapted to exploit the low dimensional geometry arising from physical phenomena, like grids in images and sequences in time series. Data points are always assumed to be statistically independent from each other, otherwise, the dependencies would have to be modeled, and identically distributed, otherwise, there is no guarantee that the model generalizes to new and unseen data. This is known as the assumption of data independent and identical distribution, *IID* (JACOBS, 1992).

The deep Convolutional Neural Network (CNN), for example, is a multi-layer gradient-based learning technique that is extensively applied to image processing and uses the kernel matrices method to process an entire image, scanning the data representation, in a grid-like structure, for target patterns (LECUN *et al.*, 1998; HUANG; LIU; WEINBERGER, 2016). It works mainly because CNNs are models that respect *translational invariance*[2], while strongly relying on the *locality assumption*[3].

Data, however, can be found in dispositions other than Euclidian tensor-like structures. When handling complex systems, interesting tasks present data in an irregular domain that can not be represented by a grid-like structure (VELICKOVIC *et al.*, 2017). There are numerous problems in which the data can happen to be a collection of objects that come with a set of pairwise interactions. Such data is then represented by networks, or *graphs*, which are a ubiquitous data structure and a universal language to describe complex systems. The power of their formalism lies both in its generality and its focus on relationships between points, rather than the properties of individual points (HAMILTON, 2020).

Formally, a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ representing relations between nodes. An edge $(u,v) \in \mathcal{E}$, goes from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$. For *directed graphs*, the origin and destination nodes describe the edge direction and the reciprocity is not guaranteed, while for *undirected graphs* $(u,v) \in \mathcal{E} \leftrightarrow (v,u) \in \mathcal{E}$. Locality on graphs is described by *neighborhoods*. The neighborhood of a node u in a graph is the set of nodes that are connected to it by edges: $\mathcal{N}(u) = \{v : (u,v) \in \mathcal{E} \vee (v,u) \in \mathcal{E}\}$.

[1] That is, the input data of a model has data points that share the same dimension

[2] When patterns are points of interest irrespective of where they are in the image

[3] The assumption is that neighboring data points (i.e., pixels) relate stronger than distant ones.

A convenient way to represent a graph is through an *adjacency matrix* $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, in which every node indexes a particular row and column. The presence of edges is then represented by entries where $\boldsymbol{A}[u,v] = 1$ if $(u,v) \in \mathcal{E}$ and $\boldsymbol{A}[u,v] = 0$ otherwise. Commonly, graph-structured data present a huge number of nodes, sometimes on the order of millions, and the number of connections per node can be highly variable (PEROZZI; AL-RFOU; SKIENA, 2014). This leads to large sparse adjacency matrices whose computation is both memory and processing inefficient. A memory-efficient way to represent a graph is through an *adjacency list*, an array-like structure in which the connectivity of the e_k edge between nodes n_i and n_j is described by a tuple (i,j) on the k-th entry of the list (SANCHEZ-LENGELING *et al.*, 2021).

Machine learning on graphs though, requires models that can learn from structured data where the assumption of independent and identical distribution, *IID*, does not hold. Rather than learning from a set of *IID* data points, it learns from an interconnected set of data points. Since such representation supports an arbitrary pairwise relational structure, learning on graphs needs a strong relational inductive bias beyond what CNNs can provide (BATTAGLIA *et al.*, 2018).

Furthermore, graph learning tasks often blur the boundaries between the traditional machine learning categories: supervised and unsupervised learning. Different from usual supervised training settings, where unlabeled data is completely removed from training, the atypical nature of the node classification task, for example, requires access to the full graph during the training phase, including unlabeled elements that would be traditionally called test data. Training only misses the labels of a given set of elements but uses any other information about it, its neighborhood, and the graph structure as a whole. Therefore, learning on graphs can be considered a *semi-supervised* task (YANG; COHEN; SALAKHUTDINOV, 2016) (HAMILTON, 2020).

2.1 Graph embeddings

Initial approaches to graph representation learning were based on methods for learning node embeddings, which are low-dimensional encoding vectors that summarize the node's graph position and the structure of its local neighborhood. These methods project nodes into a latent space where geometric relations correspond to relationships in the original graph. Embeddings are optimized so that distances in the latent space reflect the relative position of nodes in the original graph (HOFF; RAFTERY; HANDCOCK, 2002).

Such embedding methods are categorized as *shallow embedding* methods. In essence, embedding lookup methods are trained to generate a unique representation for each node. They are quite limited in what they can provide for graph representation learning tasks. First, there is no parameter sharing between nodes in the encoder, which makes the methods statistically and computationally inefficient. Additionally, it does not leverage node, edge, or graph global features during the encoding process and the majority of real graph-structured data have this type of information available to be used. Finally, they are inherently transductive i.e., they can

only generate embeddings for nodes that are available in the training phase and do not naturally generalize to unseen nodes.

Embedding methods have been applied to Road Network Graphs context (JEPSEN *et al.*, 2018) (WANG *et al.*, 2019) and results presented in Jepsen *et al.* (2018) have proved it to differ significantly from the common graph embedding contexts (e.g., social networks), in terms of node degree and level of homophily. The work presented by Jepsen *et al.* (2018) with Road Network Graph embedding is, to the best of current knowledge, the first one to start exploring the specificity of graph learning tasks on road network data.

2.2 Graph neural networks (GNNs)

While shallow embedding methods simply optimize a unique latent structural representation for each node, the Graph Neural Network (GNN) is a learning model that generates nodes' representations with a strong dependency on both the structure of the graph and any additional information that its components might have. It learns over a graph and generates an updated version of the same graph structure, which can be useful, for example, on prediction tasks where a prediction model could be trained based on the GNN output (HAMILTON, 2020).

To develop efficient GNN models though, some particularities of graphs need to be addressed. First of all, the nodes of a graph must not be assumed to be in any specific order. In fact, a graph can exist in different forms that share the same number of nodes, edges, and also the same edge connectivity. Such graphs are called *isomorphic* (WEISFEILER; LEHMAN, 1968; ZEMLYACHENKO; KORNEENKO; TYSHKEVICH, 1985) and techniques must be in place to enforce that learning over isomorphic graphs leads to the same output results.

Secondly, GNNs can leverage feature information that a graph dataset might have at the node, edge, and global levels, in contrast to shallow embedding methods. Taking node features as an example, for a given node u there is a neighborhood feature multi-set defined as $\boldsymbol{M}(\mathcal{N}(u)) = \{\{\mathbf{x}_v : v \in \mathcal{N}(u)\}\}$, where $\mathbf{x}_v$ is the node feature vector.

Therefore, the building blocks for general graph learning models should be specialized in leveraging graph characteristics, presenting a strong relational inductive bias (BATTAGLIA *et al.*, 2018). That is, such models should preserve relationships between entities (i.e., adjacency matrix) and graph symmetries to ensure better predictive performance (e.g., for isomorphic graphs). The key to developing efficient GNN models is to understand what properties are useful for operating meaningfully on graphs and what symmetries and invariances must be preserved.

Set learning tasks theory is the foundation for GNN models (QI *et al.*, 2016; ZAHEER *et al.*, 2017; XU *et al.*, 2018b). Initially, a set can be seen as a graph without edges (i.e., $\mathcal{E} = \emptyset$). Then, assuming the graph nodes have, each one, a feature vector $\mathbf{x}_i$ of size k, all nodes feature

vectors, $\mathbf{x}_i \in \mathbb{R}^k$, can be stacked into a matrix $\boldsymbol{X}_{|\mathcal{V}|\times k}$ and such matrix is a set of vectors. The simple action of stacking vectors implies a given order, and a matrix $\boldsymbol{X}_{n\times n}$ has $n!$ permutations[4].

When applying a Multi-Layer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1986) on prediction tasks with a particular input $\boldsymbol{X}_{|\mathcal{V}|\times k} = \{\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_{|\mathcal{V}|}\}$, predictions would not necessarily transfer to the same input reordered, for example, as $\boldsymbol{X}_{|\mathcal{V}|\times k} = \{\mathbf{x}_2,\mathbf{x}_{|\mathcal{V}|},...,\mathbf{x}_1\}$ and $n!$ permutations would require an exponential number of input and output examples for an MLP to be able to learn an approximation function (BATTAGLIA *et al.*, 2018). The takeaway then, is that set learning tasks require an order-independent neural network (VINYALS; BENGIO; KUDLUR, 2015).

Extrapolating the set learning theory to graphs, when using Adjacency Matrices $\boldsymbol{A}$, it is mandatory that all permutations of $\boldsymbol{A}$ lead to the same representation output. That is, predictions for graph contexts depend on symmetric functions. Then, ***Permutation Invariance*** is a required property. It is said that a function $f(\boldsymbol{X})$ is permutation invariant if, for all permutation matrices, $f(\boldsymbol{PX}) = f(\boldsymbol{X})$ holds. Such functions are useful for obtaining set-level outputs, but it is still desirable for the final method to identify element-level output. So, if elements are permuted it should not matter if it is done before or after the function is applied. This property is called ***Permutation Equivariance*** and a function $f(\boldsymbol{X})$ is permutation equivariant if, for all permutation matrices, $f(\boldsymbol{PX}) = \boldsymbol{P}f(\boldsymbol{X})$ holds.

Graph Neural Network models are then, optimizable transformations on all graph attributes that rely strongly on permutation properties to preserve graph symmetries. A generic recipe for such model would be to construct *permutation equivariant* functions $f(\boldsymbol{X},\boldsymbol{A})$, by appropriately applying local *permutation invariant* functions $g(\mathbf{x}_u, \boldsymbol{M}(\mathcal{N}(u)))$ over all nodes neighborhoods (MENA *et al.*, 2018; MURPHY *et al.*, 2018).

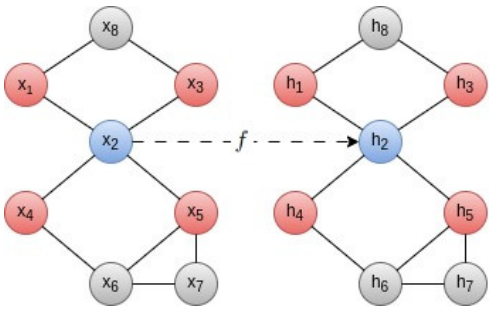
In other words, a GNN model can be understood as a local shared learnable function g that is applied to every neighborhood in isolation, mapping its multi-set set of features $\boldsymbol{M}(\mathcal{N}(u))$ to a latent space over which, later, will be applied a learnable function f that produces an updated version of the input graph, transforming its feature representations without changing its connectivity, as shown on Figure 1. The *permutation equivariant* function f and the *permutation invariant* function g are commonly refereed, respectively, as **update** and **aggregation** functions on the GNN base literature (HAMILTON, 2020).

The basic Graph Neural Network (Equation 1) is the simplification of original models proposed by Merkwirth and Lengauer (2005), Gori, Monfardini and Scarselli (2005) and Scarselli *et al.* (2009):

$$\mathbf{h}_u^{(k)} = \sigma(\boldsymbol{W}_{self}^{(k)}\mathbf{h}_u^{(k-1)} + \boldsymbol{W}_{neighbors}^{(k)} \sum_{v\in\mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \boldsymbol{b}^{(k)}) \tag{1}$$

[4] There are $n!$ permutation matrices $\boldsymbol{P}_{n\times n}$, in which every row and every column contains a single 1 and all other elements are 0, that when left-multiplied by $\boldsymbol{X}_{n\times n}$ it permutes its rows (MEYER, 2000)

Figure 1 – GNN Layer: Graph Representation Learning



$$f(\boldsymbol{X},\boldsymbol{A}) = \begin{bmatrix} [g(\mathbf{x}_1, \boldsymbol{M}(\mathcal{N}(\mathbf{x}_1))] \\ [g(\mathbf{x}_2, \boldsymbol{M}(\mathcal{N}(\mathbf{x}_2))] \\ \vdots \\ [g(\mathbf{x}_n, \boldsymbol{M}(\mathcal{N}(\mathbf{x}_n))] \end{bmatrix}$$

Source: Own work (2025).

where $\boldsymbol{W}_{self}^{(k)}, \boldsymbol{W}_{neighbors}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$ are trainable parameter matrices to control which, and how much, information should prevail between the original and updated representations; $\boldsymbol{b}^{(k)}$ is the bias term that usually is omitted for simplicity; and σ is an element-wise non-linearity.

Neighborhood information is a defining feature for most GNN models, and its incorporation depends on a neural ***message passing mechanism***. That is, vector messages are exchanged between nodes and updated using neural networks (BATTAGLIA *et al.*, 2016; GILMER *et al.*, 2017; BATTAGLIA *et al.*, 2018). A message-passing mechanism makes the learned representations aware of the graph connectivity since neighboring nodes exchange information and influence each other's updated versions. For each k-th iteration then, the hidden representation $\mathbf{h}_u^{(k)}$, for each node $u \in \mathcal{V}$, is updated according to information *aggregated* from $\mathcal{N}(u)$:

$$\mathbf{h}_u^{(k)} = UPDATE^{(k-1)}(\mathbf{h}_u^{(k-1)}, AGGREGATE^{(k-1)}(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\})) \tag{2}$$

where both the *UPDATE* and *AGGREGATE* are arbitrary differentiable functions as long as they are respectively permutation equivariant and permutation invariant, as previously described.

The *message vector* containing information aggregated from u's neighborhood by the k-th iteration can be represented by $\boldsymbol{m}_{\mathcal{N}(u)}^{(k)}$, so Equation 2 can be written as Equation 3.

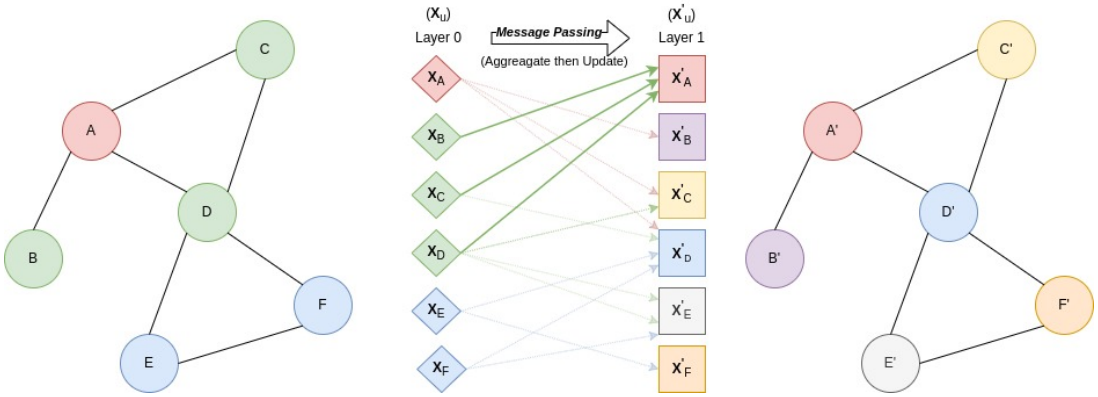
$$\mathbf{h}_u^{(k)} = UPDATE^{(k-1)}(\mathbf{h}_u^{(k-1)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k-1)}) = \sigma(\boldsymbol{W}_{self}^{(k)} \mathbf{h}_u^{(k-1)} + \boldsymbol{W}_{neighbors}^{(k)} \boldsymbol{m}_{\mathcal{N}(u)}^{(k-1)}) \tag{3}$$

Each message passing iteration is also known as a *GNN Layer* and a visual representation of one iteration can be found in Figure 2.

For simplicity, it is a common practice to add self-loops to the input graph and omit the explicit update step (HAMILTON, 2020):

$$\mathbf{h}_u^{(k)} = AGGREGATE^{(k-1)}(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u) \cup \{u\}\}) \tag{4}$$

Figure 2 – Message Passing: Updating Node A representation



Source: Own work (2025).

Self-loops are edges added to each node, connecting the node to itself. From the adjacency matrix perspective, adjacency with self-loops $\boldsymbol{A}_{self}$ is the matrix resulting from adding the adjacency matrix to the identity matrix (i.e., $\boldsymbol{A}_{self} = \boldsymbol{A} + \boldsymbol{I}$).

Usually, the initial representations, $k = 0$, are the input feature vectors of each component. For nodes, $\mathbf{h}_u^{(0)} = \mathbf{x}_u, \forall u \in \mathcal{V}$. After running k iterations of the GNN message passing the final representations would be $\boldsymbol{z}_u = \mathbf{h}_u^{(k)}, \forall u \in \mathcal{V}$ (Figure 3). At this point, the original graph structure remains exactly the same, but now it has new node representations containing structural and feature-based information about each node k-hop neighborhood. It is important to note that the initial representation (i.e., node features) $\mathbf{x}_u \in \mathbb{R}^{d^{(0)}}, \forall u \in \mathcal{V}$ are vectors of a given size $d^{(0)}$, but the updated representations at each layer have their size $d^{(k)}$ that does not have to remain the same along layers. Final representations can then be used, for example, on node-level prediction tasks in which it either already represents some information of interest or feeds a prediction model.

A graph though, can also have input feature vectors for each edge and one or more features defining global graph information. Battaglia *et al.* (2018) generalize message passing, allowing to easily integrate edge (i.e., $\mathbf{h}_{u,v}$) and global features (i.e., $\mathbf{h}_{\mathcal{G}}$):

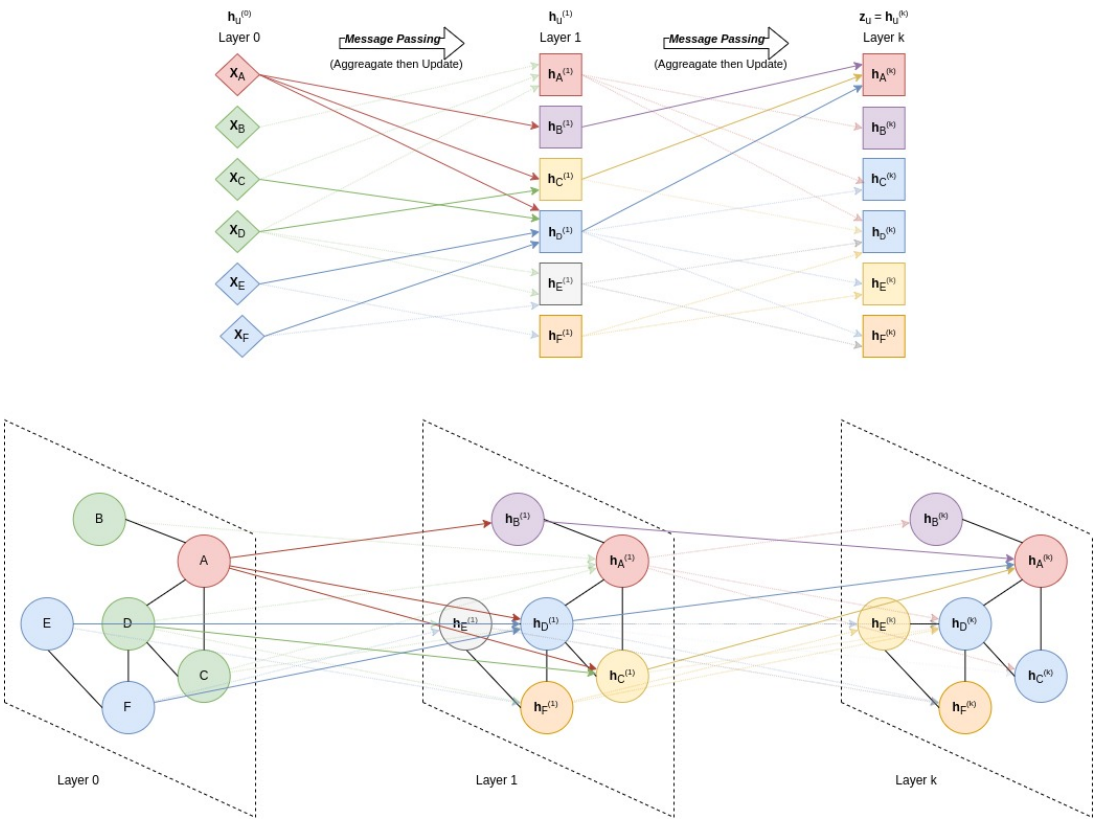
$$\mathbf{h}_{u,v}^{(k)} = UPDATE_{edge}(\mathbf{h}_{u,v}^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, \mathbf{h}_{\mathcal{G}}^{(k-1)}) \tag{5}$$

$$\boldsymbol{m}_{\mathcal{N}(u)}^{(k)} = AGGREGATE_{node}(\mathbf{h}_{u,v}^{(k)}, \forall v \in \mathcal{N}(u)) \tag{6}$$

$$\mathbf{h}_u^{(k)} = UPDATE_{node}(\mathbf{h}_u^{(k-1)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}, \mathbf{h}_{\mathcal{G}}^{(k-1)}) \tag{7}$$

$$\mathbf{h}_{\mathcal{G}}^{(k)} = UPDATE_{graph}(\mathbf{h}_{\mathcal{G}}^{(k-1)}, \mathbf{h}_u^{(k)}, \forall u \in \mathcal{V}, \mathbf{h}_{u,v}^{(k)}, \forall (u,v) \in \mathcal{E}) \tag{8}$$

Figure 3 – GNN Layers: k message passing iterations



Source: Own work (2025).

For some GNN applications, the general model presents benefits in terms of logical expressiveness, when compared to using only node features (BARCELÓ *et al.*, 2020).

In essence, the message passing feature aggregation - GNN - is analogous to convolutional kernels - CNN - since both process information of neighborhood to update elements. CNNs though, aggregate feature information from spatially-defined patches of fixed size, while on graphs, the size of nodes neighborhood varies within the same graph (SANCHEZ-LENGELING *et al.*, 2021).

In summary, Graph Neural Networks need to learn, for every component (e.g., nodes), local representations by aggregating neighborhood information in a way that it is neither affected by the neighborhood order (*permutation invariant*), nor by the order that the components are presented (*permutation equivariant*) (SANCHEZ-LENGELING *et al.*, 2021). Specialized GNN models are distinguished by changing the aggregation function, the update function, or both.

2.2.1 Aggregation methods

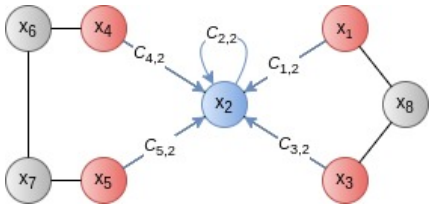
The simplest *permutation invariant* functions that can be used to aggregate neighborhood information are the sum (*SUM*), the average (*AVG*) and the max (*MAX*) or min (*MIN*) functions. The *MAX* or *MIN* functions can highlight locally salient features. The *AVG* function

normalizes features locally, something desirable especially when neighborhood sizes vary too much.

The *SUM* function is a local snapshot that highlight outliers, since it is not normalized. It is also an unstable approach, highly sensitive to node degrees, that can lead to numerical instabilities and difficulties in optimization. So, neighborhood normalization methods are usually applied to normalize the feature aggregation on node degrees (HAMILTON; YING; LESKOVEC, 2017a).

GNN models that employ symmetric-normalized aggregation and the self-loop approach are called *Graph Convolutional Networks* - GCNs - (Figure 4). It aggregates summing neighborhood features with fixed weights ($C_{u,v}$) that usually depend on the adjacency matrix $\boldsymbol{A}$ and are useful for scaling learning capacity and for learning from *Homophilous Graphs*, that is, graphs where edges encode label similarity (i.e., closer nodes are similar) (BRUNA *et al.*, 2013; DEFFERRARD; BRESSON; VANDERGHEYNST, 2016; KIPF; WELLING, 2016; WU *et al.*, 2019).

Figure 4 – Graph Convolutional Networks (GCNs) Aggregation



Source: Own work (2025).

The decision to normalize or not is application-dependent though. It helps to achieve stable and strong performance but can lead to loss of information. Various structural graph features can be obscured and it can be hard, if not impossible, to distinguish nodes with different degrees. Applications that can benefit from normalization are those where node feature information is far more useful than structural information and there is a wide range of node degrees that can lead to optimization instabilities (HAMILTON, 2020).

It is also possible to use *SUM*, *AVG* or *MAX* to reduce a set of feature vectors (i.e., neighborhood) to a single vector and then combine this reduction with a feed-forward neural network to enrich it. In fact, any permutation-invariant function that applies this set reduction approach can be approximated with arbitrary accuracy by the following model (ZAHEER *et al.*, 2017):

$$\boldsymbol{m}_{\mathcal{N}(u)} = MLP_{\theta}(\sum_{v \in \mathcal{N}(u)} MLP_{\phi}(\mathbf{h}_v)) \tag{9}$$

and the *SUM* can be replaced by any alternative reduction function (QI *et al.*, 2016). Combining such set pooling with normalization approaches was investigated by Hamilton, Ying and Leskovec (2017a) and although it can provide small increases in performance, it also increases the risk of over-fitting depending on the depth of the used MLPs. Usually, only one hidden layer is used, so it satisfies the theory without over-parameterizing the model (HAMILTON, 2020).

Finally, it is possible to add any attention mechanism (BAHDANAU; CHO; BENGIO, 2014) to the aggregation method, increasing the representational capacity of the model when prior knowledge is available. So, the model learns how to aggregate features in opposition to simply relying on the structural information. The first GNN model to apply it was the ***Graph Attention Network*** (GAN) (VELICKOVIC *et al.*, 2017) that, instead of using node degrees to normalize aggregation, used a function that calculates sum weights using the features of both the source and target node. It is also possible to add multiple attention heads to a model, computing h distinct attention weights that can represent, each one, different semantic meanings.

2.2.2 Update methods

Update methods play an equally important role in defining the power of a GNN model. It helps avoid *over-smoothing*, something that can happen after several message passing iterations when the final representation for all nodes becomes very similar to each other. It is the main problem to overcome in order to build deeper GNN models (HAMILTON, 2020). When learned embeddings become over-smoothed, approaching an almost uniform distribution, local neighborhood information is lost.

The simplest update methods that try to directly preserve information from previous rounds of message passing are the ones that use a ***Concatenation*** technique. Concatenation can be used in conjunction with the majority of other update approaches. Hamilton, Ying and Leskovec (2017b) present a simple concatenation update (Equation 10) and Pham *et al.* (2016) propose a linear interpolation method (Equation 11) in which $\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2 \in [0,1]^d$ are learnable gating vectors:

$$UPDATE_{concat}(\mathbf{h}_u, \boldsymbol{m}_{\mathcal{N}(u)}) = [UPDATE_{base}(\mathbf{h}_u, \boldsymbol{m}_{\mathcal{N}(u)}) \oplus \mathbf{h}_u] \tag{10}$$

$$UPDATE_{concat}(\mathbf{h}_u, \boldsymbol{m}_{\mathcal{N}(u)}) = \boldsymbol{\alpha}_1 \odot UPDATE_{base}(\mathbf{h}_u, \boldsymbol{m}_{\mathcal{N}(u)}) + \boldsymbol{\alpha}_2 \odot \mathbf{h}_u \tag{11}$$

where $\oplus$ denotes matrix concatenation and $\odot$ denotes elementwise multiplication. In practice, Concatenation Update techniques are more useful for node classification tasks with moderately deep GNNs (i.e., 2 to 5 GNN layers) and excel on tasks where predictions of each node are strongly related to the features of its local neighborhood (i.e., homophily).

Picturing that the aggregation function is receiving an *observation* from the neighborhood to *update* the *hidden state* of each node is another way to understand the message passing algorithm. That way, ***Gated Updates*** are possible whenever a method to update hidden states of Recurrent Neural Networks (*RNN*) is applied, as peer Li *et al.* (2015):

$$\mathbf{h}_u^{(k)} = GRU(\mathbf{h}_u^{(k-1)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}) \tag{12}$$

in which GRU is the update equation of the gated recurrent unit (CHO *et al.*, 2014) or can be replaced by updates based on the Long Short-Term Memory (LSTM) architecture (SELSAM *et al.*, 2018). Generally, any update function defined for RNNs can be used in the context of GNNs if the RNN hidden state argument is considered to be the GNN node representation and the RNN observation vector is considered to be the GNN neighborhood aggregated information.

Gated Update is an efficient technique since parameters are always shared between nodes and, in practice, are also shared between GNN layers. That is, the same GRU or LSTM cell is used to update each node. It is effective for preventing over-smoothing on deep GNNs (i.e., more than 10 layers) and is generally useful for applications where the prediction task requires complex reasoning over the global structure of the graph (HAMILTON, 2020).

Finally, it is important to mention that the implicit assumption of always using only the output of the final GNN layer is a limited strategy that motivated much of the need for residual (concatenation) and gated updates to limit over-smoothing. Another update technique is known as ***Jumping Knowledge Connections*** (XU *et al.*, 2018b) and leverages the representations at each layer of the message passing, rather than only using the final layer output (i.e., $\boldsymbol{z}_u = \mathbf{h}_u^{(k)} \forall u \in \mathcal{V}$):

$$\boldsymbol{z}_u = f_{JK}(\mathbf{h}_u^{(0)} + \mathbf{h}_u^{(1)} + ... + \mathbf{h}_u^{(k)}), \forall u \in \mathcal{V} \tag{13}$$

where f_{JK} is an arbitrary differentiable function.

2.3 Urban spatial representation learning

Any city street map can be modeled as a graph, where nodes represent intersections or dead-ends, while edges represent street segments that link those nodes (BOEING, 2017). The network representation of urban streets usually includes not only structural information but also any feature information that its components might have. Such networks are typically called Road Network Graphs[5] (ROSSI; AHMED, 2015).

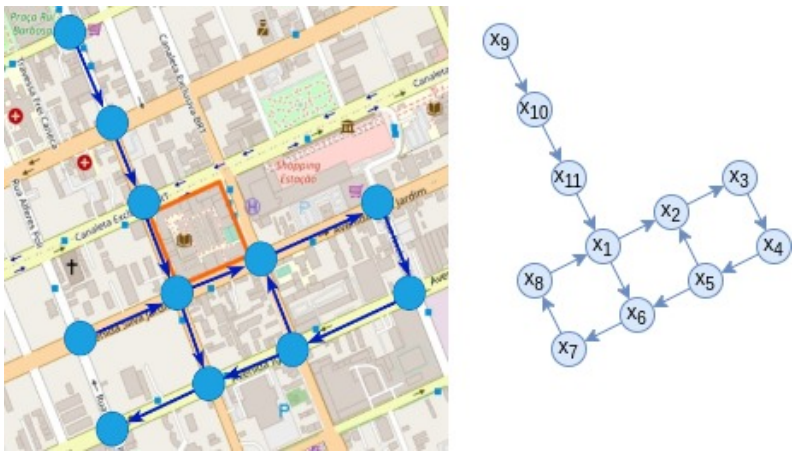
Therefore, Road Network Graphs are mathematical representations of the actual networks that structure human interactions and transportation processes inside cities. Figure 5 uses the surroundings of the Brazilian Federal University of Technology[6] to illustrate how the road map depicted in Figure 5(a) can be modeled as the directed graph depicted in Figure 5(b).

The development of graph neural networks was mainly driven by tasks in which the input data was structured in graphs with specific set of characteristics, including a high-dimensional and informative set of node features (e.g., social networks). Those GNNs were developed with the goal of representing every node of a graph as a function of its neighborhood features and structural information. The use of neighborhood node features to generate information about a

[5] In this work, the term 'graph' is preferred over 'network,' especially to avoid any confusion with the machine learning technique of neural networks, which are described later.

[6] The Brazilian Federal University of Technology - UTFPR - is located in Curitiba, Paraná

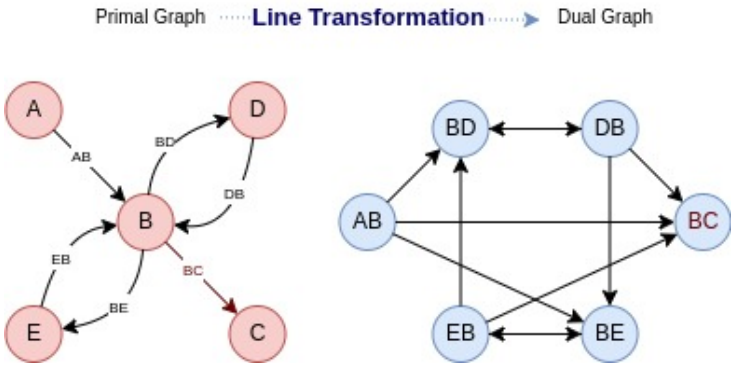
Figure 5 – Street map and its equivalent Road Network Graph



Source: Own work (2025).

node's position and role in the graph is the key to generalizing learned knowledge to unseen nodes and graphs (i.e., inductive tasks). However, the common GNN learning tasks are highly node-related, usually focusing on node classification (KIPF; WELLING, 2016; HAMILTON; YING; LESKOVEC, 2017a). Link prediction (BORDES *et al.*, 2013; ZITNIK; AGRAWAL; LESKOVEC, 2018; YING *et al.*, 2018) and graph classification (GILMER *et al.*, 2017; LI *et al.*, 2019) are common tasks too, also highly based on knowledge learned from node features.

Figure 6 – Graph Line Transformation



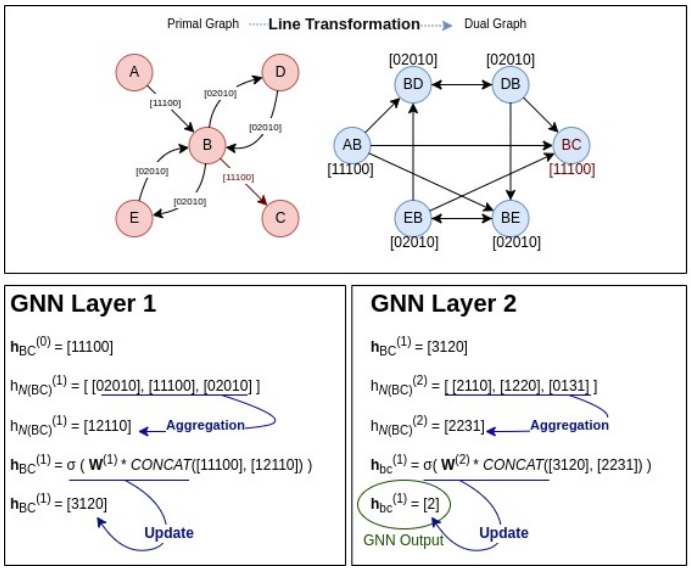
Source: Own work (2025).

When working with road networks though, the edge features are the crucial information to generate good descriptive graph representations, since they represent street segments, which are more interesting than simple street segment junctions (i.e., nodes) (JEPSEN *et al.*, 2018). Therefore, an option for learning with Road Network Graphs is to apply a Line Graph Transformation (HARARY; NORMAN, 1960; BEINEKE; BAGGA, 2021) to the input data (Figure 6). The graph representation used so far, $\mathcal{G}^{\mathcal{P}} = (\mathcal{V},\mathcal{E})$, is the Primal Graph representation and the output of a Line Graph Transformation is the Dual Graph representation, $\mathcal{G}^{\mathcal{D}} = (\mathcal{E}',\mathcal{B})$, in which every

primal graph edge $(u,v) \in \mathcal{E}$ is transformed into a dual graph node $e \in \mathcal{E}'$. The between-edge connections in the primal graph (i.e., nodes) are transformed into dual graph edges $b \in \mathcal{B}$.

A road network dual representation can then be the input data for a graph neural network approaching tasks related to urban street representations. In this case, the set of the primal graph edge feature vectors would be the same as the set of the dual graph node feature vectors and it would be the input data for the model's layers. A basic picture of the RNG dual graph for learning edge information is shown in Figure 7:

Figure 7 – Training GNNs with Road Network Graphs dual representation: a primal edge between nodes B and C is transformed into a dual node BC



Source: Own work (2025).

In this example, the primal graph edge feature vectors have size five, $\mathbf{x}_u \in \mathbb{R}^5, \forall u \in \mathcal{V}$ and the GNN has two layers. A primal edge between nodes B and C (the red edge in the graph on the left side) is transformed into a dual node BC (the red node in the graph on the right side) and the feature size is kept, that is: a dual node has the same feature vector size of a primal edge. The Aggregation function is the *MAX* function and the Update is done by concatenating the aggregated neighborhood information with the node's information, then a linear transformation is applied, followed by a non-linearity.

When passing the dual graph through the first GNN layer, the result is the same dual graph structure but the node representations have size four, $\mathbf{h}_u^{(1)} \in \mathbb{R}^4, \forall u \in \mathcal{V}$. Going through the second and last GNN layer, the final result remains the dual graph structure, but the node representation has size one $\mathbf{z}_u = \mathbf{h}_u^{(2)} \in \mathbb{R}^1, \forall u \in \mathcal{V}$. Since a dual graph node representation is equivalent to a primal graph edge representation, one can then consider the node output of the GNN as a representation for edge regression. Finally, having a proper training set and a proper error function to compare the predictions against the truth values, after several training iterations

(training epochs) with a back-propagation technique to update the MLP parameters, this GNN could be used to predict road network edge free-flow travel time.

From the fact that, when learning from road network graphs, edge features have importance precedence over node features, arises a need to use the Dual Graph representation. The structural characteristics of road networks, however, introduce an extra layer of information to a graph representation: in addition to node and edge features, road network graphs contain between-edge features characterizing the relation between road segments (i.e., edges on the dual representation). Therefore, road networks are **edge-relational** graphs (JEPSEN; JENSEN; NIELSEN, 2020). This is one of the structural characteristics that distinguish road networks from other extensively studied graph types.

Road networks are also graphs with **low density**, meaning that the average node degree value is usually in a reduced scale compared to other graph types. Having fewer adjacent road segments (i.e., smaller neighborhoods) is a structural characteristic that makes road networks sensitive to aberrant neighbors when aggregating neighborhood information (JEPSEN *et al.*, 2018).

Finally, common GNN architectures implicitly assume that the input graph data exhibit homophily, that is: neighbor nodes have similar features and changes in the graph occur gradually. On the contrary, road networks are graphs that exhibit **volatile homophily**. Such graphs might not be considered heterophilic graphs as a whole, since regions within the input data can be highly homophilic, but those graphs will usually present sharp boundaries with abrupt changes separating its homophilic regions (JEPSEN *et al.*, 2018).

The edge-relational nature of road network graphs along with its intrinsic low density and volatile homophily are the main structural characteristics that justify the necessity of specifically designed GNN architectures. Some are briefly explained in the remaining of this Chapter, Section 2.4, while the Relational Fusion Networks (JEPSEN; JENSEN; NIELSEN, 2020), the main focus of this work, are detailed in Section 3.2.

2.4 Related work

This section briefly presents some of the literature on applying graph neural networks to learn about road network graphs. The majority of the literature is interested in road type classification (JEPSEN; JENSEN; NIELSEN, 2020; GHARAEE *et al.*, 2021) or spatiotemporal regression tasks (YU; YIN; ZHU, 2018; LI *et al.*, 2018; ZHAO *et al.*, 2020; DERROW-PINION *et al.*, 2021; YAN *et al.*, 2023; JIN *et al.*, 2023; ZHANG; LONG, 2023; NIPPANI DONGYUE LI, 2024). Section 3 better details the graph neural network models explored in this work, both generic and road network targeted.

On road networks' static edge characteristics, only Jepsen, Jensen and Nielsen (2020) explore a similar static edge regression task. The present work, particularly, targets the edge regression task for a specific static edge feature: the free-flow travel time (in seconds), using in-

formation retrieved directly from OSM while Jepsen, Jensen and Nielsen (2020) use an average of historical GPS data from a private dataset on their work.

2.4.1 Spatio-Temporal graph convolutional networks

Although this work does not explore any temporal traffic data, it is worth mentioning works on spatiotemporal GNNs for road networks, usually applied to time series prediction in traffic (YU; YIN; ZHU, 2018; LI *et al.*, 2018; ZHAO *et al.*, 2020; DERROW-PINION *et al.*, 2021; YAN *et al.*, 2023; JIN *et al.*, 2023; ZHANG; LONG, 2023) or safety (NIPPANI DONGYUE LI, 2024) domains. Such spatiotemporal models are usually composed of two blocks: the temporal, which can be any time series regression architecture; and the spatial, which is usually a graph neural network. GNN layers can be the base ones, like the graph convolutional network presented in Section 2.2.1, or enhanced variations like GraphSAGE (HAMILTON; YING; LESKOVEC, 2017a) and Graph Isomorphism Network (GIN) (XU *et al.*, 2018a), which are detailed in Section 3.

Yu, Yin and Zhu (2018), for example, propose the Spatio-Temporal Graph Convolutional Networks (STGCN), a model with spatio-temporal convolutional blocks composed by temporal gated convolution layers combined with spectral graph convolutional networks (DEFFERRARD; BRESSON; VANDERGHEYNST, 2016), and compares its performance against time series regression techniques (e.g., Auto-Regressive Integrated Moving Average) and the Graph Convolutional Recurrent Unit (GCRU)[7] proposed by Li *et al.* (2018).

Similarly, Zhao *et al.* (2020) present the Temporal Graph Convolutional Network (T-GCN), combining the Graph Convolutional Network (GCN) (BRUNA *et al.*, 2013) and a Gated Recurrent Unit (GRU) to capture the spatial and temporal dependence simultaneously. Derrow-Pinion *et al.* (2021) propose a GNN model defined with three aggregation and update functions corresponding to edge, node and global (i.e., supersegment-level) operations, composed into an encode-process-decode architecture to learn a separate model for each time horizon of estimated travel time.

Finally, Nippani Dongyue Li (2024) evaluates existing deep learning methods for predicting the occurrence of accidents on road networks, testing static graph neural networks, like GraphSAGE (HAMILTON; YING; LESKOVEC, 2017a) and GIN (XU *et al.*, 2018a), and spatiotemporal graph neural networks, like DCRNN (LI *et al.*, 2018) and STGCN (YU; YIN; ZHU, 2018).

[7] This work models the traffic flow as a diffusion process on a directed graph and introduce the Diffusion Convolutional Recurrent Neural Network (DCRNN)

2.4.2 Graph Attention Isomorphism Network (GAIN)

Graph Attention Isomorphism Networks (GAIN) (GHARAEE *et al.*, 2021) is an aggregation approach proposed to overcome state-of-the-art GNNs on the road type classification task, using the road network dual representation as input data.

Additionally, Gharaee *et al.* (2021) propose using Topological Neighborhoods instead of the direct one-hop neighborhood usually selected for aggregation. The Topological Neighborhood of a given node u is sampled from the local neighborhood ($\mathcal{N}_l(u)$) and a set of global ($\mathcal{N}_g(u)$) neighbors, based on an unbiased random walk (PEARSON, 1905), to extend node representation and improve the learning procedure. The local neighborhood random walk has a size L_l and the global random walk has a size L_g where $L_g = 2 \cdot L_l$.

In the GAIN architecture, a node aggregates, with the *SUM* function, information from its neighbors based on an importance value, i.e. attention coefficient, given to each other node (GHARAEE *et al.*, 2021):

$$\mathbf{h}_u^{(k)} = MLP^{(k)}((1+\varepsilon^{(k)}) \cdot \mathbf{h}_u^{'(k-1)} + \sigma \sum_{v \in \mathcal{N}(u)} (a_{u,v}^{(k-1)} \mathbf{h}_v^{'(k-1)})) \tag{14}$$

where $\mathbf{h}_u^{'} = \boldsymbol{W}_m^{'} \cdot \mathbf{h}_u$ represents a linear transformation of a feature vector into a higher level feature space using the weight matrix $\boldsymbol{W}_m^{'}$. The nonlinearity can be an *ELU* or an *Identity* function and experiments in (GHARAEE *et al.*, 2021) showed a slightly better performance with the *Identity* function. The $MLP^{(k)}$ is a multi-layer perceptron with one hidden layer.

The attention coefficients $a_{u,v}^{(k)}$ are inspired by the Graph Attention Networks (GAT) architecture (Section 3.1.2) and are calculated with a feed-forward neural network, a $LeakyReLU$ non-linearity and a $softmax$ normalization (Section 3.1.2). Compared to GAT though, GAIN applies attention weights only to the neighbors of a given node u, excluding the node u, and uses *SUM* rather than *MEAN* to aggregate it. Gharaee *et al.* (2021) explore a multi-head attention (Equation 15) setting that did not show performance improvements for GAIN training.

$$\mathbf{h}_u^{(k)} = MLP^{(k)}(\boldsymbol{W}((1+\varepsilon^{(k)}) \cdot \mathbf{h}_u^{'(k-1)} + \sigma \sum_{m=1}^{M} \sum_{v \in \mathcal{N}(u)} (a_{m_{u,v}}^{(k-1)} \mathbf{h}_v^{'(k-1)}))) \tag{15}$$

GAIN architecture is highly based on the GIN architecture (Section 3.1.3). It is the reason why it has "Isomorphism" on its name even with the proposed aggregation not being fully injective on multisets, due to the use of non-linearity and attention weights. Similarly to the GIN formalization, on GAIN ε can be a learnable parameter or a fixed scalar, experiments performed in (GHARAEE *et al.*, 2021) use the fixed scalar $\varepsilon = 0$ setting.

3 THE ADDRESSED GRAPH NEURAL NETWORKS

Graph embedding and transductive learning are valuable, but when representation learning is aimed at helping urban planning, inductive learning with GNNs is the most suitable choice. This chapter presents four GNN models selected from the literature for this work. Section 3.1 presents general inductive learning GNNs while Section 3.2 presents GNNs particularly designed for urban road network data. All models presented in this chapter is part of the experimentation tool set.

3.1 General purpose GNNs

3.1.1 Graph Sample and Aggregate (GraphSAGE)

GraphSAGE (HAMILTON; YING; LESKOVEC, 2017a) is one of the first works to overcome the transductive nature of representation learning on graphs, presenting an inductive framework capable of recognizing structural properties of a node's neighborhood that reveal both the node's local role in the graph and its global position. Transductive approaches require that all nodes are present during the training of the embeddings however, Hamilton, Ying and Leskovec (2017a) explore the process of learning the functions that generate the embeddings for unseen data, instead of learning the embeddings themselves. They extend GCNs (KIPF; WELLING, 2016) to inductive unsupervised learning by proposing a framework that generalizes GCNs through trainable aggregation functions.

GraphSAGE aims to learn embedding functions that generalize to unseen nodes by learning the topological structure of each node's neighborhood, as well as the distribution of node features in the neighborhood and keeps the computational footprint by defining neighborhoods as uniformly sampled sets of fixed-size of a given node's original neighborhood. It also learns structural information about a node's role in a graph, despite the fact that it is inherently based on features.

The framework trains a set of *aggregator functions* that learn how to aggregate feature information from a node's local neighborhood by applying a graph-based loss function to the output representations $\boldsymbol{z}_u, \forall u \in \mathcal{V}$ and tuning K aggregator functions parameters, $AGGREGATE_k, \forall k \in \{1,...,K\}$, and weight matrices $\boldsymbol{W}^k, \forall k \in \{1,...,K\}$ via stochastic gradient descent. Each node neighborhood is uniformly sampled again at each iteration. The loss function encourages nearby nodes to have similar representations while enforcing that the representations of disparate nodes are highly distinct, usually without task-specific supervision (unsupervised loss function) but it can be augmented by a task-specific objective.

The aggregation architectures proposed by Hamilton, Ying and Leskovec (2017a) are all symmetric functions, i.e., invariant to permutations of its inputs, since they must operate over an unordered set of node's features vectors.

- **GraphSAGE-GCN:** The aggregation is nearly equivalent to the convolutional propagation rule used in the transductive GCN framework (KIPF; WELLING, 2016). First, the sampled neighborhood features are aggregated using the *MEAN* function, then concatenated to the node's feature vector, and finally, one single weight matrix is multiplied to the resulting concatenated matrix before applying the non-linearity.

$$\mathbf{h}_u^{(k)} = \sigma(\boldsymbol{W}^{(k)} \cdot CONCAT(\mathbf{h}_u^{(k-1)}, MEAN_{v \in \mathcal{N}(u)}(\mathbf{h}_v^{(k-1)}))) \tag{16}$$

- **GraphSAGE-Mean:** Neighborhood's feature vectors aggregation stills an element-wise mean of the sampled neighbors, but now there are two weight matrices when updating node's features: $\boldsymbol{W}_{self}$ for weighting the node's features contribution and $\boldsymbol{W}_{neighbors}$ for weighting the aggregated neighborhood's contribution.

$$\mathbf{h}_u^{(k)} = \sigma(\boldsymbol{W}_{self}^{(k)} \cdot \mathbf{h}_u^{(k-1)} + \boldsymbol{W}_{neighbors}^{(k)} \cdot MEAN_{v \in \mathcal{N}(u)}(\mathbf{h}_v^{(k-1)})) \tag{17}$$

- **GraphSAGE-LSTM:** For specific applications, a more complex aggregator structure based on an LSTM architecture[1] (HOCHREITER; SCHMIDHUBER, 1997) might have a larger expressive capability when compared to the *MEAN* architecture.

- **GraphSAGE-PooI:** Each neighbor's feature vector goes through a fully connected neural network[2] and then an element-wise max-pooling operation is applied to aggregate information from the transformed neighborhood set.

$$AGGREGATE_{(k)}^{Pool} = max(\{\sigma(\boldsymbol{W}_{pool} \cdot \mathbf{h}_v^{(k)} + \boldsymbol{b}), \forall v \in \mathcal{N}(u)\}) \tag{18}$$

The embedding generation is done by forward propagation (Algorithm 1), a K-layer network model that uses the learned parameters of K aggregator functions, $AGGREGATE_k$, and the learned set of K weight matrices, $\boldsymbol{W}^k$, which are used to propagate information between different layers.

[1] To overcome the fact that LSTMs are not inherently permutation invariant (i.e., process inputs in a sequential manner), it was adapted to operate on an unordered set by applying the LSTMs to a random permutation of the neighbors.

[2] The neural network can be an arbitrarily deep multi-layer perceptron but Hamilton, Ying and Leskovec (2017a) focused on simple single-layer architectures.

Algorithm 1 GraphSAGE forward propagation (HAMILTON; YING; LESKOVEC, 2017a)

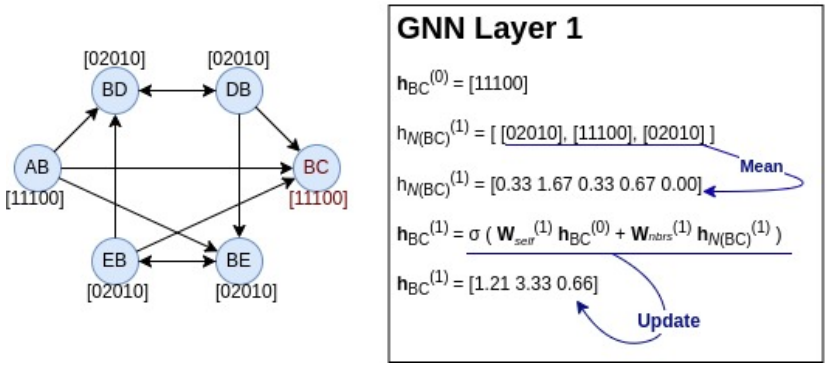
Input: Graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$; input features $\mathbf{x}_u, \forall u \in \mathcal{V}$; depth K; weight matrices $\boldsymbol{W}^k, \forall k \in \{1,...,K\}$; differentiable aggregator functions $AGGREGATE_k, \forall k \in \{1,...,K\}$; non-linearity σ; neighborhood function $\mathcal{N} : u \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\boldsymbol{z}_u$ for all $u \in \mathcal{V}$

$\mathbf{h}_u^0 \leftarrow \mathbf{x}_u, \forall u \in \mathcal{V}$
for $k \leftarrow 1$ to K **do**
  for $u \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(u)}^{(k)} \leftarrow AGGREGATE_k(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\})$
    $\mathbf{h}_u^{(k)} \leftarrow \sigma(\boldsymbol{W}^{(k)} \cdot CONCAT(\mathbf{h}_u^{(k-1)}, \mathbf{h}_{\mathcal{N}(u)}^{(k)})$
  end for
  $\mathbf{h}_u^{(k)} \leftarrow \mathbf{h}_u^{(k)} / \|\mathbf{h}_u^{(k)}\|_2, \forall u \in \mathcal{V}$
end for
$\boldsymbol{z}_u \leftarrow \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}$

GraphSAGE operates on the graph's node level, so Figure 8 pictures the application of GraphSAGE-Mean to the example of road network edge representation, from Section 2.3, using the RNG dual graph as GNN input.

Figure 8 – GraphSAGE-Mean: Training with Dual Graph



Source: Own work (2025).

3.1.2 Graph Attention Network (GAT)

An architecture that operates on graph-structured data with masked self-attentional layers (VELICKOVIC *et al.*, 2017). When layers are stacked, nodes pay attention to the features of their neighborhood. This allows for different weights to be assigned to different neighbors, making the model more capable and likely improving its interpretability

The proposed framework is agnostic to the particular choice of attention mechanism, but Velickovic *et al.* (2017) use a setup that closely follows the work of Bahdanau, Cho and Bengio (2014).

Aiming to obtain sufficient expressive power to transform input features ($\in \mathbb{R}^F$) into higher-level features ($\in \mathbb{R}^{F'}$), at least one learnable linear transformation is required. A weight matrix $\boldsymbol{W} \in \mathbb{R}^{F \times F'}$ parametrizes a shared linear transformation that leverages self-attention on nodes. Therefore, there is a shared attentional mechanism $a : \mathbb{R}^{F \times F'} \rightarrow \mathbb{R}$ that computes attention coefficients to weight the importance a given node j's features to a different node i.

$$e_{ij} = a(\boldsymbol{W} \cdot \mathbf{h}_i, \boldsymbol{W} \cdot \mathbf{h}_j) \tag{19}$$

If every node attends to every other node in the graph, the model would drop all structural information in favor of attentional information only. So, Velickovic *et al.* (2017) inject graph structure to the attention by only computing e_{ij} for nodes $j \in \mathcal{N}(i)$, where this neighborhood is a node's first-order neighborhood including the node itself. This technique is called *Masked Attention*. The $softmax$ function is applied to normalize across all choices of j, in order to make coefficients easily comparable between different nodes.

$$\alpha_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})} \tag{20}$$

Velickovic *et al.* (2017) chose the **attentional mechanism (a)** as a single-layer feed-forward neural network parameterized by a weight vector $\boldsymbol{a} \in \mathbb{R}^{2F'}$ with a $LeakyReLU(\alpha = 0.2)$ non-linearity.

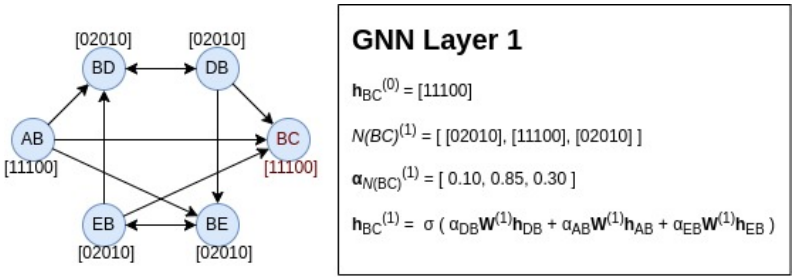
$$\alpha_{ij} = \frac{\exp(LeakyReLU(\boldsymbol{a}^T \cdot [\boldsymbol{W} \cdot \mathbf{h}_i \oplus \boldsymbol{W} \cdot \mathbf{h}_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(LeakyReLU(\boldsymbol{a}^T \cdot [\boldsymbol{W} \cdot \mathbf{h}_i \oplus \boldsymbol{W} \cdot \mathbf{h}_k]))} \tag{21}$$

where ^{T} represents transposition and $\oplus$ is the concatenation operation.

Finally, the attention coefficients α_{ij} are used to compute a linear combination of the features corresponding to them and after applying a non-linearity σ the output features vector is:

$$\mathbf{h}_u^{(k)} = \sigma\Big(\sum_{v \in \mathcal{N}(u)} (\alpha_{uv} \cdot \boldsymbol{W} \cdot \mathbf{h}_v^{(k-1)}) \Big) \tag{22}$$

Figure 9 – GAT: Training with Dual Graph



Source: Own work (2025).

Again, GAT operates on a graph's node level, so Figure 9 pictures the application of GAT to the example of road network edge representation, from Section 2.3, using the RNG dual graph as GNN input. For the sake of simplicity, this example considers that all attention coefficients $\alpha_{i,j}$ are already calculated using Equation 21 and are available to be used in the aggregation process.

Velickovic *et al.* (2017) found that extending this mechanism to employ multi-head attention (VASWANI *et al.*, 2017) is helpful to stabilize the learning process of self-attention, with the expense of multiplying the storage and parameter requirements by a factor of H.

$$\mathbf{h}_u^{(k)} = \Big\Vert_{h=1}^{H} \sigma\Big(\sum_{v \in \mathcal{N}(u)} (\alpha_{uv}^h \cdot \boldsymbol{W}^h \cdot \mathbf{h}_v^{(k-1)})\Big) \tag{23}$$

In a multi-head attention setting each layer's output $\mathbf{h}^{(k)}$ consists of $H \cdot F^{'}$, rather than $F^{'}$, features for each node. Therefore, the final (prediction) layer concatenation is then replaced by the averaging operation and the application of the final non-linearity is delayed:

$$\boldsymbol{z}_u^{(k)} = \sigma\Big(\frac{1}{H} \cdot \sum_{h \in H}\Big(\sum_{v \in \mathcal{N}(u)} (\alpha_{uv}^h \cdot \boldsymbol{W}^h \cdot \mathbf{h}_v^{(k-1)})\Big)\Big) \tag{24}$$

Since all the self-attentional computations can be parallelized across all edges and the output features computations can be parallelized across all nodes, GAT can be considered a computationally highly efficient architecture. Additionally, applying the mechanism in such a shared manner to all edges allows the input graphs to be directed and also enables inductive learning settings. In comparison to GraphSAGE, GAT also uses node features only but accesses the entire neighborhood rather than sampling it like the former.

3.1.3 Graph Isomorphism Network (GIN)

Apart from the graph neural networks' recursive message-passing scheme, it is also interesting to understand its expressiveness power when learning how to represent and distinguish between different graph structures. Xu *et al.* (2018a) hypothesize that a GNN can have as large discriminative power as the Weisfeiler-Lehman (WL) graph isomorphism test (WEISFEILER; LEHMAN, 1968) if its aggregation scheme is highly expressive and can model injective functions.

Xu *et al.* (2018a) define a multi-set as a generalized concept of a set that allows multiple instances of its elements. That is, a 2-tuple $X = (S,m)$ where S is the underlying set of distinct elements and $m : S \rightarrow \mathbb{N}_{\geq 1}$ gives the multiplicity of the elements. Multi-sets then, can be seen as the correct description of a node neighborhood, since the set of neighbors' features

can contain repeated vectors, i.e., different neighbor nodes can have exactly the same feature values.

Also, Xu *et al.* (2018a) presented a theoretical framework for analyzing the expressive power of GNNs to capture different graph structures and showed that GCN (KIPF; WELLING, 2016) and GraphSAGE (HAMILTON; YING; LESKOVEC, 2017a) are networks that fail to learn how to distinguish certain simple graph structures.

A maximally powerful GNN should map two nodes to the same location in the embedding space if, and only if, they have identical neighborhood sub-tree structures with identical features on the corresponding nodes. Moreover, a maximally powerful GNN would never map two different neighborhoods (i.e., a multi-set of feature vectors) to the same representation. Therefore, Xu *et al.* (2018a) hypothesize that to achieve maximum expressive power a GNN must have injective aggregation functions. This way, one can be sure that different graph structures are mapped to different representations in the embedding space and, also, such a GNN model should be able to solve the graph isomorphism problem for all instances that the WL-test can.

Xu *et al.* (2018a) proposition states that $\mathcal{A} : \mathcal{G} \rightarrow \mathbb{R}^d$ is a GNN with a sufficient number of layers and it maps any $\mathcal{G}_1$ and $\mathcal{G}_2$ graphs that the WL-test decides as non-isomorphic to different embeddings if:

1. f, which operates on multi-sets, and ϕ are injective functions.

$$\mathbf{h}_u^{(k)} = \phi(\mathbf{h}_u^{(k-1)}, f(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\})) \tag{25}$$

2. when applied to graph classification tasks, $\mathcal{A}$'s graph-level readout, which operates on the multi-set of node features $\{\boldsymbol{z}_i^{(K)}\}$, is injective.

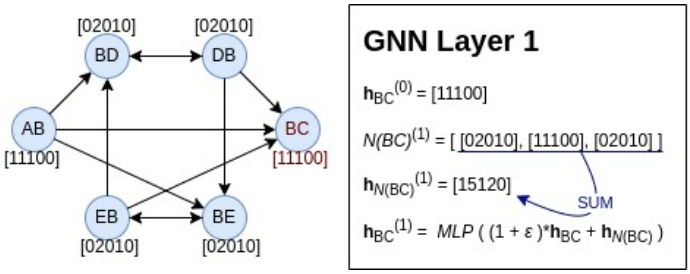
The Graph Isomorphism Network (GIN) (XU *et al.*, 2018a) have neighbor aggregation done through injective multi-set functions - "deep multi-sets" - parametrized with neural networks, based on the universal approximation theorem (RUMELHART; HINTON; WILLIAMS, 1986). Let ε be a learnable parameter or a fixed scalar:

$$\mathbf{h}_u^{(k)} = MLP^{(k)}((1 + \varepsilon^{(k)}) \cdot \mathbf{h}_u^{(k-1)} + \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)}) \tag{26}$$

The *SUM* aggregation is both permutation invariant and injective on multisets, therefore it should be able to distinguish different graph structures. Xu *et al.* (2018a) experiment with two GIN variants: GIN-ε, in which ε is learned during the training, and GIN-0, in which $\varepsilon = 0$ is a fixed scalar. Figure 10 pictures the application of GIN to the same example of Road Network Graph edge representation, from Section 2.3, using the RNG dual graph as GNN input.

Other existing GNNs use a 1-layer perceptron ($\sigma \circ \boldsymbol{W}$) (DUVENAUD *et al.*, 2015) (KIPF; WELLING, 2016) (ZHANG *et al.*, 2018) and, although those are valid examples of generalized

Figure 10 – GIN: Training with Dual Graph

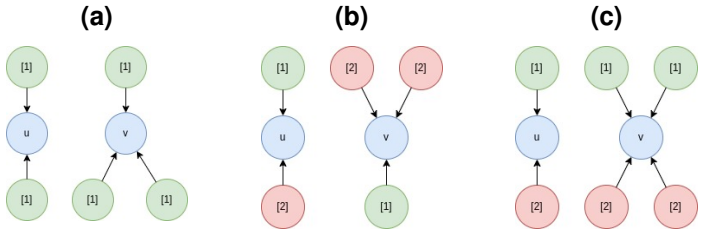


Source: Own work (2025).

linear models (MCCULLAGH; NELDER, 1972), are not enough for graph learning. First, such GNN layers degenerate into simply summing neighborhood features. Secondly, it lacks the bias term. Even with bias though, 1-layer perceptrons are not a universal approximator of multiset functions, while MLPs are. Xu *et al.* (2018a) also point out that 1-layer perceptrons can severely under-fit training data and show worse test accuracy than MLPs on tasks of graph classification.

To stress their point, Xu *et al.* (2018a) list some simple neighborhood structures that can confuse *MEAN* and *MAX-POOLING* based aggregators. Although those are the aggregation functions analyzed, the theoretical framework is supposed to be general enough to analyze other more complicated variants of aggregation functions (e.g., attention mechanisms, LSTMs).

Figure 11 – Simple graphs structures that confuse *MEAN* and *MAX-POOLING*



Source: (XU *et al.*, 2018a).

Figure 11 (a) shows two different neighborhood structures that have the same aggregation result when using *MEAN* or *MAX-POOLING* based aggregators, since the features of all neighbors are the same[3]. Both aggregators have trouble with nodes having repeated features, while a *SUM* aggregation would be able to differ between the two neighborhoods[4]. In Figure 11 (b), the *MEAN* aggregator does not fail to differ both neighborhoods, but the *MAX-POOLING*

[3] *{MEAN(1,1), MEAN(1,1,1)} = {1, 1}* and *{MAX-POOLING(1,1), MAX-POOLING(1,1,1)} = {1, 1}*

[4] *{SUM(1,1), SUM(1,1,1)} = {2, 3}*

aggregator fails[5]. Finally, on Figure 11 (c), *MEAN* and *MAX-POOLING* aggregators would fail again[6,7], while a *SUM* aggregation would not[8].

The neighborhood examples in Figure 11 show the representational power of a multi-set injective aggregation function, i.e., *SUM*. But since GNNs designed with *MEAN* and *MAX-POOLING* aggregators have been proven efficient on determined tasks, those aggregators do not have only weaknesses (XU *et al.*, 2018a).

Actually, from those examples analysis, it is possible to realize that the *MEAN* captures the proportion of elements in a multi-set, but not the exact multi-set. It may be a good choice for tasks in which statistical and distributional information in the graph is more important than the exact structure. This may explain why GNNs with *MEAN* aggregators are effective for node classification tasks where node features are rich and the neighborhood distributions provide a strong signal for the task. Also, from those examples analysis, it is possible to get that *MAX-POOLING* considers nodes with the same features as a single node, i.e., handles a multi-set as a set. It doesn't capture the exact structure or distribution of features, but it may perform well for tasks where it's important to identify representative elements of a multi-set rather than the exact structure or distribution.

3.2 Relational Fusion Network (RFN)

The state-of-the-art in Graph Neural Networks for learning on Road Network Graphs, Relational Fusion Networks (RFNs) (JEPSEN; JENSEN; NIELSEN, 2020) was designed specifically for machine learning on road networks. All previously mentioned GNNs are capable, in theory, of leveraging the structure of an Road Network Graph, but those were architectures designed for node classification tasks on graphs (e.g., social networks, proteins) that differ too much from Road Network Graphs in terms of information available and structural characteristics (e.g., average node degree, homophily). Therefore, some implicit assumptions of general purpose GNNs do not apply for RNG-related tasks and experiments of Jepsen, Jensen and Nielsen (2020) found that such models fail to leverage Road Network Graph structure and even to outperform regular MLPs on such tasks. Additionally, the authors argue that simply using the Road Network Graph dual representation is not enough to make common GNNs suitable for road graphs.

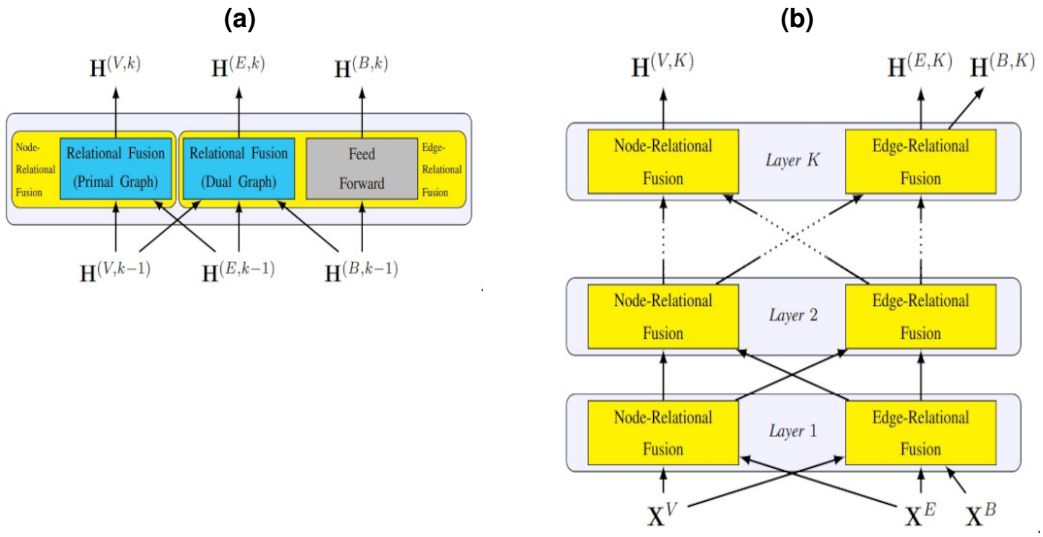
First of all, Road Network Graphs have a low number of edge and node features, information that is actually often incomplete. Moreover, it contains not only node and edge information, but also between-edge attributes characterizing the relation between road segments and, therefore, those graphs are edge-relation. Secondly, Road Network Graphs are usually low-density

[5] *{MAX-POOLING(1,2), MAX-POOLING(1,2,2)} = {2, 2}*
[6] *{MEAN(1,2), MEAN(1,1,2,2)} =* $\{\frac{3}{2}, \frac{3}{2}\}$
[7] *{MAX-POOLING(1,2), MAXPOOLING(1,1,2,2)} = {2, 2}*
[8] *{SUM(1,2), SUM(1,1,2,2)} = {3, 6}*

networks, meaning that it has low average node degree (i.e., few adjacent street segments). The small size of neighborhoods makes aggregation on such graphs highly sensitive to aberrant neighbors. Finally, GNNs commonly are based on the implicit assumption that graphs exhibit homophily. That is, neighbor nodes have similar features and changes in the graph occur gradually. On the contrary, Road Network Graphs exhibit volatile homophily: regions can be highly homophilic but have sharp boundaries with abrupt changes.

The Relational Fusion Network (JEPSEN; JENSEN; NIELSEN, 2020) is a spatial, inductive and general-purpose method proposed for Road Network Graph learning tasks, designed to overcome all the before-mentioned specificities of this type of graph. It aggregates over representations of relations, instead of representations of neighbors. It is designed to disregard relations with aberrant neighbors using an attention mechanism. It learns representations based, not only on dual graph but on both the primal (i.e., node relational) and the dual (i.e., edge relational) representations simultaneously.

Figure 12 – Relational Fusion: (a) one RF layer and (b) a K-layered RF network



Source: (JEPSEN; JENSEN; NIELSEN, 2020).

A RFN layer receives an input graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ with a set of node features $X^{\mathcal{V}} \in \mathbb{R}^{|V|\times d^{\mathcal{V}}}$, a set of edge features $X^{\mathcal{E}} \in \mathbb{R}^{|E|\times d^{\mathcal{E}}}$ and a set of between-edge features $X^{\mathcal{B}} \in \mathbb{R}^{|B|\times d^{\mathcal{B}}}$, where $d^{\mathcal{V}}, d^{\mathcal{E}}$ and $d^{\mathcal{B}}$ represent, respectively, the node, the edge and the between-edge feature vectors dimensions. The input graph can be broken on its node-relational (i.e., primal), $\mathcal{G}^{\mathcal{P}} = (\mathcal{V},\mathcal{E}), X^{\mathcal{V}}, X^{\mathcal{E}}$, and its edge-relational (i.e., dual), $\mathcal{G}^{\mathcal{D}} = (\mathcal{E},\mathcal{B}), X^{\mathcal{E}}, X^{\mathcal{B}}$, representations. Each fusion layer (Figure 12 (a)) performs both a node-relational fusion and an edge-relational fusion, to learn representations from both views simultaneously, and captures the interdependence between both views by using as input the node and edge representations (i.e, outputs) from previous layers (Figure 12 (b)).

Algorithm 2 The Relational Fusion Operator (JEPSEN; JENSEN; NIELSEN, 2020)

function $RelationalFusion^k(\mathcal{G} = (\mathcal{V},\mathcal{E}), \mathbf{h}^{\mathcal{V}',k-1}, \mathbf{h}^{\mathcal{E}',k-1})$
  let $\mathbf{h}^{\mathcal{V}',k-1}$ be an arbitrary $|\mathcal{V}'| \times d^{F^k}$ real feature matrix.
  for each $u' \in \mathcal{V}'$ **do**
    $F_{u'} \leftarrow FUSE^k(\mathbf{h}_{u'}^{\mathcal{V}',k-1}, \mathbf{h}_{u',v'}^{\mathcal{E}',k-1}, \mathbf{h}_{v'}^{\mathcal{V}',k-1} | v' \in \mathcal{N}(u'))$
    $\mathbf{h}_{u'}^{\mathcal{V}',k} \leftarrow AGGREGATE^k(F_{u'})$
    $\mathbf{h}_{u'}^{\mathcal{V}',k} \leftarrow NORMALIZE^k(\mathbf{h}_{u'}^{\mathcal{V}',k})$
  end for
  return $\mathbf{h}^{\mathcal{V}',k}$

A relational fusion (Algorithm 2) is a graph convolutional operator in which neighborhood aggregation is replaced by a relational aggregation. That is, aggregation is performed over representations of relations (u,v) that a given node $u \in \mathcal{V}$ participates, as opposed to aggregation over-representation of u and its neighbors. A relation representation includes representations of nodes $u \in \mathcal{V}$, $v \in \mathcal{V}$ and the representation of the edge $(u,v) \in \mathcal{E}$, all fused by a Fusion Function ($FUSE^k$). Then, Aggregation is done over the set of fused representations, generating a single latent representation of u, called u'. Finally, there is an optional normalization, important when the relational aggregation has different scales across elements with different neighborhood sizes (e.g. when aggregation is based on SUM). Jepsen, Jensen and Nielsen (2020) used L2-normalization when needed.

A Fusion Function is the relational fusion layer component responsible for extracting the right information from each relation. It was designed to capture the Road Network Graphs' intrinsic volatile homophily, allowing the creation of sharp boundaries at the edge of homophilic regions. (JEPSEN; JENSEN; NIELSEN, 2020) propose and experiment with two different Fusion Functions.

The *ADDITIVEFUSE* function transforms each representation individually, using three trainable weight matrices:

$$\begin{aligned} ADDITIVEFUSE^k(\mathbf{h}_u^{(\mathcal{V},k-1)}, \mathbf{h}_v^{(\mathcal{V},k-1)}, \mathbf{h}_{u,v}^{(\mathcal{E},k-1)}) = \\ \sigma(\boldsymbol{W}^s \cdot \mathbf{h}_u^{(\mathcal{V},k-1)} + \boldsymbol{W}^t \cdot \mathbf{h}_v^{(\mathcal{V},k-1)} + \boldsymbol{W}^r \cdot \mathbf{h}_{u,v}^{(\mathcal{E},k-1)} + \boldsymbol{b}) \end{aligned} \tag{27}$$

where σ is an activation function, $b \in \mathbb{R}^{1xd^o}$ is a bias term and where $\boldsymbol{W}^s \in \mathbb{R}^{d^s xd^o}$, $\boldsymbol{W}^t \in \mathbb{R}^{d^t xd^o}$ and $\boldsymbol{W}^r \in \mathbb{R}^{d^r xd^o}$ are, respectively, the source, target and relation weights. Here, d^s and d^t are the source and target nodes feature dimensionality, d^r is the edge feature dimensionality and d^o is the output dimensionality of the fusion function. Equation 27 can be simplified as:

$$\begin{aligned} ADDITIVEFUSE^k(\mathbf{h}_u^{(\mathcal{V},k-1)}, \mathbf{h}_v^{(\mathcal{V},k-1)}, \mathbf{h}_{u,v}^{(\mathcal{E},k-1)}) = \\ \sigma(\boldsymbol{W}^R \cdot \mathbf{h}_{u,v}^{(\mathcal{R},k-1)} + \boldsymbol{b}) \end{aligned} \tag{28}$$

where $\boldsymbol{W}^R \in \mathbb{R}^{d^R x d^o}$ is a weight matrix with row dimensionality $d^R = d^s + d^t + d^r$ and $\mathbf{h}_{u,v}^{(\mathcal{R},k-1)} = \mathbf{h}_u^{(\mathcal{V},k-1)} \oplus \mathbf{h}_v^{(\mathcal{V},k-1)} \oplus \mathbf{h}_{u,v}^{(\mathcal{E},k-1)}$ is a relational feature matrix where $\oplus$ denotes matrix concatenation.

While the *ADDITIVEFUSE* function summarizes the relationship between u and v, it does not explicitly model interactions between representations. Therefore, the *INTERACTIONALFUSE* function is proposed to improve the modeling capacity of *ADDITIVEFUSE* function, enabling it to better address the challenge of volatile homophily at the cost of an increase in parameters, quadratic in the number of input features. *INTERACTIONALFUSE* captures and weight interactions at a much finer granularity:

$$\begin{aligned} INTERACTIONALFUSE^k(\mathbf{h}_u^{(\mathcal{V},k-1)}, \mathbf{h}_v^{(\mathcal{V},k-1)}, \mathbf{h}_{u,v}^{(\mathcal{E},k-1)}) = \\ \sigma((\mathbf{h}_{u,v}^{(\mathcal{R},k-1)} \cdot \boldsymbol{W}^I \odot \mathbf{h}_{u,v}^{(\mathcal{R},k-1)}) \cdot \boldsymbol{W}^R + \boldsymbol{b}) \end{aligned} \tag{29}$$

where $\boldsymbol{W}^I \in \mathbb{R}^{d^R x d^R}$ is a trainable weight matrix and $\odot$ denotes element-wise multiplication.

Finally, Algorithm 2 relational aggregator function is composed of an attention mechanism that filters out irrelevant and/or aberrant neighbors by weighting their contribution. Such filtering capacity is highly desirable for RGNs that, as previously mentioned, present volatile homophily and low density (i.e., small average node degree) that amplifies the aggregation noise:

$$AGGREGATE^k(\mathcal{F}_u) = \sum_{\boldsymbol{f}_v \in \mathcal{F}_u} A(u,v) \cdot \boldsymbol{f}_v \tag{30}$$

where $\mathcal{F}_u = \{\boldsymbol{f}_v | v \in \mathcal{N}_u\}$ is the set of fused relational representations of node u's relations to each of its neighbors $v \in \mathcal{N}_u$.

However, current graph attention mechanisms rely on a common transformation of each neighbor, which is incompatible with RFNs that rely on context-dependent neighbor transformation. The proposed Attentional Aggregator (JEPSEN; JENSEN; NIELSEN, 2020) computes a weighted mean over the relational representations, that corresponds to computing $softmax$ over the attention coefficients of all neighbors v of u, where the attentions weights sum to one and are calculated by:

$$A(u,v) = \frac{\exp(\mathcal{C}(\mathbf{h}_{u,v}^{(\mathcal{R},k-1)}))}{\sum_{n \in \mathcal{N}_u}(\exp(\mathcal{C}(\mathbf{h}_{u,n}^{(\mathcal{R},k-1)})))} \tag{31}$$

where $\mathcal{C} : \mathbb{R}^{d^R} \rightarrow \mathbb{R}$ is an attention coefficient function:

$$\mathcal{C}(\mathbf{h}_{u,v}^{(\mathcal{R},k-1)}) = \sigma(\mathbf{h}_{u,n}^{(\mathcal{R},k-1)} \cdot \boldsymbol{W}^C) \tag{32}$$

in which σ is an activation function and $\boldsymbol{W}^C \in \mathbb{R}^{d^R}$ is a weight matrix.

Finally, a K-layered Relational Fusion Network (Figure 12 (b)) uses the forward propagation algorithm (Algorithm 4) to propagate latent space representations of nodes, edges and

Algorithm 3 RFN Forward Propagation - JOIN (JEPSEN; JENSEN; NIELSEN, 2020)

function $JOIN^k(\mathbf{h}^{\mathcal{V}}, \mathbf{h}^{\mathcal{B}})$
  for each $((v,u),(u,w)) \in \mathcal{B}$ **do**
    $\mathbf{h}^{\mathcal{B}'}_{((v,u),(u,w))} \leftarrow \mathbf{h}^{\mathcal{B}}_{((v,u),(u,w))} \oplus \mathbf{h}^{\mathcal{V}}_{u}$
  end for
  return $\mathbf{h}^{\mathcal{B}'}$

Algorithm 4 The Relational Fusion Network Forward Propagation (JEPSEN; JENSEN; NIELSEN, 2020)

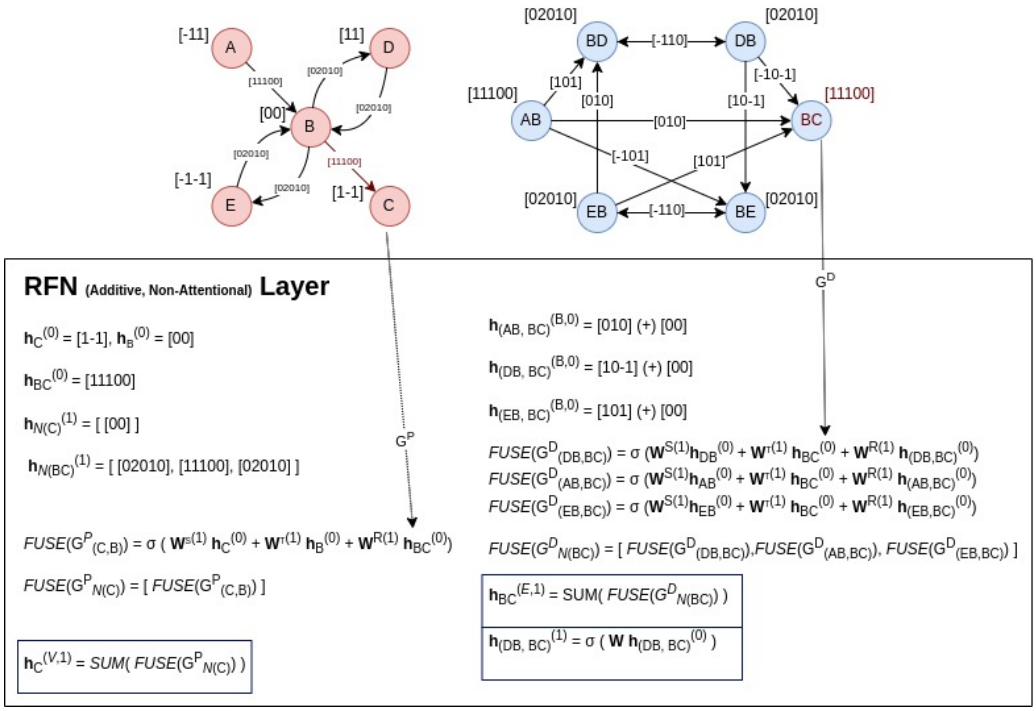
function $ForwardPropagation(X^{\mathcal{V}}, X^{\mathcal{E}}, X^{\mathcal{B}})$
  let $\mathbf{h}^{\mathcal{V},0} = X^{\mathcal{V}}$, $\mathbf{h}^{\mathcal{E},0} = X^{\mathcal{E}}$ and $\mathbf{h}^{\mathcal{B},0} = X^{\mathcal{B}}$
  for $k \leftarrow 1$ to K **do**
    $\mathbf{h}^{(\mathcal{V},k)} \leftarrow RelationalFusion^k(\mathcal{G}^{\mathcal{P}}, \mathbf{h}^{(\mathcal{V},k-1)}, \mathbf{h}^{(\mathcal{E},k-1)})$
    $\mathbf{h}^{(\mathcal{B}',k-1)} \leftarrow JOIN^k(\mathbf{h}^{(\mathcal{V},k-1)}, \mathbf{h}^{(\mathcal{B},k-1)})$
    $\mathbf{h}^{(\mathcal{E},k)} \leftarrow RelationalFusion^k(\mathcal{G}^{\mathcal{D}}, \mathbf{h}^{(\mathcal{E},k-1)}, \mathbf{h}^{(\mathcal{B}',k-1)})$
    $\mathbf{h}^{(\mathcal{B},k)} \leftarrow RelationalFusion^k(\mathbf{h}^{(\mathcal{B},k-1)})$
  end for
  return $\mathbf{h}^{\mathcal{V},K}$, $\mathbf{h}^{\mathcal{E},K}$, $\mathbf{h}^{\mathcal{B},K}$

between edges through its layers. In this setting, both Road Network Graph representations ($\mathcal{G}^{\mathcal{P}}$ and $\mathcal{G}^{\mathcal{D}}$) contain structural information about both the node and the edge-relation views.

Before applying a relational fusion function to the dual representation though, node and between-edge representations from the previous layer are concatenated (Algorithm 3) to capture the information from both sources that describe the relationships between two edges. The between-edge representations are transformed by a single feed-forward operator at each layer to learn increasingly abstract between-edge representations. Algorithm 4 has three outputs: node, edge and between-edge latent representations, that can be used for predictions on any of the three components or even for multi-task learning settings.

Having both primal and dual representations of a given Road Network Graph it is possible to replicate the example from Section 2.3 using RFN, in a similar way that was previously done for the general purpose GNNs. For the sake of simplicity, the RFN in this example encompasses an Additive Fuse function and a Non-attentional aggregator. Figure 13 illustrates how a single RFN would update node ($\mathbf{h}^{(\mathcal{V},k)}_{C}$)), edge ($\mathbf{h}^{(\mathcal{E},k)}_{BC}$)) and between-edge ($\mathbf{h}^{(\mathcal{B},k)}_{(DB,BC)}$)) representations. All three representation outputs could then be fed to the next RFN layer or used by another prediction/classification model.

Figure 13 – RFN: Training with Primal and Dual Graphs



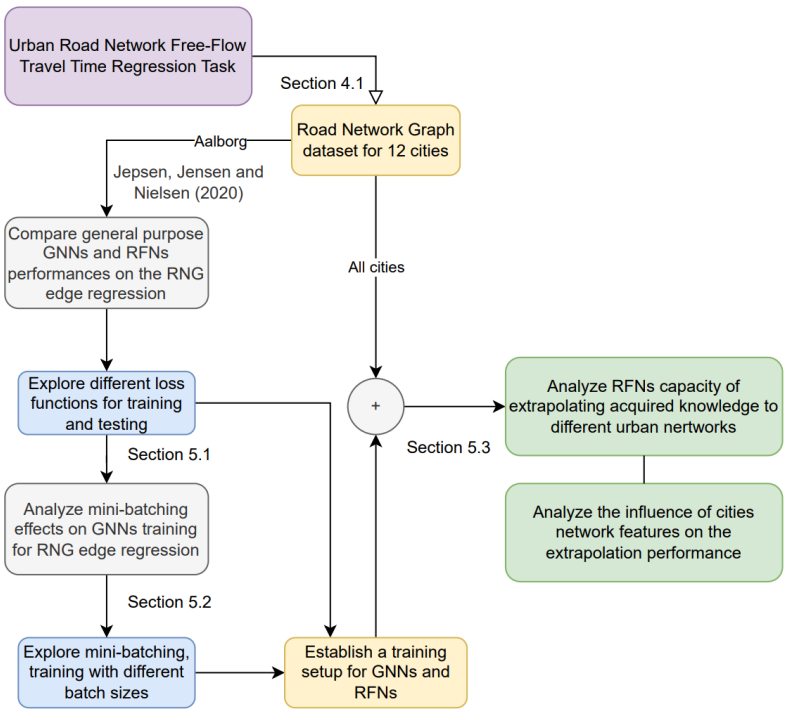
Source: Own work (2025).

4 METHODOLOGY

Making use of collaborative and open data sources, this work retrieves information from a selected group of cities to model each one as a Road Network Graph. It is then possible to explore GNNs on the free-flow travel time regression task and understand their performance and limitations. Section 4.1 details the Road Network Graph dataset, and Section 4.2 details the setup for the experiments aimed at answering this work's research questions.

Figure 14 shows a roadmap of the methodology followed to obtain the results of the experiments. By starting from Road Network Graphs of 12 cities, the branch on the left of Figure 14 considers only Aalborg city to compare general purpose GNNs and RFNs, whose results are given in Section 5.1. Experiments with mini-batches are then executed and results are presented in Section 5.2. After defining a training setup for GNNs and RFNs, other cities are considered to analyze the extrapolation capacity of RFNs, whose results are given in Section 5.3. Colors are used in Figure 14 to differentiate information: gray elements represent information already available in the literature; yellow elements represent operational steps; blue elements represent minor/exploratory contributions and green elements represent this work's main contributions.

Figure 14 – Roadmap of the methodology



Source: Own work (2025).

4.1 The Road Network Graph dataset

This section presents the addressed Road Network Graphs, including an introduction to the data source and tools used, as well as the list of selected cities along with the structural and informational models of their corresponding Road Network Graphs.

4.1.1 Data source

Data to study urban street networks are usually made available by city, state, or national data repositories, typically in the form of shape-files of digitized streets (BOEING, 2017). Acquiring, processing and ensuring consistency of street network information collected from various data sources can be challenging and time-consuming. Restricting the data gathering to a single data source though, can heavily reduce the sample size, limiting the data representation power.

There are proprietary and open access options for available data sources, including those originating from GPS-gathered data and map applications that usually include network structural information with temporal traffic information. This work uses OpenStreetMap® (OpenStreetMap contributors, 2017) as a single data source for retrieving urban street graphs. It avoids the overhead of gathering information from multiple sources and solves the sample size problem by providing enough centralized data.

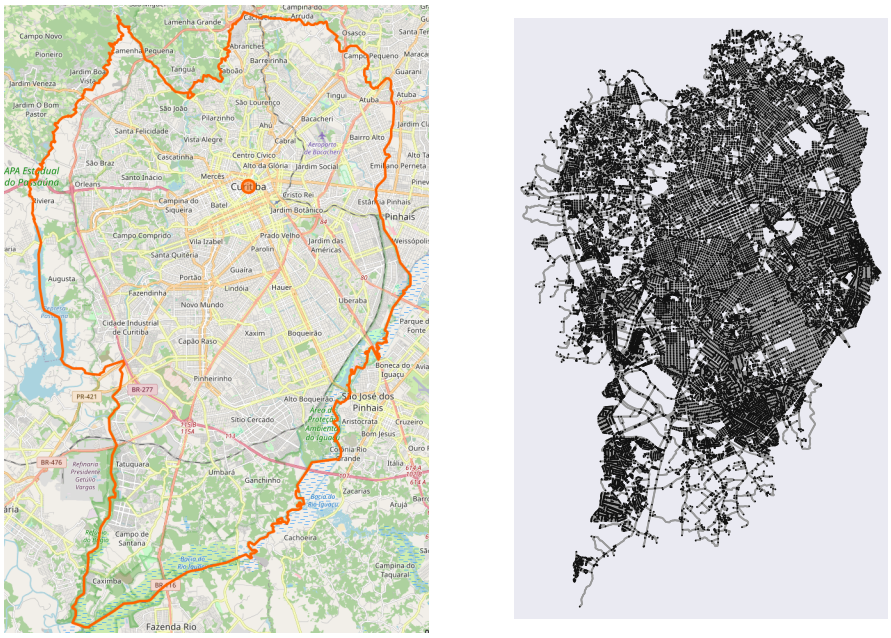
OpenStreetMap® (OSM) is an editable map database, built and maintained by volunteers and distributed under Open Data Commons Open Database License (ODbL) by the OpenStreetMap Foundation (OSMF). It is a collaborative worldwide mapping project, in which mapping takes place from geospatial information and data collected by volunteers during car trips, casual jogs, photos, videos and GPS traces. There are several OSM models to be downloaded (e.g. walkable, drivable, bikeable) depending on the urban transportation object of interest. This work focuses only on drivable network information. Data quality may vary among countries, but in general, there is high-quality street information available in the tool (HAKLAY, 2010; BARRON; NEIS; ZIPF, 2014; BOEING, 2019).

The OSMnx Python package licensed under the MIT license (BOEING, 2017) is used to retrieve data from OpenStreetMap®. OSMnx library facilitates the download of geospatial data from OSM to model, project, visualize, and analyze real-world street networks. OSMnx strongly relies on NetworkX (HAGBERG; SCHULT; SWART, 2008), a Python package developed for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Figure 15 shows the Brazilian city of Curitiba: (a) its OSM boundary polygon contour; (b) its OSMnx retrieved drivable network.

OSMnx internal functions are used to enrich the road network edge feature information. Function *add_edge_bearings()* adds compass bearing attributes to all graph edges. A bearing represents the clockwise angle in degrees between the north and the geodesic line from the origin node to the destination node. The function ignores self-loop edges as their bearings are

Figure 15 – Curitiba, Brazil: (a) OpenStreetMap® boundary and (b) OSMnx drivable network

(a) **(b)**



Source: (OpenStreetMap contributors, 2017).

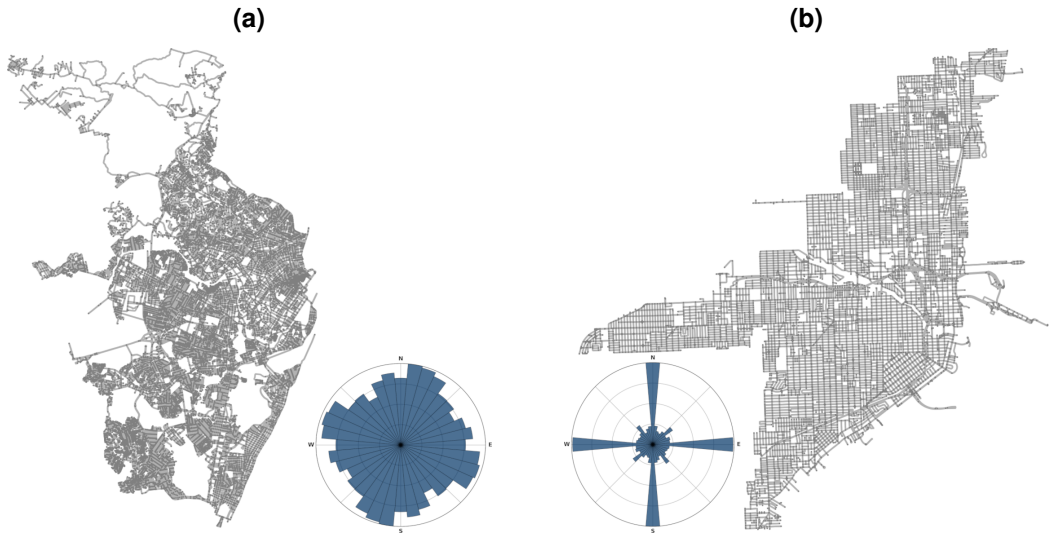
undefined. OSMnx *add_edge_speeds()* imputes free-flow travel speeds for the *speed_kph* attribute of edges, via the average maximum speed value of edges in a highway class. For a highway class with missing maximum speed values on all edges, it assigns the average of all maximum speed values of the graph. Because OSM has a high volume of data, it is expected that all the segments are labeled with high-quality maximum speed values information though. Finally, OSMnx *add_edge_travel_times()* calculates free-flow travel time along each edge, based on both *length* and *speed_kph* attributes. In this case, all edges must have both attributes and their values must be non-null. Actually, this feature is exactly what should be predicted in this work by a regression task.

Moreover, OSMnx retrieved information encompasses several useful graph global statistics on the selected road network, including the average circuity and the orientation-order, both proposed by Boeing (2019) and detailed in Appendix A. Figure 16 presents two road network examples and their respective edge orientation distribution in a polar plot, divided into 36 sections, whose bars represent the count of edges with that bearing orientation. It is possible to visualize how a "grid-like" street map affects the distribution of bars in the orientation plot.

4.1.2 Selected cities

This work selects relevant cities from all continents to have a globally representative sample of cities and to avoid the commonplace of restraining the analysis to the author's geo-

Figure 16 – City Road Network Graphs: (a) Recife, Brazil and (b) Miami, USA.



Source: Own work (2025).

graphic location. Twelve cities have been selected: the European city of Aalborg (the same city used in the experiments of Jepsen, Jensen and Nielsen (2020)); Brisbane in Australia; Nara in Japan; Manhattan island, Miami, and Seattle, all in the USA; Damascus in Syria; Cape Town and Johannesburg in South Africa; and three Brazilian cities of Curitiba, Niteroi, and Recife.

When an entire city map (i.e. a Road Network Graph including primal, dual graphs, and feature sets) does not fit in GPU memory, it is cropped, focusing on cities' central areas. Appendix A contains images detailing each city road network and its relevant edge feature information. The complete data set of all Road Network Graphs, on both primal and dual representation, along with node, edge, and between-edges features have been made publicly available[1].

For this work, a Road Network Graph is characterized by 12 global features as shown in Tables 1 and 2: number of nodes, number of edges, average node degree, average streets per node, average circuity, entropy, average and median edge length (meters), average and median free-flow travel time (seconds), and average and median legal speed (km/h). Moreover, a metric called Major Highway Class Rate[2] is used to represent the edge class distribution of a road network. Global features characterize a Road Network Graph but are not directly used for GNN training. Instead, they are used to search relationships between model performance and Road Network Graph global features.

[1] https://github.com/tcervi

[2] It measures how concentrated an edge class distribution is towards a specific class by computing the count ratio between the number of edges in the most common class and the total number of edges.

Table 1 – Road Network Graph global features by city

City	# Nodes	# Edges	Avg. node degree (k)	Avg. streets per node	Avg. circuitry (ς)	Entropy (ϕ)	Major Hwy class rate
Curitiba	23,815	60,866	5.112	2.954	1.025	3.501	0.736
Niterói	6,771	15,298	4.519	2.668	1.082	3.568	0.747
Recife	19,435	49,441	5.088	2.895	1.039	3.556	0.779
Manhattan	4,420	9,572	4.331	3.587	1.015	2.622	0.472
Miami	8,500	22,617	5.322	3.246	1.023	2.492	0.677
Seattle	19,059	50,265	5.275	3.046	1.028	2.779	0.705
Aalborg	14,187	31,305	4.413	2.386	1.063	3.508	0.700
Damascus	9,206	21,749	4.725	2.867	1.073	3.529	0.648
Cape Town	23,890	62,456	5.229	2.944	1.077	3.497	0.802
Johannesburg	32,635	78,718	4.824	2.873	1.057	3.555	0.758
Nara	13,069	34,187	5.232	2.836	1.088	3.356	0.821
Brisbane	18,496	43,472	4.701	2.851	1.047	3.315	0.628

Table 2 – Road Network Graph global features by city (cont.)

City	Avg. edge length (m)	Median edge length (m)	Avg. travel time (s)	Median travel time (s)	Avg. legal speed (km/h)	Median legal speed (km/h)
Curitiba	103.591	91.977	9.643	8.5	39.407	38.2
Niterói	114.285	77.525	10.923	7.5	38.752	35.9
Recife	84.483	63.329	9.468	7.1	32.739	29.8
Manhattan	116.672	82.224	10.284	7.5	40.202	40.2
Miami	108.547	91.557	8.653	7.3	45.875	42.5
Seattle	109.653	92.569	11.182	9.2	35.442	32.2
Aalborg	162.229	76.315	10.980	5.9	49.129	45.1
Damascus	102.515	68.518	8.594	6.0	42.941	38.1
Cape Town	104.995	74.459	6.392	4.5	59.4091	60.0
Johannesburg	122.028	81.836	7.427	5.0	58.9230	58.6
Nara	83.510	53.825	6.622	4.3	45.547	46.0
Brisbane	113.803	91.996	8.206	6.5	50.773	50.0

4.1.3 Road network graph sets of node, edge and between-edges features

Although OSM can have rich information on both the network nodes and edges, the size of the available feature vector might vary from city to city, and even within the same city. The present work defines a default setup for all considered Road Network Graphs to ensure that they all have the same set of node, edge, and between-edges features as shown in Table 3.

The node features are *x* and *y* geographical coordinates (latitude and longitude) of an intersection. The set of between-edges features contains *x* and *y* geographical coordinates of

Table 3 – Node, edge and between-edge features of an RGN

Node features	Edge features	Between-Edge features
x-coordinate latitude	lanes	x-coordinate
y-coordinate longitude	length	y-coordinate
	one_way	bearing
	class (one-hot(5))	turn_direction (one-hot(4))

the intersection node between edges, a one-hot-encoded vector for the *turn_direction* (*right*, *left*, *forward*, *backward*) and the *bearing* value representing the turn angle in degrees between edges. The set of edge features contains *lanes* representing how many lanes a street segment has, a *length* feature with the length in meters of that segment, a *one_way* boolean feature indicating if the street segment has one or two traffic ways and a one-hot-encoded vector for the edge classes.

Originally, urban road networks encompass 13 edge classes: *{"residential", "living_street", "primary", "primary_link", "secondary", "secondary_link", "tertiary", "tertiary_link", "unclassified", "trunk", "trunk_link", "motorway", "motorway_link"}*. Details on each edge class meaning can be found in Appendix A. However, an unbalanced distribution of edge classes is expected in urban road networks. An urban scenario may have few or no “motorway” and “trunk” road segments because they are classes representing high-performance roads that connect two or more different cities (i.e., inter-cities) and usually only outline or cross an urban network (i.e., intra-city). Major arterial roads in a city are usually connecting different regions of the city. Therefore, edges classified as “primary” and “secondary” are expected, but on a very lower count when compared to “residential” or “living_street” edges. The same applies to “tertiary” and “unclassified” edges, although minor public roads are expected to appear in higher count than major arterial roads.

Figure 17 shows the edge class distribution in the Aalborg Road Network Graph. Aalborg is not a densely concentrated city on its territory according to Figure 17 (a). Indeed, it is considerably dispersed with a high presence of arterial roads in its network. Figure 17 (b) presents a very unbalanced class distribution towards residential and minor public road classes.

Figure 17 – Road Network Graph: (a) Aalborg RNG and (b) edge class distribution.

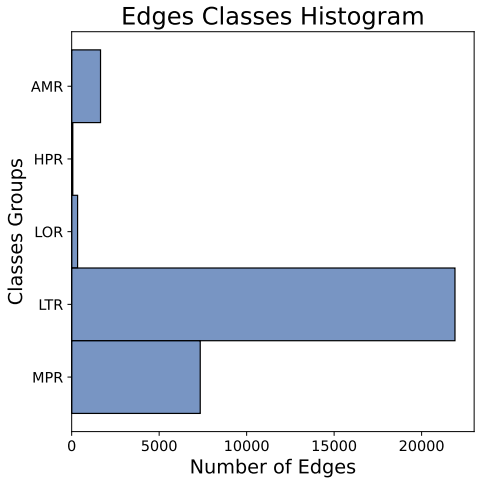
Source: Own work (2025).

This work does not address the natural unbalanced distribution of classes in Road Network Graphs. On the contrary, the experiments test GNN architectures and evaluate their performance for these unbalanced datasets. However, the unbalanced distribution of 13 different classes might jeopardize the learning process. Therefore, we semantically group different edge classes in order to reduce the number of features as in Table 4 (although unbalance still remains as shown in Figure 18).

Table 4 – Edge classes grouping

Class Group	Acronym	Grouped Classes
Highest Performance Roads	HPR	*"motorway"*; *"trunk"*
Arterial Main Roads	AMR	*"primary"*; *"secondary"*
Minor Public Roads	MPR	*"unclassified"*; *"tertiary"*
Local Traffic Roads	LTR	*"living_street"*; *"residential"*
Link Only Roads	LOR	*"primary_link"*; *"secondary_link"*; *"tertiary_link"*; *"trunk_link"*; *"motorway_link"*

Figure 18 – Edge class distribution of Aalborg after grouping classes

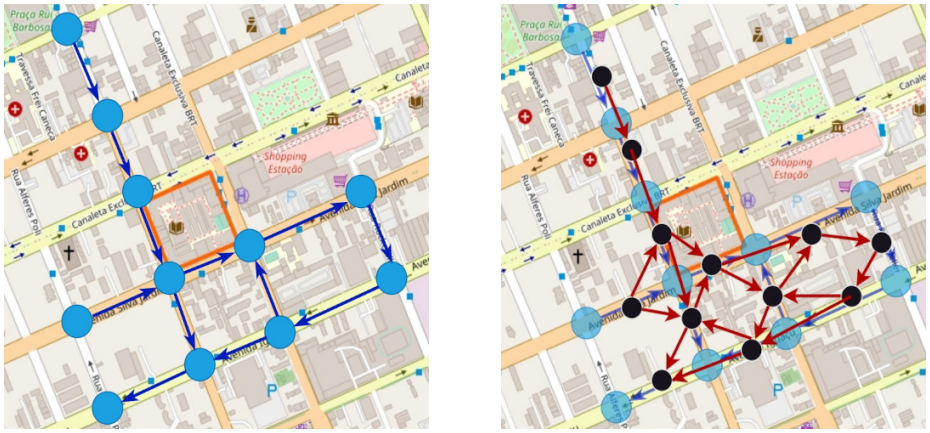


Source: Own work (2025).

In the present work, the final Road Network Graph for both representations can be described by the same model used by Jepsen, Jensen and Nielsen (2020). The primal representation $\mathcal{G}^{\mathcal{P}} = (\mathcal{V},\mathcal{E})$ of an RNG is defined by a set of node features $X^{\mathcal{V}} \in \mathbb{R}^{|V|\times d^{\mathcal{V}}}$, a set of edge features $X^{\mathcal{E}} \in \mathbb{R}^{|E|\times d^{\mathcal{E}}}$ and a set of between-edge features $X^{\mathcal{B}} \in \mathbb{R}^{|B|\times d^{\mathcal{B}}}$ with dual representation $\mathcal{G}^{\mathcal{D}} = (\mathcal{E},\mathcal{B})$. Figure 19 shows an example of an addressed Road Network Graph on its primal (a) and dual (b) representations.

Figure 19 – (a) Primal and (b) Dual Road Network Graph representations of UTFPR (Brazil) surroundings.

(a) **(b)**



Source: Own work (2025).

Samples of primal node and dual edge feature matrices of a city's Road Network Graph can be found in Table 5 and Table 6, respectively. The matrix of features is normalized using the min-max strategy to ensure that the training values are controlled.

Table 5 – Example of a primal Road Network Graph matrix of node features (raw and normalized)

Node ID	Primal Road Network Graph Node Features (Raw data)	
	Latitude (X)	Longitude (Y)
0	−25.431946	−49.240482
1	−25.430780	−49.240360
...	...	...
1925	−25.440885	−49.247937
1926	−25.426805	−49.264894
Node ID	**Primal Road Network Graph Node Features (Normalized data)**	
	Latitude (X)	**Longitude (Y)**
0	0.57097299	0.99645865
1	0.60611587	0.9985922
...	...	...
1925	0.30153123	0.86608804
1926	0.72594345	0.56953985

Table 6 – Example of a dual Road Network Graph matrix of edge features (raw and normalized)

Edge ID	Dual Road Network Graph edge features (Raw data)						
	Latitude (X)	Longitude (Y)	Turn angle (degrees)	Turn direction (encoded)			
0	−25.430256	−49.243278	92.2	0	1	0	0
1	−25.432489	−49.240560	116.6	0	1	0	0
...	...	...	...	...	...	...	...
9213	−25.441254	−49.236597	180.0	0	1	0	0
9214	−25.426156	−49.222144	180.0	0	1	0	0
Edge ID	**Dual Road Network Graph Edge Features (Normalized data)**						
	Latitude (X)	**Longitude (Y)**	**Turn angle (degrees)**	**Turn direction (encoded)**			
0	0.69677038	0.84767126	0.28343068	0	1	0	0
1	0.43360503	0.87563569	0.35843836	0	1	0	0
...	...	...	...	...	...	...	...
9213	0.65635343	0.63224805	0.55333538	0	1	0	0
9214	0.1645586	0.70625364	0.55333538	0	1	0	0

Both primal and dual representations of a city's Road Network Graph are stored as the PyG data format from PyTorch Geometric, a library licensed under the MIT license and built upon PyTorch[3] to easily write and train GNNs for a wide range of applications. A PyG data object contains at least the following properties: *x*, storing the graph node features tensor; *edge_index*, storing the graph adjacency list; *edge_attr*, storing the graph edge features tensor; *y*, storing the labels for classification tasks, or features values for prediction tasks. Both the primal and dual feature matrices are normalized before being stored on their respective PyG Data object.

[3] PyTorch is an optimized Python tensor library for deep learning using GPUs and CPUs, licensed under the Caffe2 license, which can be easily extended by other Python packages (e.g. NumPy, SciPy, Cython) when needed.

4.2 Experimental setup

This work conducts a set of experiments aiming to explore GNN representation learning on Road Network Graphs, specifically for edge regression tasks, considering the research questions presented in Table 7.

Table 7 – Performed experiments

Name	Research question	Justification
Baseline	Are GNNs a good fit for Road Network Graph edge regression tasks?	According to JEPSEN; JENSEN; NIELSEN and GHARAEE *et al.* results, only GNNs specifically designed for RNG task will outperform MLPs on the RNG edge regression task.
Mini-batch	How does mini-batch in gradient descent algorithm influence GNNs' performance?	The stochastic nature of mini-batching might reduce the training time and reduce the training set over-fitting.
RFN extrapolation	How well RFNs extrapolate to new Road Network Graphs?	GNNs specifically designed for Road Network Graphs should learn representations that extrapolate to other/similar cities.

4.2.1 Addressed graph neural network architectures

This work explores the application of spatial node-level GNNs on the Road Network Graph representation learning problem, using the dual (i.e., line transformed) graph representation as input data, except for the RFN architecture (Section 3.2). RFNs are an spatial relational-level GNN operating on both primal and dual graph representations. All GNNs explored in this work are designed for inductive learning. They have been detailed in Chapter 3 and their implementation codes, used in the experiments, have been made available by the original authors. The complete list of architectures can be found in Table 8.

GraphSAGE-GCN, GraphSAGE-Mean, GraphSAGE-LSTM, and GraphSAGE-Pool are variants of the GraphSAGE[4] framework, using the graph convolutional aggregator, mean based aggregator, LSTM based aggregator, and max-pooling based aggregator, respectively. GAT-H3 and GAT-H6 are GAT[5] architectures with three and six attention-headers, respectively. GIN-E and GIN-0 are GIN[6] architectures with learnable parameter ε, and $\varepsilon = 0$, respectively. RFN-AI, RFN-NAI, RFN-AA and RFN-NAA are RFNs variants[7] with attentional aggregation and interacional

[4] https://github.com/williamleif/GraphSAGE/
[5] https://github.com/PetarV-/GAT/
[6] https://github.com/weihua916/powerful-gnns
[7] https://github.com/TobiasSkovgaardJepsen/relational-fusion-networks

Table 8 – Models addressed in this work

Type	GNN	Characteristics
General	GraphSAGE-GCN	GCN aggregator
	GraphSAGE-MEAN	Mean aggregator
	GraphSAGE-LSTM	LSTM aggregator
	GraphSAGE-POOL	Max-Pooling aggregator
	GAT-H3	3 Attention headers
	GAT-H6	6 Attention headers
	GIN-0	SUM aggregator with $\varepsilon = 0$
	GIN-ε	SUM aggregator with learnable ε
Road Network Graph specific	RFN-AI	Interactional fusion; Attentional aggregator
	RFN-NAI	Interactional fusion; Non-attentional aggregator
	RFN-AA	Additive fusion; Attentional aggregator
	RFN-NAA	Additive fusion; Non-attentional aggregator

fusion, non-attentional aggregation and interacional fusion, attentional aggregation and additive fusion, and non-attentional aggregation and additive fusion functions, respectively.

According to the results obtained by Jepsen, Jensen and Nielsen (2020) and Gharaee *et al.* (2021), in road network-related tasks, only specifically designed GNNs can outperform MLPs. Consequently, an MLP trained with the road network features data is considered the baseline prediction model for comparison. It uses only edge features to learn a regression model for the desired output and therefore, disregards all structural information found on a graph dataset.

4.2.2 Training setup and performance metrics

A complete description of the test framework can be found in Section 5, but it is important to highlight common configurations shared by all models during the experimental phase. For any GNN training, regardless of the selected city, the input (Road Network Graph data) is split into five disjoint folds (each one with 20% of data) for cross-validation. Each model is trained once with 80% of the road network edges (four folds) as training data and 20% (one fold) as testing data. The testing fold is then changed to another fold with the remaining folds as training data in a second training round. Therefore, five models are obtained (one for each round). Training data is additionally randomly split into training (60%) and validation (20%) subsets unless otherwise specified.

During all experiments training phases, models' weights are optimized using the Adam method. It is an algorithm for first-order gradient-based optimization of stochastic objective functions known to be well suited for problems with many data or parameters (KINGMA; BA, 2014). A learning rate of 0.001 and a weight decay of 0.0001 are always the hyper-parameters chosen for the Adam optimizer. The loss function on the validation set is constantly checked. For every epoch in which the current validation loss is lower than the current best value, these model's parameters are saved, overriding any previously saved parameters, and the best validation value is

also updated accordingly. Finally, the last version of the model's parameters (i.e., after all training epochs) is also saved.

Given that the regression task is the prediction of free-flow travel time along each primal RNG edge, all GNNs are assembled in such a way that regardless of their layers count, the last layer has an output dimension of one. This way, the output of all GNNs is expected to have a free-flow travel time estimated as a single value for each primal Road Network Graph edge.

For all explored architectures, linear weights are initialized using Glorot uniform initialization. GraphSAGE-Pool and GraphSAGE-LSTM extra weights, as well as all GAT attention weights and RFN weights, are initialized using Xavier initialization. All activation functions used between layers are ReLU functions, except for the RFN input layer which adopts the ELU function as in Jepsen, Jensen and Nielsen (2020) work. Also in the original paper, RFNs are designed with an $L2$-normalization in place on hidden layers and no normalization on the final (i.e., output) layer. So, the same is applied for the MLP and general-purpose GNNs hidden and output layers.

Training a graph neural network, like any other neural network, is an optimization problem aimed at finding network weights and biases that reduce the value of the loss function evaluated on a training sample. In Section 5.1, experiments use the entire training set (i.e., batch gradient descent - BGD) for calculating the loss function during training sessions (GOODFELLOW; BENGIO; COURVILLE, 2016). Road Network Graph related problems though, especially when handling cities with high populations (e.g., megalopolis), can result in graph neural networks being trained with large datasets. When training a GNN, the whole training set is loaded into memory, which can make batch training with bigger RNGs very expensive, and sometimes not feasible, in terms of computational memory.

The opposite option is the stochastic gradient descent, SGD, which presents a single example per optimization step while training (GOODFELLOW; BENGIO; COURVILLE, 2016). It might also not be desirable though, due to the consequent increase in training time, result of the size of nodes/edges scale in bigger Road Network Graphs training sets. With SGD the randomness on the choice of single examples can lead to updates with higher variance, making the optimization process fluctuate significantly. The sequential nature of SGD's updates can lead to inefficient use of modern hardware designed for parallel processing and capable of performing multiple calculations simultaneously, by forcing it to spend a considerable amount of time waiting for the next training example to arrive.

Mini-batch Gradient Descent is a trade-off between stochastic and batch techniques. Instead of using a single training sample per iteration, it calculates the loss function average over a small number of samples, called mini-batch (GOODFELLOW; BENGIO; COURVILLE, 2016). This averaging process reduces update variance compared to the SGD approach. As a result, it tends to provide more stable optimization, with smoother convergence during the training of GNNs. Additionally, it updates the model's parameters more frequently than BGD, which can result in faster training convergence. It also efficiently utilizes the parallel processing capabilities of the hardware, ensuring that multiple (i.e., mini-batch size) calculations can be

performed simultaneously, maximizing computational resource usage and accelerating training times.

Mini-batch Gradient Descent is explored in detail as part of the experiments of Section 5.2, not only to find a good balance between resource usage and computational time but especially to understand the role of the mini-batch size, a training parameter, on the learning capacity of graph neural networks when applied to Road Network Graph related tasks. Different batch sizes are considered and their effects on the training of models are evaluated. Section 5.2 results support the choice of the mini-batch size used in Section 5.3. Details on mini-batching can be found in Appendix D.

Training loss function or inference performance metrics are, throughout this work, calculated by three different error measures: the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or Mean Absolute Percentage Error (MAPE) (ANTHONY; BARTLETT, 1999).

The Mean Absolute Error (MAE) is defined by Equation 33.

$$\mathbf{MAE} = \frac{1}{n} \cdot \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{33}$$

where y_i and $\hat{y}_i$ are the actual and predicted travel times of n values, respectively.

The Root Mean Squared Error (RMSE) is defined by Equation 34.

$$\mathbf{RMSE} = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{34}$$

Due to a quadratic form, the gradient descent has a single global minimum.

The Mean Absolute Percentage Error (MAPE), also known as mean absolute percentage deviation (MAPD), is a normalized error (scale independent). It is suitable for prediction evaluation when predicted values vary on different scales. It can be expressed by Equation 35.

$$\mathbf{MAPE} = \frac{100}{n} \cdot \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{35}$$

However, near-zero actual travel times can deteriorate MAPE value.

The loss function can have a significant impact on the model's regression performance, depending on the domain of the problem and it is important to perform an exploratory analysis on the effect of different choices of loss function for the problem presented in this work.

For Road Network Graph datasets with varying edge lengths and driving speeds, it is expected that free-flow travel times can also vary too much. In this case, MAE and RMSE might become poor evaluation metrics. For example, a prediction absolute error of 2 seconds is relevant if the target value is 5 seconds (40%) but is almost irrelevant (2%) if the target value is

100 seconds. A discussion on the influence of those functions is detailed in the experiments of Section 5.1.

5 EXPERIMENTS AND RESULTS

The experimental framework used for this work has been designed to ensure that the same GNN structural assembling and training routines are shared between all the evaluated models, providing classes that can be parameterized to change only the GNN layer architecture when needed. Source code is publicly available[1].

All experiments have been executed in a PC with the following specs: 12th Gen Intel(R) Core(TM) i7-12700F Processor with 12 cores (2 Threads per core), CPU max frequency of 4900.0000 MHz and minimum frequency of 800.000 MHz; 32GB RAM; NVIDIA GeForce RTX 3060 Ti Graphic Card with 8 GB GDDR6 memory available and 4864 CUDA cores.

The performed experiments have been designed to find to find answers to this work's research questions presented in Chapter 1. Section 5.1 compares different GNN models on the addressed edge regression task, using an MLP as the comparison baseline. Section 5.2 explores various batch sizes in the mini-batch gradient descent training algorithm. Section 5.3 describes the results obtained from RFN variants when learning from each one of the twelve selected cities and extrapolating learned knowledge to the remaining cities.

5.1 Are GNNs a good fit for Road Network Graph edge regression tasks?

Experiments in this section aim to evaluate the selected GNN architectures (Section 4.2) on their representation learning capacities when trained with Road Network Graphs. The evaluation criterion is the edge free-flow travel time (in seconds) prediction performance. The objective here is to verify whether only specifically designed GNNs can outperform baseline models, like MLPs, as per the results of (JEPSEN; JENSEN; NIELSEN, 2020) and (GHARAEE *et al.*, 2021).

5.1.1 Data

The chosen input road network for the experiments performed in this section is Aalborg City in Denmark (Figure 20), the same used in (JEPSEN; JENSEN; NIELSEN, 2020). The used Road Network Graph is similar to the one used in the reference work[2]. Its primal representation has a total of 14187 nodes and 31305 edges, while its dual representation has 31305 nodes and 82600 edges. The set of features is the default for the assembled Road Network Graph dataset (see Section 4.1.3 for details):

[1] https://github.com/tcervi

[2] Aalborg's network in (JEPSEN; JENSEN; NIELSEN, 2020) has 16294 nodes, 35947 edges and 94718 between-edges, with 3 features per node (city zones), 16 features per edge (length, 3 origin and 3 destination node features and 9 possible edge classes) and 5 between-edge features (1 turning angle and 4 turn directions).

- 2 Node features: x (latitude) and y (longitude) coordinates of the intersection;
- 8 Edge features: length (meters), number of lanes, one-way (boolean flag) and a vector encoding 5 possible edge classes;
- 7 Between-edge features: x (latitude) and y (longitude) coordinates of the connecting node, turning angle and a vector encoding 4 possible turn directions;

Figure 20 – Aalborg's (Denmark) road network graph.

Source: Own work (2025).

The present work targets the free-flow travel time regression, using information retrieved directly from OSM (Section 4.1.3) as the travel time ground truth values, while Jepsen, Jensen and Nielsen (2020) use an average of historical GPS data from a private dataset on their work.

In Aalborg's dataset, edge lengths (in meters) and travel times (in seconds) distributions are represented by a peak of samples of small values, followed by a long tail that reaches bigger values (Appendix A - Figure 31 (c) and (d)). That is, both characteristics appear on a scale with a large width. There is also a clear unbalanced distribution (Appendix A - Figure 31 (e) and (f)) of legal speed (in kilometers per hour) and edge classes (see Section 4.1.3). This is another characteristic of this dataset that is expected for cities, where most of the road segments would belong to residential areas, connected by fewer expressways, eventually surrounded by motorways.

5.1.2 Training with different loss functions

Although this experiment aims to evaluate the selected GNN architectures on the edge free-flow travel time (in seconds) regression performance, multiple training loss function are evaluated for each model. Jepsen, Jensen and Nielsen (2020) use the Mean Absolute Error (MAE), in seconds, however, the present work explores the effects of different loss functions on the

gradient-based learning, using first the same metric (MAE), then the Root Mean Square Error (RMSE) and later the Mean Absolute Percentage Error (MAPE).

Aiming to ensure a fair comparison between the final validation and test results, all training processes described in this section share the same experimental setup, summarized in Table 9.

Table 9 – Experimental Parameters Setup

Parameter	Value	Parameter	Value
Layers	2	Hidden Size	16
Input Size	Feature vector size[3]	Output Size	one[4]
Activation Function	ReLU[5]	Training Epochs	1000
Learning Rate	0.001	Weight Decay	0.0001
Optimizer	Adam[6]	Cross-Validation	5-fold
Training Set	60%	# Batches	one[7]
Validation Set	20%	Test Set	20%

The MLP and general-purpose GNNs are configured with 2 layers each: the input layer with an input dimension of 15 (i.e., number of edges features) and and output dimension of 16 (i.e., hidden layer dimension); and the output layer with an input dimension of 16 and an output dimension of 1. RFNs are also configured with 2 layers each, with the exception that such models operate on both the primal and the graph representations simultaneously, therefore the input dimension may vary: 2 for node (i.e, primal) features, 15 for edge (i.e., dual) features and 7 for between-edge features (i.e., the single feed-forward transformation included on each RFN layer according to Section 3.2).

Aalborg's RNG dataset is split into 5 disjoint sub-groups of edges and a 5-fold cross-validation procedure is adopted to mitigate the data partition bias. The same 5-fold separation described in Chapter 4 is used to train all models, including the same training, validation and test samples, such that there is a common base for comparison between models. There is no batching division in this experiment, so the full dataset is always fed to the training routine, naturally respecting the Test, Validation and Training divisions.

For all models, a total of 1000 epochs of training is executed and for each epoch, the current loss function is evaluated over the validation set. At the end of each training session, the best model (i.e., the model with the lowest loss on the validation set) and the last model (i.e., the model after 1000 epochs) are saved to predict the free-flow travel time for each of the five test samples. Since this experiment uses 5-fold cross-validation, the presented results are the averages of five tests.

[3] Node, edge, or between-edge.
[4] Only the edge representation output is considered since the experiment explores edge regression.
[5] Except for the RFN input layer, designed with the ELU function (JEPSEN; JENSEN; NIELSEN, 2020).
[6] See (KINGMA; BA, 2014)
[7] A single batch containing the entire training set is used.

Table 10 – Trained Models Information

Model	Avg. Training Time (seconds)	Number of Parameters
MLP	$2.9_{0.32}$	161
SAGE-MEAN	$4.8_{0.3}$	305
SAGE-GCN	$4.4_{0.1}$	161
SAGE-POOL	$5.2_{0.1}$	649
SAGE-LSTM	$20.1_{0.4}$	3057
GAT-H3	$6.8_{0.2}$	585
GAT-H6	$12.4_{0.1}$	1170
GIN-0	$4.9_{0.2}$	873
GIN-E	$4.6_{0.2}$	873
RFN-AI	$266.5_{0.5}$	5783
RFN-NAI	$262.5_{0.1}$	5682
RFN-AA	$259.0_{0.3}$	918
RFN-NAA	$254.8_{0.2}$	817

Table 10 summarizes the baseline and GNN's total (i.e., for 1000 epochs) training time (in seconds) and the total number of learned parameters. It is important to highlight that in this work models are compared based on testing performance after being trained for the same number of epochs[8], regardless of the time required for computing that amount of epochs. Different models might have different numbers of learnable parameters and, therefore, might require more computational time for each optimization step. SAGE-GCN, for example, has the same number of parameters as the MLP. However, as it uses the message-passing mechanism, the training time is higher than a simple MLP which completely discards the graph structure during training.

Appendix B details the learning curves of each one of the explored models. Figure 21 (a) presents all models' learning curves when using MAE as the loss function. Results show that after 1000 epochs of training with this function, the baseline model - MLP - presents a final validation MAE of 8.26, reducing its initial loss value by about 25%. Both GIN variants are unable to overcome the baseline MLP validation performance, reducing the initial loss only to about 3% and 12% before converging to values of 9.27 for GIN-E and 10.44 for GIN-0. Compared to the baseline, GIN models perform worse on the validation set. The remaining general-purpose GNNs converge to similar values, differing mainly on the curve shape and the convergence epoch. GAT-H3, GraphSAGE-Pool and GraphSAGE-LSTM present a final validation MAE of 7.90 while, while GAT-6 presents a final validation MAE of 7.92 and both the GraphSAGE-GCN and GraphSAGE-Mean present a final validation MAE of 8.02.

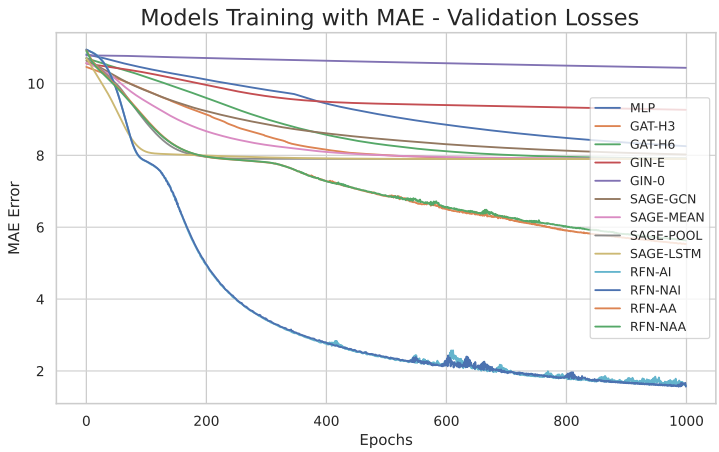
On the RFNs results, both models with additive fusion present similar performance of 5.53 (RFN-AA) and 5.63 (RFN-NAA), both about 32% better than the MLP baseline and 29% better than the majority of the explored general purpose GNNs (disregarding the GIN models). Additive RFN learning curves do not present a convergence pattern within 1000 epochs, yet they show a fast loss decrease during the first 200 epochs, and then a step followed by a slower loss decrease persists for the epochs. RFNs with interactional fusion (i.e., RFN-AI and RFN-NAI),

[8] i.e., after the same amount of gradient optimization steps.

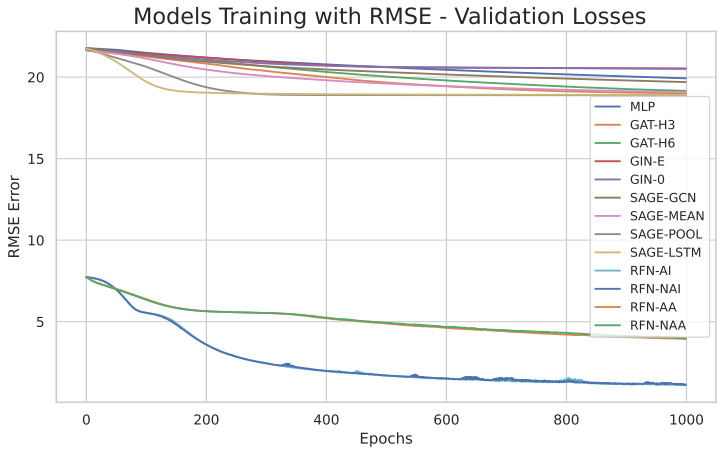
on the other hand, have the best MAE performances of 1.55, about 81% better than the MLP baseline and 80% better than the general purpose GNNs performance.

Figure 21 – Training with different loss functions: (a) MAE, (b) RMSE and (c) MAPE

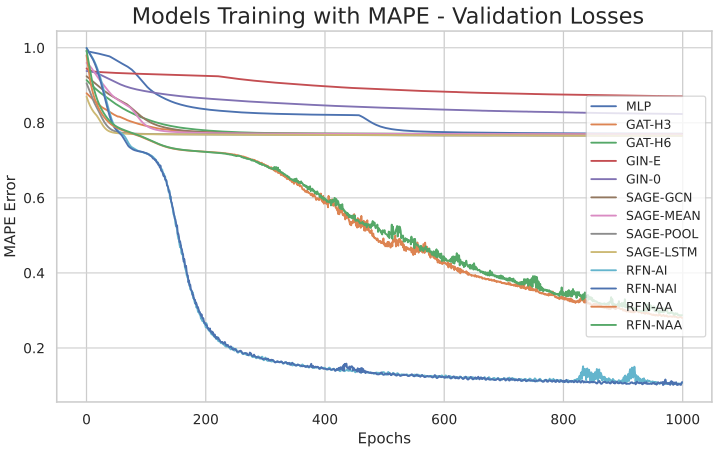
(a)



(b)



(c)



Source: Own work (2025).

Figure 21 (b) shows all models' training curves when using RMSE as the loss function. When training with RMSE none of the general-purpose GNNs can relevantly outperform the MLP baseline final validation result of 19.93. Moreover, the difference between those models' final validation values is even smaller. Again, both GIN variations have the worst performance with an RMSE value of 20.53 for GIN-E and 20.51 for GIN-0, 3% worse than baseline.

Results show a relevant difference in learning power during the initial training epochs. It is important to highlight that neither of the two additive fusion variations when trained with the loss function, converges within 1000 training epochs. The final validation value and the learning curves differ a lot from their interactional counterpart. Although additive RFNs are still relevantly better on this regression task than the baseline MLP and the general purpose GNNs, the difference to its interactional counterparts might be indicative that for this task, with this urban road network, only summarizing relationships (i.e., additive fusion) is desirable but modeling interactions (i.e., interactional fusion) can increase even more the learning power.

Although no relevant performance gains are verified when using general purpose GNNs instead of a simple MLP with RMSE as the loss function, Figure 21 (b) pictures a result that this function accentuates the regression performance differences between RFNs and other models. On the very first training epoch, all RFNs have already an RMSE validation value lower than 10, while baseline and general purpose GNNs have an RMSE value higher than 20 for the same epoch. Again, there is no verified performance enhancement when using attentional aggregators. And finally, the interactional RFNs are still the best performant models with RFN-AI converging to an RMSE value of 1.07 and RFN-NAI converging to a value of 1.10, both 95% better than the baseline, while RFN-AA converges to an RMSE value of 3.93 and RFN-NAA converges to a value 3.99, both 80% better than the baseline.

Finally, Figure 21 (c) illustrates all models' training curves when using MAPE as the loss function. Learning curves have similar shapes to the ones presented when MAE is used as the loss function (Figure 21 (a)). It is important to highlight though, that when training with MAPE none of the general-purpose GNNs can outperform the MLP baseline final validation MAPE (around 0.77) significantly. With MAPE, both GIN variations are still the worst-performing models. GIN-E converges to a MAPE value of 0.87, 13% worse than the baseline, while GIN-0 converges to a MAPE value of 0.82, 6% worse than the baseline. Compared to the baseline, both GAT and all four GraphSAGE variants converge to similar values, differing only on the curve shape and the convergence epoch. No relevant performance gains, when using general-purpose GNNs instead of a simple MLP, occur when MAPE is used as the training loss function.

RFN-AA converges to a MAPE value of 0.28, 64% better than the baseline, while RFN-NAA converges to a value of 0.29, 62% better than the baseline. The interactional RFNs are again the best models, converging to MAPE values of about 0.10, 87% better than the baseline. Trained with MAPE, interactional RFNs also present a learning curve with accentuated gains between epochs 100 and 200, when the validation MAPE value drops from 0.7 to 0.2.

When comparing the different GNNs learning curves against the MLP baseline, results presented in this section show that even when the final validation performed is not relevantly enhanced, the use of GNNs introduces learning patterns that differ from the simple baseline. GraphSAGE-GCN and GraphSAGE-Mean present a faster learning curve, especially when trained with MAPE. GraphSAGE-Pool and GraphSAGE-LSTM present even faster learning curves, but also are the models that present over-fitting patterns when trained with RMSE and MAPE. Between both GAT-H3 and GAT-H6 models, the increase in the number of attention headers does not enhance the learning power of those models. Among the general-purpose GNNs selected for this experiment, GIN-0 and GIN-E have the worst validation performance, with learning curves that are even slower than the baseline.

When analyzing RFNs, it is possible to notice a more significant increase in the steepness of the learning curve decrease. More than that, none of the explored RFNs present over-fitting patterns. The RNF's attentional aggregator, proposed by Jepsen, Jensen and Nielsen (2020), does not significantly enhance the learning behavior of RFNs on Aalborg's Road Network Graph version explored in this work. Also, the additive fusion-based RFNs present a slighter steepness on the learning curves compared to their interactional counterparts, probably due to the difference in both numbers of learnable weights and biases.

Table 11 – Average Aalborg's training errors with standard deviation as subscript

Model	Loss Function		
	MAE	**RMSE**	**MAPE**
MLP	$8.26_{0.32}$	$19.93_{0.85}$	$0.771_{0.006}$
SAGE-GCN	$8.02_{0.30}$	$19.70_{0.77}$	$0.770_{0.006}$
SAGE-MEAN	$8.02_{0.30}$	$19.70_{0.77}$	$0.770_{0.006}$
SAGE-POOL	$7.90_{0.27}$	$18.88_{0.72}$	$0.765_{0.005}$
SAGE-LSTM	$7.90_{0.27}$	$18.91_{0.73}$	$0.765_{0.008}$
GAT-H3	$7.90_{0.27}$	$19.00_{0.77}$	$0.767_{0.006}$
GAT-H6	$7.92_{0.28}$	$19.16_{0.80}$	$0.768_{0.006}$
GIN-0	$10.44_{0.78}$	$20.51_{1.66}$	$0.823_{0.100}$
GIN-E	$9.27_{1.51}$	$20.53_{0.91}$	$0.870_{0.119}$
RFN-AI	$\mathbf{1.55_{0.09}}$	$\mathbf{1.07_{0.06}}$	$\mathbf{0.102_{0.004}}$
RFN-NAI	$\mathbf{1.55_{0.07}}$	$\mathbf{1.10_{0.04}}$	$\mathbf{0.099_{0.004}}$
RFN-AA	$5.53_{0.27}$	$3.93_{0.17}$	$0.279_{0.005}$
RFN-NAA	$5.63_{0.24}$	$3.99_{0.18}$	$0.285_{0.003}$

Table 11 summarizes the training information. The conclusion of this training and validation analysis states that, regardless of the chosen loss function, general-purpose GNNs do not outperform a simple MLP on this free-flow travel time regression task.

This is initially not expected for GNNs[9] but aligns with results provided by Jepsen, Jensen and Nielsen (2020). Small validation gains of general purpose GNNs do not justify the increase in the number of model parameters and training time (Table 10). When comparing the RFN variants, it is clear that RFNs outperform not only the simple MLP, but also all the general purpose GNN variants on this edge regression task. Despite the big increase in the number of parameters and training time (Table 10), the results point out that RFNs (especially the variants with interaction fusion) are the better-suited models for this edge regression task.

Additionally, for all loss functions, the role of different fusion functions is clear and still aligned with Jepsen, Jensen and Nielsen (2020). The original paper and the present work's results differences are mainly on the scale of the performance enhancements and the scale of the difference between interactional and additive models. RFNs with additive fusion functions have the worst performance among the RFN variants and the attentional aggregation does not show any relevant help on the models' regression performance. This might indicate that the low-density of Aalborg's road network (i.e., small node average degree) does not play a role as important as the volatile homophily. In other words, outlier neighbors do not affect the training process as much as the sharp boundaries between homophilic regions of the Road Network Graph. The next section explores the results of predicting free-flow travel times on the test sets, i.e. sets that are not used to measure the regression loss during the training process.

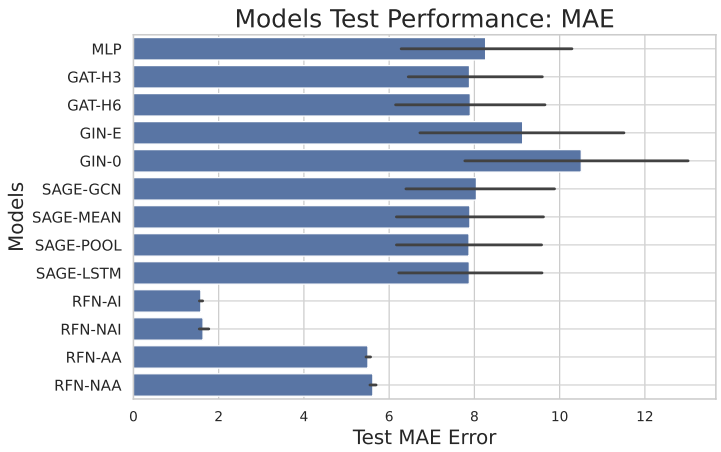
5.1.3 Testing the trained models

The results for the free-flow travel time regression performance on test sets are summarized in Figure 22, where each model is represented by a blue bar corresponding to its average regression performance, and black lines represent the 95% confidence interval. Figure 22 (a) shows that, except for the GIN model, the baseline GNNs are on average 5% better than the MLP baseline when trained and evaluated using the Mean Absolute Error (MAE), the same cost function used as evaluation criteria in Jepsen, Jensen and Nielsen (2020).

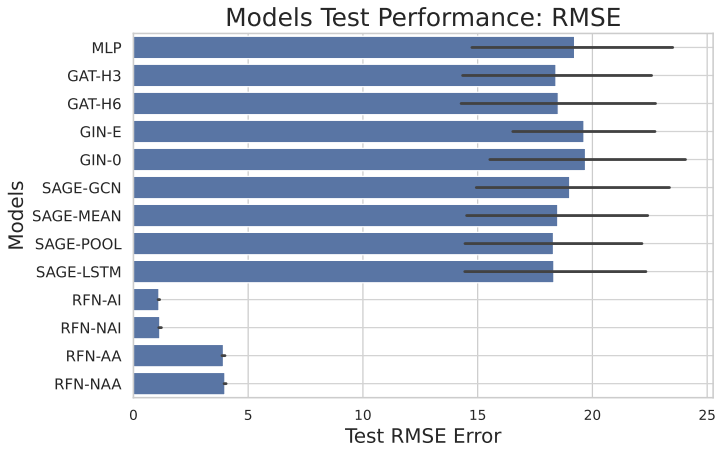
[9] GNNs are models specifically designed to leverage structural information from graph datasets and, therefore, should outperform models that entirely disregard that information

Figure 22 – Test results for models trained with different loss functions: (a) MAE, (b) RMSE and (c) MAPE

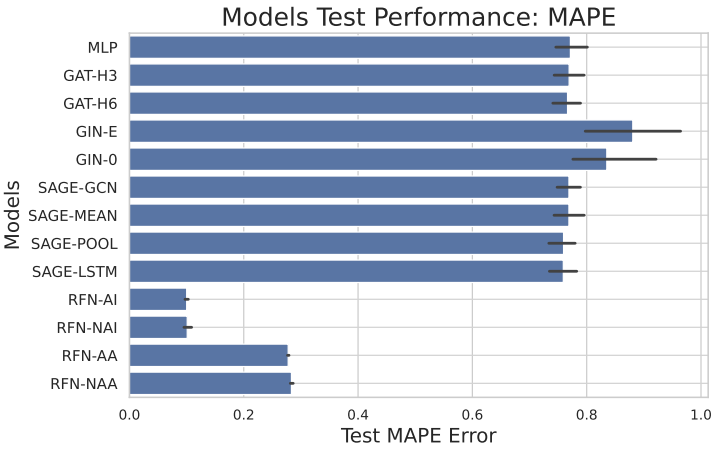
(a)



(b)



(c)



Source: Own work (2025).

Figure 22 also shows that RFNs are the best models for this road network task, but in scales that differ from Jepsen, Jensen and Nielsen (2020) original work. For the original authors, additive RFNs present 27% of performance enhancement when compared to the MLP baseline, while this work results show 32% of enhancement, and interactional RFNs present 33% of performance enhancement, while this work results show 81% of regression performance enhancement. The main difference between both results is the performance gap between additive and interactional RFNs. There are two possible reasons to explain it: the small differences in the road network modeling (Section 5.1.1) or the possible differences in the target values (i.e., travel-time data) used. Finally, it is also possible to verify that RFNs are not only better in average performance but also that their results with 95% confidence intervals are narrower than the general purpose and baseline intervals.

Table 12 summarizes the baseline and GNN test performance - i.e., regression performance on the test - for all the three used loss functions: MAE, RMSE and MAPE. Models are trained with a single loss function and tested for all three functions. Bold values in each column highlight the best performance.

Table 12 – Average Aalborg's testing errors with stardard deviation as subscript

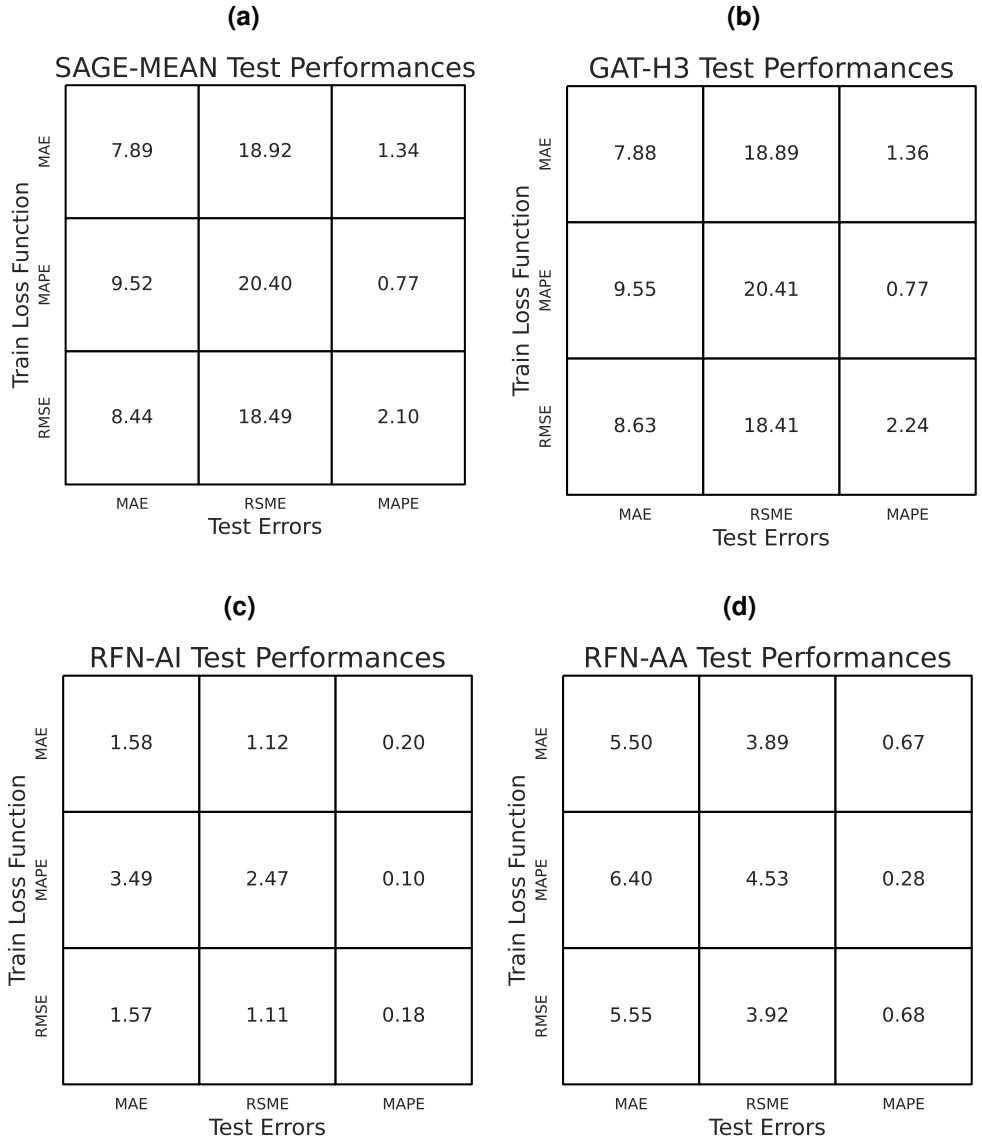
Model	Training with MAE			Training with RMSE			Training with MAPE		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
MLP	$8.26_{2.55}$	$19.50_{5.74}$	$0.971_{0.365}$	$8.02_{2.44}$	$19.22_{5.70}$	$1.102_{0.432}$	$9.58_{2.87}$	$20.43_{5.77}$	$0.772_{0.035}$
SAGE-GCN	$8.05_{2.35}$	$19.22_{5.66}$	$1.149_{0.559}$	$7.95_{2.20}$	$19.01_{5.55}$	$1.300_{0.672}$	$9.58_{2.84}$	$20.44_{5.75}$	$0.769_{0.027}$
SAGE-MEAN	$7.89_{2.21}$	$18.92_{5.57}$	$1.340_{0.630}$	$8.44_{1.71}$	$18.49_{5.28}$	$2.101_{1.235}$	$9.52_{2.87}$	$20.40_{5.77}$	$0.769_{0.038}$
SAGE-POOL	$7.87_{2.19}$	$18.86_{5.55}$	$1.372_{0.614}$	$9.40_{1.57}$	$18.30_{4.91}$	$2.738_{1.247}$	$9.47_{2.85}$	$20.37_{5.76}$	$0.760_{0.030}$
SAGE-LSTM	$7.88_{2.19}$	$18.87_{5.56}$	$1.370_{0.622}$	$9.37_{1.59}$	$18.32_{4.95}$	$2.710_{1.194}$	$9.47_{2.83}$	$20.36_{5.75}$	$0.759_{0.032}$
GAT-H3	$7.88_{2.19}$	$18.89_{5.56}$	$1.356_{0.623}$	$8.63_{1.67}$	$18.41_{5.19}$	$2.238_{1.127}$	$9.55_{2.86}$	$20.41_{5.77}$	$0.769_{0.034}$
GAT-H6	$7.90_{2.24}$	$18.97_{5.61}$	$1.296_{0.596}$	$8.21_{1.86}$	$18.51_{5.40}$	$1.879_{0.905}$	$9.55_{2.87}$	$20.42_{5.77}$	$0.767_{0.032}$
GIN-0	$10.51_{3.42}$	$20.92_{6.01}$	$0.937_{0.092}$	$9.69_{2.37}$	$19.70_{5.42}$	$1.351_{0.668}$	$10.05_{2.59}$	$20.70_{5.61}$	$0.835_{0.095}$
GIN-E	$9.13_{3.11}$	$19.96_{6.03}$	$0.941_{0.139}$	$9.59_{1.78}$	$19.63_{4.15}$	$1.407_{0.677}$	$10.32_{2.39}$	$20.84_{5.46}$	$0.881_{0.112}$
RFN-AI	$\mathbf{1.58_{0.05}}$	$\mathbf{1.12_{0.04}}$	$\mathbf{0.205_{0.015}}$	$\mathbf{1.57_{0.06}}$	$\mathbf{1.11_{0.04}}$	$\mathbf{0.184_{0.035}}$	$\mathbf{3.49_{0.13}}$	$\mathbf{2.47_{0.09}}$	$\mathbf{0.100_{0.004}}$
RFN-NAI	$\mathbf{1.63_{0.15}}$	$\mathbf{1.16_{0.11}}$	$\mathbf{0.216_{0.022}}$	$\mathbf{1.63_{0.11}}$	$\mathbf{1.15_{0.08}}$	$\mathbf{0.197_{0.036}}$	$\mathbf{3.40_{0.08}}$	$\mathbf{2.41_{0.06}}$	$\mathbf{0.101_{0.008}}$
RFN-AA	$5.50_{0.07}$	$3.89_{0.05}$	$0.666_{0.010}$	$5.55_{0.12}$	$3.92_{0.09}$	$0.677_{0.013}$	$6.40_{0.07}$	$4.53_{0.05}$	$0.278_{0.002}$
RFN-NAA	$6.40_{0.07}$	$4.53_{0.05}$	$0.278_{0.002}$	$5.64_{0.08}$	$3.99_{0.06}$	$0.692_{0.010}$	$6.45_{0.12}$	$4.56_{0.08}$	$0.283_{0.003}$

According to Table 12, testing results do not show relevant differences between using MAE or RMSE for training a model. Models trained with both loss functions present similar performance results even when a model trained with MAE is tested with RMSE (and vice versa). The same does not apply for tests with MAPE though. Baseline and general purpose GNNs models trained with MAE or RMSE present bad MAPE performance (i.e., higher than 0.9), while RFNs models present test results of 0.2 for interactional and 0.7 for additive variants. Since this work targets a free-flow travel time regression it is important to consider the average percentage error, especially if models are aimed to provide predictions to drivers or urban planners, use cases in which the percentage error might be more meaningful than a small mean absolute error.

When trained with MAPE, obviously, models achieve lower MAPE test results in comparison to models trained with MAE and RMSE. It is interesting to point out though, that when those models are tested with MAE or RMSE, it present values that are only slightly worse than the test

values of models trained with MAE and RMSE. That is: when trained with MAPE, models are not only able to reduce the final average percentage error but by doing that they also achieve relevantly good average absolute errors. This is true for the free-flow travel time regression task on urban networks since target values (i.e., travel times in seconds) are concentrated on a small scale (e.g., below 20 seconds). Baseline and general purpose GNNs achieve quite similar average absolute errors, but present MAPE values of 0.7 instead of the values above 0.9 presented when trained with MAE or RMSE. RFNs achieve slightly worse average absolute errors compared to those trained with MAE or RMSE, but on the other hand, achieve MAPE values of 0.1 for interactional and 0.3 for additive variants.

Figure 23 – Sample of GNN Models' Train vs. Test Matrices: (a) GraphSAGE-MEAN, (b) GAT-H3, (c) RFN-AI and (d) RFN-AA



Source: Own work (2025).

Figure 23 pictures test results with different performance metrics, providing a different view of results presented in Table 12. It shows metrics matrices for four models, presenting for each loss function its respective regression test performance when using each of three cost functions (i.e., MAE, RMSE and MAPE). Models are sampled to represent the simplest GNN model with more parameters than the baseline MLP, GraphSAGE-MEAN; the best performant between all GAT and GIN variants, GAT-H3; and both RFN variants, one with each fusion operator, RFN-AI and RFN-AA. Each row corresponds to the loss function used to train the model and each column corresponds to the trained model test regression performance, one for each error metric (i.e., loss value).

Appendix D presents in detail, for all models explored in this section, prediction plots picturing how far each free-flow travel time prediction is from its correspondent truth value (in seconds). It has one plot for each model trained with each loss function.

5.2 How does mini-batch in gradient descent algorithm influence GNNs' performance?

Section 5.1.2 experiments have explored different models by training with the entire set of training examples when computing an epoch loss. This technique is called batch gradient descent (BGD), or deterministic learning, in which the gradient is optimized only once per epoch (GOODFELLOW; BENGIO; COURVILLE, 2016). In their experiments though, Jepsen, Jensen and Nielsen (2020) applied the Mini-batch Gradient Descent technique, dividing their training set into batches of size 256. Although there is no deep discussion on the mini-batch size choice, the authors point models' convergence within 20 to 30 epochs, a significant decrease on the order of 200 to 400 epochs required for the models to converge on previous experiments (Section 5.1.2). This sub-section explores different sizes of mini-batches for training GNNs with Road Network Graphs, described below:

Table 13 – Aalborg's Road Network Graph Training: batch sizes

	Batch Size	Batch Count
Full training set	23478	1
Mini-Batch	2048	11
	1024	22
	512	45
	256	91

Previously, during the Aalborg road network BGD training (Section 5.1.2), for each training epoch the loss average was calculated for 23478 samples (i.e., the entire training set) and only a single optimization step was taken. When using mini-batch, for each epoch the loss function is averaged for b samples, where b is the mini-batch size, and the gradient is optimized once for each mini-batch. Clearly, the processing time of an epoch would increase with mini-batching,

since there would be n optimization steps[10] instead of a single one, for each epoch. The counterpart of the increase in the processing time is that, with this technique, model's parameters are updated more frequently, reducing the number of epochs required for a model to converge. Additionally, the stochastic nature of mini-batch training can help the training optimization to avoid being tied to local minimums.

Additionally, to avoid biasing the training towards any particular edge feature, training batches are generated without ensuring that: i) each batch maintains the same distribution for any edge feature, and ii) each batch exhibits a distribution similar to the entire training set. Apart from biasing the training process, ensuring that batches maintain the distribution of edge features from the entire training set would require preprocessing steps, which extrapolate the scope of this work. Also, the mini-batches order is randomly shuffled for each epoch, such that the mini-batches are not always presented in the same order as the trained model.

For this mini-batch experiment all the three loss functions - MAE, RMSE and MAPE - are again used but each training session takes only 80 epochs, instead of 1000 epochs of the previous experiment. According to Jepsen, Jensen and Nielsen (2020) results, 80 epochs are enough for GNN models to converge. So, each model is trained, for each loss function, once for each mini-batch size presented in Table 13, with the same experimental setup presented in Table 9. Appendix D details each training session in learning curve plots for all the different mini-batch sizes. It also presents Tables 16, 17, 18 and 19 summarizing all mini-batch training session results, where bold values are the best achieve performance.

Table 14 – Mini-batch training: Computational time analysis

Model	Average Epoch Computation Time (seconds)				
	$b = 23478$[11]	$b = 2048$	$b = 1024$	$b = 512$	$b = 256$
MLP	< 0.01	0.01	0.02	0.03	0.06
SAGE-GCN	< 0.01	0.02	0.04	0.07	0.13
SAGE-MEAN	< 0.01	0.02	0.04	0.07	0.13
SAGE-POOL	0.01	0.03	0.06	0.11	0.21
SAGE-LSTM	0.02	0.14	0.26	0.51	1.02
GAT-H3	0.01	0.05	0.09	0.17	0.32
GAT-H6	0.01	0.09	0.18	0.34	0.67
GIN-0	< 0.01	0.02	0.05	0.07	0.13
GIN-E	< 0.01	0.02	0.05	0.07	0.14
RFN-AI	0.27	2.85	5.71	11.66	23.76
RFN-NAI	0.26	2.90	5.64	11.50	23.47
RFN-AA	0.26	2.84	5.56	11.55	22.94
RFN-NAA	0.26	2.81	5.48	11.48	22.52

Table 14 presents the increase on the average computation time of each training epoch when the mini-batch size is reduced. As expected, the smaller the batch size the higher the number of batches in which training data can be split into, as per Table 13, and for each batch

[10] Where n is the number of mini-batches of size b (e.g., $b = 256$) the training set can be divided into.

[11] This column presents, for comparison, the average epoch computation time during the batch training executed as part of Section 5.1 experiments

a new optimization step is taken. Additionally, a comparison between the MLP baseline and any of the graph neural networks highlights the effect of the message-passing mechanism on the computation time. GNNs will make use of it to leverage structural information but it has a processing cost of one message passing round per optimization step.

The bigger the Road Network Graph, the bigger the expected increase on processing time. It is expected though, despite of the training time increase, that the stochastic nature of mini-batch training will enhance the final test performance of models after training. The performance results should always be compared keeping the computation time in mind, such that a good balance between both is found.

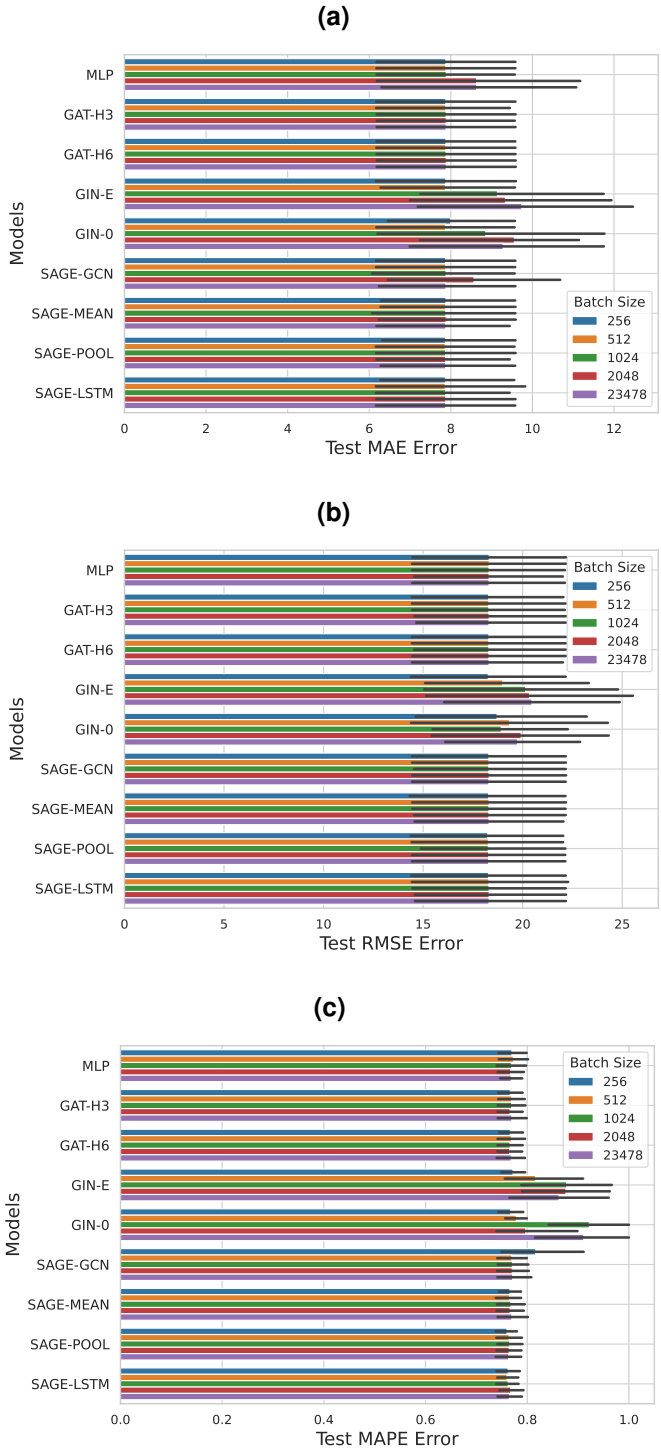
The analysis of different batch sizes, for the baseline and general-purpose GNN models, is depicted in Figure 24. It shows the test performance of each model, when trained with each batch size, for all the three loss functions: MAE (a), RMSE (b) and MAPE (c). The test performance is measured using the trained models to predict values on the test set, that is edges not used in the training phase, and the final result is an average of five tests (i.e., 5-fold cross-validation). Horizontal bars represent the average test values and colors are used to differentiate the batch sizes adopted during the mini-batch training.

Regardless of the loss function used for training, the effect of mini-batch on the final regression test performance is almost irrelevant for the generic purpose GNNs, except for the GIN models that seem to struggle with the batch training and with mini-batch training with bigger sizes of batches, when its test result is worst in comparison with other GNNs. For 512 and 256 batch sizes, GIN models can achieve test performances very similar to all other general-purpose GNNs. But again, since its final loss reduction is almost insignificant, even the best suitable mini-batch sizes can only bring GIN performance closer to the baseline and general-purpose GNNs.

Figure 25 pictures the effect of different batch sizes on the test performance of the relational fusion network variants when trained with the three loss functions: MAE (a), RMSE (b) and MAPE (c). The test performance is measured by using the trained models to predict values on the test set, that is not used in the training phase, and the final result is an average of five tests (i.e., 5-fold cross-validation). Horizontal bars represent the average test values and colors are used to differentiate the batch sizes used during the mini-batch training.

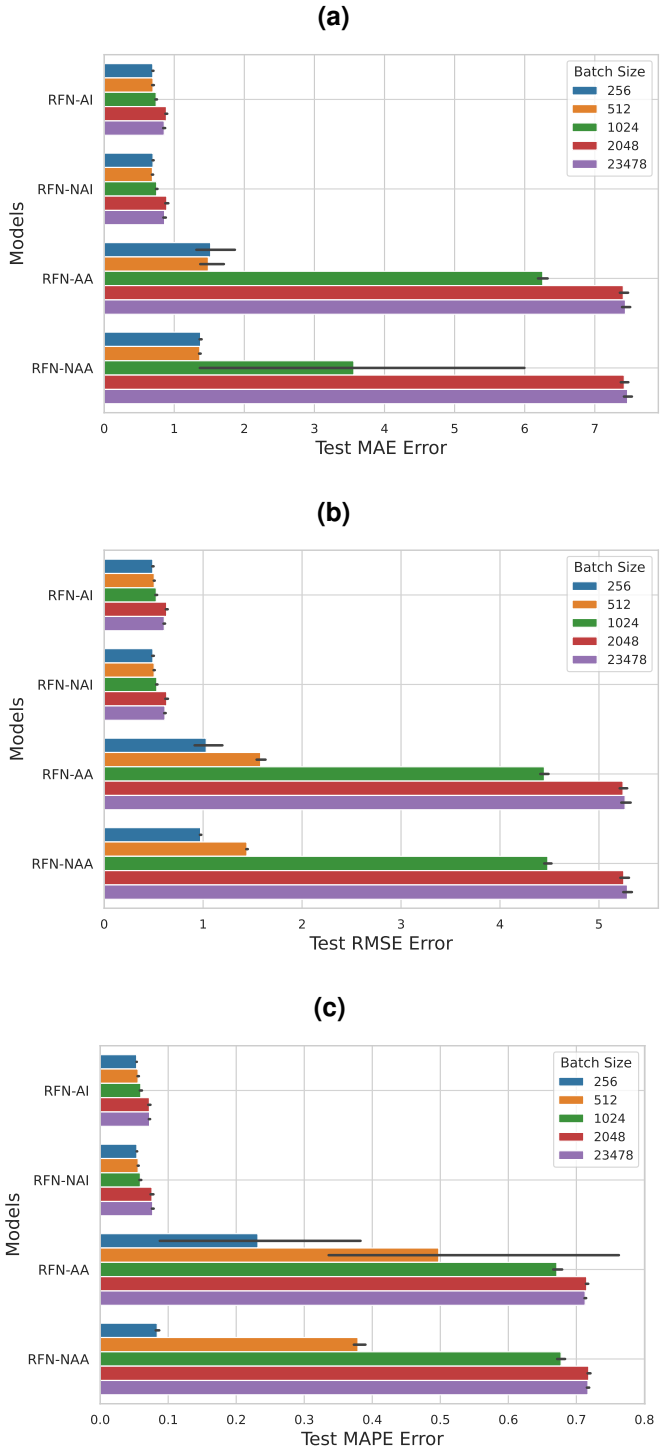
Results show the relevant effect of mini-batching training for additive RFNs. For all loss functions, the smaller the batch size the better the RFN test performance. The scale of the performance enhancement is bigger for additive RFN variants, which seem to benefit even more from mini-batch training. Overall, with batches of size 256, additive RFNs present test performances relatively closer to their interactional counterparts. This result is now aligned with Jepsen, Jensen and Nielsen (2020) results, where additive RFNs performed slightly worse than interactional RFNs on the regression task.

Figure 24 – Mini-batch test performance for the MLP and generic purpose GNNs trained with (a) MAE, (b) RMSE and (c) MAPE



Source: Own work (2025).

Figure 25 – Mini-batch test performance for the RFN variants trained with (a) MAE, (b) RMSE and (c) MAPE



Source: Own work (2025).

Applying Mini-batch Gradient Descent reveals the limited learning power of baseline MLPs and general-purpose GNNs for the Road Network Graph problem, with negligible performance improvements. In contrast, Relation Fusion Networks show significant performance

enhancements with this technique. For the explored dataset, Interactional RFNs outperform Additive RFNs, but both benefit from Mini-batch Gradient Descent.

5.3 How well RFNs extrapolate to new Road Network Graphs?

Although Jepsen, Jensen and Nielsen (2020) have presented a new GNN architecture that is proven to be best suited for Road Network Graphs learning tasks, there is still no discussion on the generalization power of RFN learned knowledge. This section explores how well a relation fusion network can extrapolatewhat is learned from a Road Network Graph to a different one. General-purpose GNNs explored in Sections 5.1.2 and 5.2 are not used in this section.

Each model is trained once with each of twelve different Road Network Graphs and tested not only on its own test set but also on the other eleven cities. The number of network layers (i.e., 2), the hidden size (i.e., 16) and the learning rate (i.e., 0.001) remain the same from Section 5.1.2, but based on the results of Section 5.2 this section uses mini-batch training with batches of size 512 and 100 training epochs. Every trained model is evaluated using the mean average error (MAE) as the performance metric.

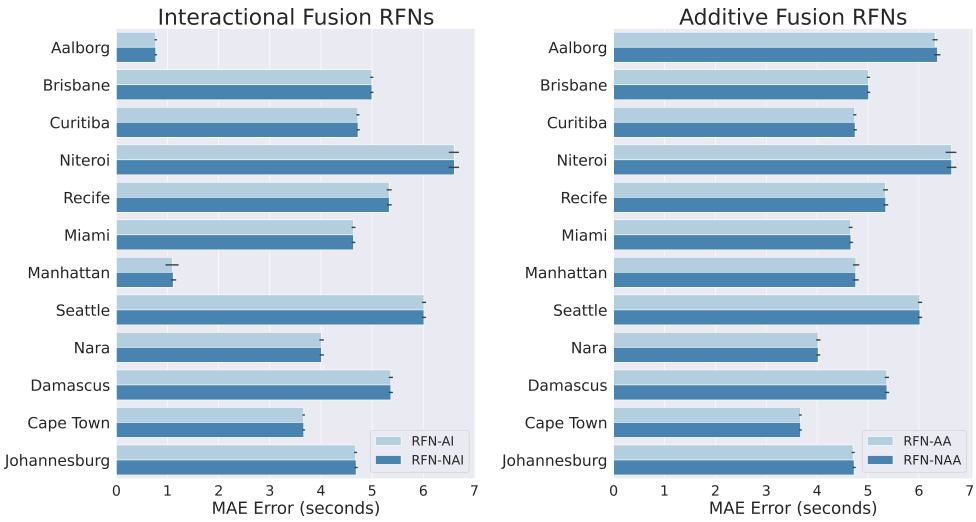
Two regression scenarios are considered in this section: i- self-test, which trains the model on a city A and tests it on unseen data (edges) for the same city, i.e., this scenario evaluates the generalization of regression on travel time with a fixed topology; and ii- extrapolationtest, which trains the model on a city A and tests it on an entirely new data from other cities, i.e., this scenario evaluates extrapolationof regression on travel time for a varying topology – this is useful to explore how well an RFN can extrapolatewhat is learned from a Road Network Graph to a different one.

Figure 26 shows the results, for each, city of the average *self-test* regression error of interactional fusion RFNs (RFN-AI and RFN-NAI) and additive fusion RFNs (RFN-AA and RFN-NAA). Light blue bars show results for attentional RFNs, dark blue bars for non-attentional RFNs, and black lines represent the 95% confidence intervals.

Notice in Figure 26 that the same RFN presents different regression errors when applied to different cities. An example is RFN-AI which presents lower self-test MAE values for Aalborg and Manhattan than it does for Niteroi and Seattle. In general, interactional RFNs present better regression results than additive RFNs. Moreover, there are no relevant differences between RFN-AI and RFN-NAI, or between RFN-AA and RFN-NAA, for any city. Therefore, results show that the attentional aggregator does not enhance the learning capacity of those regression models on this regression task. For all RFNs the confidence interval is clearly narrow, indicating a good predictability of the *self-test* regression errors.

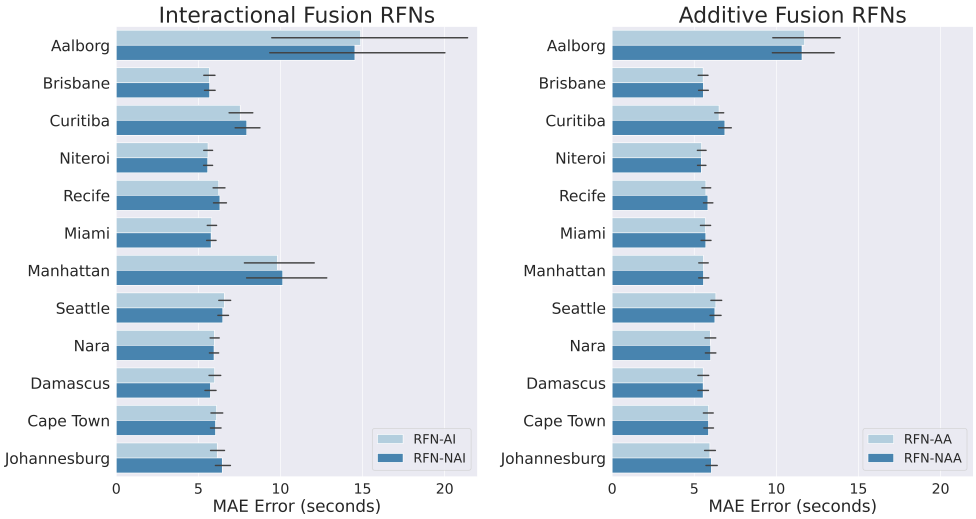
Figure 27 shows the average *extrapolation-test* regression error of interactional and additive RFNs trained with a single city data and tested with data from all other cities. As expected, Figure 27 shows higher regression errors than Figure 26, indicating that the task of training an RFN for one city and testing it in other cities is even more challenging. Similar to the *self-test*

Figure 26 – Self-test regression MAE performance for interactional and additive fusion RFNs.



Source: Own work (2025).

Figure 27 – Extrapolation-test regression MAE performance for different RFNs when trained for one city (vertical labels) but tested with data from all other cities.

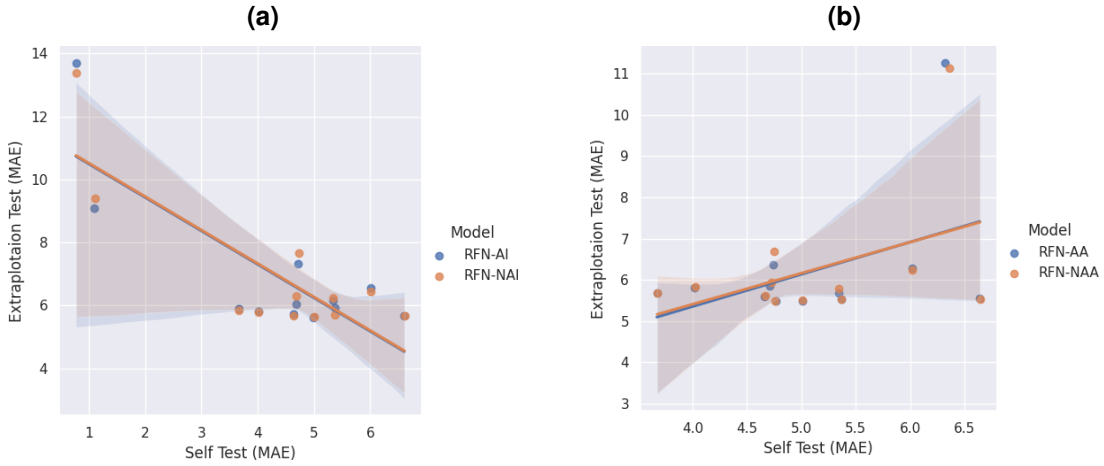


Source: Own work (2025).

results, the attentional aggregator does not enhance the learning capacity of models. However, the results are slightly better for additive variants than for interactional fusion variants. It is important to point out that RFNs trained with road networks that performed best in the *self-test* (i.e., Aalborg and Manhattan) are the two worst performers on the *extrapolation* test.

Figure 28 presents the Pearson correlation between *self-test* and *extrapolation-test* regression errors of cities for interactional and additive fusion RFNs. The Pearson correlations are 0.43, 0.42, -0.81, and -0.82 for RFN-AA, RFN-NAA, RFN-AI, and RFN-NAI, respectively. The positive correlations between their *self* and *extrapolation* tests enforce that additive RFNs extrapolatebetter than interactional RFNs which exhibit negative correlations.

Figure 28 – Correlation between test performances (MAE) for RFNs with (a) interactional and (b) additive fusion.



Source: Own work (2025).

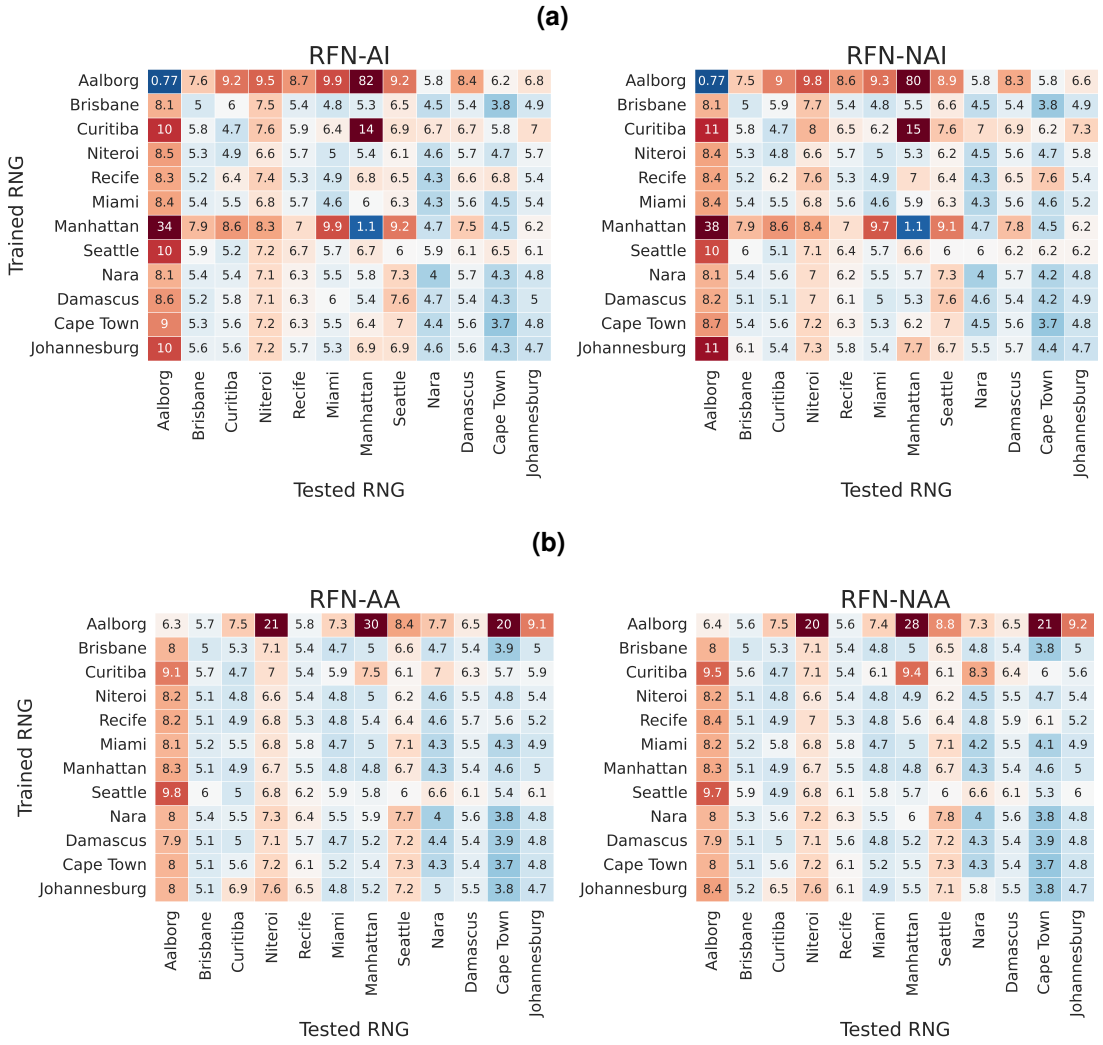
Figure 29 details the *extrapolation-test* of each RFN variant for each city. Each row contains the average *extrapolation-test* regression error of a model trained with data of a city (row label) and tested individually on all remaining cities (column labels). The lowest error in a row often occurs in the diagonal entry as expected, i.e., for the *self-test* scenario. In addition, note that entries in the heatmap are not symmetric. Therefore, the results of training an RFN for city A and testing in city B are not the same as training for B and testing in A.

The idea of extrapolating learned knowledge to other cities should be better investigated by relating road network global features to the corresponding regression errors. Figure 30 presents the heatmap of Pearson correlations between road network features and the corresponding regression error of each RFN variant, using data of all cities for both tests (*self-test* and *extrapolation-test*). Information from all cities is used to generate this heatmap, and the Pearson correlation coefficient (PCC) is the coefficient used to measure the linear correlation between regression performance and feature values. The correlation coefficient ranges from -1 to 1, where the value 1 implies that a linear equation perfectly describes the relationship between X and Y, and the value 0 implies that there is no linear dependency between the variables.

For the task of free-flow travel time regression, according to Figure 30, the numbers of nodes and edges do not show any relevant effect in the RFN regression performance, i.e., no relevant correlation between those features and test performance is found. The circuity average (ς) and orientation order (ϕ) are also not relevantly correlated with the test performance for the set of explored cities.

Additionally, the road network global features that present relevant correlations with the regression performance are semantically connected to what is expected to affect the static free-flow time: the average degree, the average streets per node (i.e., street intersections), the edge length, legal speed distribution, and travel-time distribution. This result adds up to Jepsen,

Figure 29 – Heatmap of average *extrapolation-test* regression error (MAE) of each RFN variant when trained for one city (row label) and tested with data of another city (column label)



Source: Own work (2025).

Jensen and Nielsen (2020) findings, indicating that RFNs can weight semantically connected features for the explored regression task.

The average edge length (in meters) presents a relevant positive correlation with additive RFN performance for both tests. A positive correlation means that as the average edge length increases the same occurs to the test MAE value. The median edge length (in meters), on the other hand, does not present the same correlation pattern. Appendix A pictures all cities' edge length histograms and it is possible to verify that all cities present a similar edge length distribution extremely concentrated between 0 and 500 meters. Therefore, the absence of correlation between median edge length and RFN performance might be caused by the similarity of the cities on this feature.

Results showing the average edge length correlation indicate that RFN performance can be affected by the edge length distribution on the training data. Additive RFNs then, might per-

Figure 30 – Heatmap of Pearson correlations between a road network feature (column label) and the corresponding regression error of cities for self-testand extrapolation-testin each RFN variant (row label).

Source: Own work (2025).

form better on regressions when trained with road networks with narrow edge length distribution around smaller values. On the contrary, interactional RFNs can extrapolate better when trained with lower average edge length networks, yet its *self-test* presents a relevant negative correlation with such feature. That is, the higher the average edge length, the lower the MAE value for *self-test*.

The average degree and the average streets per node correlate negatively with the regression performance for both additive RFN tests and for the interactional RFNs *extrapolation-test*. There is though a slight positive correlation between such features and the interactional RFNs *self-test*. Results show that interactional RFNs can benefit from road networks' low-density characteristics for the *self-test*, but might be negatively affected by them when trying to extrapolatelearned knowledge. Additionally, results indicate that additive RFNs may not benefit from road networks' low-density characteristics.

The legal speed, indistinctly for median or average (both in kilometers per hour), does not present a relevant correlation with any RFNs *extrapolation test* performance but correlates

negatively with all RFNs *self-test average* performances. The magnitude of the correlation is smaller for interactional than for additive RFNs, but both seem to benefit from legal speed distributions centered on higher values. It is important to highlight that for Road Network Graph, the scale of legal speed distribution is usually concentrated between 10 and 80 kilometers per hour, hardly exceeding 100 or 120 kph. Appendix A shows how the edge legal speed distribution varies between studied cities.

The *Major Highway Class Rate* does not present a relevant correlation for additive RFNs. This indicates that additive RFN performance is not affected by edge class distributions. However, *Major Highway Class Rate* presents a medium positive correlation of 0.41 for RFN-AI and RFN-NAI *self-test*. Therefore, although interactional RFNs do specialize the learned knowledge to the training set (i.e., good *self-test* performance), they are not overfitting towards the edge class information, since the higher the concentration towards a class group, the higher the *self-test* MAE value.

While the travel time (in seconds) is not a road network feature considered for training (i.e., it is the feature being predicted), the correlation between cities' average travel time and the RFN performance demonstrates that the distribution of travel time values does not affect the interactional RFNs *self-test* performance. It negatively affects (i.e., positive correlation) interactional RFNs *extrapolation test* performance though, similarly to how it affects additive RFNs *self-test* performance. That is, the higher the average travel time (i.e., the travel time values distribution wideness), the higher the MAE value (i.e., the performance decreases). Appendix A pictures all cities' edge free-flow travel time histograms and it is possible to verify that all cities present a similar travel time distribution extremely concentrated between 0 and 50 seconds.

6 CONCLUSION

The present work explored Graph Neural Networks (GNNs) applied to the free-flow travel time regression task on the Road Network Graph contetxt. Models were evaluated on their representation learning capacities aiming to answer the three selected research questions.

Concerning the **first research question** about the suitability of the graph representation to solve the edge regression task, general purpose GNNs, in particular, GraphSAGE, Graph Attention Networks (GAT) and Graph Isomorphic Networks (GIN), and specialized GNNs, specifically Relation Fusion Networks (RFNs), were explored on the Aalborg Road Network Graph. It is the same city from Denmark explored in the original paper that proposes RFNs for road network-related tasks. Although the road network features were considerably similar to the ones used by Jepsen, Jensen and Nielsen (2020) in their original work, there might be hidden differences between target travel times used in their regression task and those used in this work. Particularly, the present work targeted the free-flow travel time regression using information retrieved directly from OSM as ground truth travel times. On the other hand, Jepsen, Jensen and Nielsen (2020) used an average of historical GPS data from a private dataset.

All the experiments added up to those presented in (JEPSEN; JENSEN; NIELSEN, 2020). The results showed that the general-purpose GNNs do not present relevant regression performance gains compared to the Multi-Layer Perceptron (MLP) baseline to justify the increase in the model's parameters and training computation time. Besides, RFNs presented a road network regression performance, on Aalborg's Road Network Graph, that was not achieved by any addressed general-purpose GNN. Despite the significant increase in the number of parameters and training time, results indicated that RFNs with interaction fusion are the most suitable model for the edge regression task.

The role of different fusion functions was clear and aligned with the proposal of Jepsen, Jensen and Nielsen (2020). Differences between this and the original work were mainly on i) the scale of the performance enhancements for RFNs when compared with other GNNs, and ii) the scale of the difference between interactional and additive RFNs. Additive fusion functions had the worst performance among the four RFN variants. Moreover, the attentional aggregation did not provide any relevant improvement in the models' regression performance. This might indicate that the low density of the Aalborg road network (i.e., small node average degree) does not play a role as important as its volatile homophily. In other words, outlier neighbors do not affect the training process as much as sharp boundaries do between homophilic regions.

On the different training loss functions explored in this work, regardless of the chosen function, the previous conclusion still holds. When training with the Mean Absolute Error (MAE), i.e. the same loss function used by Jepsen, Jensen and Nielsen (2020), results showed that RFNs were indeed the best models for this road network task, but in scales that differ from the original work. For the original work, additive RFNs presented 27% of performance enhancement when compared to the MLP baseline, while this work's results showed only 14% of enhancement,

and interactional RFNs presented 33% of performance enhancement, while this work's results showed almost 90% of regression performance enhancement. These scale differences might be explained by the differences in the set of truth travel times aforementioned. When training with the Root Mean Squared Error (RMSE), results showed that this loss function accentuates the regression performance differences between RFNs and generic-purpose GNNs. When training with the Mean Absolute Percent Error (MAPE), the final validation performance enhancement of additive RFNs concerning the baseline was amplified.

Training RFNs with MAPE might be the best choice for this specific regression task, since models trained with MAPE could achieve reasonably good MAE and RMSE values while also achieving the best MAPE values. This is true for the free-flow travel time regression task on urban networks as the set of target values (i.e., travel times in seconds) are concentrated on a small scale (e.g., below 20). Predicting travel times with a reasonably small MAPE is good since the percentual error might be more meaningful for a driver interested in an estimated time to arrive than the small mean absolute error, especially for travel time outliers.

For the **first research question**, the results showed that general-purpose GNNs do not outperform MLP for regression on Aalborg's Road Network Graph dataset. By comparing learning curves, SAGE-GCN and SAGE-MEAN showed faster learning than MLP. However, considering that SAGE-GCN is an MLP with message-passing, SAGE-MEAN would be preferred for future tests. SAGE-POOL and SAGE-LSTM also presented fast learning, but they presented overfitting for RMSE and MAPE. GAT-H3 and GAT-H6 also presented fast learning, but GAT-H6 (with more attention headers) did not improve the results. Therefore, a simple model GAT-H3 would be preferable. GIN-0 and GIN-E presented poor performances, even worse than MLP, with a marginal advantage for GIN-E. RFN-AI and RFN-NAI showed the fastest learning. In 300 epochs, they had greatly reduced training errors represented by MAE, RMSE, or MAPE. They did not show overfitting even for small errors. No differences were noticed from the results of applying attentional aggregators (RFN-AI and RFN-NAI). RFN-AA and RFN-NAA (additive) showed slower learning than (interactional) RFN-AI and RFN-NAI, not converging until 1000 epochs. However, they had almost five times fewer model parameters to be learned.

Regarding the **second research question** on the influence of Mini-batch Gradient Descent training with different batch sizes, regardless of the loss function used for training, the effect of the mini-batch on the final regression test performance was almost irrelevant for the general-purpose GNNs. This can be taken as an additional result to reinforce the lack of general-purpose GNNs' capacity to learn meaningful urban road network representations.

On the other hand, the results showed a relevant effect of mini-batching values in the RFN performance. For all tested loss functions, the smaller the batch size the better the RFN test performance. RFNs using additive fusion seemed to benefit even more from mini-batch training since its error scale was bigger. This is also aligned with Jepsen, Jensen and Nielsen (2020) results, where additive RFNs performed slightly worse than interactional RFNs on the regression task, although the performance gap between both RFN variants is bigger in this work.

One explanation is the differences in dataset features, which can affect differently the learning capacity of RFNs with interactional or additive fusion operators. Both variants are still clearly learning meaningful information since both showed performance enhancements when trained using the Mini-batch Gradient Descent.

For the **third research question**, regarding the RFN extrapolation capability, this work explored the four RFN variants on the same regression task, now training models with Road Network Graphs from 12 different cities. The comparison between RFNs using interactional or additive fusion revealed that both variations present much higher errors when trying to extrapolate knowledge than when applying it to the same city used for training. However, additive fusion exhibited lower mean absolute errors when extrapolating the learned knowledge to other cities.

Experiments also showed that, regardless of the fusion operator employed, the attentional aggregator does not enhance RFN's learning capacity compared to the non-attentional approach for this regression task. RFN-AI and RFN-NAI presented relevant and negative correlations between self-test and the extrapolation test, indicating that RFNs with interactional fusion operators tend to specialize better to the training dataset and do not extrapolate well for different cities. The same did not apply to RFN-AA or RFN-NAA (additive fusion), where there was a medium and positive correlation between self and generalization test performances. Again, this can be explained by how much the dataset characteristics can affect differently the learning capacity of RFNs, with interactional or additive fusion operators, and future works on the exploration of extended datasets can help to better guide conclusions on this topic.

When trying to find correlations between road network global features and their learning and generalization powers, results showed that RFNs did weight road network global features that are semantically connected to the predicted edge feature (i.e., the free-flow travel time), adding up to the results on RFNs for road network learning.

6.1 Future work

Future work can explore extended datasets (i.e., datasets with increased feature information) and also different edge regression tasks, swapping the training and target features to explore whether RFNs always do weight training features that are semantically connected to the target one.

Moreover, this work did not address the problem of the Road Network Graph batch division. Future work could test batches that keep edge feature distributions when the dataset consists of a single city. It could also explore Road Network Graph segmentation, along with different training approaches that can help diversify road network information available on training, possibly extending the number of available training features and creating datasets of multiple road networks, considering each city as a dataset entry (i.e., batch).

Distributed learning techniques can help towards extrapolation enhancements, increasing the range of structural samples available on training. Further exploration of road network

features, including a wider range of cities, is important to better understand which characteristics affect the regression and extrapolation capacity of RFNs. Advancements towards this direction can serve various goals, such as city-specific specialization or extrapolation across different urban environments of distinct cities.

Exploring different edge feature regression problems is one path for investigation, but also the edge classification problem can be explored. This was not explored in this work and it might be possible to enhance general-purpose GNNs classification performance, when compared to RFNs, by using models that do consider edge direction when passing messages.

Future works exploring better GNN models for the Road Network Graph context, could also address the model size problem. It was mentioned that RFNs are comparatively larger models than the general-purpose RFNs. The size increase was justified by the performance enhancement in this work. However, future applications might be interested in smaller models. It could be possible to explore new, and smaller, RFN fusion operators or experiment reducing RFNs to its edge-relational component only. It means a compromise between performance and size, allowing for the comparison of different regression performances against the complete RFN counterpart. A different research path could explore knowledge distillation from RFNs to simpler models, probably general-purpose RFNs and measure its regression performance. Smaller models could be applied, for instance, as the structural components in Spatio-Temporal GNNs, benefiting from road network latent representations to enhance the performance of temporal regression, such as traffic patterns prediction and traffic accident forecasting.

REFERENCES

ALBINO, V.; BERARDI, U.; DANGELICO, R. M. Smart cities: Definitions, dimensions, performance, and initiatives. **Journal of Urban Technology**, v. 22, p. 3–21, 2015. ISSN 1063-0732.

ANTHONY, M.; BARTLETT, P. L. **Neural Network Learning: Theoretical Foundations**. [*S.l.*]: Cambridge University Press, 1999.

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. **International Conference on Learning Representations**, abs/1409.0473, 2014.

BARCELÓ, P. *et al.* The logical expressiveness of graph neural networks. *In*: **International Conference on Learning Representations**. [*S.l.*: *s.n.*], 2020.

BARRON, C.; NEIS, P.; ZIPF, A. A comprehensive framework for intrinsic openstreetmap quality analysis. **Transactions in GIS**, v. 18, 2014.

BATTAGLIA, P. W. *et al.* Relational inductive biases, deep learning, and graph networks. **ArXiv**, abs/1806.01261, 2018.

BATTAGLIA, P. W. *et al.* Interaction networks for learning about objects, relations and physics. *In*: **NIPS**. [*S.l.*: *s.n.*], 2016.

BEINEKE, L. W.; BAGGA, J. S. Line graphs and line digraphs. **Developments in Mathematics**, 2021.

BOEING, G. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. **Computers, Environment and Urban Systems**, v. 65, p. 126–139, 2017. Disponível em: https://doi.org/10.1016/j.compenvurbsys.2017.05.004.

BOEING, G. Urban spatial order: street network orientation, configuration, and entropy. **Applied Network Science**, Springer Science and Business Media LLC, v. 4, n. 1, aug 2019. Disponível em: https://doi.org/10.1007\%2Fs41109-019-0189-1.

BORDES, A. *et al.* Translating embeddings for modeling multi-relational data. *In*: **NIPS**. [*S.l.*: *s.n.*], 2013.

BRUNA, J. *et al.* Spectral networks and locally connected networks on graphs. **CoRR**, abs/1312.6203, 2013.

CERVI, T. E. *et al.* Regression performance of relational fusion networks on urban road networks. *In*: **Anais do VIII Workshop de Computação Urbana (to appear)**. Porto Alegre, RS, Brasil: SBC, 2024. p. 141–154.

CHO, K. *et al.* Learning phrase representations using rnn encoder–decoder for statistical machine translation. *In*: **Conference on Empirical Methods in Natural Language Processing**. [*S.l.*: *s.n.*], 2014.

DEFFERRARD, M.; BRESSON, X.; VANDERGHEYNST, P. Convolutional neural networks on graphs with fast localized spectral filtering. *In*: **Thirtieth Conference on Neural Information Processing Systems (NIPS)**. [*S.l.*: *s.n.*], 2016.

DERROW-PINION, A. *et al.* Eta prediction with graph neural networks in google maps. **Proceedings of the 30th ACM International Conference on Information & Knowledge Management**, 2021.

DONNAT, C. *et al.* Spectral graph wavelets for structural role similarity in networks. **ArXiv**, abs/1710.10321, 2017.

DUVENAUD, D. K. *et al.* Convolutional networks on graphs for learning molecular fingerprints. *In*: **Twenty-ninth Conference on Neural Information Processing Systems (NIPS)**. [*S.l.*: *s.n.*], 2015. abs/1509.09292.

EGER, J. M. Smart growth, smart cities, and the crisis at the pump a worldwide phenomenon. **I-WAYS, Digest of Electronic Commerce Policy and Regulation**, v. 32, p. 47–53, 2009. ISSN 10844678.

GHARAEE, Z. *et al.* Graph representation learning for road type classification. **Pattern Recognit.**, v. 120, p. 108174, 2021.

GILMER, J. *et al.* Neural message passing for quantum chemistry. *In*: **Proceedings of the 34th International Conference on Machine Learning - Volume 70**. [*S.l.*]: JMLR.org, 2017. (ICML'17), p. 1263–1272.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S.l.*]: MIT Press, 2016. http://www.deeplearningbook.org.

GORI, M.; MONFARDINI, G.; SCARSELLI, F. A new model for learning in graph domains. **Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.**, v. 2, p. 729–734 vol. 2, 2005.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using networkx. *In*: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). **Proceedings of the 7th Python in Science Conference**. Pasadena, CA USA: [*s.n.*], 2008. p. 11 – 15.

HAKLAY, M. M. How good is volunteered geographical information? a comparative study of openstreetmap and ordnance survey datasets. **Environment and Planning B: Planning and Design**, v. 37, p. 682 – 703, 2010.

HAMILTON, W. L. Graph representation learning. **Synthesis Lectures on Artificial Intelligence and Machine Learning**, 2020.

HAMILTON, W. L.; YING, R.; LESKOVEC, J. Inductive representation learning on large graphs. *In*: **Proceedings of the 31st International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 1025–1035. ISBN 9781510860964.

HAMILTON, W. L.; YING, R.; LESKOVEC, J. Representation learning on graphs: Methods and applications. **CoRR**, abs/1709.05584, 2017. Disponível em: http://arxiv.org/abs/1709.05584.

HARARY, F.; NORMAN, R. Z. Some properties of line digraphs. **Rendiconti del Circolo Matematico di Palermo**, v. 9, p. 161–168, 1960.

HARRISON, C. G. *et al.* Foundations for smarter cities. **IBM J. Res. Dev.**, v. 54, p. 1–16, 2010.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, p. 1735–1780, 1997.

HOFF, P. D.; RAFTERY, A. E.; HANDCOCK, M. S. Latent space approaches to social network analysis. **Journal of the American Statistical Association**, v. 97, p. 1090 – 1098, 2002.

HUANG, G.; LIU, Z.; WEINBERGER, K. Q. Densely connected convolutional networks. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 2261–2269, 2016.

JACOBS, K. Independent identically distributed (iid) random variables. *In*: ____. **Discrete Stochastics**. Basel: Birkhäuser Basel, 1992. p. 65–101. ISBN 978-3-0348-8645-1. Disponível em: https://doi.org/10.1007/978-3-0348-8645-1_4.

JEPSEN, T. S.; JENSEN, C. S.; NIELSEN, T. D. Relational fusion networks: Graph convolutional networks for road networks. **IEEE Transactions on Intelligent Transportation Systems**, v. 23, p. 418–429, 2020.

JEPSEN, T. S. *et al.* On network embedding for machine learning on road networks: A case study on the danish road network. **2018 IEEE International Conference on Big Data (Big Data)**, p. 3422–3431, 2018.

JIN, G. *et al.* Dual graph convolution architecture search for travel time estimation. **ACM Trans. Intell. Syst. Technol.**, Association for Computing Machinery, New York, NY, USA, v. 14, n. 4, jun 2023. ISSN 2157-6904. Disponível em: https://doi.org/10.1145/3591361.

KARIMI, K. A configurational approach to analytical urban design: 'space syntax' methodology. **URBAN DESIGN International**, v. 17, p. 297–318, 2012.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **International Conference on Learning Representations**, abs/1412.6980, 2014.

KIPF, T.; WELLING, M. Semi-supervised classification with graph convolutional networks. **ArXiv**, abs/1609.02907, 2016.

LECUN, Y. *et al.* Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, p. 2278–2324, 1998.

LI, Y. *et al.* Graph matching networks for learning the similarity of graph structured objects. *In*: **International Conference on Machine Learning**. [*S.l.*: *s.n.*], 2019.

LI, Y. *et al.* Gated graph sequence neural networks. **CoRR**, abs/1511.05493, 2015.

LI, Y. *et al.* Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. **International Conference on Learning Representation**, 2018.

MCCULLAGH, P.; NELDER, J. A. **Generalized Linear Models**. [*S.l.*]: Chapman and Hall, 1972. ISBN 9780203753736.

MENA, G. E. *et al.* Learning latent permutations with gumbel-sinkhorn networks. **ArXiv**, abs/1802.08665, 2018.

MERKWIRTH, C.; LENGAUER, T. Automatic generation of complementary descriptors with molecular graph networks. **Journal of chemical information and modeling**, v. 45 5, p. 1159–68, 2005.

MEYER, C. **Matrix Analysis and Applied Linear Algebra**. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2000. (Other Titles in Applied Mathematics). ISBN 9780898719512. Disponível em: https://books.google.com.br/books?id=-7JeAwAAQBAJ.

MURPHY, R. L. *et al.* Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. **ArXiv**, abs/1811.01900, 2018.

NIPPANI DONGYUE LI, H. J. H. N. K. H. R. Z. A. Graph neural networks for road safety modeling: Datasets and evaluations for accident analysis. **Neural Information Processing Systems**, 2024.

OpenStreetMap contributors. **Planet dump retrieved from https://planet.osm.org** . 2017. https://www.openstreetmap.org.

PARTHASARATHI, P.; LEVINSON, D. M.; HOCHMAIR, H. H. Network structure and travel time perception. **PLoS ONE**, v. 8, 2013.

PEARSON, K. The problem of the random walk. **Nature**, v. 72, 1905.

PEROZZI, B.; AL-RFOU, R.; SKIENA, S. Deepwalk: Online learning of social representations. *In*: **Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2014. (KDD '14), p. 701–710. ISBN 9781450329569. Disponível em: https://doi.org/10.1145/2623330.2623732.

PHAM, T. *et al.* Column networks for collective classification. *In*: **AAAI Conference on Artificial Intelligence**. [*S.l.*: *s.n.*], 2016.

PORTA, S.; CRUCITTI, P.; LATORA, V. The network analysis of urban streets: A primal approach. **Environment and Planning B: Planning and Design**, v. 33, p. 705 – 725, 2006.

QI, C. *et al.* Pointnet: Deep learning on point sets for 3d classification and segmentation. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 77–85, 2016.

ROSSI, R. A.; AHMED, N. K. The network data repository with interactive graph analytics and visualization. *In*: **AAAI**. [*s.n.*], 2015. Disponível em: https://networkrepository.com.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, p. 533–536, 1986.

SANCHEZ-LENGELING, B. *et al.* A gentle introduction to graph neural networks. **Distill**, 2021. Https://distill.pub/2021/gnn-intro.

SCARSELLI, F. *et al.* The graph neural network model. **IEEE Transactions on Neural Networks**, v. 20, n. 1, p. 61–80, 2009.

SELSAM, D. *et al.* Learning a sat solver from single-bit supervision. **ArXiv**, abs/1802.03685, 2018.

SEUNG, H.; SOMPOLINSKY, H.; TISHBY, N. Statistical mechanics of learning from examples. **Physical review. A**, v. 45, p. 6056–6091, 1992.

SHIELDS, R. Cultural topology: The seven bridges of königsburg, 1736. **Theory, Culture & Society**, v. 29, p. 43 – 57, 2012.

SILVA, T. H.; SILVER, D. Using graph neural networks to predict local culture. **arXiv**, 2024.

STRANO, E. *et al.* Urban street networks, a comparative analysis of ten european cities. **Environment and Planning B: Planning and Design**, v. 40, p. 1071 – 1086, 2013.

VAPNIK, V. The nature of statistical learning theory. *In*: ____. [*S.l.*]: Springer, 2000. v. 8, p. 1–15. ISBN 978-1-4419-3160-3.

VASWANI, A. *et al.* Attention is all you need. *In*: **NIPS**. [*S.l.*: *s.n.*], 2017.

VELICKOVIC, P. *et al.* Graph attention networks. **ArXiv**, abs/1710.10903, 2017.

VINYALS, O.; BENGIO, S.; KUDLUR, M. Order matters: Sequence to sequence for sets. **CoRR**, abs/1511.06391, 2015.

WANG, M. xiang *et al.* Learning embeddings of intersections on road networks. **Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems**, 2019.

WEISFEILER, B.; LEHMAN, A. A. A reduction of a graph to a canonical form and the albegra arising during this reduction. **Nauchno-Technicheskaya Informatsia**, 1968.

WILSON, D. R.; MARTINEZ, T. R. The general inefficiency of batch training for gradient descent learning. **Neural networks: the official journal of the International Neural Network Society**, v. 16 10, p. 1429–51, 2003. Disponível em: https://api.semanticscholar.org/CorpusID:652801.

WU, F. *et al.* Simplifying graph convolutional networks. **36th International Conference on Machine Learning (ICML)**, abs/1902.07153, 2019.

XU, K. *et al.* How powerful are graph neural networks? **ArXiv**, abs/1810.00826, 2018.

XU, K. *et al.* Representation learning on graphs with jumping knowledge networks. *In*: **35th International Conference on Machine Learning (ICML)**. [*S.l.*: *s.n.*], 2018. (ICML'18).

YAN, H. *et al.* Jointly modeling intersections and road segments for travel time estimation via dual graph convolutional networks. *In*: **Spatial Data and Intelligence: Third International Conference, SpatialDI 2022, Wuhan, China, August 5–7, 2022, Revised Selected Papers**. Berlin, Heidelberg: Springer-Verlag, 2023. p. 19–34. ISBN 978-3-031-24520-6. Disponível em: https://doi.org/10.1007/978-3-031-24521-3_2.

YANG, Z.; COHEN, W. W.; SALAKHUTDINOV, R. Revisiting semi-supervised learning with graph embeddings. *In*: . [*S.l.*]: JMLR.org, 2016. (ICML'16), p. 40–48.

YING, R. *et al.* Hierarchical graph representation learning with differentiable pooling. *In*: **Neural Information Processing Systems**. [*S.l.*: *s.n.*], 2018.

YU, T.; YIN, H.; ZHU, Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *In*: **International Joint Conference on Artificial Intelligence**. [*S.l.*: *s.n.*], 2018.

ZAHEER, M. *et al.* Deep sets. Curran Associates, Inc., v. 30, 2017.

ZEMLYACHENKO, V. N.; KORNEENKO, N. M.; TYSHKEVICH, R. Graph isomorphism problem. **Journal of Soviet Mathematics**, v. 29, p. 1426–1481, 1985.

ZHANG, L.; LONG, C. Road network representation learning: A dual graph-based approach. **ACM Transactions on Knowledge Discovery from Data**, v. 17, p. 1 – 25, 2023.

ZHANG, M. *et al.* An end-to-end deep learning architecture for graph classification. *In*: **AAAI Conference on Artificial Intelligence**. [*S.l.*: *s.n.*], 2018.

ZHAO, L. *et al.* T-gcn: A temporal graph convolutional network for traffic prediction. **IEEE Transactions on Intelligent Transportation Systems**, v. 21, p. 3848–3858, 2020.

ZHOU, D. *et al.* Learning with local and global consistency. *In*: **NeurIPS**. [*S.l.*: *s.n.*], 2004.

ZITNIK, M.; AGRAWAL, M.; LESKOVEC, J. Modeling polypharmacy side effects with graph convolutional networks. **Bioinformatics**, v. 34, p. i457 – i466, 2018.

APPENDIX A – Road Network Graph Dataset

This Appendix presents more details on the assembled Road Network Graph dataset, including details on the usage of OSMnx Python library in this work and relevant notes from OpenStreetMap® (OpenStreetMap contributors, 2017) documentation regarding the used road network information.

A.1 Details on the dataset design

When accessing OpenStreetMap® worldwide information, one of the benefits of using OSMnx is that it has been developed to solve two common issues when working with street networks: data over-simplification and repeatability of results (BOEING, 2017). It is usual to find simplified representations of street networks, usually transformed into planar or undirected graphs. Assuming network planarity is a fair simplification choice for a great number of cities. However, cities with numerous grade-separated expressways, bridges and tunnels (e.g., Los Angeles in the USA) have truly non-planar networks that could not fit such simplification (BOEING, 2017). Additionally, the network direction information may not matter for studies focused on pedestrian movement, but it is highly important for representing correctly drivable networks, the main focus of this work. OSM available data not only provides a unified way of collecting topological (i.e., not necessarily planar) network structure, but also provides it in an open way for everyone interested in verifying or reproducing experimental results later.

Moreover, OSMnx retrieved information includes several useful graph global statistics on the selected road network, including the number of nodes, number of edges, the average node degree, the average street per node count, the average length of edges, the self-loop proportion, the average circuity and orientation-order. The average circuity ς indicates how much more circuitous a Road Network Graph is than it would be if all of its edges were straight-line paths between nodes. That is the relation between all edge lengths in the graph and the sum of all great-circle distances between all pairs for connected nodes (BOEING, 2019). The orientation-order ϕ is defined by calculating the bearings of all edges and dividing them into 36 bins of 10 degrees each, measuring then the entropy of the binned bearings (BOEING, 2019).

OSMnx retrieves a MultiDiGraph object (directed graph with parallel edges), containing the road network representation of a city. The retrieved network buffers 500 meters by default to get a graph larger than requested. Nodes outside the boundary polygon are truncated even if they have neighbors inside of it. Finally, only the largest connected component (if the network is not fully connected) is returned. It is also possible too simplify a graph topology by removing all nodes that are not intersections or dead-ends.

When OSMnx retrieves a city's Road Network Graph, it provides a unique integer identifier for each node and this value is the respective OSM global node ID. This work processes all selected Road Network Graphs such that their node IDs are mapped into a set of indexes $\{0, ..., |\mathcal{V}|\}$ where $|\mathcal{V}|$ is the number of nodes in the final connected component. The NetworkX *line_graph()* function is used to generate the dual representation of the Road Network Graph.

The primal edge feature set is added to the dual node feature set, and the primal between-edge feature set is added to the dual edge feature set in the dual representation.

All OSMnx graphs contain both node and edge features. An important edge feature is the “highway class” which contains information on the type of the road, distinguishing the importance of the road segment within the road network graph[1]. The “motorway” tag is used to identify the highest-performance roads within a territory. It usually implies on a road segment in which the maximum legal speed is high, the surface is paved, has at least two lanes in each traffic direction with some degree of separation and allows only motor vehicle access (i.e., no pedestrians or bicycles are allowed). The “trunk” tag is used for high-performance or high-importance roads that don’t meet the requirements for motorway but are still high in a degree order that compares roads tagged as “primary", which are usually highways linking large towns or major arterial roads within cities.

Next, the “secondary” tag is used for highways that are not part of major routes, but form a link in the national route network and, within cities, for major arterial roads of lesser importance than “primary” ones. The “tertiary” tag is used for roads connecting smaller settlements, and within large settlements for roads connecting local centres. Within larger urban centers “tertiary” roads are roads with low to moderate traffic linking local centres of activity such as shopping facilities, schools, or suburbs. There is also a special and misleading tag “unclassified” used for minor public roads, typically at the lowest level of whatever administrative hierarchy. This tag does not indicate that the road is not classified, but rather that this road is used only for local traffic or has low importance near the outskirts of towns and cities. All tags listed so far, except the “unclassified” have their “link” variant (e.g., “trunk_link", “primary_link") used to describe roads leading to and from a certain class of roads.

Finally, roads not qualifying for any of the groups above, with residences along their sides, are tagged “residential", because they usually provide access to, or within, residential areas, but are not normally used as through routes. For urban environments, there is also a special type of residential road, the “living_street” tag, used for residential or shared spaces with lower speed limits, and special traffic and parking rules compared to roads tagged as “residential".

An edge can have one or more “highway class” tags and to ensure a common ground for all cities in the dataset. Any class that is not described in this section has been discarded from Road Network Graphs used in this work. Examples of classes discarded are: “disused”, “services”, “rest_area”, “crossing”, “road” and “busway”. If an edge has only one “highway class” not described in this section, it is removed from the network. If an edge has more than one class and one of them is not described here, that tag is removed to keep the edge with only the remaining tags. After this edge filtering, an network can become a disconnected graph. If it is the case, once again only the largest weakly connected component is retrieved.

[1] OpenStreetMap® official documentation (OpenStreetMap contributors, 2017)

A.2 Selected cities

This work selects relevant cities from all continents to have a globally representative sample of cities and to avoid the commonplace of restraining the analysis to the author's local geographic position. Twelve cities have been selected for the experiments: the European city of Aalborg (the same city used in the experiments of Jepsen, Jensen and Nielsen (2020)); Brisbane in Australia; Nara in Japan; Manhattan island, Miami, and Seattle, all in the USA; Damascus in Syria; Cape Town and Johannesburg in South Africa; and three Brazilian cities of Curitiba, Niteroi, and Recife. The list of cities selected to be modeled as Road Network Graphs, and their approximate population sizes, can be found in Table 15.

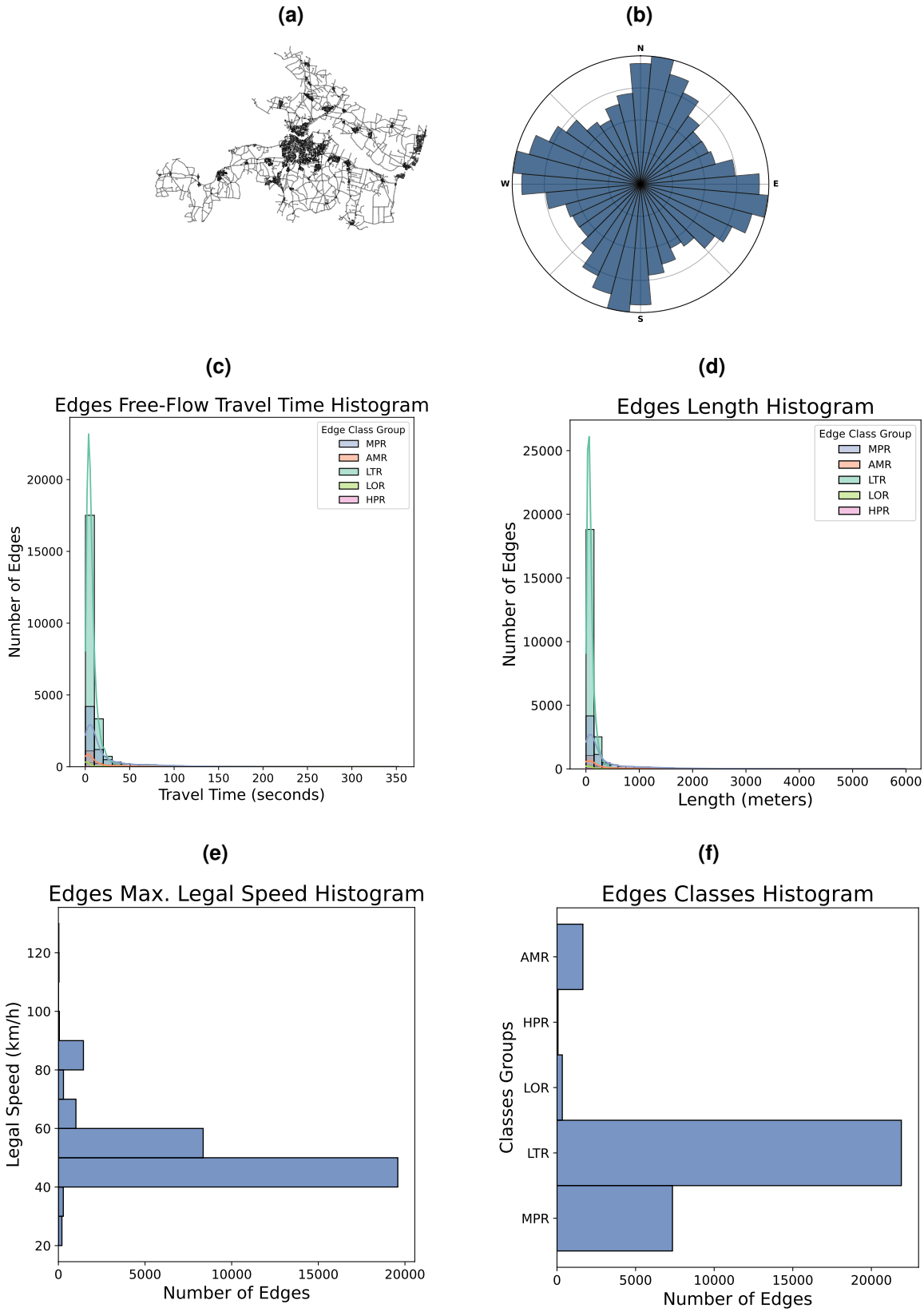
Table 15 – Selected Road Network Graph datasets of cities from all continents

City	Country	Approx. Population (people)	Population density (people/km^2)
Curitiba	Brazil	1,773,718	4,078.5
Niterói	Brazil	515,317	3,601.7
Recife	Brazil	1,488,920	6,816.3
Manhattan	USA	1,694,251	19,320.9
Miami	USA	442,241	3,045.1
Seattle	USA	737,015	2,002.9
Aalborg	Denmark	119,862	2,364.1
Damascus	Syria	2,079,000	3,628.3
Cape Town	South Africa	4,005,015	1,631.4
Johannesburg	South Africa	5,926,668	3,602.8
Nara	Japan	35,666	1,299.2
Brisbane	Australia	2,274,600	385.3

The complete data set of all Road Network Graphs, on both primal and dual representation, along with node, edge, and between-edges features have been made publicly available[2]. This appendix now presents figures picturing and detailing each city. Each figure will first present the city road network structure. Then, a polar plot divided into 36 sections of 10^o each, in which each section has a bar with height representing the count of edges that have that bearing orientation. After, there will be histograms picturing the distribution of the road network edge features in the following order: free-flow travel time (in seconds), edge length (in meters), maximum legally allowed speed (in kilometers per hour) and edge class (grouped as described in Section 4.1.3). Both free-flow travel time and edge length histograms will have information grouped by edge class groups, each one pictured in a different color.

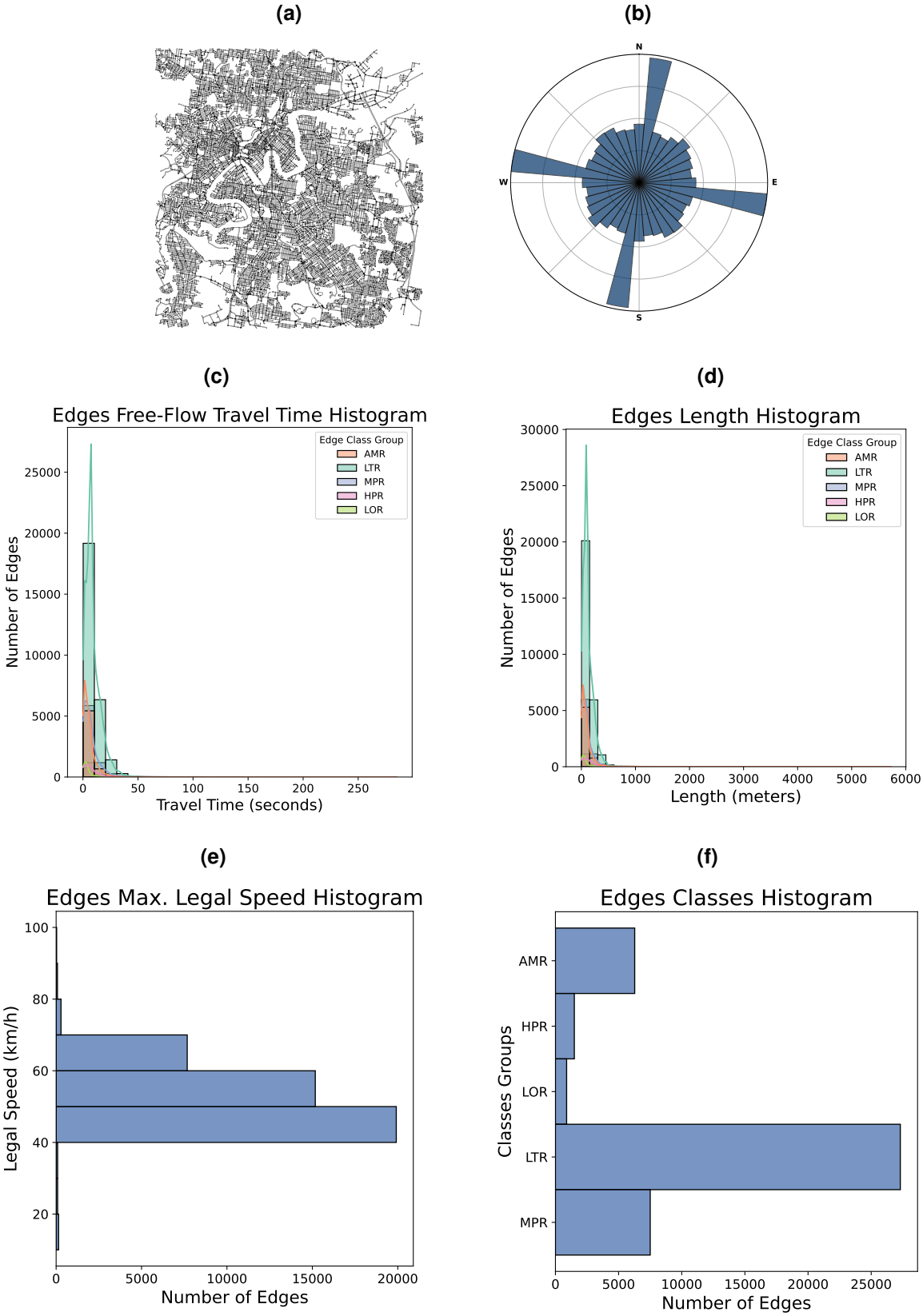
[2] https://github.com/tcervi

Figure 31 – Aalborg (Denmark) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



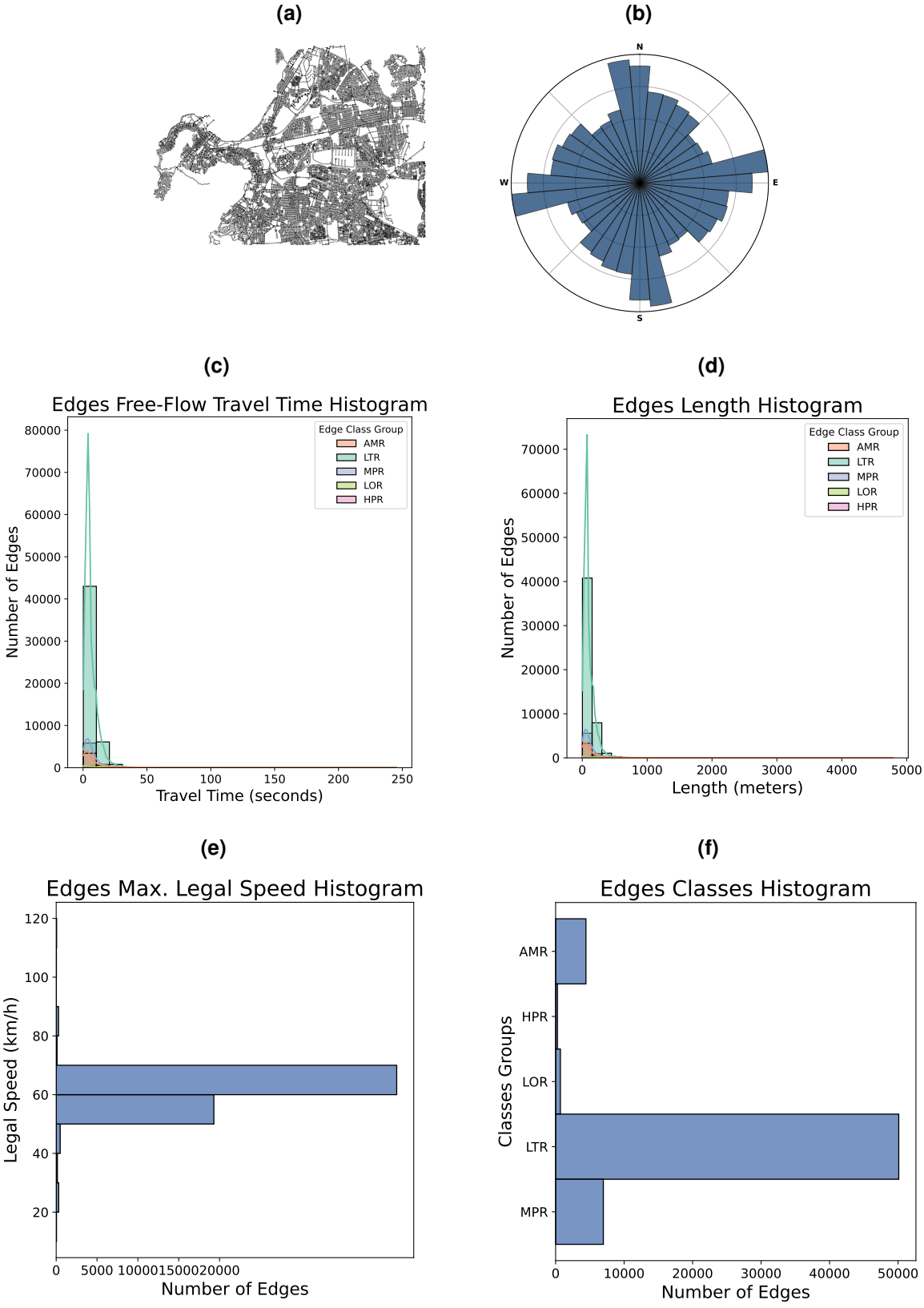
Source: Own work (2025).

Figure 32 – Brisbane (Australia) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



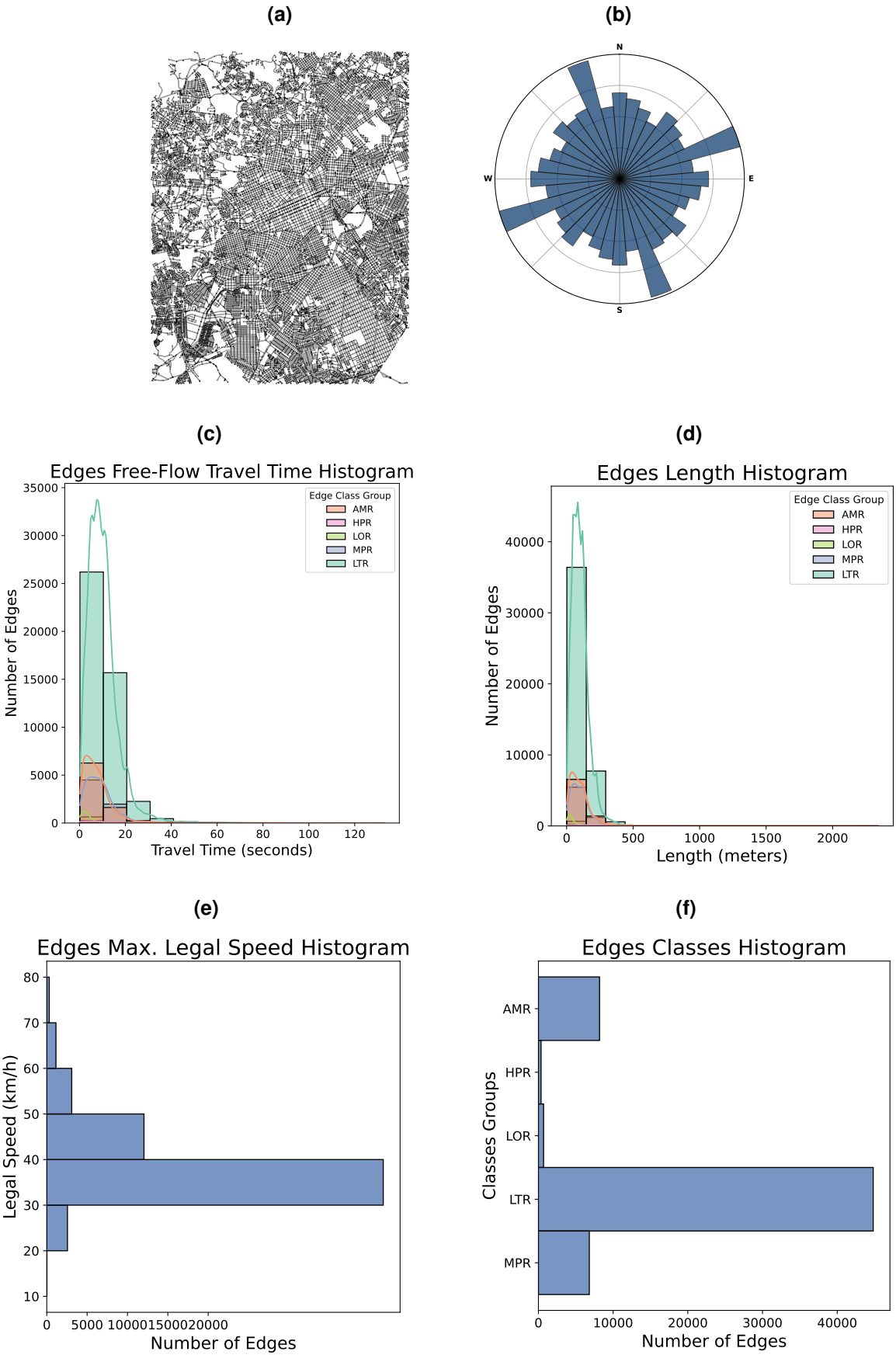
Source: Own work (2025).

Figure 33 – Cape Town (South Africa) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



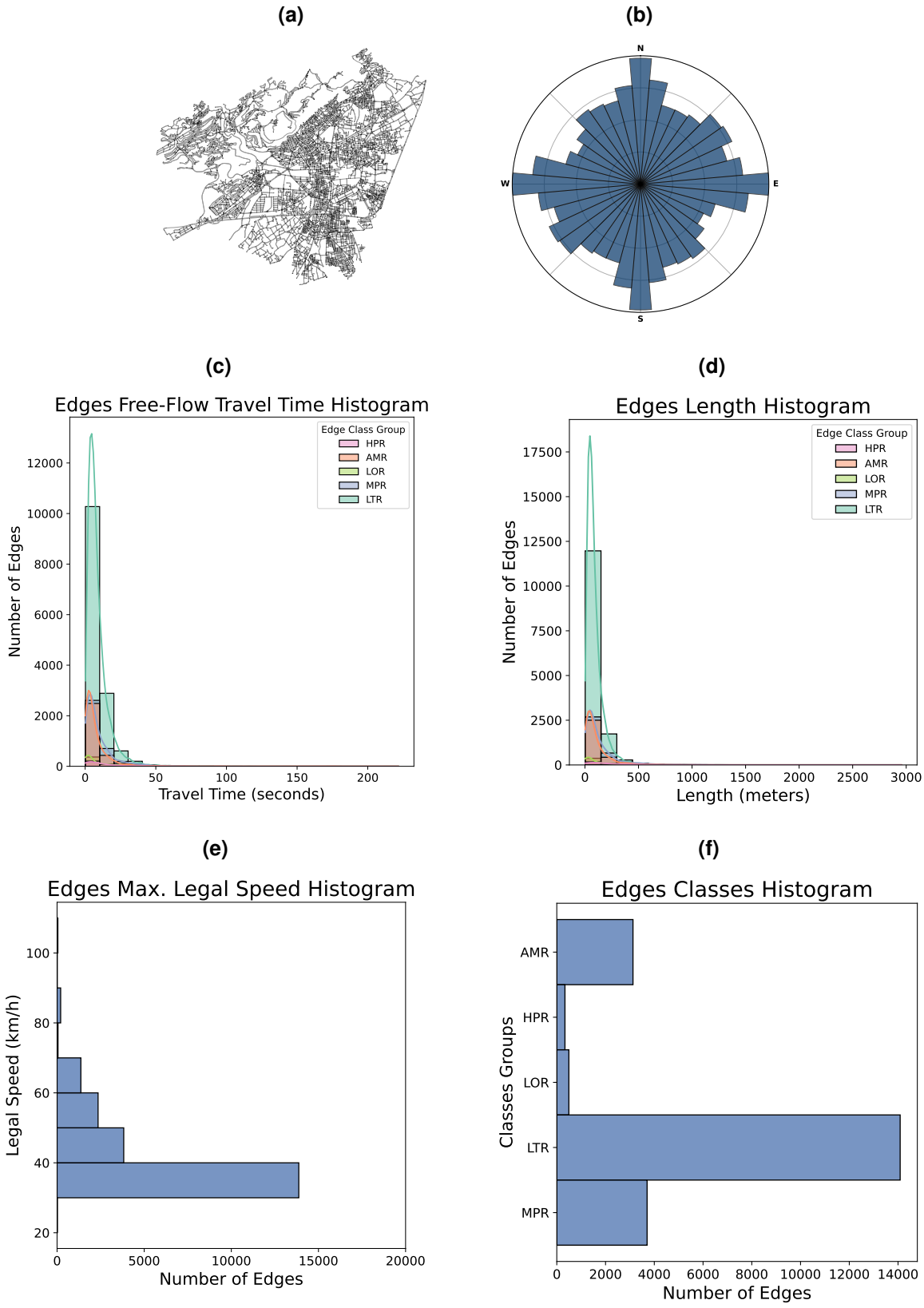
Source: Own work (2025).

Figure 34 – Curitiba (Brazil) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



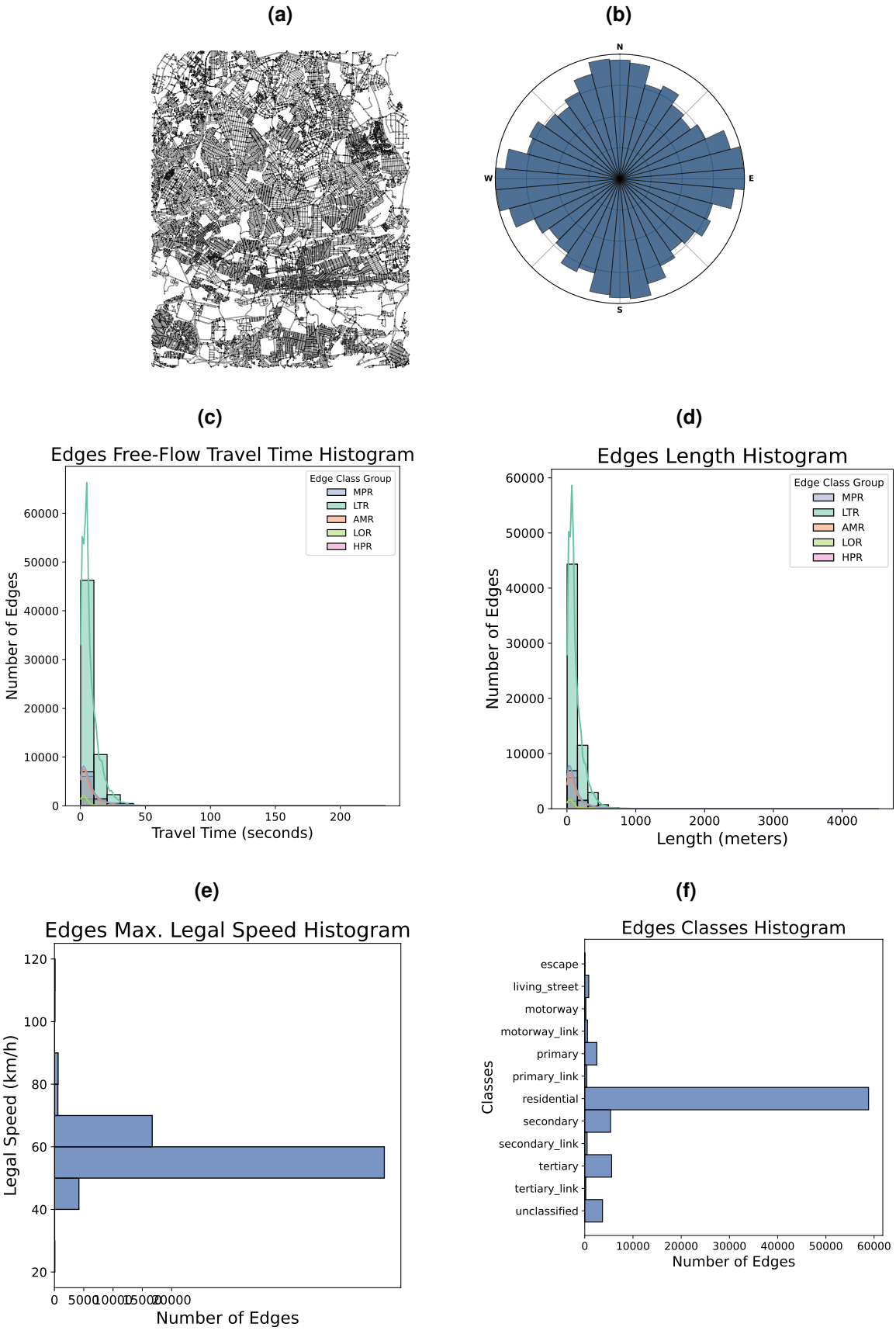
Source: Own work (2025).

Figure 35 – Damascus (Syria) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



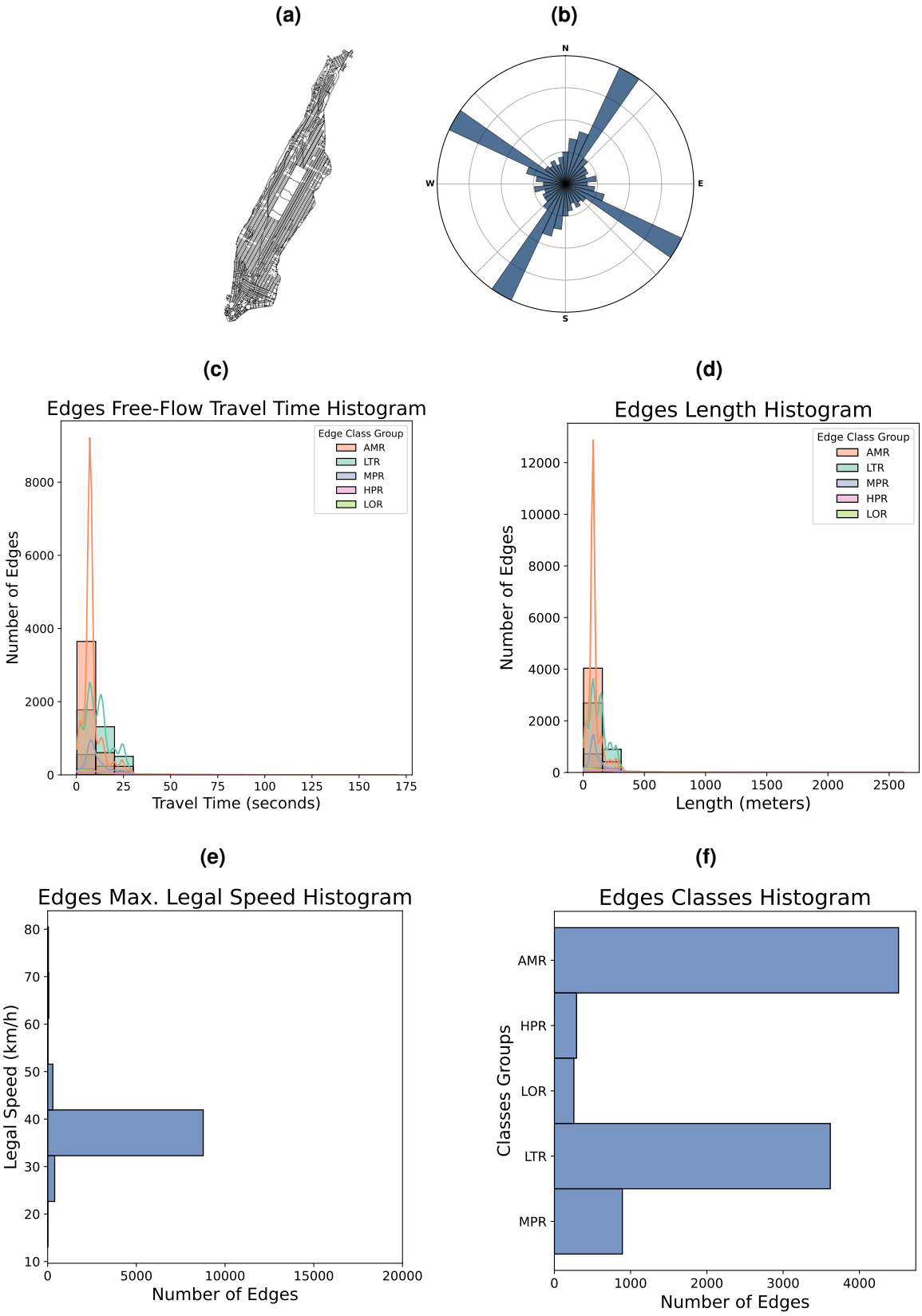
Source: Own work (2025).

Figure 36 – Johannesburg (South Africa) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



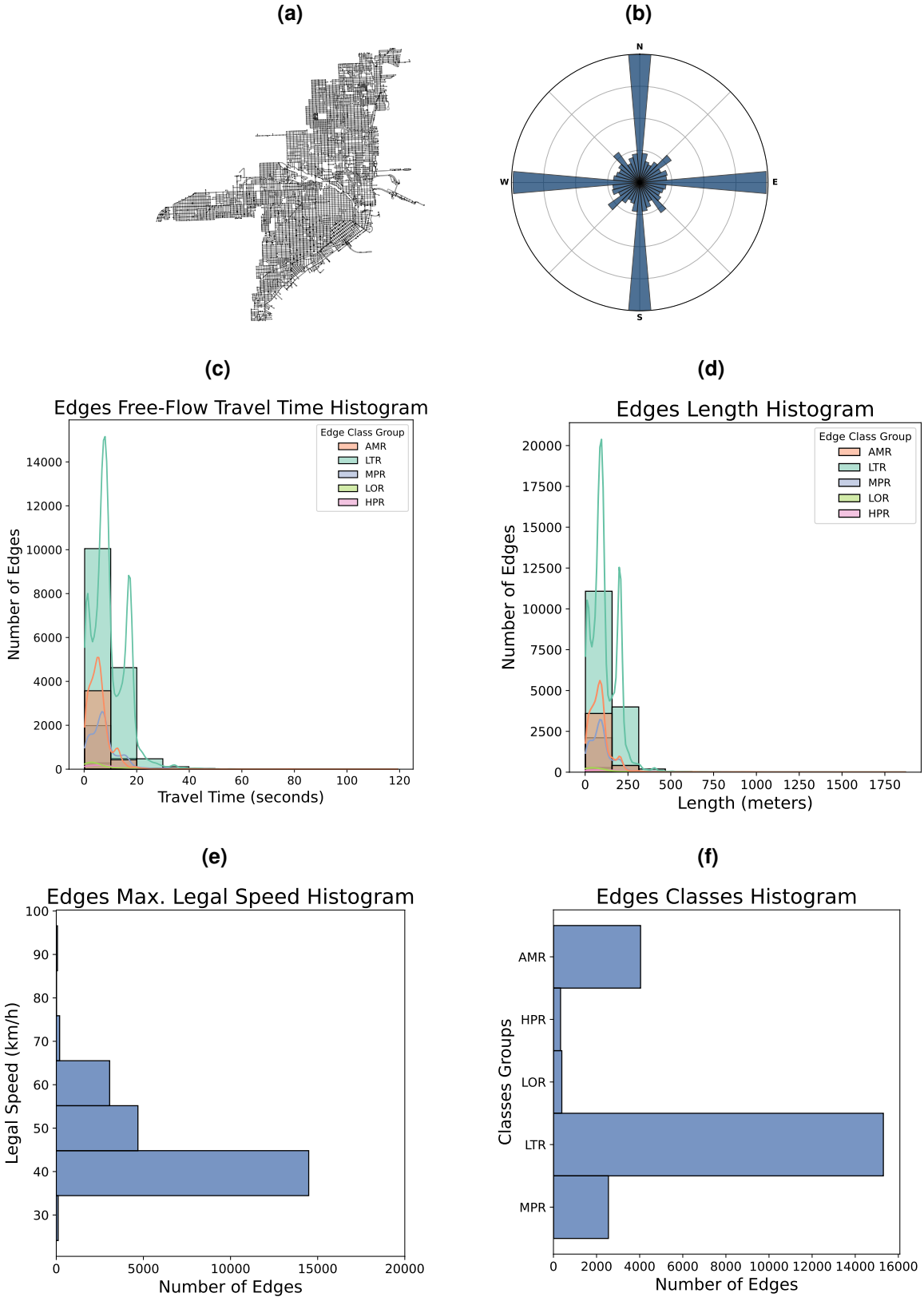
Source: Own work (2025).

Figure 37 – Manhattan Island (United States of America) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



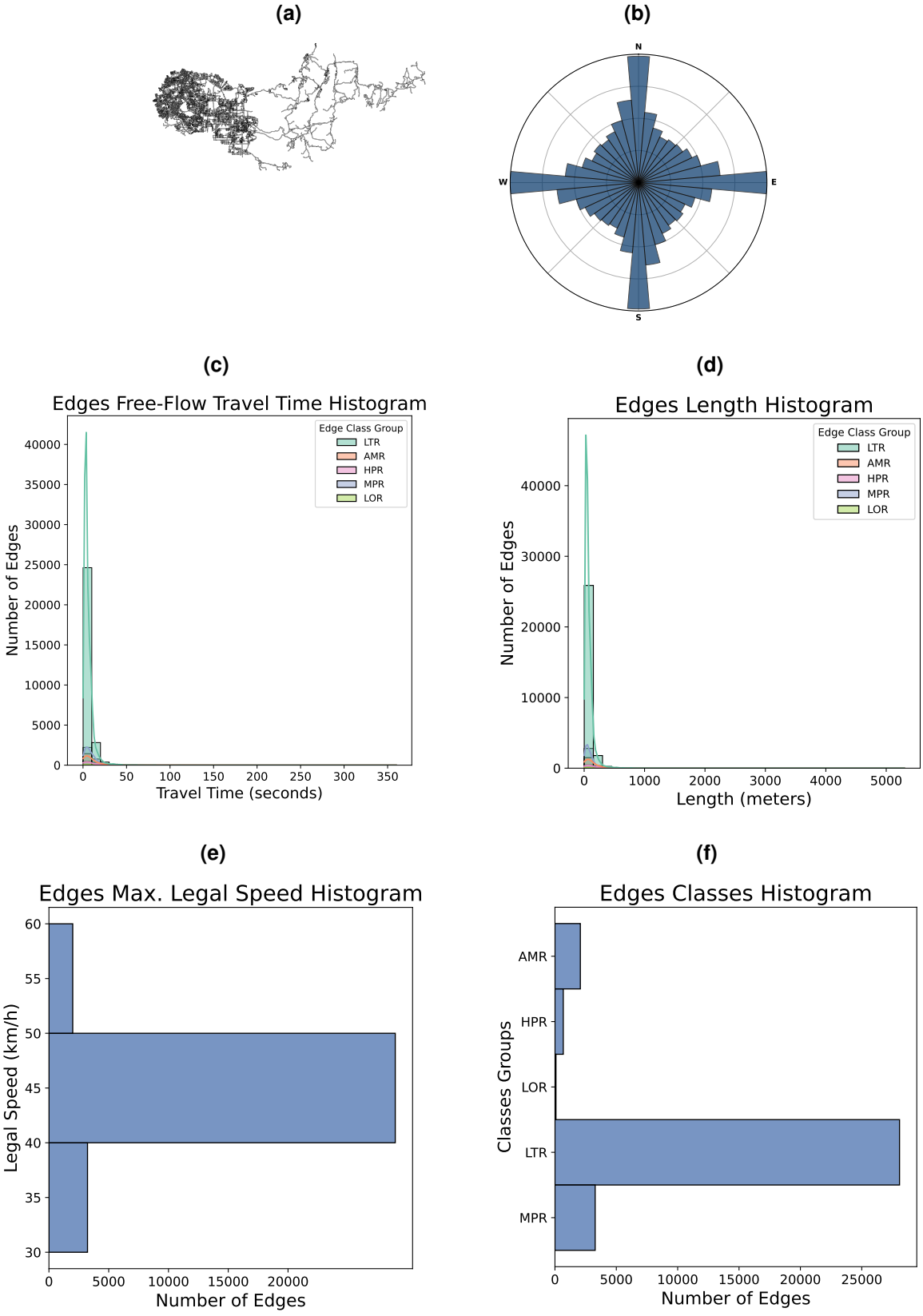
Source: Own work (2025).

Figure 38 – Miami (United States of America) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



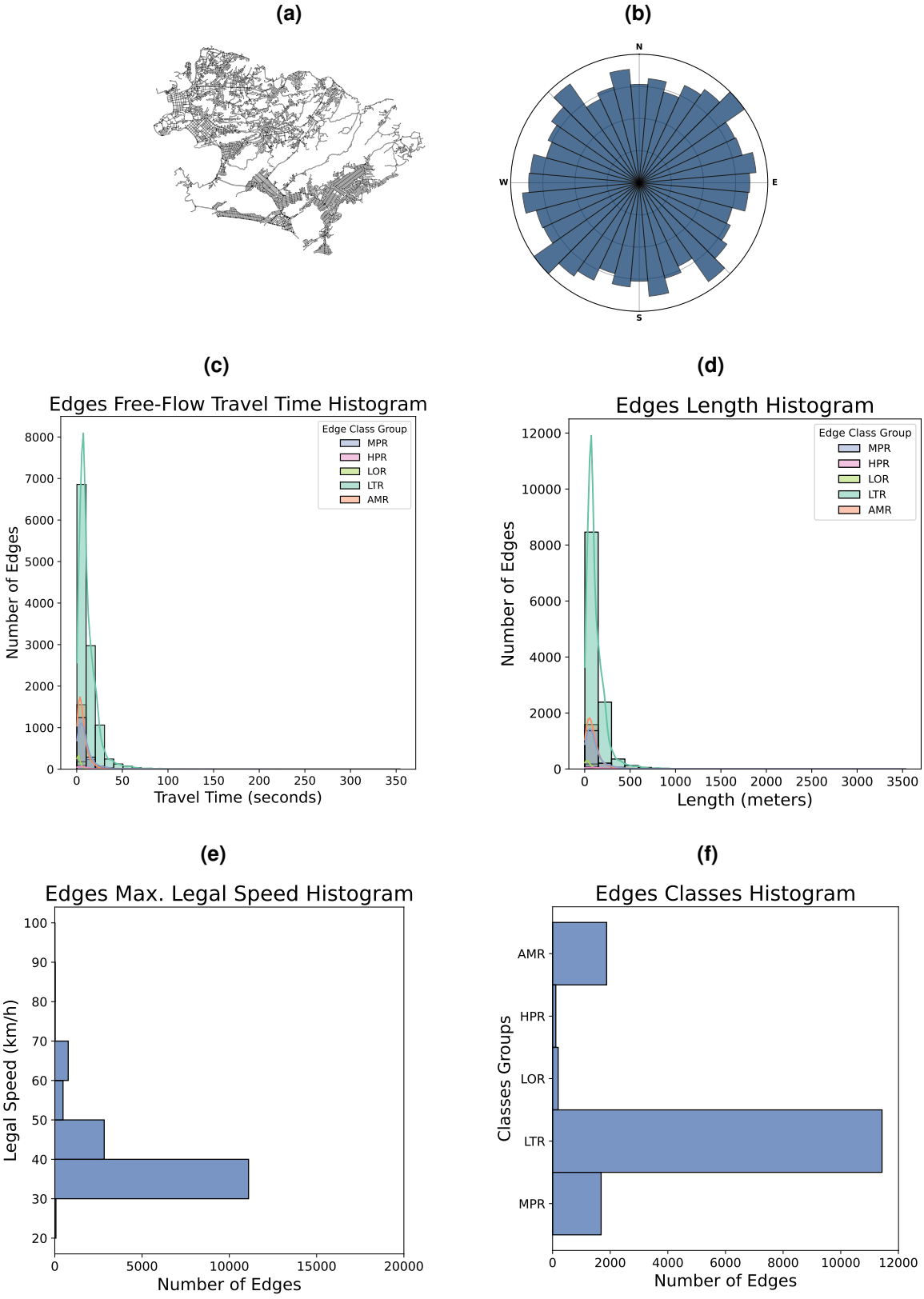
Source: Own work (2025).

Figure 39 – Nara (Japan) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



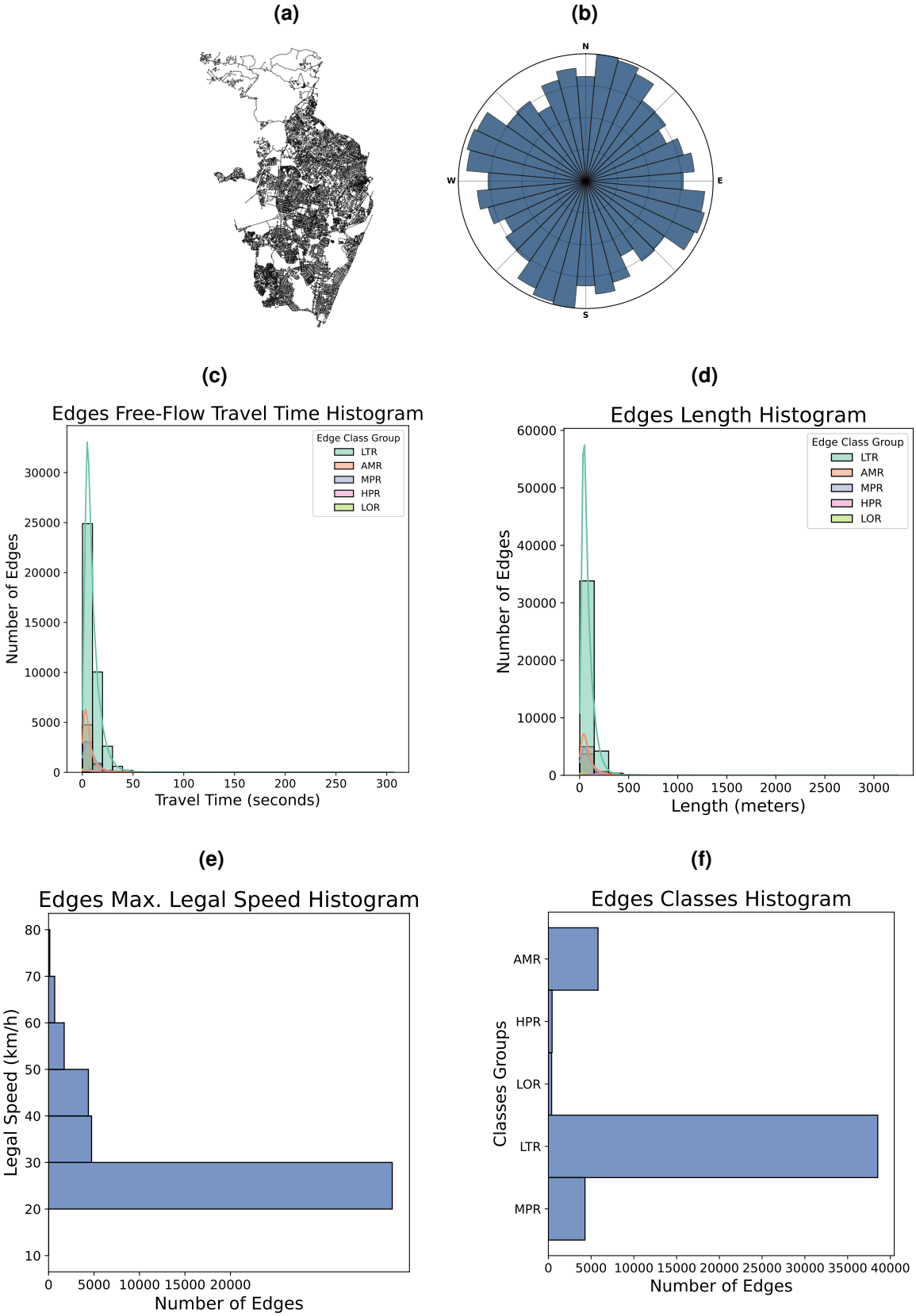
Source: Own work (2025).

Figure 40 – Niterói (Brazil) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



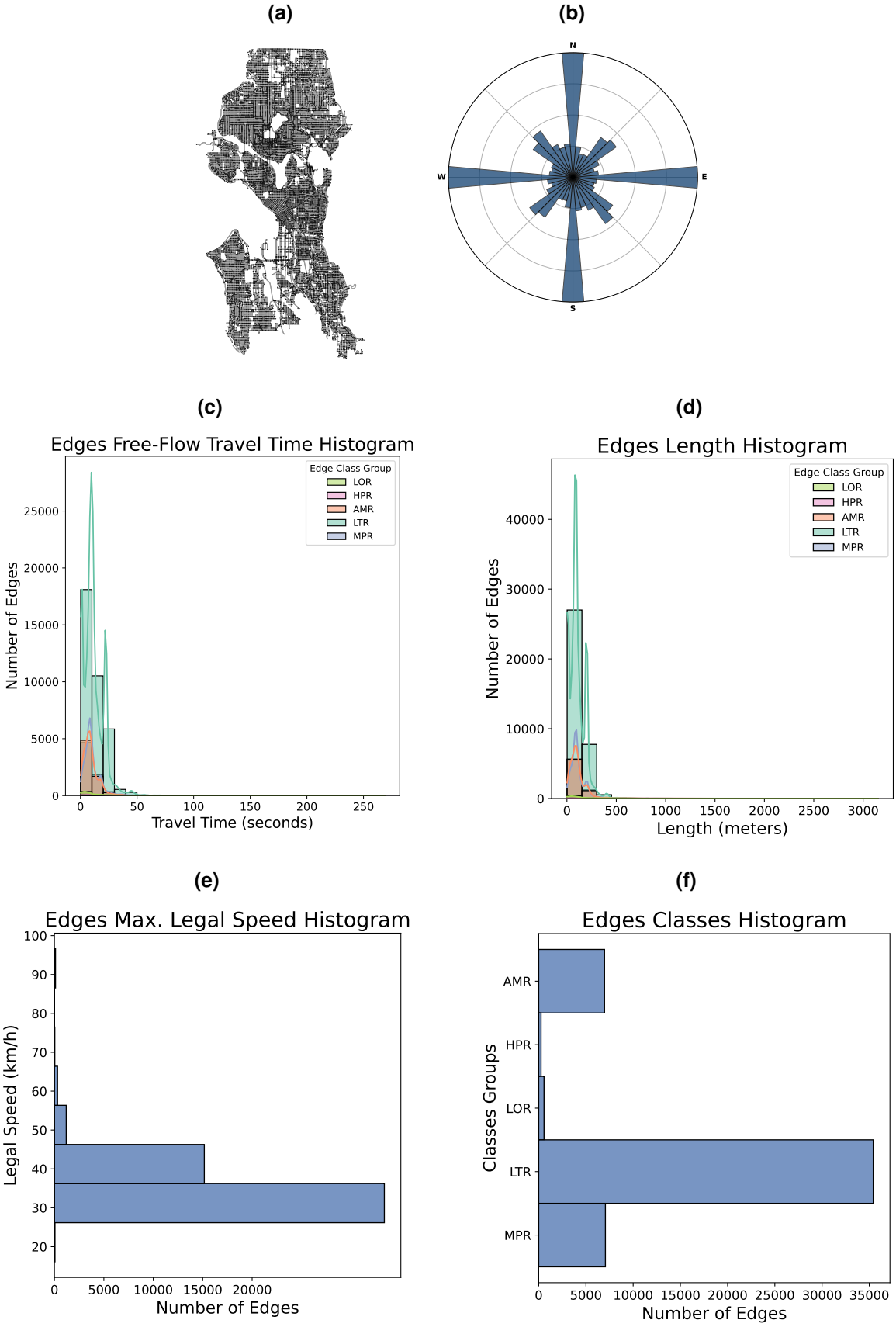
Source: Own work (2025).

Figure 41 – Recife (Brazil) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.



Source: Own work (2025).

Figure 42 – Seattle (United States of America) information: (a) Road network; (b) Edge orientation polar plot; Edge histograms for: (c) Free-flow travel time, (d) Length, (e) Max. Legal Speed and (f) Classes distribution.

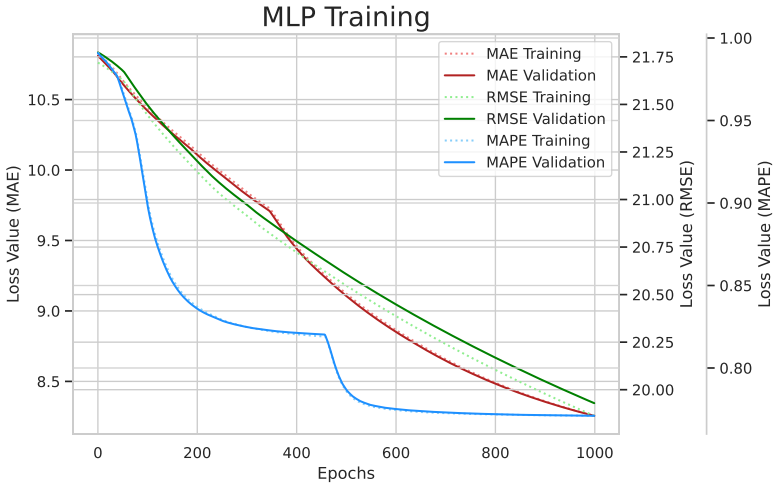


Source: Own work (2025).

APPENDIX B – Training details

This appendix details the training phase of Section 5.2 experiments, presenting figures picturing the learning behaviour of each explored learning architecture. In all figures, the vertical scale on the right corresponds to MAE loss values (in red), the vertical scale on the left corresponds to RMSE loss values (in green) and the outer-left vertical scale corresponds to MAPE loss values (in blue). Each curve shows 1000 training epochs, averaged for all five training rounds. Continuous lines represent the validation loss and dashed lines represent the training loss during the epochs.

Figure 43 – MLP: Full Batch training with Aalborg's Road Network Graph



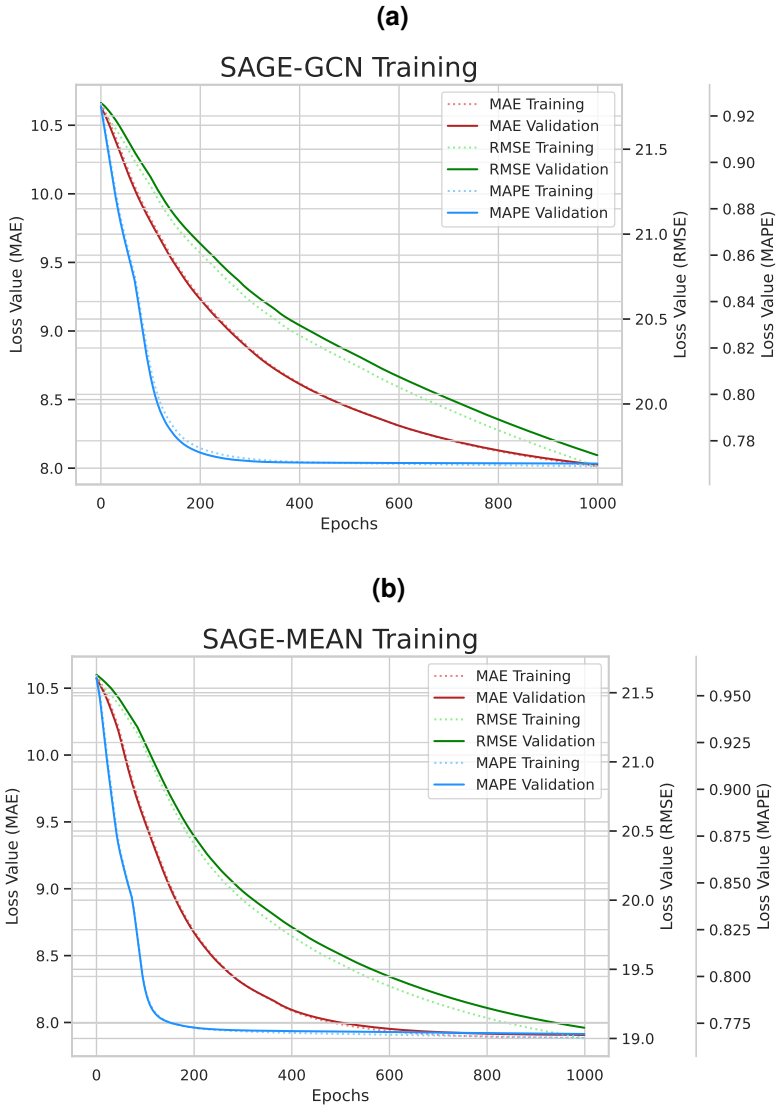
Source: Own work (2025).

It is interesting to note that GraphSAGE-GCN - Figure 44 (a) - presents, for all loss functions, a learning curve quite similar to the MLP learning curve (except for MLP's step on the MAPE validation curve). This is expected since the GCN aggregator is the simplest GraphSAGE variation. It has the same parameter size and performs the same mathematical operations as an MLP, except the message-passing mechanism added to leverage structural information from the graph, i.e. averaging and concatenating neighborhood information. GraphSAGE-Mean, Pool and LSTM variants aggregate neighborhood information with more advanced techniques and the difference in their training curves is clear.

Additionally, GraphSAGE-Pool and GraphSAGE-LSTM variants present a pattern of over-fitting when trained with RMSE and MAPE loss functions. Over-fitting happens when the model's weights and biases are so exaggerated and adapted for decreasing the loss function error values on the training sample that the error on the validation sample, and probably also on the test sample, stabilizes or starts to increase (i.e., prediction performance start to decay for anything outside the training sample). In Figure 45 this behavior appears when a continuous line stabilizes and the correspondent (color) dashed line continues to decrease.

It is important to highlight that the difference between the initial and final loss values for GIN variants is the smallest among all models presented so far, for all loss functions. That is,

Figure 44 – GraphSAGE: Full Batch training with Aalborg's Road Network Graph: (a) GCN and (b) Mean

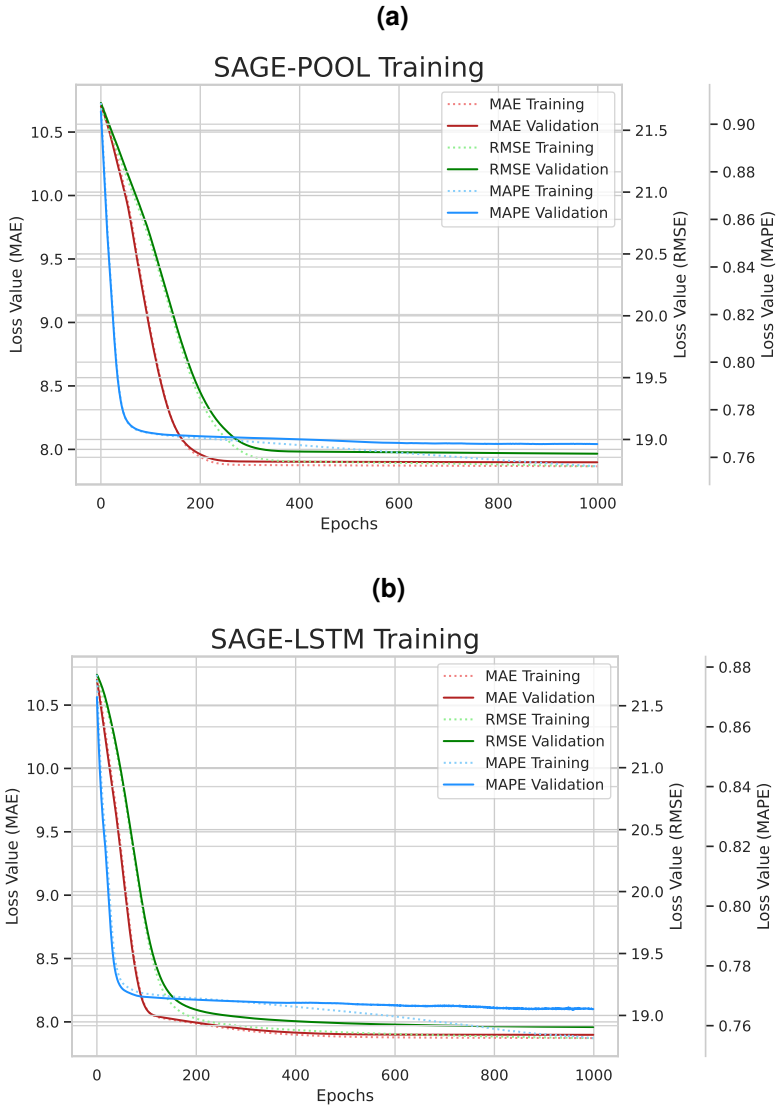


Source: Own work (2025).

trained GIN models cannot reduce significantly any of the loss functions loss values. This is the first indication that this GNN model struggles to learn correct road network representations, even more than the baseline MLP. Additionally, neither of the GIN models presents a training convergence pattern for either training loss functions.

Figures 48 and 49 present the relational fusion networks training curves. RFNs seem to have the highest representational learning power among all tested models, being able to reduce all validation errors to relevantly small values within 1000 training epochs. RFNs with Interactional fusion operators - Figure 48 (a) and (b) - do not present over-fitting patterns and do not converge before 800 epochs for either one of the loss functions. There are no relevant differences between the RFN-AI and RFN-NAI learning curves and the loss values scale, including the convergence

Figure 45 – GraphSAGE: Full Batch training with Aalborg's Road Network Graph: (a) Pool and (b) LSTM

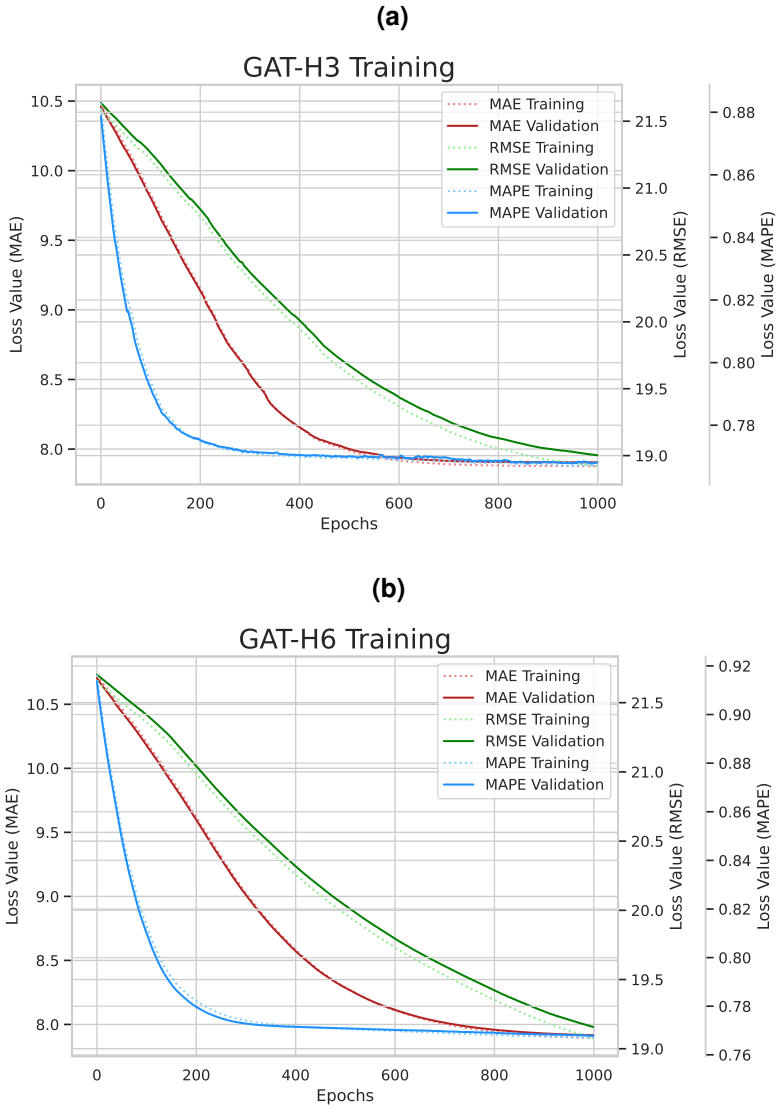


Source: Own work (2025).

value, suggesting that the RFN's attentional aggregator does not enhance the learning capacity of those models for this edge regression task. It is possible to identify learning instability after 300 training epochs, when small oscillations on the validation loss values start to appear. This might be related to the gradient optimization algorithm operation on losses in an already very small scale and to a model with thousands of learnable parameters (Table 10).

The initial learning epochs for interactional RFNs seem to be relevant, since for all loss functions there is a relevant value decrease during the first 200 epochs. It is not clear if this is a huge decrease in learning power or convergence since the absolute loss values are already very small (i.e., below two). For MAPE, the curve towards a steep decrease is even more accentuated and after 600 epochs there is still a small decrease in loss values that are already below 0.2.

Figure 46 – GAT: Full Batch training with Aalborg's Road Network Graph: (a) GAT-h3 and (b) GAT-h6

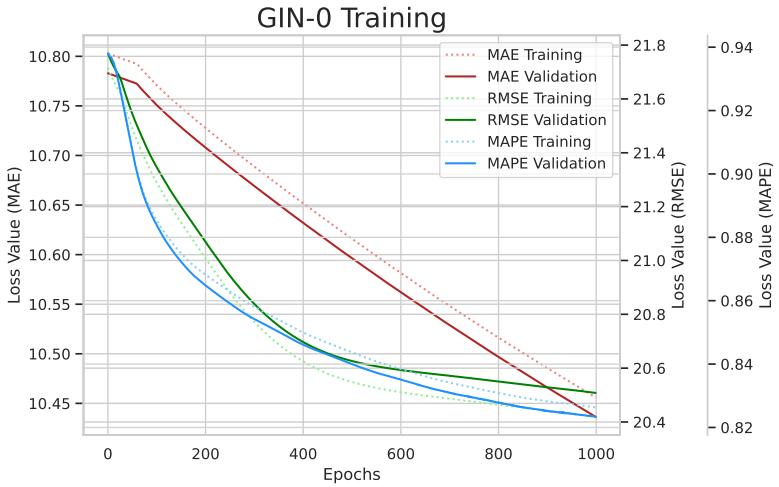


Source: Own work (2025).

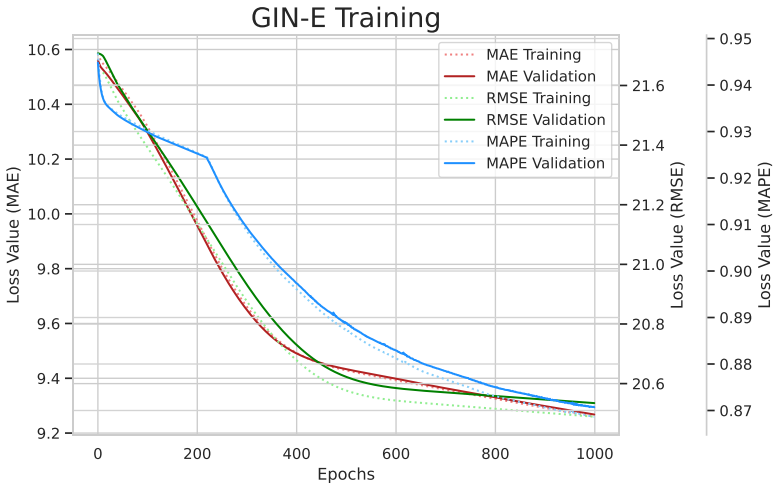
Additive fusion operators - Figure 49 (a) and (b) - on the other hand, do not match the final validation loss of its interactional counterparts for neither loss function. Also, there is no relevant difference between RFN-AA and RFN-NAA learning curves and the loss scale, including the convergence value, suggesting again that the RFN's attentional aggregator does not enhance RFNs' learning capacity for the explored edge regression task. Additive RFNs have about five times fewer learnable parameters than their interactional counterparts (Table 10) and do not reach validation values as small as they do. Yet, they present a more accentuated learning instability pattern after 300 epochs, with notable oscillations in the validation loss while the curve keeps decreasing.

Figure 47 – GIN: Full Batch training with Aalborg's Road Network Graph: (a) GIN-0 and (b) GIN-E

(a)



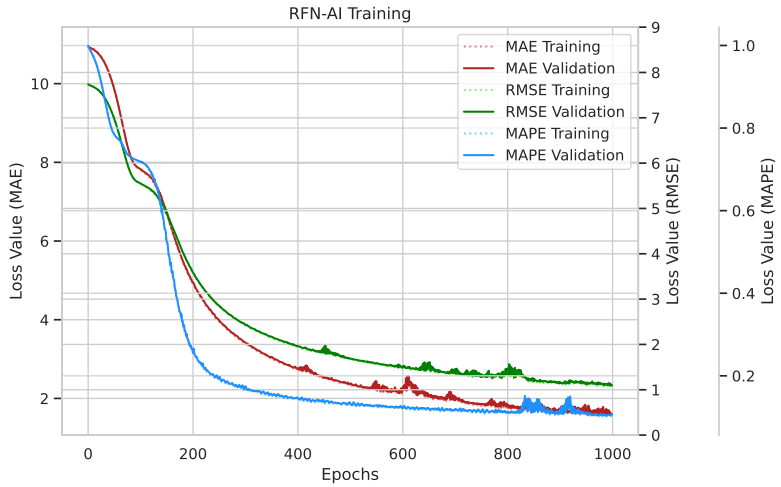
(b)



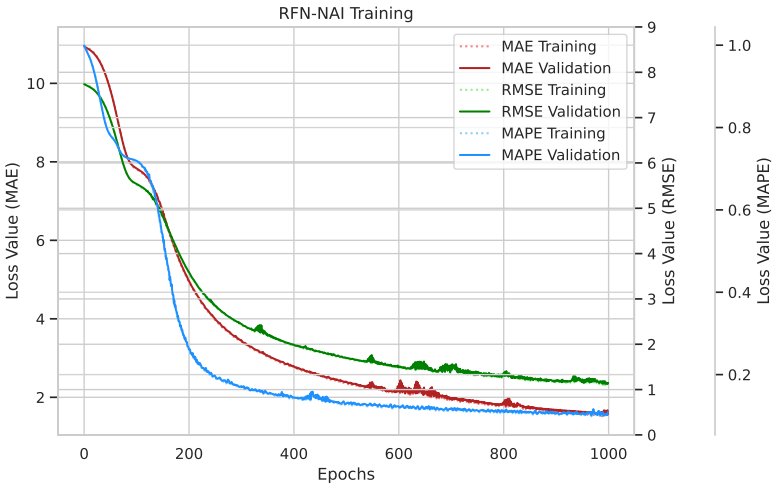
Source: Own work (2025).

Figure 48 – RFN: Full Batch training with Aalborg's Road Network Graph: (a) AI and (b) NAI

(a)

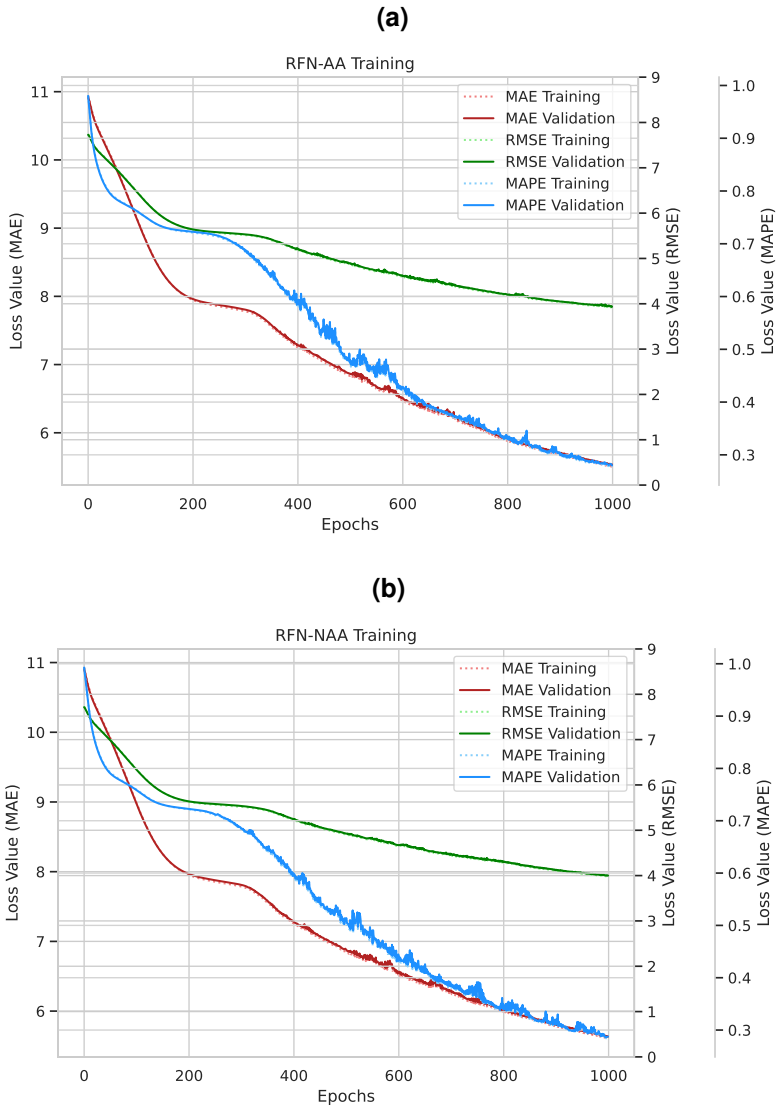


(b)



Source: Own work (2025).

Figure 49 – RFN: Full Batch training with Aalborg's Road Network Graph: (a) AA and (b) NAA



Source: Own work (2025).

APPENDIX C – Models Free-Flow Travel Time (s) predictions

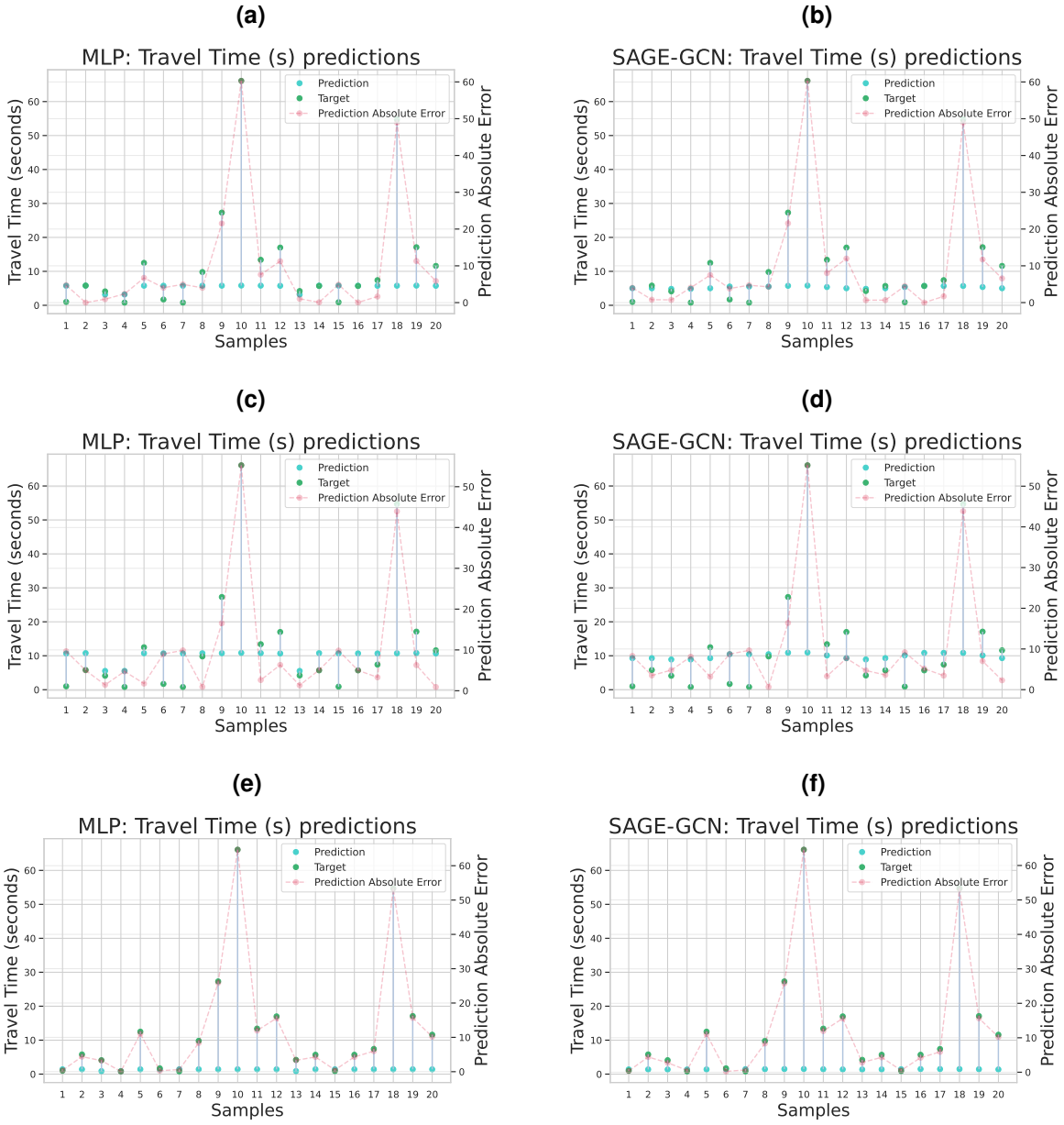
This appendix presents, for all models explored in Section 5.1.3, prediction plots picturing how far each free-flow travel time prediction is from its correspondent truth value (in seconds). All figures in this appendix picture the same sample of the 20 first test predictions with different models, each one trained with MAE, RMSE and MAPE cost functions. Green dots represent the truth values (i.e., prediction targets) and blue dots represent the model prediction for a given edge travel-time, with the scale (in seconds) on the left. The gray line between each pair of target/prediction pictures the real difference (in seconds) between both. Red dots represent the absolute error for each target/pair, with the scale (in seconds) on the right.

It is important to notice the effect of MAPE on model's learning capabilities. Models trained with MAE and RMSE seem to have a "higher absolute value" around which the predictions are distributed, while models trained with MAPE seem to have a "lower absolute value". It happens because MAPE penalizes more the errors for prediction value higher than the truth value in comparison to predictions that have value lower than the truth value[1]. That is why the predictions plot give the impression that models trained with MAPE seem to "approach" truth values from bellow. This characteristic might not be desirable for predicting travel-times, a scenario in which a wrong prediction of early arrival is less desired than a wrong prediction of delayed arrival. Although, for the generic purpose GNNs trained with MAE and RMSE there are several errors of "early arrival" on the test sample as well, since the models are not actually capable of leveraging enough information about the road network.

When analysing the RFN variants, the same characteristic of MAPE trained model appears for models that use additive fusion functions - Figures 54 and 55. Models with interactional fusion function are so capable of learning information about the Road Network Graph that does not really matter which cost function is used for training. Even when predicting outliers values, the error is relevantly low when using interactional RFNs. For this case, of the best suitable models, it might be valuable to use MAPE as the training cost to reduce the overall average percentage error of predictions. Enhancements on the overall percentage error can be seen when comparing Figures 54 and 55 scale on the right vertical axis (i.e., the prediction absolute error value axis).

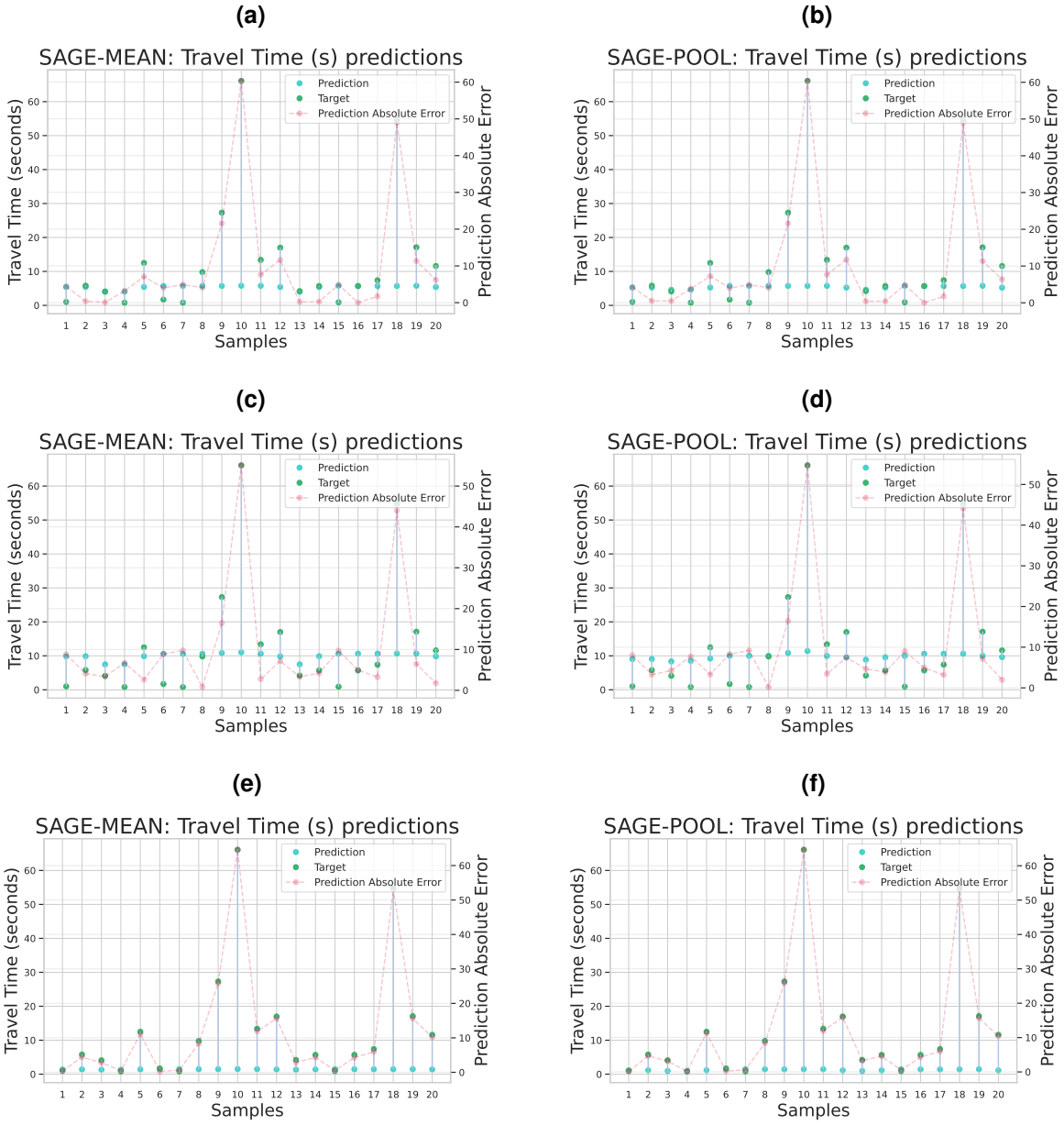
[1] A simple example would be a travel-time prediction of value 7.5 seconds: for a truth value of 10 seconds, MAPE would consider it an error of 25%, while for a truth value of 5 seconds, MAPE would consider it an error of 50%.

Figure 50 – Free-flow travel-time prediction plots - MLP trained with (a) MAE, (c) RMSE and (e) MAPE and GraphSAGE-GCN trained with (b) MAE, (d) RMSE and (f) MAPE



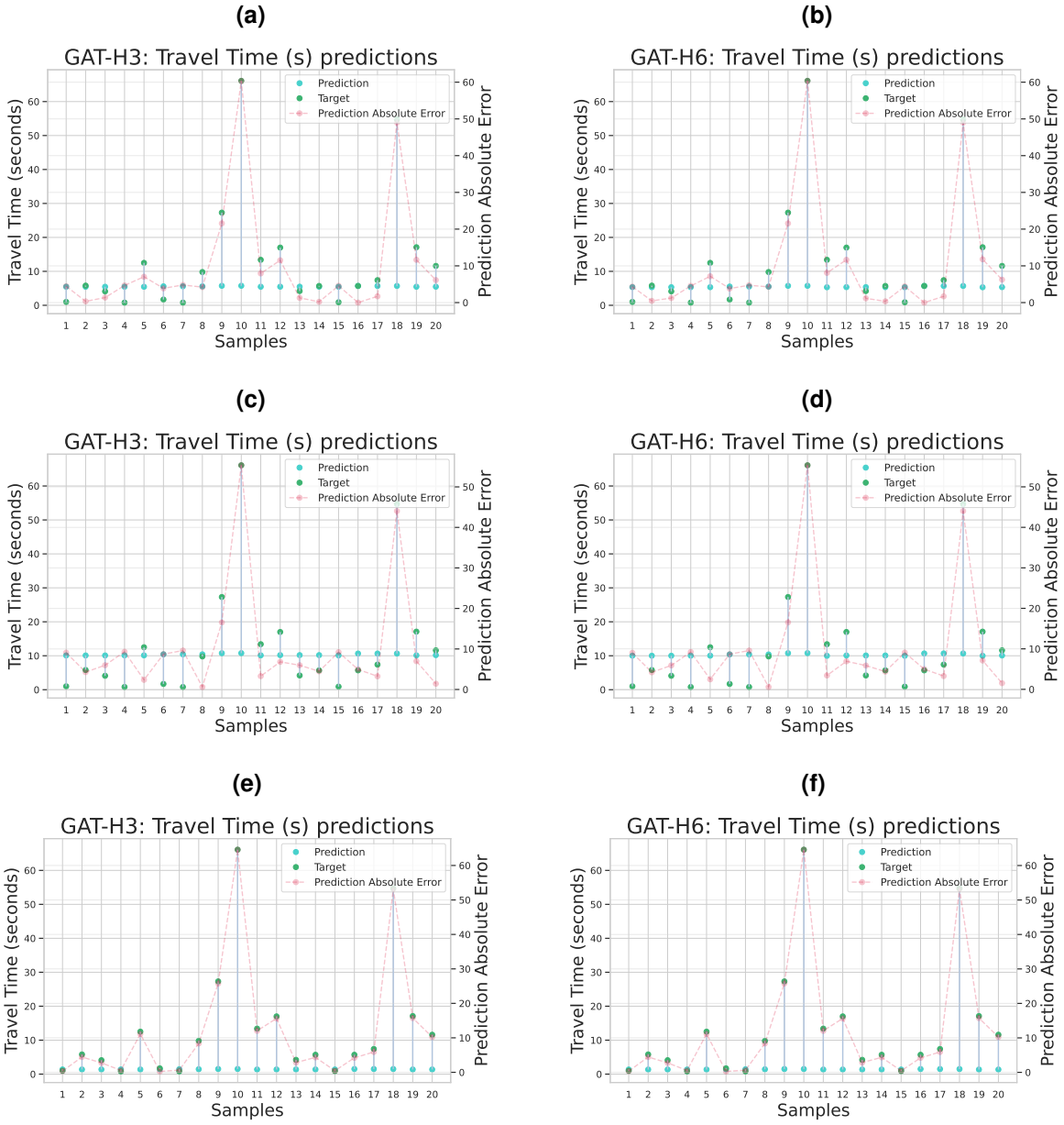
Source: Own work (2025).

Figure 51 – Free-flow travel-time prediction plots - GraphSAGE-MEAN trained with (a) MAE, (c) RMSE and (e) MAPE and GraphSAGE-POOL trained with (b) MAE, (d) RMSE and (f) MAPE



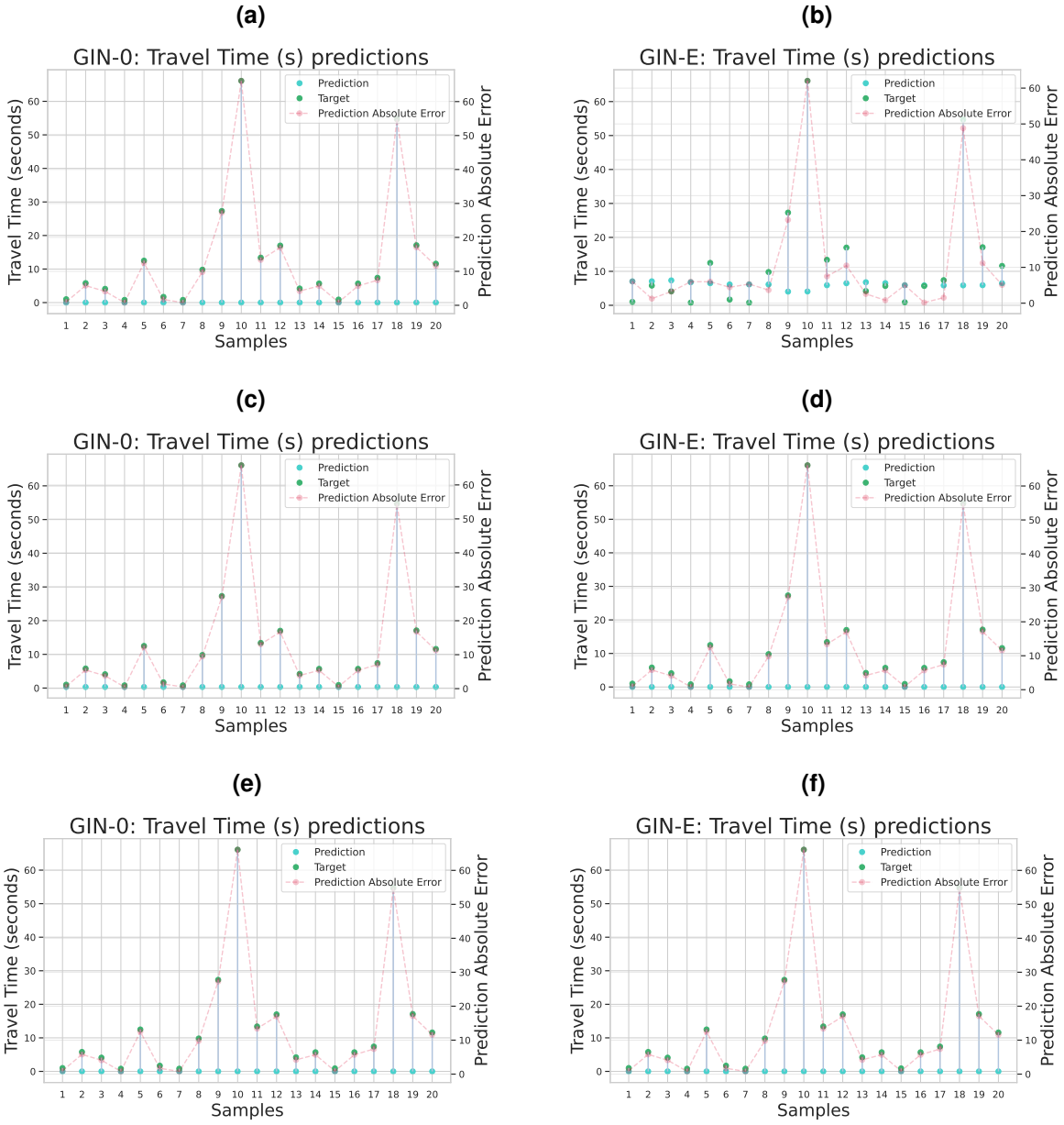
Source: Own work (2025).

Figure 52 – Free-flow travel-time prediction plots - GAT-H3 trained with (a) MAE, (c) RMSE and (e) MAPE and GAT-H6 trained with (b) MAE, (d) RMSE and (f) MAPE



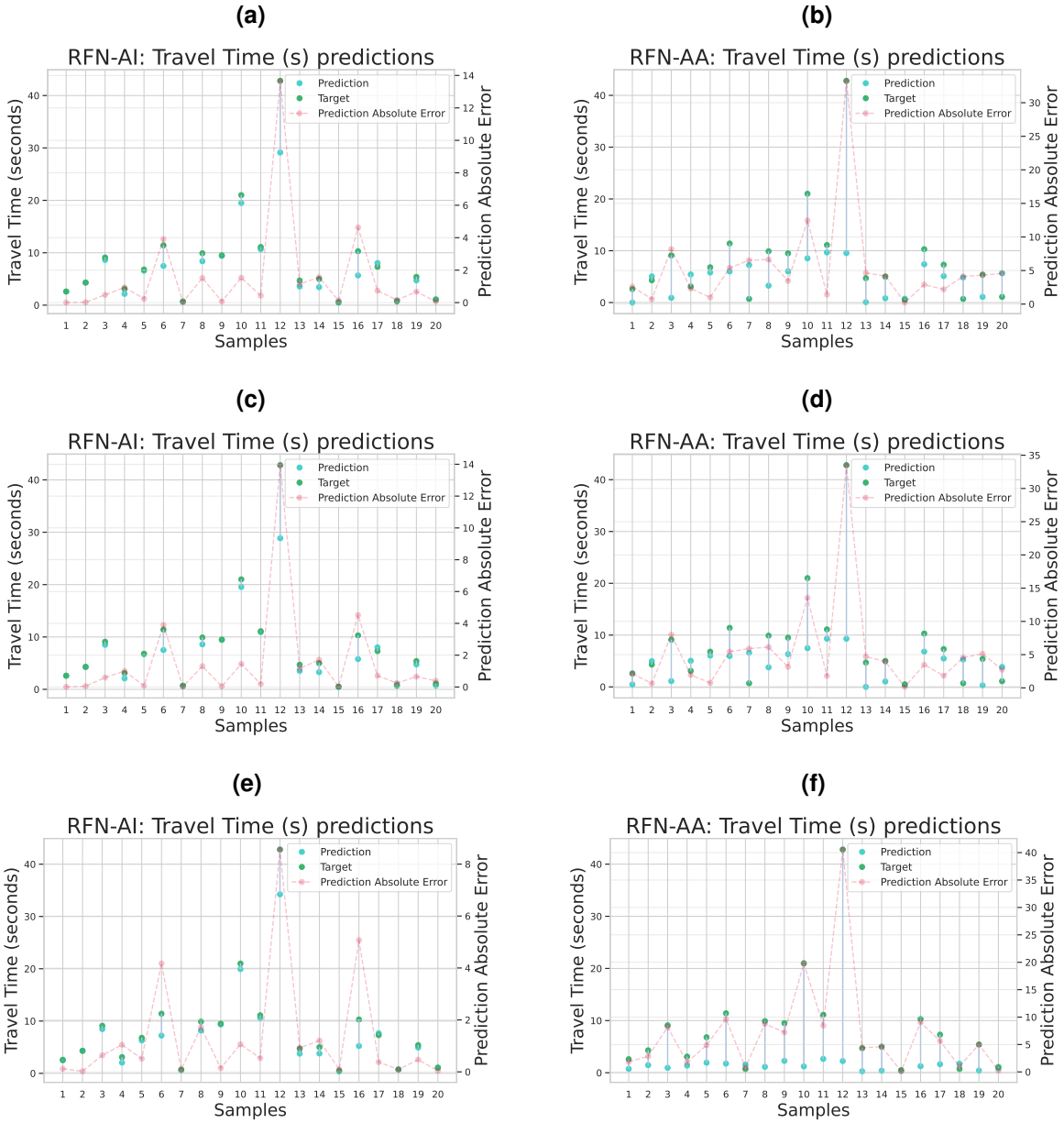
Source: Own work (2025).

Figure 53 – Free-flow travel-time prediction plots - GIN-0 trained with (a) MAE, (c) RMSE and (e) MAPE and GIN-E trained with (b) MAE, (d) RMSE and (f) MAPE



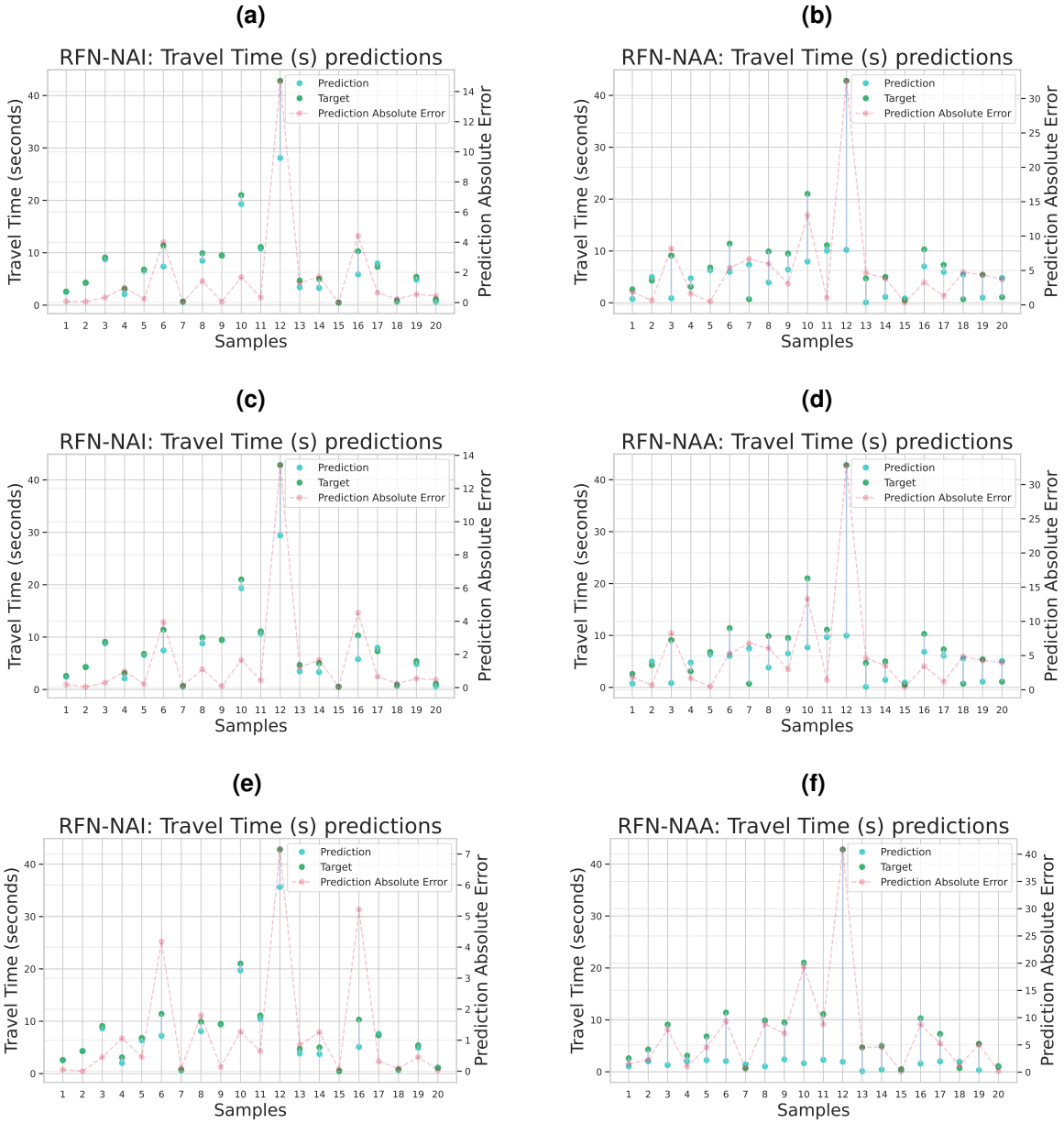
Source: Own work (2025).

Figure 54 – Free-flow travel-time prediction plots - RFN-AI trained with (a) MAE, (c) RMSE and (e) MAPE and RFN-AA trained with (b) MAE, (d) RMSE and (f) MAPE



Source: Own work (2025).

Figure 55 – Free-flow travel-time prediction plots - RFN-NAI trained with (a) MAE, (c) RMSE and (e) MAPE and RFN-NAA trained with (b) MAE, (d) RMSE and (f) MAPE



Source: Own work (2025).

APPENDIX D – Mini-batching details

This appendix presents details of the explored models trained using three loss functions - MAE, RMSE and MAPE - and different sizes of batches listed on Table 13, as part of Section 5.2 experiments.

Training a neural network is an optimization problem aimed at finding network parameters that significantly reduce a cost function typically evaluated on a training sample of a given size. Using the entire training set when computing the cost is a technique known as "batch gradient descent" or "deterministic learning" (GOODFELLOW; BENGIO; COURVILLE, 2016). The need to calculate the average cost of all training samples makes batch training prone to stuck in local minimums and less likely to find a global minimum. Also, since the whole training set is loaded into memory, batch training is expensive in terms of computational resources. The larger the training set, the more expensive is to compute the cost function for all samples. Therefore, the computation cost will grow in direct proportion with the dataset size.

For complex problems, larger datasets are crucial to avoid over-fitting and to attain higher generalization accuracy of neural networks (WILSON; MARTINEZ, 2003). For such cases batch training might not be desirable. The option is to train a model presenting a single example at a time. This type of technique is called "stochastic gradient descent" or "online learning". When using stochastic techniques the gradient optimization takes many steps per epoch, one per data point. So, an epoch of stochastic training will take more time than an epoch of batch training, but it will also take several optimization steps in contrast to the single step that batch training takes per epoch. It is reasonable to expect that with such stochastic sampling, there will be fewer epochs needed for the model to converge.

Wilson and Martinez (2003) compared batch and stochastic techniques and stated that practitioners were choosing batch training under the false assumption that it is faster. Since batch training takes a single step per epoch it estimates the gradient only at the starting point in weight space and thus can not follow curves in the error surface as the stochastic gradient descent does. Also, as the size of the training set increases the batch training is more prone to lose the learning stability. Wilson and Martinez (2003) state that the stochastic technique is usually a better approximation of a true gradient descent optimization and can safely overcome the efficiency problems of batch training when using larger datasets.

Mini-batch gradient descent is a trade-off between stochastic and batch techniques. Instead of using a single training sample per iteration, it calculates the cost function average over a small number of test samples, called mini-batch. The mini-batch (i.e., sample) size is orders of magnitude bigger than one and orders of magnitude smaller than the entire training set size. Mini-batch smooths out some of the training noise, but not all of it. It keeps just enough noise such that the optimization can step out of local minimums, while still keeping the performance benefits of stochastic training. According to Wilson and Martinez (2003), mini-batch sizes are generally driven by the following factors:

- **Hardware utilization**: multi-core architectures are usually underutilized by small batches, so there is a theoretical absolute minimum on the batch size bellow which the

time to process a mini-batch will not decay. Additionally, since typically all the examples in the mini-batch are processed in parallel, the amount of memory required for training scales with the batch size. Finally, some types of hardware achieve better performance with specific sizes of arrays[1].

- **Linear returns**: larger batches provide a more accurate gradient estimate but with less than linear returns.
- **Total runtime**: Smaller batches can offer a regularization effect and generalization error is often best for batches of size one. Training with smaller batches might require smaller learning rates to maintain stability and the total runtime can grow as a result of the increase on the number of steps per epoch.

All figures in this appendix present the training process for all the different mini-batch sizes, each one plotted with a different color, including the first 100 epochs of the batch training from Section 5.1.2. All the three training loss functions are pictured, each on in its own plot. The continues curves are the validation set loss checks on each epochs, while the dotted lines are the training set loss checks on each epoch.

Tables 16, 17, 18 and 19 present all mini-batch training session results, where bold values are the best achieve performance.

Table 16 – Aalborg's travel-time (s) regression test performance - Batch size: 2048

Model	MAE	RMSE	MAPE	Avg. Training Time (seconds)	Avg. Epoch Time (seconds)
MLP	$8.63_{3.20}$	$18.31_{4.91}$	$0.767_{0.032}$	2.4	0.01
SAGE-GCN	$8.56_{2.69}$	$18.31_{4.91}$	$0.770_{0.039}$	4.5	0.02
SAGE-MEAN	$7.88_{2.19}$	$18.30_{4.91}$	$0.766_{0.034}$	4.7	0.02
SAGE-POOL	$7.87_{2.18}$	$18.29_{4.90}$	$0.764_{0.032}$	6.1	0.03
SAGE-LSTM	$7.87_{2.19}$	$18.31_{4.91}$	$0.767_{0.032}$	28.0	0.14
GAT-H3	$7.88_{2.18}$	$18.31_{4.90}$	$0.76_{0.03}$	9.57	0.05
GAT-H6	$7.88_{2.18}$	$18.31_{4.91}$	$0.765_{0.031}$	18.47	0.09
GIN-0	$9.55_{2.51}$	$19.92_{5.99}$	$0.797_{0.114}$	4.9	0.02
GIN-E	$9.33_{3.24}$	$20.34_{6.57}$	$0.876_{0.114}$	5.0	0.02
RFN-AI	$\mathbf{0.88_{0.01}}$	$\mathbf{0.63_{0.01}}$	$\mathbf{0.072_{0.002}}$	571.6	2.85
RFN-NAI	$\mathbf{0.89_{0.02}}$	$\mathbf{0.63_{0.01}}$	$\mathbf{0.076_{0.002}}$	581.0	2.90
RFN-AA	$7.40_{0.07}$	$5.24_{0.05}$	$0.715_{0.002}$	568.2	2.84
RFN-NAA	$7.42_{0.07}$	$5.25_{0.05}$	$0.718_{0.003}$	561.9	2.81

[1] When using GPUs it is common to use power of 2 batch sizes to achieve better runtime.

Table 17 – Aalborg's travel-time (s) regression test performance - Batch size: 1024

Model	MAE	RMSE	MAPE	Avg. Training Time (seconds)	Avg. Epoch Time (seconds)
MLP	$7.88_{2.18}$	$18.31_{4.91}$	$0.768_{0.037}$	3.9	0.02
SAGE-GCN	$7.88_{2.18}$	$18.30_{4.91}$	$0.770_{0.038}$	7.2	0.04
SAGE-MEAN	$7.87_{2.18}$	$18.30_{4.90}$	$0.767_{0.036}$	7.9	0.04
SAGE-POOL	$7.87_{2.18}$	$18.27_{4.93}$	$0.765_{0.032}$	11.1	0.06
SAGE-LSTM	$7.87_{2.19}$	$18.31_{4.90}$	$0.762_{0.026}$	51.9	0.26
GAT-H3	$7.88_{2.18}$	$18.30_{4.91}$	$0.769_{0.035}$	18.1	0.09
GAT-H6	$7.88_{2.18}$	$18.30_{4.91}$	$0.766_{0.032}$	36.9	0.18
GIN-0	$8.85_{3.36}$	$18.92_{4.43}$	$0.922_{0.106}$	9.0	0.05
GIN-E	$9.14_{2.99}$	$20.15_{6.58}$	$0.877_{0.115}$	9.3	0.05
RFN-AI	$\mathbf{0.74_{0.01}}$	$\mathbf{0.52_{0.01}}$	$\mathbf{0.059_{0.002}}$	1141.6	5.71
RFN-NAI	$0.75_{0.01}$	$\mathbf{0.53_{0.01}}$	$\mathbf{0.059_{0.001}}$	1127.9	5.64
RFN-AA	$6.25_{0.09}$	$4.44_{0.05}$	$0.671_{0.008}$	1112.6	5.56
RFN-NAA	$3.56_{3.0365}$	$4.48_{0.04}$	$0.677_{0.007}$	1095.1	5.48

Table 18 – Aalborg's travel-time (s) regression test performance - Batch size: 512

Model	MAE	RMSE	MAPE	Avg. Training Time (seconds)	Avg. Epoch Time (seconds)
MLP	$7.87_{2.18}$	$18.31_{4.92}$	$0.772_{0.036}$	6.7	0.03
SAGE-GCN	$7.87_{2.19}$	$18.30_{4.91}$	$0.769_{0.036}$	13.3	0.07
SAGE-MEAN	$7.87_{2.18}$	$18.31_{4.91}$	$0.765_{0.031}$	14.2	0.07
SAGE-POOL	$7.86_{2.18}$	$18.28_{4.91}$	$0.764_{0.032}$	21.2	0.11
SAGE-LSTM	$7.86_{2.18}$	$18.30_{4.92}$	$0.760_{0.026}$	102.6	0.51
GAT-H3	$7.873_{2.189}$	$18.30_{4.91}$	$0.769_{0.034}$	33.1	0.17
GAT-H6	$7.87_{2.18}$	$18.30_{4.92}$	$0.76_{0.03}$	67.8	0.34
GIN-0	$7.86_{2.17}$	$19.33_{6.19}$	$0.778_{0.028}$	13.7	0.07
GIN-E	$7.86_{2.19}$	$18.98_{5.28}$	$0.816_{0.105}$	14.6	0.07
RFN-AI	$\mathbf{0.69_{0.01}}$	$\mathbf{0.50_{0.01}}$	$\mathbf{0.055_{0.001}}$	2331.5	11.66
RFN-NAI	$\mathbf{0.68_{0.01}}$	$\mathbf{0.50_{0.01}}$	$\mathbf{0.055_{0.001}}$	2300.8	11.50
RFN-AA	$1.4868_{0.24}$	$1.58_{0.06}$	$0.497_{0.286}$	2310.1	11.55
RFN-NAA	$1.3656_{0.01}$	$1.44_{0.01}$	$0.379_{0.012}$	2296.6	11.48

Table 19 – Aalborg's travel-time (s) regression test performance - Batch size: 256

Model	MAE	RMSE	MAPE	Avg. Training Time (seconds)	Avg. Epoch Time (seconds)
MLP	$7.87_{2.18}$	$18.32_{4.91}$	$0.770_{0.037}$	12.2	0.06
SAGE-GCN	$7.87_{2.19}$	$18.30_{4.92}$	$0.816_{0.108}$	25.2	0.13
SAGE-MEAN	$7.87_{2.18}$	$18.29_{4.91}$	$0.766_{0.029}$	27.8	0.13
SAGE-POOL	$7.86_{2.19}$	$18.24_{4.92}$	$0.760_{0.025}$	41.2	0.21
SAGE-LSTM	$7.86_{2.19}$	$18.28_{4.94}$	$0.762_{0.029}$	205.2	1.02
GAT-H3	$7.87_{2.19}$	$18.29_{4.92}$	$0.766_{0.031}$	63.5	0.32
GAT-H6	$7.88_{2.18}$	$18.30_{4.92}$	$0.766_{0.031}$	134.0	0.67
GIN-0	$7.99_{2.09}$	$18.71_{5.54}$	$0.767_{0.032}$	26.2	0.13
GIN-E	$7.87_{2.20}$	$18.27_{4.95}$	$0.772_{0.030}$	27.7	0.14
RFN-AI	$\mathbf{0.69_{0.01}}$	$\mathbf{0.49_{0.01}}$	$\mathbf{0.053_{0.000}}$	4751.7	23.76
RFN-NAI	$\mathbf{0.69_{0.01}}$	$\mathbf{0.49_{0.01}}$	$\mathbf{0.054_{0.000}}$	4694.8	23.47
RFN-AA	$1.51_{0.37}$	$1.03_{0.17}$	$0.232_{0.197}$	4589.4	22.94
RFN-NAA	$1.37_{0.01}$	$0.97_{0.01}$	$0.083_{0.003}$	4504.4	22.52

Figure 56 – MLP mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

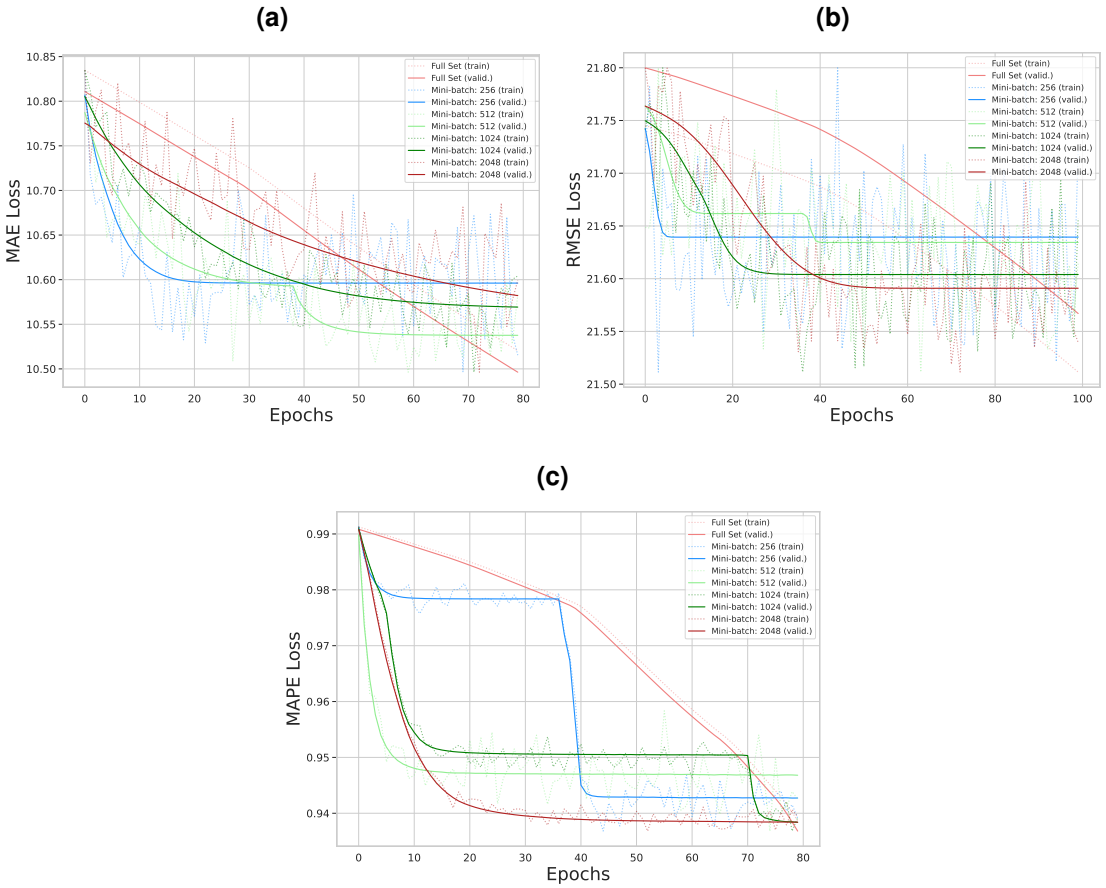


Figure 57 – GraphSAGE-GCN mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

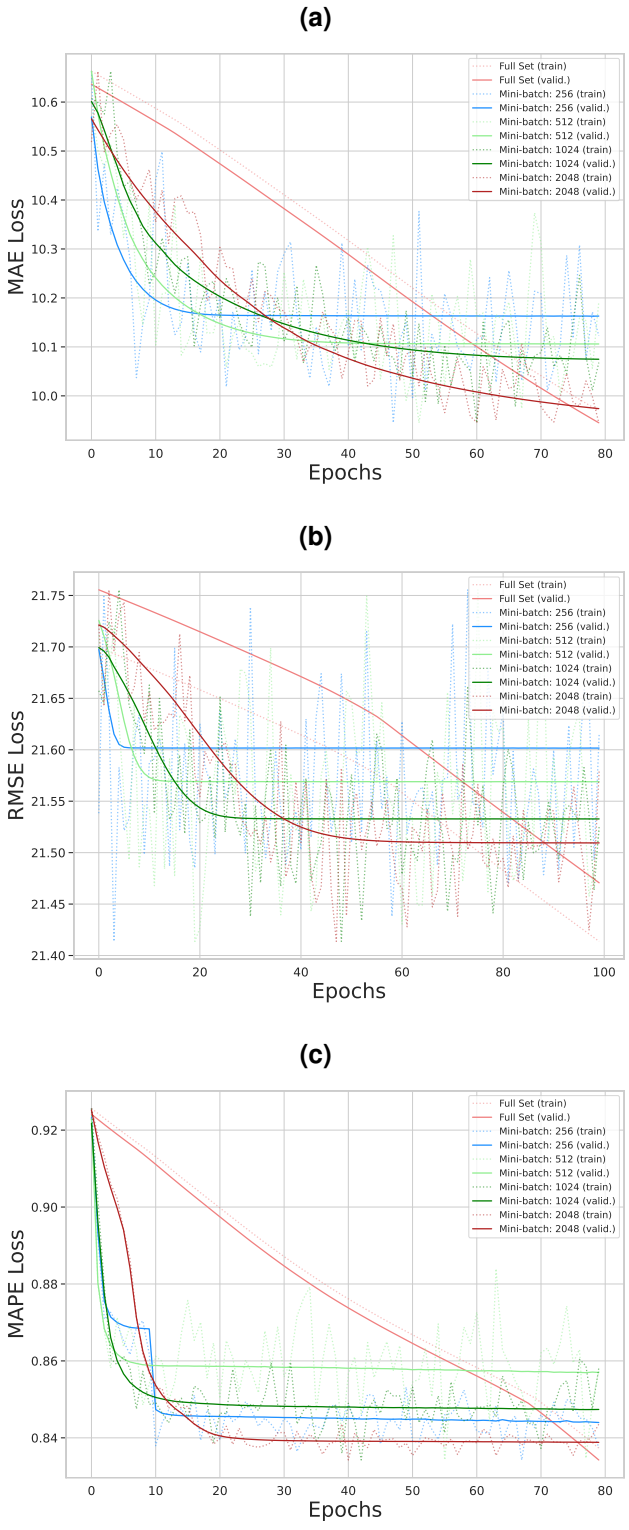


Figure 58 – GraphSAGE-MEAN mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

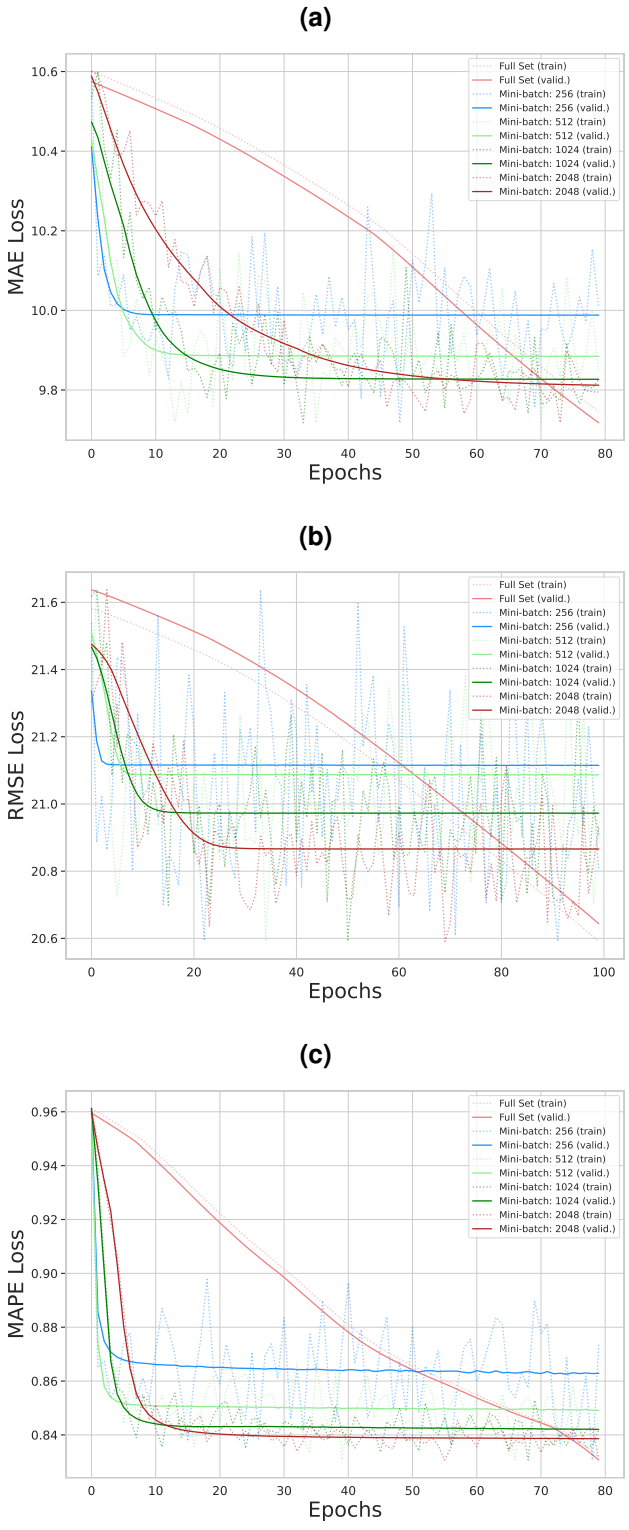


Figure 59 – GraphSAGE-POOL mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

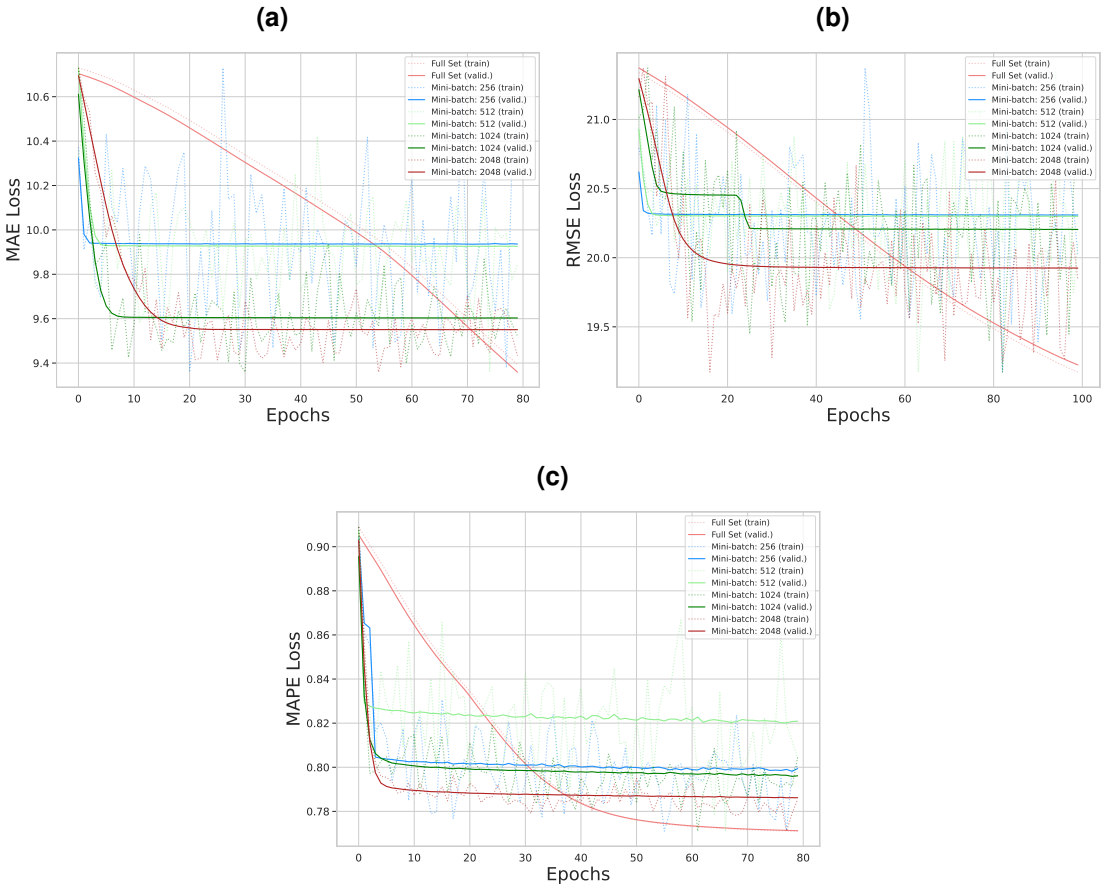


Figure 60 – GraphSAGE-LSTM mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

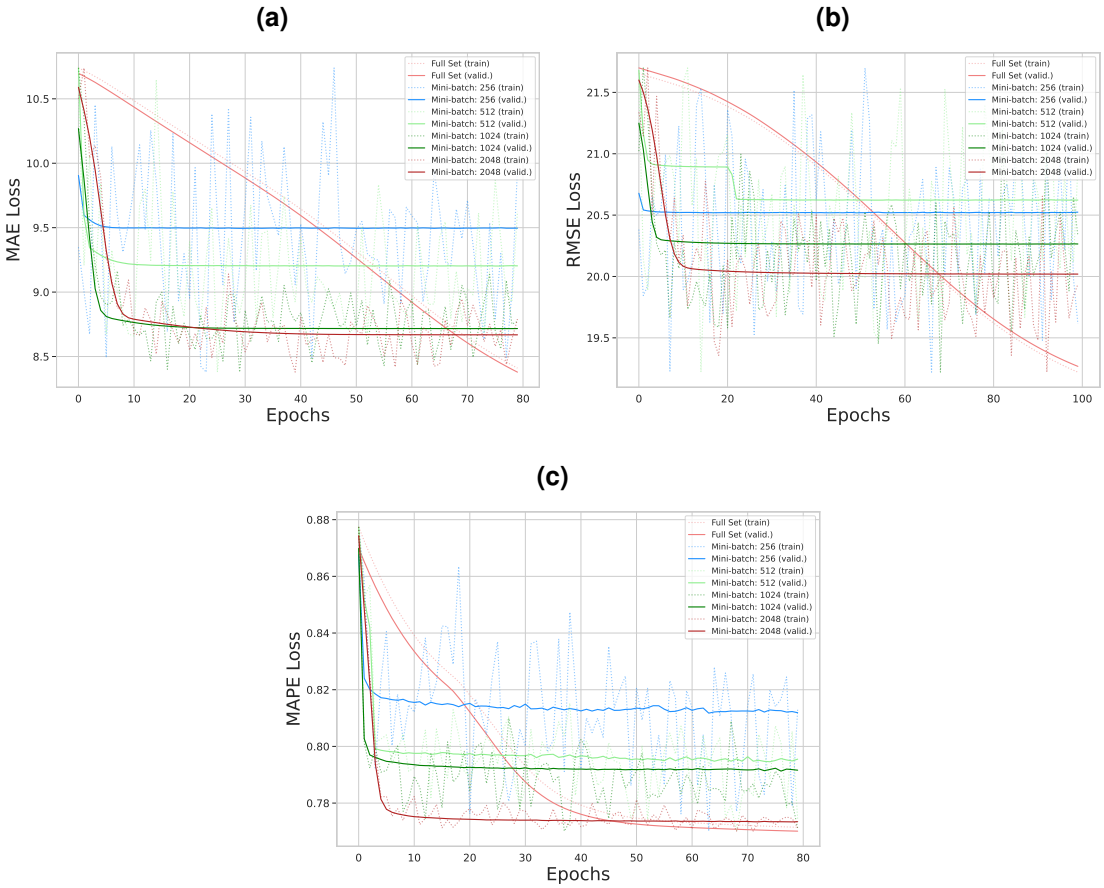


Figure 61 – GAT-H3 mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

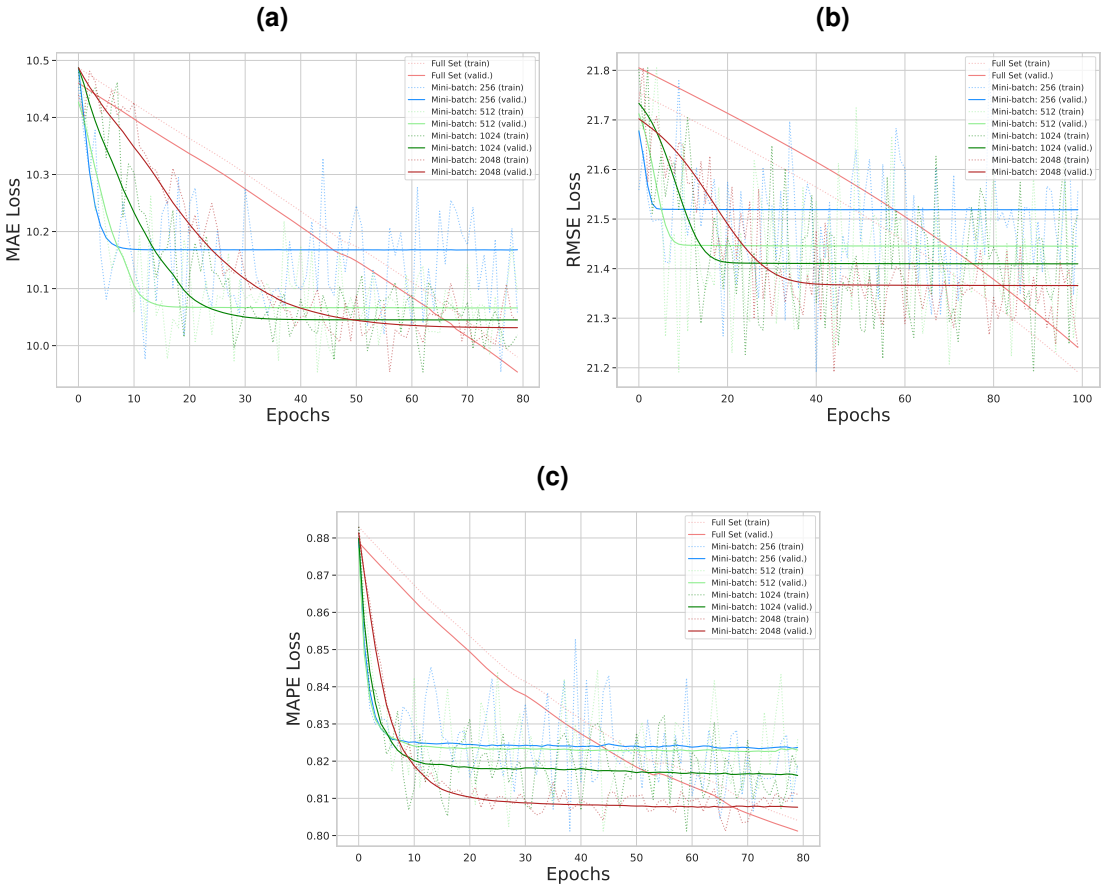


Figure 62 – GAT-H6 mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

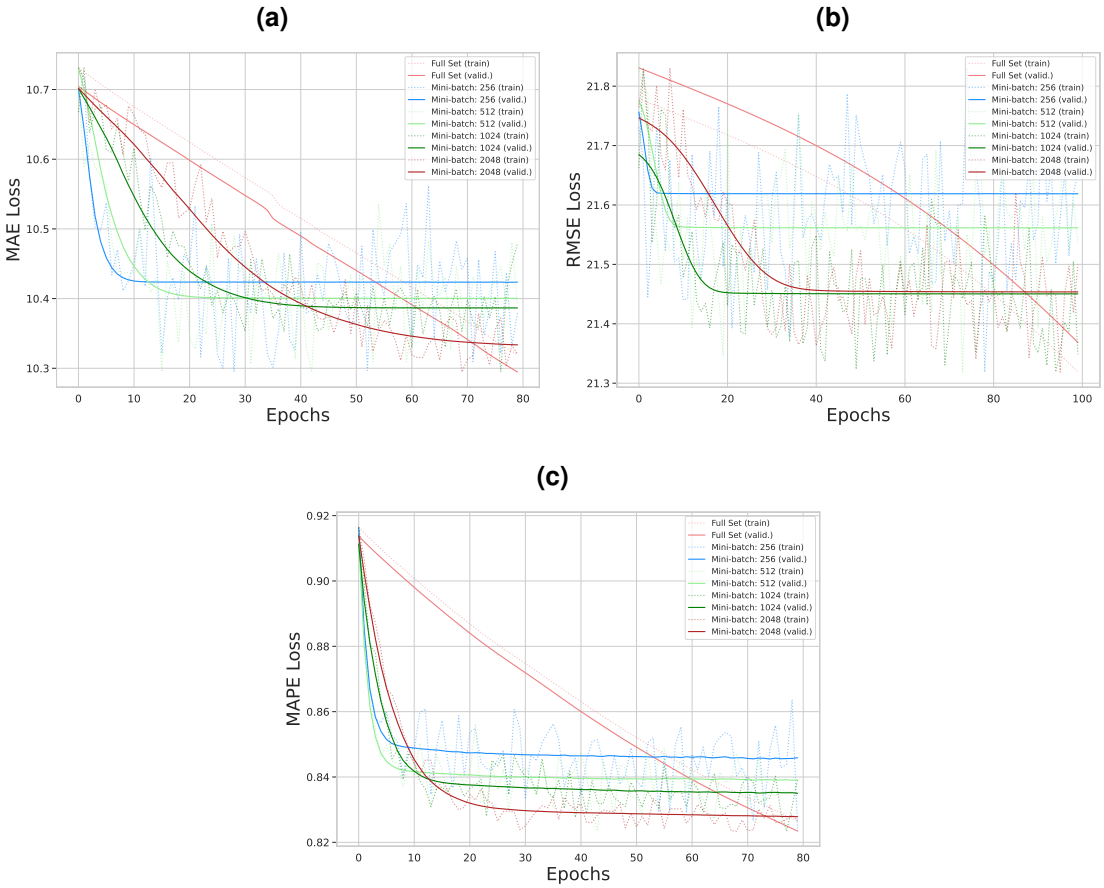


Figure 63 – GIN-0 mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

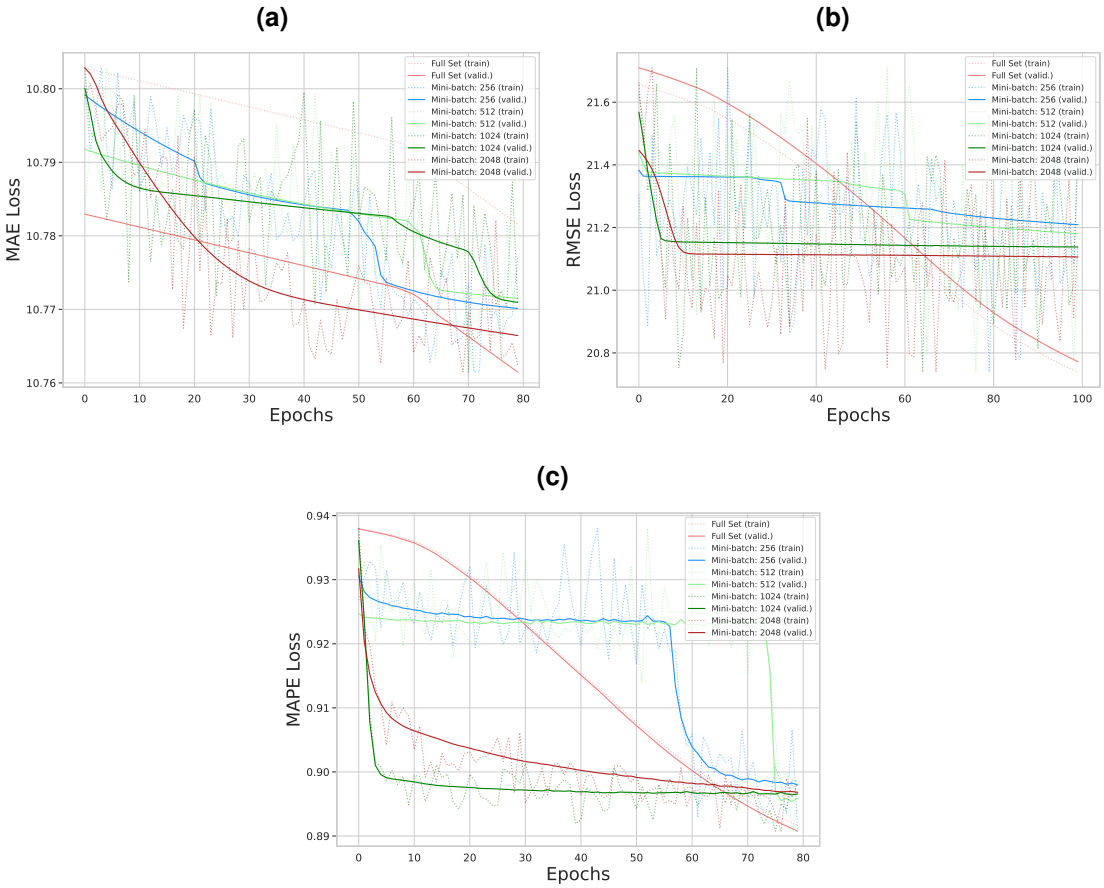


Figure 64 – GIN-E mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

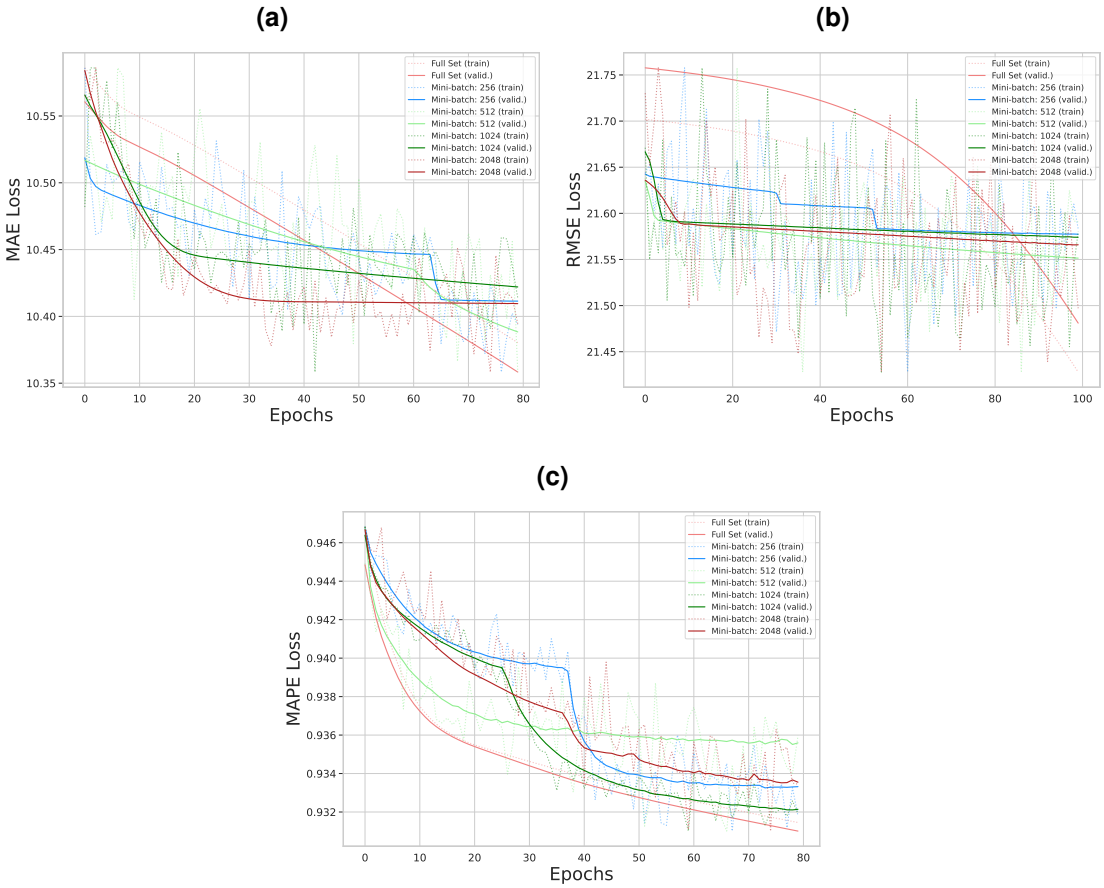


Figure 65 – RFN-AI mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

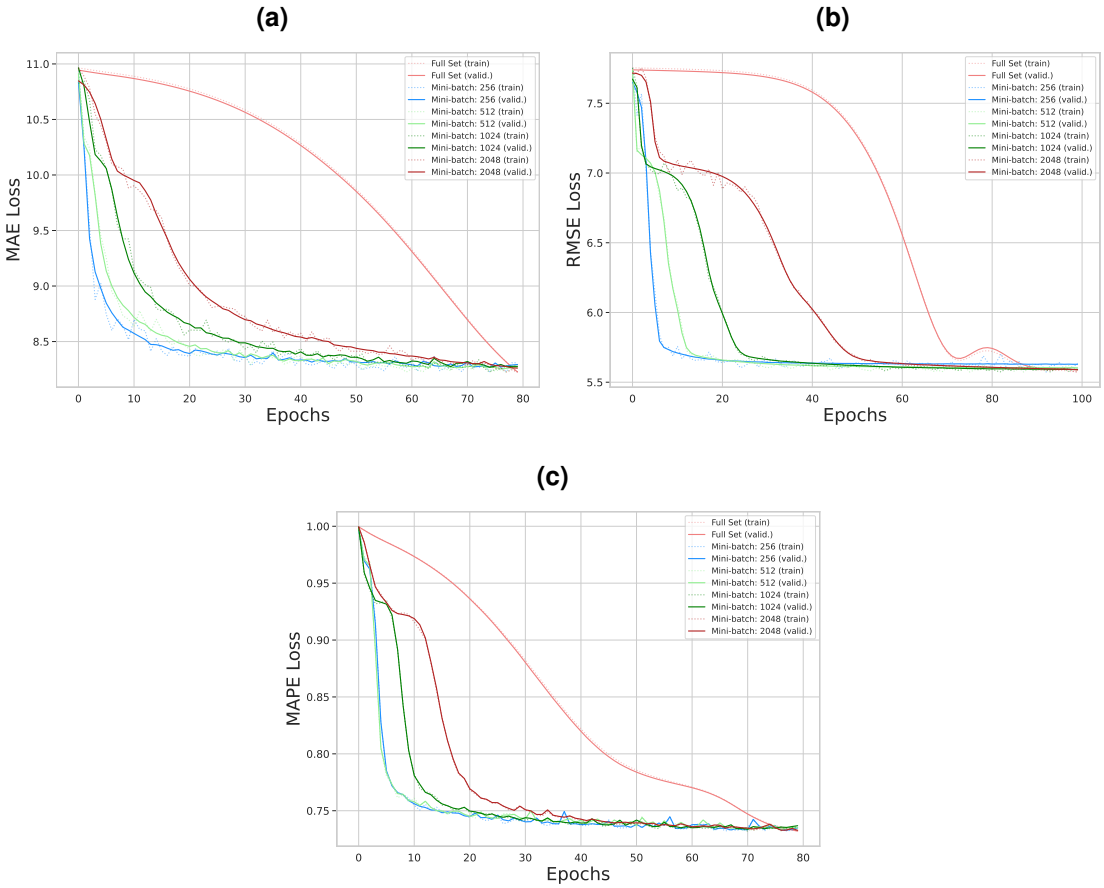


Figure 66 – RFN-AA mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

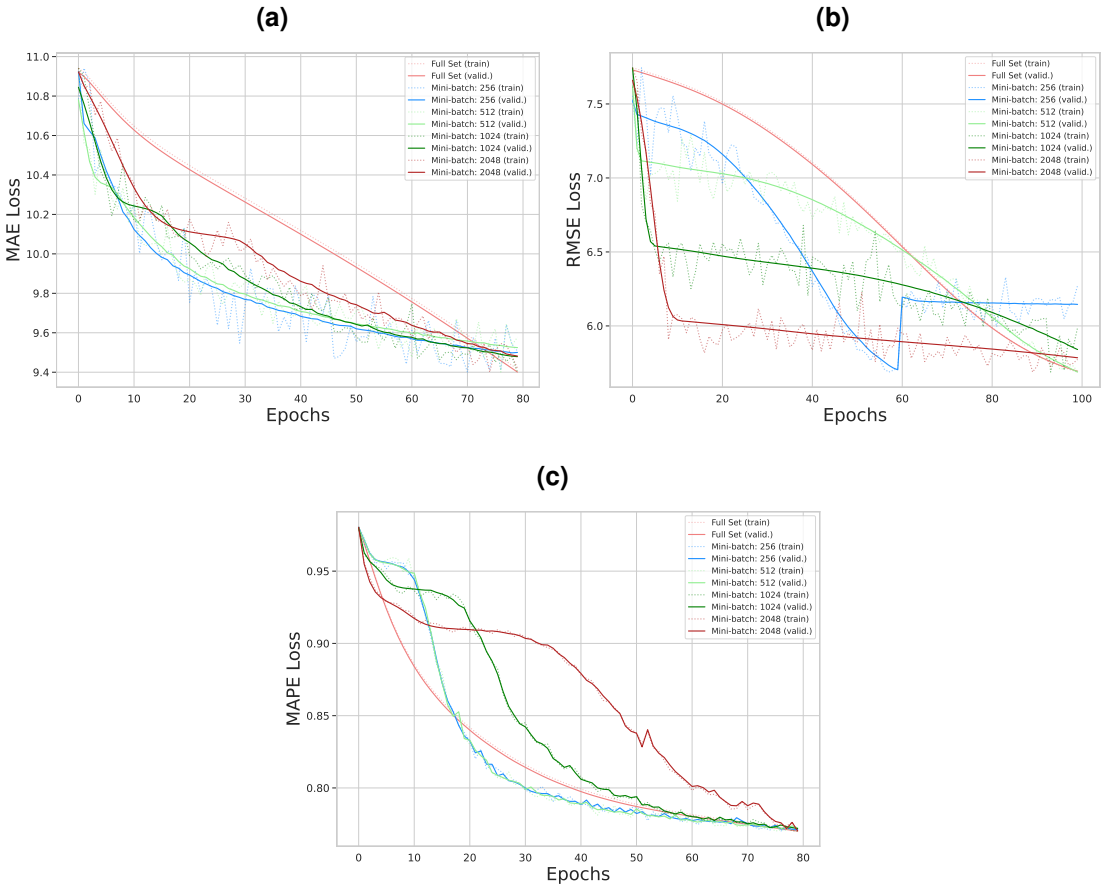


Figure 67 – RFN-NAI mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions

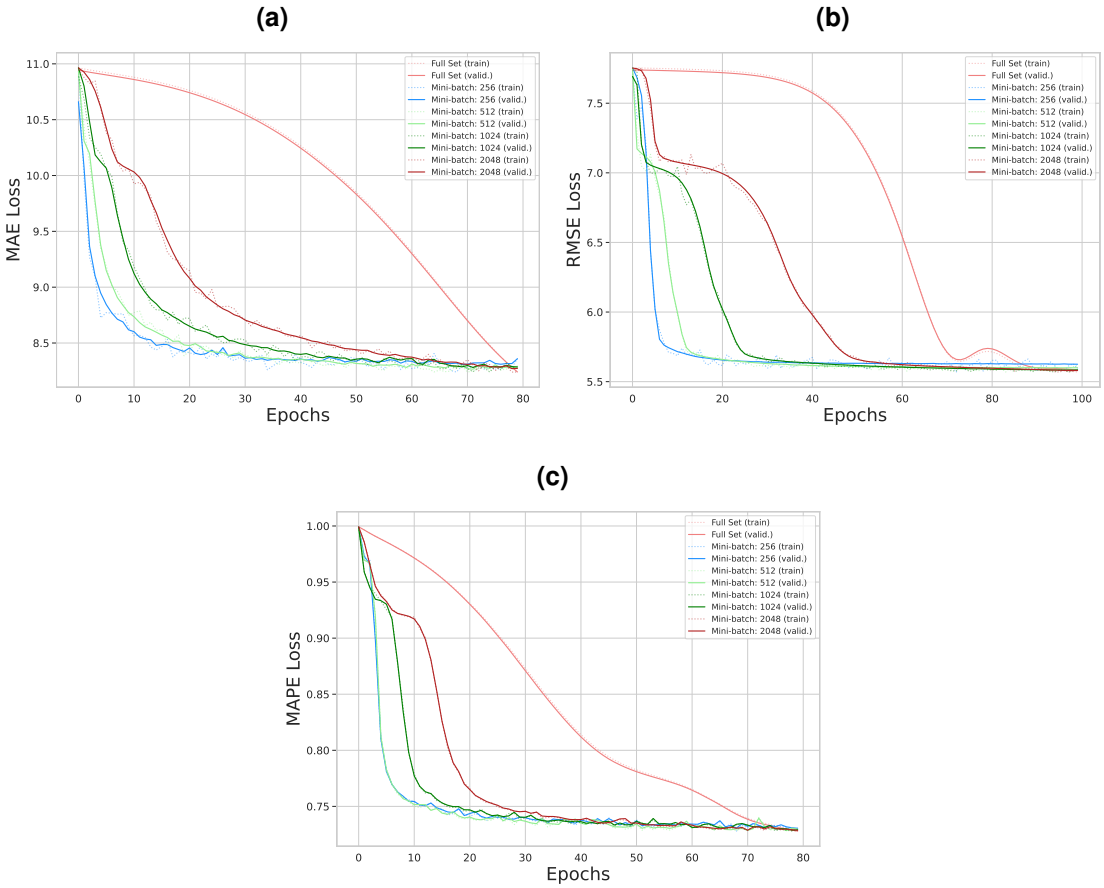


Figure 68 – RFN-NAA mini-batch training with: (a) MAE, (b) RMSE and (c) MAPE loss functions