# GRAPH NEURAL NETWORKS FOR ACCELERATED 2D TRUSS OPTIMIZATION

Master of Science dissertation

G. N. C. ARIYASINGHE


Department of Civil Engineering
University of Moratuwa, Sri Lanka

January 2025

# GRAPH NEURAL NETWORKS FOR ACCELERATED 2D TRUSS OPTIMIZATION

A dissertation submitted to the
Department of Civil Engineering, University of Moratuwa
in partial fulfilment of the requirements for the
degree of Master of Science

by

**GEEKIYANAGE NISAL CHAMIDU ARIYASINGHE**

Supervised by: Dr. Sumudu Herath, Prof. Chinthaka Mallikarachchi and Dr. Hansani Weeratunga

**Department of Civil Engineering**
**University of Moratuwa, Sri Lanka**

**January 2025**

# Declaration

The work submitted in this dissertation is the result of my own investigation, except where otherwise stated.

It has not already been accepted for any degree, and is also not being concurrently submitted for any other degree.

Nisal Chamidu Ariyasinghe

We endorse the declaration by the candidate.

Dr. Sumudu Herath

***UOM Verified Signature***

Prof. Chinthaka Mallikarachchi

***UOM Verified Signature***

Dr. Hansani Weeratunga

# Abstract

Structural optimization of skeletal forms plays a crucial role in weight-sensitive applications, but conventional iterative methods often demand significant computational effort, especially under varying design parameters and constraints. This dissertation introduces a novel Graph Neural Network (GNN)-based surrogate model designed for real-time structural optimization, offering a computationally efficient alternative. By representing trusses (composed of pin joints and members) as mathematical graphs, where nodes correspond to joints and edges denote member cross-sectional areas, the GNN predicts topology and size-optimized truss structures based on input parameters such as geometry, loading conditions, and boundary constraints. The model eliminates the need for iterative processes by learning from a dataset of problem definitions and their corresponding optimized solutions, significantly reducing computational costs while maintaining high accuracy. Testing across various initial design domains with node configurations of 24, 45, and 60 demonstrated robust performance. For topology optimization, the recall rate consistently reached unity or near-unity, and the Jaccard similarity index exceeded 0.95 in all cases with 60 nodes trained on 9600 data points, with minor exceptions of 0.94 and 0.87, still indicating reliable predictions. In size optimization, the Mean Absolute Percentage Error and Symmetric Mean Absolute Percentage Error remained below 8% across all results for cantilever and simply supported loading scenarios. It is important to highlight that these highly accurate optimization results were generated using the GNN-based framework, which reduces the computational time by over 95% compared to traditional optimization methods. These results highlight the model's ability to generalize across diverse design scenarios, enabling rapid, accurate, and material-efficient optimization. By offering near-optimal solutions with minimal computational effort, this GNN-based approach holds significant potential for sustainable and resource-efficient structural design across various applications.

# Acknowledgements

There are many people who have been a constant source of encouragement and support throughout my MSc journey. While it is impossible to mention everyone, several individuals deserve special recognition.

First and foremost, I am deeply grateful to my supervisors. Dr. Sumudu Herath, who has been a constant source of inspiration and motivation, has guided me since the very beginning of my research journey as an undergraduate, teaching me not only how to approach research but also how to navigate life with integrity and purpose. I owe immense thanks to Prof. Chinthaka Mallikarachchi for his unwavering support throughout my research, and for choosing to fund my work when he could have extended that opportunity to others. His mentorship has not only shaped my academic path but also provided a role model for me to aspire to. I am also sincerely thankful to Dr. Hansani Weeratunga for her invaluable feedback, support, and assistance in solving research challenges, as well as her guidance on navigating the ups and downs of academic life.

It has been a privilege to share the lab with Lasith Kuruppu and Eroshan Sandeepa, whose collaboration, insightful discussions, and shared expertise have greatly enriched my research experience. I would also like to express my sincere gratitude to Mrs. Dilani Ranawaka, the lab assistant. Furthermore, I am profoundly thankful for the financial support provided by the Senate Research Grant and the National Research Grant, without which this research would not have been possible.

I would also like to thank my friends and roommates for their patience and for listening to me vent during the stressful times of progress reviews.

Above all, my deepest gratitude goes to my family, whose unwavering support has carried me through this journey. To my wife, who has been my pillar of strength and encouragement, I dedicate this thesis to you.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In recent years, advancements in structural optimization have transformed the way engineers and researchers approach design challenges. However, notable research gaps persist, particularly in integrating data-driven methods with optimization frameworks. This thesis, titled "Graph Neural Networks for Accelerated 2D Truss Optimization" aims to address these gaps by exploring the intersection of advanced manufacturing techniques and data-driven approaches.

Chapter 1 begins by outlining the key motivations behind this research, addressing current limitations in the field and exploring potential applications. It then reviews recent advancements in structural optimization and manufacturing processes, as well as the increasing importance of data-driven methods, providing essential background for this study. Additionally, this chapter highlights the specific contributions made to various research communities, focusing on how this work aims to address existing gaps. The chapter concludes by presenting an overview of the thesis structure, serving as a roadmap for the subsequent chapters.

## 1.1 Motivation

The increasing demand for efficient and sustainable structural designs has intensified interest in structural optimization. Traditional steel construction, while prevalent, presents significant environmental challenges due to its high carbon footprint and energy consumption. As sustainability becomes an urgent priority, the construction industry is under growing pressure to innovate and minimize its environmental impact. Structural optimization is pivotal in advancing sustainability by facilitating the design of resource-efficient and environmentally responsible structures. By optimizing material usage, it significantly reduces waste and mitigates the environmental impact associated with the extraction, processing, and transportation of construc-

tion materials. For example, Dillen et al. [1] demonstrated that optimizing the steel structure of the Market Hall in Ghent led to a 15% reduction in weight, decreasing the total from 226.6 tons to approximately 192.7 tons. This optimization not only reduced material costs but also lowered embodied carbon emissions, benefiting both economic and environmental aspects of the project. Additionally, optimization enhances energy efficiency by minimizing the energy required during construction and throughout the lifecycle of the structure. Wang and Sigmund [2] showed that the topology optimization of multi-material active structures can significantly reduce both energy consumption and greenhouse gas (GHG) emissions. Their study found that an embodied energy-optimized active design consumed 26% less energy and emitted 27% less GHG compared to a passive design. These reductions underscore the power of optimization in creating more sustainable structures. By incorporating optimization techniques into the design process, the construction industry can contribute to global efforts to lower carbon emissions and promote a sustainable built environment.

Steel straight members produced through formative techniques offer significant advantages for large-scale construction projects due to their ease of production, transportation, and assembly. Their simplicity not only lowers costs but also streamlines the construction process. Moreover, straight members are well-suited to modular construction, where standardized components can be efficiently assembled into larger structures. Their straightforward disassembly further enhances sustainability, as components can be easily reused or recycled, thereby reducing environmental impact [3]. This efficient and eco-friendly approach has driven increased research into the use of straight members, such as beams and columns, in optimized structural designs. Recent advancements in optimization methods, alongside innovations in manufacturing, have significantly advanced structural optimization, leading to more efficient material usage, improved performance, and minimized environmental impact. However, applying these innovations in large-scale production poses challenges, as the increasing complexity of structures and optimization processes often results in higher computational demands. The primary motivation for this research is to reduce computational power and time in the optimization process, particularly as large-scale structures incorporating straight members become increasingly common. Researchers and industry practitioners across various fields have developed reliable frameworks utilizing data-driven techniques to reduce computational costs in different applications. This research seeks to apply such data-driven methods to structural optimization, providing design-informed outputs capable of translating design parameters into structurally optimized configurations with straight members. By exploring innovative approaches that merge data-driven insights with structural

optimisation, this thesis aims to advance sustainable construction practices while addressing the critical need for more efficient design methodologies.

## 1.2 Background

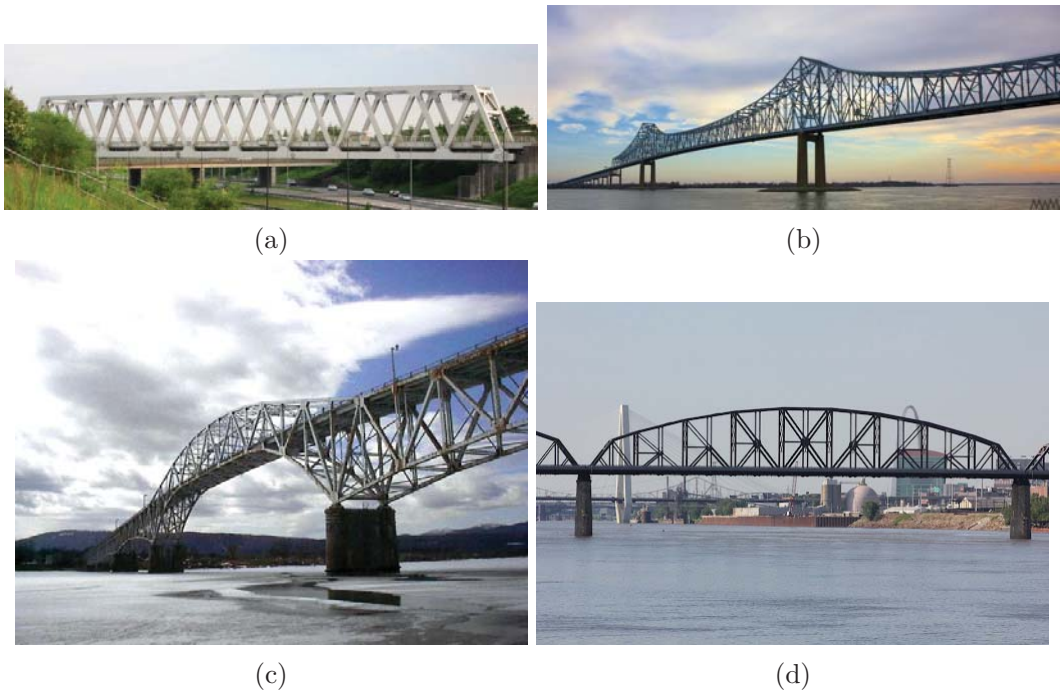
This section provides an overview of the key concepts and developments underpinning this research, offering a foundation in the domain of structural optimisation, steel manufacturing techniques, and data-driven methods. By outlining the latest trends in each of these areas, this background aims to bridge the gap between theoretical research and practical application in the field of construction.

### 1.2.1 Steel Truss Structures

Steel truss structures play a pivotal role in civil engineering, commonly used in the design of bridges, buildings, and industrial facilities due to their strength, durability, and cost-effectiveness. Composed of steel members, these frameworks are designed to support significant loads and ensure structural stability. Steel, an alloy of iron and carbon, is highly valued in construction for its exceptional load-bearing capacity, resistance to forces such as wind and earthquakes, and ability to span large distances with minimal material. Compared to materials like concrete and wood, steel offers superior flexibility and resilience, making it ideal for a wide range of infrastructure projects, from skyscrapers and bridges to stadiums and industrial complexes. Its versatility supports innovative architectural designs, while its durability ensures long-lasting performance in challenging environments. The effectiveness of steel truss structures lies in the strategic arrangement of their components—typically straight steel members connected at nodes—to form rigid, stable frameworks capable of withstanding both static and dynamic loads.

The design of steel truss structures relies on three key factors: topology, layout, and member sizing, each of which plays a critical role in the overall performance and efficiency of the structure. Topology refers to the arrangement and connectivity of the truss members, determining how forces are transmitted and loads are distributed. Layout involves the spatial arrangement of the truss, including its depth, height, and overall geometry, influencing both load paths and structural behaviour. Member sizing focuses on selecting appropriate cross-sectional dimensions for individual members to ensure that they can bear the required loads without excess material use. Different topologies offer distinct advantages depending on the application. This is clear by investigating the different truss configurations available in bridges where different configurations are more suited under a certain load, span and other factors might not be suitable under another combination of factors [4].

which is clear looking at the different truss configurations in Figure 1.1 where truss layouts and section sizes are varied depending on span, load-bearing capacity, and environmental factors.



(a) (b)

(c) (d)

Figure 1.1: (a) Brinnington Railway Bridge in Manchester (UK), (b) Commodore Barry Bridge in Pennsylvania (USA), (c) Lake Champlain Bridge between New York and Vermont (USA), and (d) McKinley Bridge across the Mississippi River (USA)

### 1.2.2 Structural Optimization Techniques

Structural optimization involves refining a structure's design to minimize or maximize a specific objective function, while adhering to a set of constraints. Recent advancements in computational power and algorithms have enabled the application of these optimization techniques to increasingly complex structures. For instance, Laurain (2018) developed a method to find the optimal shape of a structure that minimizes compliance [5]. Villalba et al. (2022) explored structural topology optimization aimed at minimizing weight while incorporating local stress constraints [6]. Kang et al. (2020) focused on maximizing specified natural frequencies in large-scale dynamic topology optimization problems [7]. Additionally, Mitjana et al. (2019) investigated structural optimization for minimal mass designs under both stress and buckling constraints [8]. These studies highlight the diverse objectives and complexities tackled in modern structural optimization.

Topology optimization is crucial in steel truss structures, determining the most efficient material layout within a given design space. Combined with shape and size

optimization, which refine geometry and member dimensions, this ensures that the final structure meets performance requirements with minimal material use. Optimizing steel trusses requires careful selection of topology, layout, and size to strike an optimal balance between material usage, cost, and structural performance [9]. As the demand for efficient, large-scale infrastructure projects grows, optimizing truss designs becomes increasingly critical. While optimization can greatly enhance structural performance, it necessitates a thorough understanding of how various configurations affect factors such as load distribution, material consumption, and constructability. A comprehensive explanation of truss structure optimization is provided in Chapter 2 Section 2.1.

### 1.2.3 Manufacturing Techniques for Steel Structures

Recent advancements in steel manufacturing have opened new possibilities for designing and constructing structures. There are three primary manufacturing techniques in steel construction: additive, subtractive, and formative methods. Understanding the strengths and limitations of each helps determine their suitability for various applications.

**Additive Manufacturing (AM)**: Often referred to as 3D printing, additive manufacturing builds structures layer by layer from raw material. This technique offers significant flexibility in design, allowing for the creation of complex geometries that would be impossible or prohibitively expensive using traditional methods. However, its application in large-scale steel construction is still limited due to material constraints and the relatively slow speed of the process. (Figure 1.2a)

**Subtractive Manufacturing**: This technique involves removing material from a larger block to create the desired shape, typically through machining or milling. Subtractive manufacturing is well-suited for creating precise, high-quality components, but it can lead to significant material waste. Its use in steel construction is often limited to custom or specialized components where precision is critical. (Figure 1.2b)

**Formative Manufacturing**: This traditional method shapes materials through processes such as forging, casting, or rolling. It is widely used in steel construction due to its cost-effectiveness and scalability. Formative manufacturing is particularly suitable for producing large, standardized components, making it the dominant approach in large-scale steel truss construction. (Figure 1.2c)

Each technique offers distinct advantages depending on the project's design, material, and scale. Since the motivation of this research focuses on optimization for large-scale construction, we emphasize the use of straight members fabricated

through formative techniques due to their efficiency and suitability for mass production.

(a) (b) (c)

Figure 1.2: (a) Additive manufacturing, (b) subtractive manufacturing, and (c) formative manufacturing

### 1.2.4 Data-Driven Techniques in Structural Optimization

The rise of data-driven approaches, especially those using machine learning (ML), has brought a transformative shift to structural optimization. ML, a branch of artificial intelligence (AI), empowers computers to identify patterns within data and make predictions or informed decisions with minimal human input [10]. In structural optimization, ML techniques offer a powerful solution to the high computational demands of traditional simulation-based methods by providing predictive models that operate with substantially lower computational costs. This ability to streamline computational processes has seen wide application across various fields. For instance, ML-based frameworks are increasingly used in the medical domain [11, 12], aerospace engineering [13, 14], weather forecasting [15, 16], and material modeling [17, 18], among others. Although the training of ML models can be computationally intensive, the trained models enable rapid predictions of optimal design solutions, offering a significant reduction in time and resource consumption compared to conventional methods. This makes ML an attractive tool for accelerating the optimization process in various complex systems.

This approach is particularly beneficial for optimization problems where traditional methods are too slow or computationally expensive. By utilizing data-driven techniques, researchers and engineers can enhance computational efficiency while maintaining accuracy and reliability in their results. The success of such optimization efforts largely depends on selecting the appropriate ML techniques. Supervised, unsupervised, and reinforcement learning are commonly applied in this context, each offering distinct advantages and limitations. Incorporating ML into the optimization process aims to bridge the gap between academic research and practical applications in steel construction. The significant potential for reducing computational time and improving design efficiency also creates new opportunities for more

sustainable construction practices. A detailed examination of these techniques is provided in Chapter 2 Section 2.3 where the current advances and limitations are explored to identify the research gap that is addressed in this study.

## 1.3 Contribution

This research proposes a novel data-driven technique for design-informed structural optimization, addressing the increasing need for computationally efficient methods capable of handling real-time design adjustments. In the field of structural optimization, the diversity of design parameters and the unique demands of each structure present significant challenges, particularly in large-scale, steel-based constructions. By incorporating ML techniques, this work contributes to reducing the computational burden typically associated with traditional optimization methods, thereby making real-time structural optimization more feasible in practical applications.

One of the key contributions of this research is the development of a framework capable of predicting both cross-sectional properties and member topologies in steel truss structures. Current optimization techniques often struggle with the complexity of structural design, where slight changes in parameters, such as load distribution or material constraints, can dramatically alter the final design. This work addresses this challenge by integrating data-driven models that can rapidly adjust to these changes, ensuring the structure meets the required performance specifications while minimizing material usage and cost.

Furthermore, this research advances the state of the art in data-driven optimization by emphasizing the importance of balancing computational efficiency with design flexibility. While ML has been explored in structural optimization, its application in predicting both member topologies and cross-sectional dimensions has remained underdeveloped. This study fills this gap by offering a method that not only speeds up the design process but also ensures that the resulting structures are optimized for sustainability and practicality in real-world applications.

In summary, the contributions of this research are threefold:

1. **Data-driven optimization framework**: The development of a ML-based framework for real-time structural optimization, capable of predicting design parameters with reduced computational costs.

2. **Efficient cross-sectional and topology prediction**: A novel approach to simultaneously optimizing both cross-sectional dimensions and member topologies in steel truss structures, addressing key challenges in the field.

3. **Practical application for large-scale structures**: A focus on real-world applications, demonstrating the framework's potential to significantly improve the efficiency and sustainability of large-scale steel construction projects.

This contribution represents a meaningful advancement in the domain of structural optimization, bridging the gap between traditional methods and modern, data-driven techniques, ultimately paving the way for more efficient and sustainable construction practices.

## 1.4 Thesis Layout

This thesis is organized into seven chapters, each building upon the last to provide a comprehensive exploration of the research problem, methodology, and findings. A brief overview of each chapter is provided below.

**Chapter 2** provides a comprehensive literature review on structural optimization, manufacturing techniques for steel structures using straight steel members, and the application of data-driven methods in structural optimization. This chapter identifies key research gaps and highlights the contributions of past studies that have shaped the development of this thesis.

**Chapter 3** offers an in-depth review of Graph Neural Networks (GNN), the ML technique employed in this research. It covers the background and fundamental principles of GNNs, providing a foundational understanding of the technique applied in this study.

**Chapter 4** focuses on the structural optimization techniques and the process of database generation. It explains the traditional ground truth optimization method, which this research aims to replace with a more computationally efficient, data-driven model. Additionally, it details the steps involved in generating the database used for training the model.

**Chapter 5** outlines the development of the data-driven model, describing the model architecture, the selected hyperparameters, and the validation process. This chapter also explains how the model's performance was evaluated.

**Chapter 6** presents the performance metrics and evaluation results of the trained models, focusing on two case studies. It illustrates how the model performed under various design scenarios and assesses its effectiveness.

**Chapter 7** summarizes the key findings of the research and discusses potential directions for future work in this area.

# Chapter 2

# Literature Review

This chapter provides an in-depth exploration of the various structural optimization techniques that have been developed or are currently under research, with a specific focus on steel truss structures. The discussion continues by examining the key frameworks that employ these optimization techniques, particularly in the context of assembly-based manufacturing, where components are primarily produced using formative manufacturing techniques. Following this, the chapter explores the role of ML in structural optimization, highlighting its potential for reducing computational costs. By critically assessing the challenges and limitations of existing methods, this section establishes the basis for the research problem and motivates the need for innovative approaches to further advance the field.

## 2.1 Structural Optimization

As discussed in Chapter 1, structural optimization in steel construction plays a pivotal role in aligning engineering practices with modern sustainability demands. By minimizing material usage and maximizing structural performance, structural optimization ensures that designs meet the necessary strength, stability, and environmental standards. This section provides an overview of the foundational concepts in structural optimization, focusing on the discretization approaches and optimization algorithms commonly used in the design and analysis of steel truss structures.

### 2.1.1 Domain Discretization

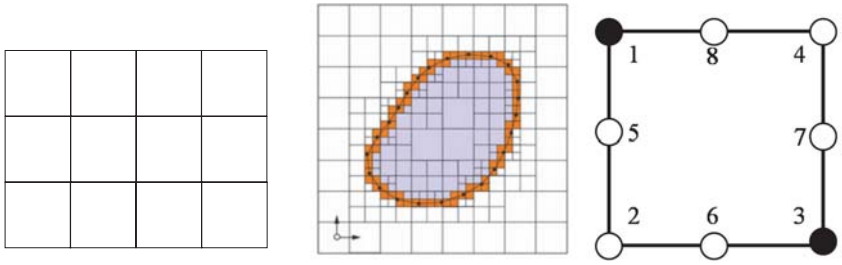
Structural optimization begins with the discretization of the design domain, which is critical for simplifying the complex continuous design space into manageable subdomains. This process is divided into two broad categories: *continuous domain discretization* and *discrete domain discretization*. The choice between these de-

pends on the nature of the structural problem and the type of optimization being performed.

### Continuous Domain Discretization

In continuous domain discretization, the design space is divided into finite elements to approximate the structure's geometry. Several approaches used in continuous domain discretization:

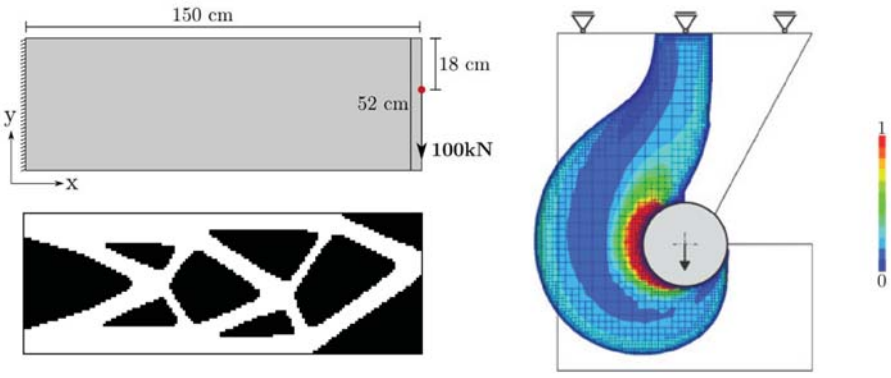
- **Orthogonal Grid Division**: This is the most commonly used discretization technique, where the design domain is divided into a grid of orthogonal elements, often resembling finite element meshes [19]. (Figure 2.1a)
- **Embedding Element Discretization**: This method embeds smaller elements within larger ones, allowing for greater flexibility in capturing complex geometries. It is useful in problems where localized stress concentrations need to be examined in greater detail [20]. (Figure 2.1b)
- **Higher-Order Finite Element Discretization**: Higher-order finite elements are used to model more complex structures with greater accuracy. This technique is advantageous in problems where higher precision in stress analysis is required, such as in the design of steel trusses under dynamic loads [21]. (Figure 2.1c)




(a) (b) (c)

Figure 2.1: (a) Orthogonal grid division [19], (b) embedding element discretization [20], and (c) higher-order finite element discretization (8 node square element [21])

The optimized structure is typically represented by a continuous material distribution, where the density values of the finite elements range between 0 and 1. These density values can be interpreted to determine the cross-sectional areas of the structural members, with higher densities corresponding to thicker members and lower densities to thinner ones [20] (see Figure 2.2b). When a binary image is generated, skeletonization can be used to identify the member locations, and the combined width of the pixels can then be analyzed to infer the cross-sectional areas of the members [19] (see Figure 2.2a).




(a) (b)

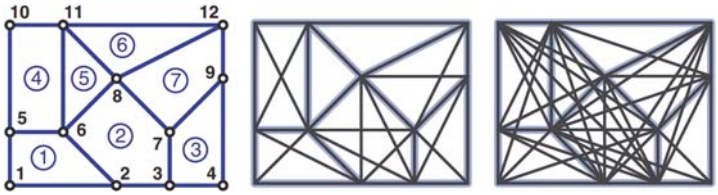
Figure 2.2: (a) Design parameters and binary image generated after optimization [19], (b) strain energy density for the optimal design layout scaled between 0 and 1 [20]

### Discrete Domain Discretization

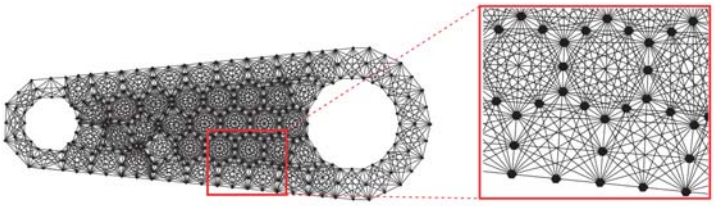
Discrete domain discretization involves breaking down the design space into a finite set of possible member configurations. Some common techniques in discrete domain discretization include:

- **Connectivity-Level Discretization**: The design domain is discretized according to the connectivity of the structural elements. At connectivity level 1, all possible members are established between directly neighboring nodes. Connectivity level 2 includes not only connections between neighbors but also introduces members between the neighbors of those neighbors. This pattern continues for higher levels [22]. (Figure 2.3a)
- **Macro Element and Macro Patch Methods**: These methods discretize the domain into larger macro-elements or patches, which the possible members are considered within those macro patches. These methods are particularly effective in large-scale structures, where entire sections of the structure can be treated as modular components [23]. (Figure 2.3b)
- **Principal Stress Trajectory Path Approach**: This method discretizes the structure based on the path of principal stress trajectories. By aligning structural members with these paths, designers can create more efficient load-bearing structures, especially in steel trusses where stress distributions vary across the framework [24]. (Figure 2.3c)

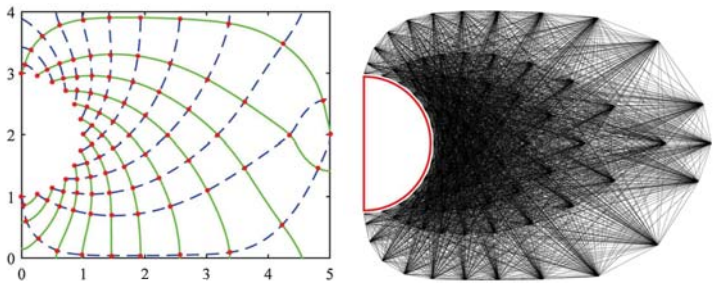
After the topology optimization, the 0 or 1 for presence or absence of members in the final structure, and the members present in the final configuration will have quantitative values corresponding to the cross sectional areas. (see Figure 2.4)




(a) Base mesh with elements and nodes, ground structure for level 1 connectivity and level 2 connectivity [22]



(b) Macro elements with possible connections between nodes within each macro patch of a wrench domain [23]

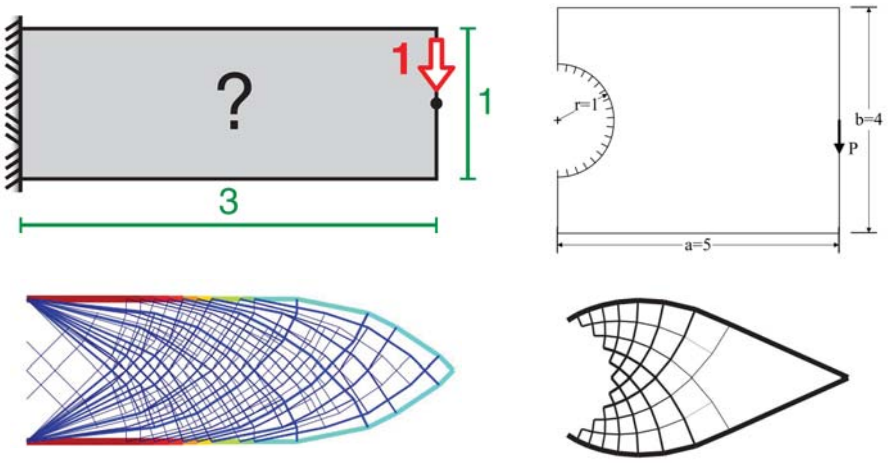



(c) Principal stress trajectories and all possible connections between nodes in concave ground structure of a Michell cantilever [24]

Figure 2.3: Example discrete domain discretizations

### 2.1.2 Optimization Algorithms

Once the design domain is discretized, optimization algorithms take over to find the final optimized structure. These algorithms seek the best structural design by balancing specific objectives, such as minimizing weight or maximizing stiffness, while satisfying constraints like load capacity, deflection limits, or equilibrium conditions. The objective function represents the design goal mathematically, and the optimization process ensures that all constraints are met. Optimization algorithms can be classified into two main types: *deterministic*, which follow a specific sequence of operations (e.g., gradient descent), and *stochastic*, which incorporate randomness (e.g., genetic algorithms). The process typically begins with an initial guess and progressively refines the solution by adjusting variables according to predefined rules. This continues until a stopping criterion is reached, such as meeting a threshold for improvement or completing a set number of iterations. Below are various optimization




(a) (b)

Figure 2.4: Design domain and parameters, and optimized structures of (a) connectivity level approach [22] and (b) principal stress trajectory path approach [24]

algorithms commonly used in both continuous and discrete domains.

**Optimization Algorithms in Continuous Domains**

Common optimization algorithms used in continous design domanis include:

- **Solid Isotropic Material with Penalization (SIMP)**: SIMP is one of the most commonly employed algorithms in topology optimization, particularly for continuum structures. It solves the optimization problem by iteratively updating material distribution within the design space, using a penalization factor to push intermediate densities towards binary values (either solid or void). The algorithm typically minimizes an objective such as compliance (stiffness) under a volume constraint. The penalization ensures that the solution avoids gray-scale regions, which are undesirable in real-world manufacturing. SIMP is gradient-based and often uses the method of moving asymptotes (MMA) to handle large-scale optimization problems efficiently. [25, 26]

- **Particle Swarm Optimization (PSO)**: PSO is a stochastic, nature-inspired optimization algorithm that models the social behavior of swarms (e.g., flocks of birds or schools of fish). In the context of structural optimization, each ”particle” represents a candidate solution, where the position of the particle defines the design variables (such as material distribution or member sizes). The particles move through the design space based on their own experience and the experience of neighboring particles, updating velocities and positions iteratively. PSO is particularly effective for non-linear, discontinuous, or multimodal objective functions and is commonly used in problems where gradient

information is unavailable. Its flexibility allows it to tackle both continuous and discrete optimization problems, often solving complex, multi-objective structural optimization challenges. [27]

- **Evolutionary Structural Optimization (ESO)**: ESO is a heuristic, geometry-based method that eliminates inefficient material from a structure by iteratively removing elements with low strain energy density. The optimization process starts with an over-sized structure, and as material is gradually removed, the structure evolves towards an optimized design that meets performance criteria such as stress or stiffness. ESO's strength lies in its simplicity and ease of implementation, as it does not require gradients or sensitivity analysis. It is particularly well-suited for minimum-weight design problems but may struggle with multi-objective optimization or cases where material needs to be added. [28]

- **Bi-Directional Evolutionary Structural Optimization (BESO)**: BESO improves upon ESO by allowing both material removal and addition during the optimization process. This bi-directional capability makes it more adaptable and efficient in finding optimal solutions, especially in problems where material might need to be reintroduced to meet design constraints. The method typically uses sensitivity analysis to determine where to add or remove material based on strain energy density or other performance criteria. BESO integrates multiple constraints, such as buckling or dynamic performance, making it effective for a wider range of structural optimization problems, including multi-material designs and complex loading conditions. [29]

### Optimization Algorithms in Discrete Domains

Common algorithms in discrete domain optimization include:

- **Standard Linear Programming with Interior Point Algorithm**: This algorithm is a deterministic approach designed to solve optimization problems characterized by linear objective functions and linear constraints. In structural optimization for discrete domains, it is particularly effective when the relationships between design variables are linear and well-defined. The Interior Point Method (IPM) is a preferred solver for large-scale linear programming due to its polynomial-time complexity, making it suitable for high-dimensional discrete problems. IPM approaches the optimal solution by traversing the interior of the feasible region, avoiding the inefficiency of traditional simplex methods, and is often integrated with branch-and-bound methods for mixed-integer linear programming (MILP) problems in discrete structural design. [22]

- **Particle Swarm Optimization (PSO)**: In structural optimization for discrete domains, where all possible members are predefined, PSO is adapted to select the optimal configuration from the set of predefined structural members. Each particle in the swarm represents a possible design solution, encoded as a vector where each element corresponds to a predefined member's presence or size. The position of a particle in the discrete space signifies the selection of specific structural elements. The particles explore the design space by updating their positions based on both their own best-found solution and the global best solution in the swarm, all while ensuring the design adheres to constraints. [30, 31]

- **Genetic Algorithms (GA)**: GA is a stochastic, heuristic-based optimization algorithm that mimics the process of natural selection. In discrete domain structural optimization, potential solutions (e.g., structural configurations) are encoded as binary strings or discrete chromosomes, where each gene represents a design variable (e.g., presence or absence of a member, material type, or cross-sectional area). GA operates through genetic operators like selection, crossover, and mutation to evolve a population of solutions toward optimality. GA is particularly powerful in combinatorial optimization where the solution space is vast, such as determining the optimal arrangement of structural elements in a truss system. It is widely used in discrete domains for its ability to handle multi-modal landscapes and non-smooth objective functions without requiring derivative information. [32, 33]

- **Grammatical Evolution (GE)**: GE is an advanced form of genetic algorithm that evolves solutions based on formal grammars, offering a flexible way to explore the design space in discrete structural optimization. Instead of evolving direct variables, GE evolves rules and symbolic expressions (via Backus-Naur Form grammar) that generate solutions. This is particularly useful for problems governed by complex constraints or hierarchical rules, such as modular structural systems or lattice structures. GE is capable of generating topologically diverse solutions by evolving a grammar that dictates the arrangement and connection of structural elements. The algorithm is ideal for solving problems where the design space is too large or constrained to be represented by simple variables, allowing for the automated generation of structural layouts from a set of encoded design rules. [34]

## 2.2 Formative Manufacturing and Optimization

As discussed in Chapter 1, Section 1.2 on manufacturing techniques, the three main methods—additive, subtractive, and formative manufacturing—each offer distinct advantages and challenges in the production process. As highlighted earlier in Section 1.1, the use of straight members like tubular sections, I-beams, L-angles, or solid rods produced through formative techniques can significantly enhance sustainability in the construction industry, particularly in large-scale projects where structural repetition is common. Acknowledging these benefits, researchers have developed optimization frameworks that integrate structural optimization specifically designed for straight members that can be manufactured using formative processes. These optimization approaches are applied in both continuous and discrete domains.

In the context of optimizing structures for construction with straight members, Cuillière et al. [35] (see Figure 2.5a) implemented a SIMP-based optimization framework to achieve an optimal design. To transition this optimized design into a construction-ready structure using straight members, they employed a series of post-processing steps. These included shrinking induced by a combination of Taubin and Laplacian-type smoothing, followed by curve skeleton extraction. Next, they distributed the cross-sectional radius along the skeleton branches, normalized the skeleton, and distributed beam cross-section radii, ultimately arriving at an optimized beam structure suitable for straight member construction. Similarly, Sarma et al. [19] (see Figure 2.5b) applied SIMP-based optimization to generate a binary image of the optimized design. This image underwent skeletonization to extract the graph model, enabling topology extraction, size and layout optimization, resulting in an optimized structure that can be constructed using straight members.

In contrast to frameworks that utilize continuous domains, those operating in discrete domains offer a distinct advantage: the discretization process naturally employs straight connections between nodes to generate the ground structure, as illustrated in Figure 2.3. For instance, the optimization framework proposed by [36] (Figure 2.6a), which employs gradient-based optimization algorithms, is designed for structures intended to be constructed using steel I-beams. After defining the initial design parameters and generating a ground structure with level 2 connectivity, the final optimized structure is construction-ready without the need for additional post-processing steps. Similarly, in the framework presented by [37] (Figure 2.6b), which uses a GA for optimization, the predefined ten-bar truss discrete domain results in an optimized structure that is immediately ready for construction. Both approaches highlight the efficiency and practicality of discrete domain frameworks in producing construction-ready designs.

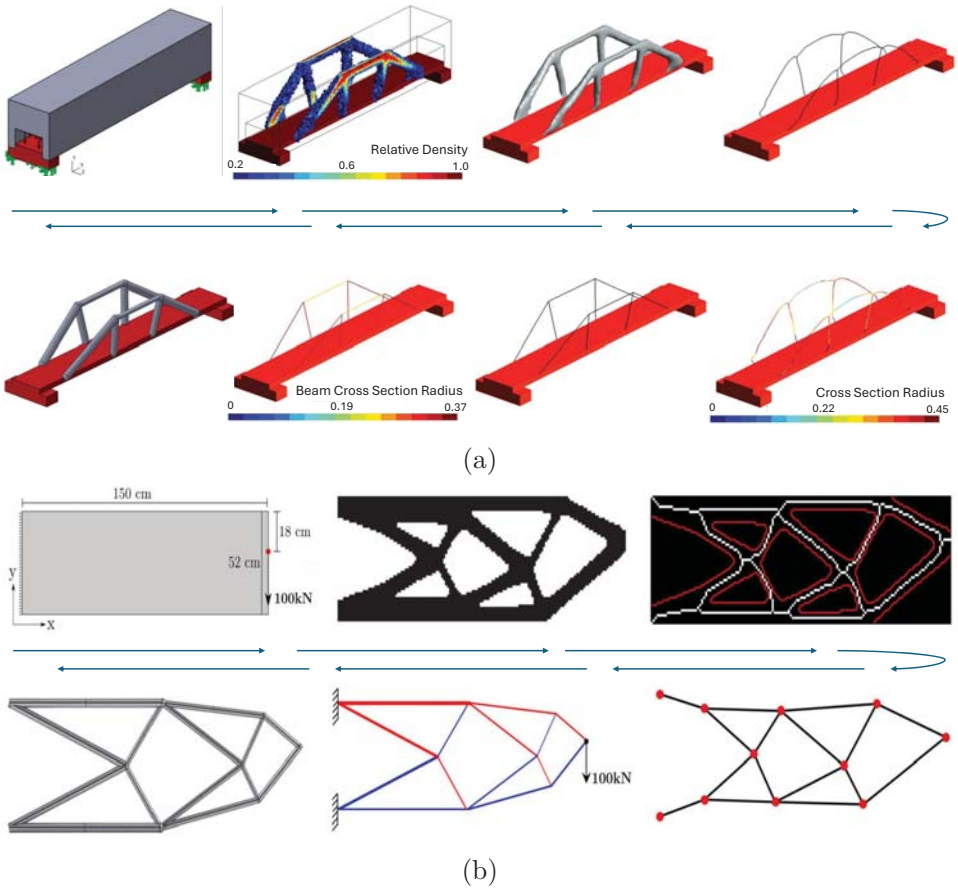



Figure 2.5: Optimization frameworks utilizing continous design domains to arrive at structure constructable using straight members used by (a) [35] and (b) [19]

Understanding the benefits of discrete domains in assembly-based structural optimization, particularly for designs utilizing members manufactured through formative techniques, is essential for defining the methodology of this research. This study will establish a data-driven framework built on the principles of discrete domain structural optimization.

## 2.3 Machine Learning Implementation in Structural Optimization

Due to the high computational costs associated with traditional optimization techniques, both industrial and academic sectors have increasingly embraced ML to enhance the efficiency of structural optimization processes. A significant challenge in structural optimization is the need for repeated iterations across design spaces, which can be computationally demanding. By leveraging ML models to uncover the underlying relationships between design parameters and optimized structures—re-

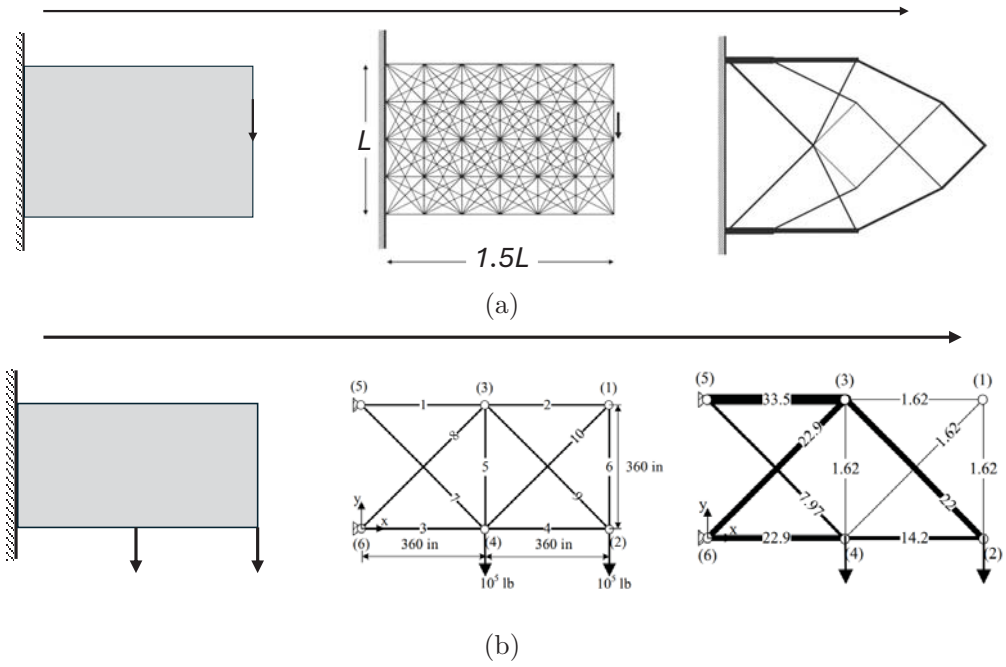



Figure 2.6: Optimization frameworks utilizing discrete design domains to arrive at structure constructable using straight members used by (a) [36] and (b) [37]

lationships that is difficult to capture through analytical methods but can be effectively modeled with data-driven approaches—optimization can be accelerated without sacrificing accuracy. This section aims to broaden the understanding of the reader of current methods for integrating ML into structural optimization and to justify the research approach adopted in this study.

### 2.3.1 Continuous Domains

Researchers have incorporated data-driven techniques to reduce the computational time in optimization processes within continuous design domains. Du et al. [38] introduced a structural optimization approach based on a generative model, using Least Squares Generative Adversarial Networks. In this approach, the generator creates new samples that replicate the distribution of real data, while the discriminator evaluates whether these samples are genuine or artificially generated. Here, an initial main truss is created using 3D modelling software and a topology optimization solver is then applied under varying working conditions, generating results that are used as the basis for the dataset. To increase the size and diversity of the dataset, several data augmentation techniques are applied to the images, resulting in a dataset comprising of images labeled with the corresponding working conditions of the truss. These augmented images are used to enhance the learning capabilities and the overall efficiency of the model.

When considering the deep learning approaches taken in the structural optimization, Seo & Kapania [39] used an image-based approach utilizing a deep learning-based surrogate model, specifically a Convolutional Neural Network architecture known as U-Net. This model is designed to predict optimum material density layouts for structural topology optimization based on static analysis results. For the data, Seo & Kapania created baseline finite element models using Abaqus/CAE and static analyses were performed on these models. The resulting output data was then generated through SIMP-based topology optimization. Finally, the data from the static analysis was organized and mapped to align with the integration points of the finite element mesh, ensuring a consistent and structured dataset for further analysis. The input data consists of multiple variables, such as displacements and strains, while the output data represents the optimum density values derived from the topology optimization process. Zheng et al. [40] employed several ML models, primarily focusing on deep learning techniques, to evaluate their performance in structural optimization. The models included U-Net, Res-U-Net, Attention-U-Net, Attention-Res-U-Net (the proposed model), as well as traditional algorithms like K-Nearest Neighbors (KNN) and Support Vector Regression. Data for model training was generated using the direct Moving Morphable Component method. The results indicate that the Attention-Res-U-Net model outperformed the others, achieving a precision of 0.8134, an F1 Score of 0.8001, and a Jaccard Index of 0.6710. In comparison, U-Net had a 3.3% lower precision and a 2.1% lower F1 Score. Both Res-U-Net and Attention-U-Net performed similarly, with slightly lower precision and Jaccard indices. The superior performance of Attention-Res-U-Net is clear across all metrics.

Hybrid approaches to structural optimization integrate multiple ML models within a single framework. For instance, Wang et al. [41] combined Principal Component Analysis (PCA) with a feedforward neural network to enhance prediction accuracy. First, the collected training data is processed using PCA to reduce its dimensionality. The reduced data is then fed into a feedforward neural network, where load information corresponding to the training set is input, and the weights are labeled accordingly. This training process establishes a nonlinear mapping between the load and the optimized structure. The study found that when the number of samples (D) exceeds 500, the network error converges, significantly improving prediction accuracy. The minimum prediction error is around 2.663% when the number of eigen-images (M) is set to approximately 22% of the sample size. Moreover, the error stabilizes between 3.6% and 3.8% when the number of neurons (N) is within 15% to 30% of the sample size. Similarly, Seo & Min [42] used a hybrid model combining GNNs and Multi-Layer Perceptrons (MLPs) to address topology optimization prob-

lems involving complex non-Euclidean data, such as finite element meshes. They generated a dataset of 3,840 samples with varying graph vertex counts, which was divided into training, validation, and test sets. In this approach, GNNs are used to encode the topology, with inputs such as vertex features (spatial coordinates, displacement, and stress fields) and an adjacency matrix representing the graph's connectivity. The GNN outputs an encoded vector that captures critical information for topology optimization. This encoded vector is then passed to the MLP, which combines Fourier features (for high-frequency spatial details) and vertex features to predict the optimal material distribution. The final output is a one-dimensional vector representing material density (ranging from 0 to 1), indicating where material should be present or absent in the structure.

Jeong et al. [43] took a different approach by utilizing Physics-Informed Neural Networks (PINNs) to solve solid mechanics problems, particularly for topology optimization. The PINN is designed to approximate displacement fields while adhering to the governing physical laws expressed by partial differential equations (PDEs), effectively serving as an alternative to traditional FEA. It numerically computes the structure's deformation states, which are crucial for evaluating its response under different loading conditions. Unlike conventional ML models, the PINN does not rely on labeled data; instead, it is trained using a physics-informed loss function that incorporates the governing PDEs. This enables the model to learn directly from the physics of the problem. The training process focuses on minimizing the system's potential energy, and within the PINNTO framework, the goal is to minimize the mean compliance of the structure. The displacement fields approximated by the PINN are used to calculate stress, strain, and perform sensitivity analyses, which help update design variables. Additionally, a density interpolation scheme is employed to allocate elemental densities in a mesh-free manner, enhancing the flexibility of the optimization process. This iterative process continues as the PINN repeatedly computes displacement fields and performs sensitivity analyses, updating the design variables until convergence is achieved.

For clarity, several outputs from the previously discussed studies are provided in Figure 2.7. While the optimized structural topologies can be successfully obtained, it is evident that accurately determining the cross-sectional sizes from these images remains challenging. This difficulty arises due to the pixel dependency of cross-sectional properties, which can introduce significant error, especially when scaled across different structures. Consequently, without post-processing techniques for size optimization, it is difficult to develop a ML model that accurately predicts both topology and section sizes in continuous design domains. Recognizing this limitation, this study will focus on the application of ML models in discrete design

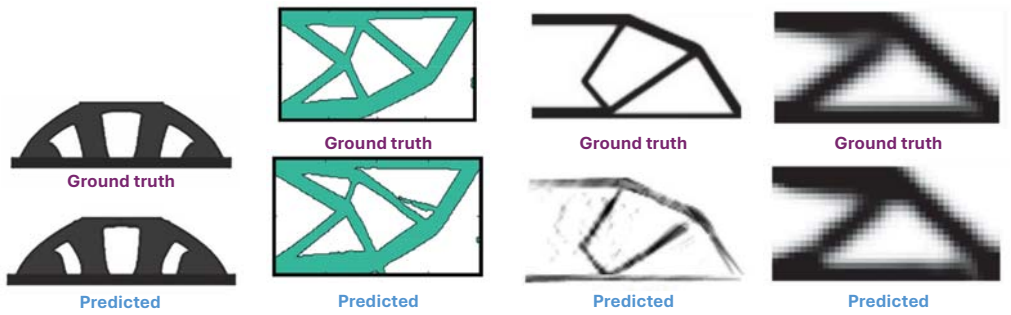



Figure 2.7: Ground truth vs predicted outputs of ML adopted optimization frameworks in continuous domains (a) [38], (b) [40], (c) [41] and (d) [43]

domains, where these challenges can be better addressed.

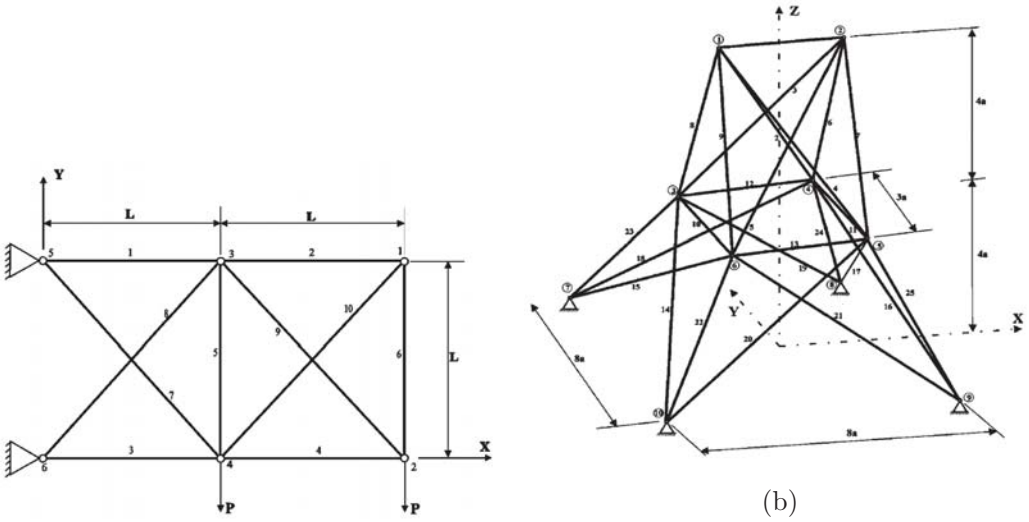
### 2.3.2 Discrete Domains

Optimization techniques in the discrete domain have evolved significantly, driven by contributions from various studies. One approach gaining traction is the integration of ML to approximate FEA results. For instance, Mai et al. [44] incorporated a Deep Neural Network (DNN) into the optimization process, combining it with a Differential Evolution (DE) algorithm in a structured manner. The input data for the model consisted of cross-sectional areas of truss members, while the output data (i.e. the displacements at the nodes) were obtained through geometrically nonlinear analysis using the arc-length method. Once trained, the DNN served as a surrogate model, allowing it to predict nodal displacements without the need for time-intensive FEA. The model's accuracy was validated with benchmark tests, and for the 30-bar dome truss, the mean squared error (MSE) values were reported to be as low as 0.059 and 0.051 for training and validation datasets, respectively, with corresponding root mean squared error (RMSE) values of approximately 2.429% and 2.273%. In a more recent study, Pham et al. [45] employed a k-nearest neighbor comparison (k-NNC) approach in optimization. The k-NNC model assesses new design candidates by comparing them to their k-nearest neighbors within the current population of solutions. If the majority of neighboring candidates are inferior, the new design is considered suboptimal and discarded without undergoing costly structural analysis. Unlike traditional ML models that require a training phase, k-NNC operates without one, using the existing candidate population for evaluation. This eliminates the need for additional memory to store training data. By integrating k-NNC with Rao algorithms, Pham et al. achieved significant reductions in computational cost, decreasing the number of structural analyses required by 40% to 60% compared to conventional methods.

Nguyen and Yu [46] adopted an AdaBoost classifier to enhance the performance of the Differential Evolution (DE) optimization algorithm, focusing on classifying the safety state of structures during optimization. This approach allows the rejection of unpromising trial vectors without conducting costly structural analyses. When a new trial vector is generated, the AdaBoost classifier predicts whether it is likely to violate design constraints. If the classifier deems the vector ”unsafe” and its objective function value is worse than the current best solution, the trial vector is discarded without further analysis. This method effectively reduces the need for expensive structural evaluations, improving the overall efficiency of the optimization process. Similarly, Nourian et al. [47] proposed a framework aimed at minimizing the number of finite element model (FEM) analyses during optimization by employing a GNN. The GNN is trained to predict nodal displacements based on different cross-sectional areas of truss members. In approximately 90% of the optimization iterations, the GNN is used to approximate nodal displacements, while FEM analyses are performed in the remaining 10% to ensure accuracy. This selective use of FEM analyses ensures both efficiency and precision. For the 10-bar planar truss benchmark test, the final mean squared error (MSE) of the GNN model was reported to be 0.022, highlighting its effectiveness in reducing the computational load while maintaining accuracy.

Unsupervised ML techniques have also been employed in structural optimization. Mai et al. [48] utilized a DNN where the network's weights and biases were treated as design variables for the structural optimization problem. In this approach, the DNN takes the middle spatial coordinates of truss elements as input. The loss function is constructed to minimize the total weight of the structure while ensuring that all design constraints are satisfied. This loss function integrates both the objective of weight minimization and the structural responses obtained from FEA. After training, the optimal weights of the truss structures are determined. In the case of the 10-bar planar truss benchmark problem, the optimal weight obtained through this method was 524.719 kg, which closely matches the result of 524.463 kg achieved by the Differential Evolution (DE) algorithm, with a minimal weight error of 0.051%. In a related study, Mai et al. [49] further enhanced the design optimization process by integrating a DNN with Bayesian Optimization (BO) to tune hyperparameters automatically. This approach was applied to the optimization of geometrically nonlinear truss structures. Similar to their earlier work, a loss function was constructed based on predicted cross-sectional areas and deflection constraints, which were derived from FEA using the arc-length method. The objective was to minimize this loss function, thereby finding the optimal structural weight. BO was employed to optimize DNN hyperparameters, such as the number of hidden layers,

the number of neurons per layer, activation functions, and learning rate. Hyperparameter tuning was treated as a black-box optimization problem, with the objective of minimizing the structure's weight as predicted by the DNN. A Gaussian process was used as a surrogate model within the BO framework to improve the efficiency of the optimization.

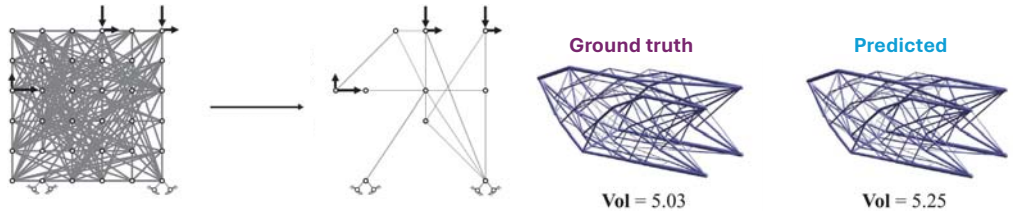



(a) (b)

Figure 2.8: Structural optimization benchmark problems in discrete domains - (a) 10 bar truss, (b) 25 bar truss [50]

The optimization frameworks discussed so far have primarily focused on size optimization, where the topology of the structure is predefined. However, Zhu et al. [51] introduced a framework for topology optimization, where unnecessary members are removed based on stress ratios, and a fully-stressed design assigns cross-sectional areas to the remaining members. In this study, they employed reinforcement learning (RL) to automate the generation of machine-specified ground structures (MGS). The process is formulated as an RL task, with a policy network determining the probability of selecting nodes from a predefined node-set. The RL agent is trained to maximize cumulative rewards, which aids in designing stable binary trusses. The policy network uses a graph embedding framework to incorporate structural information into a feature matrix that effectively represents the system's state. This matrix enables the RL agent to evaluate the probability of selecting each node accurately. The stochastic nature of the policy network allows it to generate diverse structures under varying conditions, making it crucial for exploring different topologies and increasing the chances of finding a global optimal solution. The agent was trained using a randomized $4 \times 4$ node-set, learning through trial and error to maximize cumulative rewards. The policy network parameters were updated using a gradient-based algorithm to refine its decision-making capabilities. Zheng et al. [40] further

advanced topology optimization by using Graph Convolutional Networks (GCN) to predict structural designs for truss optimization. They framed the problem as a link prediction task, where the GCN learns to predict optimal layouts based on graph representations of truss structures. The dataset was generated using a connectivity-level-based ground structure optimization process. The GCN was implemented in an encoder-decoder framework, with the encoder extracting relevant features from the input graph, while the decoder reconstructed the optimal truss layout based on these features. The model's effectiveness was evaluated using metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and $R^2$ score. For a total of 2,600 test cases, the GCN achieved an average MSE of 0.0833, an RMSE of 0.2886, and an $R^2$ score of 0.7863, demonstrating the model's capability in predicting optimal layouts with reasonable accuracy.




(a) (b)

Figure 2.9: (a) Optimised output of the RL based framework proposed in [51], (b) ground truth vs predicted structure of the GCN based framework proposed in [40]

With reviewing these recent studies, there is a deficiency in the use of ML techniques in optimization frameworks that can optimize both topology and size of the discrete structural domains. And the existing structural optimization techniques like Zhu handles optimization in a low number of nodes and studies like Zheng has the room for improvement. With the consideration of these and the potential advantage of GNNs as a ML technique for handling graph structured problem which is the basis of discrete domains in truss optimization, we will focus on the using GNNs for the optimization which will be discussed in Chapter 3.

# Chapter 3

# Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural networks specifically designed to work with graph-structured data [52]. Unlike traditional neural networks, which are primarily suited for grid-like data such as images or sequences, GNNs excel in scenarios where the relationships between entities are naturally represented as graphs [53]. These networks use the inherent structure of graphs, where nodes represent entities and edges depict the relationships between them, allowing GNNs to model complex, relational data across a wide range of applications. From social network analysis [54, 55], traffic prediction [56, 57] to molecular chemistry [48, 58].

## 3.1 Preliminaries

This section is intended for the reader to obtain a broader understanding of the technicalities of GNNs.

### 3.1.1 Graph Representation

A graph $G$ is composed of two primary components:

- **Nodes (or Vertices)**: These represent entities or data points. For a graph $G = (\mathbf{V}, \mathbf{E})$, $\mathbf{V}$ denotes the set of nodes, where each node $v \in \mathbf{V}$ can represent various data types such as molecules, social network users, or physical locations. Each node can also be associated with a feature vector $\mathbf{x}_v \in R^d$, where $d$ is the feature dimensionality. These features can represent properties like atom types, user profiles, or spatial coordinates.
- **Edges**: These represent relationships between pairs of nodes. The set $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ defines edges between nodes, where $(u, v) \in \mathbf{E}$ indicates an edge from node $u$ to node $v$. Edges can either be *directed* or *undirected*, depending on

the application domain. Each edge can also carry information, such as weights or attributes, denoted by $w_{uv}$, which could represent distances, similarities, or interaction strengths.

Thus, the graph $G$ can be represented as $G = (\mathbf{V}, \mathbf{E}, \mathbf{X}, \mathbf{W})$, where $\mathbf{X} = \{\mathbf{x}_v \mid v \in \mathbf{V}\}$ denotes the set of node features, and $\mathbf{W} = \{w_{uv} \mid (u, v) \in \mathbf{E}\}$ represents the edge attributes.

In GNNs, the objective is to learn node (or graph-level) representations by using the structure and relationships within the graph. The learning process involves aggregating and transforming information from each node's local neighborhood through multiple layers of neural networks (explained in the following section).

### 3.1.2 Message Passing and Aggregation

The fundamental operation in GNNs is the message passing mechanism, which facilitates the exchange of information between neighboring nodes. At each layer of the GNN, nodes communicate by sending and receiving messages to update their feature representations.

**General Message Passing Framework**

The message passing process for node $v$ involves three main steps: *message generation*, *aggregation*, and *update*. These steps are iteratively applied over multiple layers (or time steps), allowing information to propagate across the graph [59].

1. **Message Generation**: Each node $u$ sends a message to its neighbors. The message depends on the current state (i.e., feature representation) of node $u$ and potentially the edge feature between $u$ and $v$. The message from node $u$ to node $v$ at layer $k$ can be denoted as:

$$\mathbf{m}_{u \to v}^{(k)} = \text{MESSAGE}^{(k)}(\mathbf{h}_u^{(k)}, \mathbf{h}_v^{(k)}, w_{uv}), \tag{3.1}$$

   where $\mathbf{h}_u^{(k)}$ is the feature vector of node $u$ and $\mathbf{h}_v^{(k)}$ is the feature vector of node $v$ at layer $k$, and $w_{uv}$ is the edge feature.

2. **Aggregation**: Each node $v$ collects the messages from its neighbors $N(v)$ and aggregates them using a predefined aggregation function, such as sum, mean, or max. The aggregated message at layer $k$ is:

$$\mathbf{m}_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\{\mathbf{m}_{u \to v}^{(k)} \mid u \in N(v)\}\right). \tag{3.2}$$

   The choice of aggregation function is crucial since it must be *permutation-invariant*, meaning it should not depend on the order of the neighbors. Com-

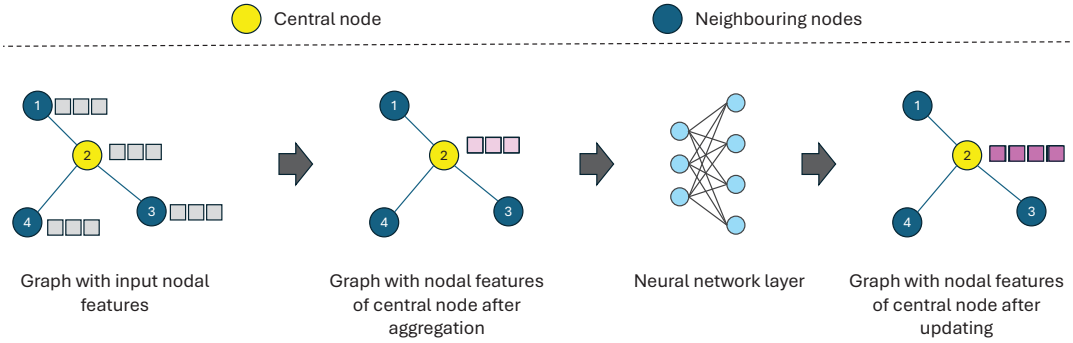



Figure 3.1: Message passing framework of GNNs

mon aggregation functions include:

- **Sum**:
$$\mathbf{m}_v^{(k)} = \sum_{u \in N(v)} \mathbf{m}_{u \to v}^{(k)}$$

- **Mean**:
$$\mathbf{m}_v^{(k)} = \frac{1}{|N(v)|} \sum_{u \in N(v)} \mathbf{m}_{u \to v}^{(k)}$$

- **Max**:
$$\mathbf{m}_v^{(k)} = \max_{u \in N(v)} \mathbf{m}_{u \to v}^{(k)}$$

3. **Update**: After aggregation, each node updates its feature representation based on the aggregated message and its previous state. The update function is typically a neural network layer, such as a MLP or a linear transformation followed by a non-linearity. The updated node feature for node $v$ at layer $k+1$ is:
$$\mathbf{h}_v^{(k+1)} = \text{UPDATE}^{(k)}(\mathbf{h}_v^{(k)}, \mathbf{m}_v^{(k)}), \tag{3.3}$$
where the update function can incorporate non-linear activation functions (e.g., ReLU, tanh) to introduce complexity and model non-linearities.

Thus, the complete message-passing operation at layer $k$ for each node $v$ from equations (3.1), (3.2) and (3.3) is:

$$\mathbf{h}_v^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_v^{(k)}, \text{AGGREGATE}^{(k)}\left(\{\text{MESSAGE}^{(k)}(\mathbf{h}_u^{(k)}, \mathbf{h}_v^{(k)}, w_{uv}) \mid u \in N(v)\}\right)\right). \tag{3.4}$$

### Graph Convolution as a Special Case

In the case of **Graph Convolutional Networks (GCNs)**, the message passing is simplified by assuming the messages between nodes depend only on their feature vectors and a fixed weighting scheme based on the adjacency matrix $A$. The

convolution operation at layer $k$ is written as:

$$\mathbf{h}_v^{(k+1)} = \sigma\left(\sum_{u \in N(v)} \frac{1}{\sqrt{d_u d_v}} \mathbf{W}^{(k)} \mathbf{h}_u^{(k)}\right), \tag{3.5}$$

where:

- $\sigma$ is a non-linear activation function (e.g., ReLU),
- $\mathbf{W}^{(k)}$ is the learnable weight matrix at layer $k$,
- $d_u$ and $d_v$ are the degrees of nodes $u$ and $v$,
- $\frac{1}{\sqrt{d_u d_v}}$ is a normalization term to prevent overemphasis of high-degree nodes.

This formulation of GCN is a specific form of message passing where the messages are weighted by the adjacency matrix, and the aggregation step is performed using a weighted sum.

**Multi-Layer Message Passing**

The power of GNNs comes from applying multiple layers of message passing. Each layer expands the receptive field of each node, allowing it to gather information from nodes that are farther away in the graph. After $k$ layers, the feature vector of a node $v$ contains information aggregated from its $k$-hop neighborhood.

By stacking several layers of message passing, GNNs can capture complex interactions between nodes and edges, making them suitable for various tasks such as node classification, link prediction, and graph-level tasks (e.g., molecular property prediction).

### 3.1.3 Types of Predictions with GNNs

Graph Neural Networks (GNNs) can be applied to various types of tasks, depending on the structural level of the graph (node, edge, or graph) and the nature of the prediction required. The core idea of GNN-based prediction tasks is to leverage both node features and the topological structure of the graph to infer hidden patterns or properties at different levels.

**Node-Level Predictions**

In **node-level prediction** tasks, the goal is to infer specific attributes or outcomes related to individual nodes within a graph. These predictions leverage both the node's own features and the structural information from the graph, which includes relationships with its neighbors. Node-level prediction encompasses tasks such as node classification, node regression, and anomaly detection [60].

- **Problem Definition**: Given a graph $G = (\mathbf{V}, \mathbf{E}, \mathbf{X})$, where $\mathbf{V}$ is the set of nodes, $\mathbf{E}$ is the set of edges, and $\mathbf{X} \in R^{|V| \times d}$ is the matrix of node features, the goal is to predict specific outcomes for each node $v \in \mathbf{V}$. The prediction task can vary, ranging from node classification, which involves predicting a discrete label $y_v$ for each node, to node regression, which involves predicting a continuous value $y_v$. Another task is anomaly detection, where the objective is to identify whether a node exhibits abnormal patterns based on its features and neighborhood.

- **Applications**: Node-level predictions are applicable in a variety of domains. In social networks, they can be used to predict the role or category of a user (e.g., classification) [61] or to estimate user engagement (e.g., regression) [62]. In citation networks, node classification can be applied to categorize papers by subject area [63], while regression tasks may involve predicting the number of future citations [64]. Additionally, anomaly detection [53] can identify rare or unusual entities within the graph.

- **Methodology**: Graph Neural Networks (GNNs) are commonly used for node-level prediction tasks. GNNs use a multi-layer message-passing mechanism, allowing nodes to aggregate information from their neighbors to refine their own feature representation. This refined embedding is used as input to a prediction model, such as a classifier or regressor, depending on the task.

  The feature update process at each GNN layer derived from function (3.4) follows the form:

$$\mathbf{h}_v^{(k+1)} = \sigma\left(\mathbf{W}^{(k)} \cdot \text{AGGREGATE}\left(\mathbf{h}_v^{(k)}, \{\mathbf{h}_u^{(k)} \mid u \in N(v)\}\right)\right), \tag{3.6}$$

  where $\mathbf{W}^{(k)}$ is a learnable weight matrix. The aggregation function combines information from the node's neighbors $N(v)$, allowing the model to capture both local and global graph structure.

### Edge-Level Predictions

In **edge-level prediction** tasks, the objective is to predict specific attributes or outcomes related to edges between two nodes. These tasks can involve predicting the presence or absence of an edge (link prediction), estimating an edge's weight, or even predicting the type of relationship that the edge represents. Edge-level predictions leverage both node features and the graph's structural information to make informed predictions.

- **Problem Definition**: Given a graph $G = (V, E, \mathbf{X})$, where $V$ is the set of

nodes, $E$ is the set of edges, and $\mathbf{X} \in R^{|V| \times d}$ represents node features, the goal is to predict specific outcomes for each edge $e_{uv} \in E$ between two nodes $u$ and $v$. The task could involve predicting whether an edge exists (link prediction), estimating an edge weight $w_{uv}$, or determining the type of relationship the edge encodes. Edge-level predictions can thus be framed as binary classification, regression, or multi-class classification problems depending on the task.

- **Applications**: Edge-level predictions are essential in various domains. In social networks, link prediction helps in forecasting future interactions between users [65]. In recommendation systems, edge-level prediction can suggest potential items to users [66]. In biological networks, these tasks help predict interactions between proteins [67].

- **Methodology**: Graph Neural Networks (GNNs) are commonly used for edge-level prediction tasks. GNNs learn low-dimensional node embeddings by aggregating information from neighboring nodes. For edge-level tasks, a score is computed for each potential edge, derived from the embeddings of the two connected nodes. This score can be computed using functions such as cosine similarity, dot product, or a learned function over the node embeddings:

$$\hat{y}_{uv} = f(\mathbf{h}_u, \mathbf{h}_v), \tag{3.7}$$

  where $f(\mathbf{h}_u, \mathbf{h}_v)$ is a scoring function that predicts the presence of an edge, its weight, or its type between nodes $u$ and $v$. For tasks like link prediction, this score is often passed through a sigmoid function to produce a probability estimate.

- **Edge Features**: If edge features $w_{uv}$ are available, they can be incorporated into the message-passing process. In this case, edge attributes can directly influence the prediction process, with message passing follows function (3.1) which allows the edge features $w_{uv}$ to interact with the node embeddings during the model learning process.

### Graph-Level Predictions

In **graph-level prediction** tasks, the objective is to predict an attribute or outcome for an entire graph, rather than individual nodes or edges. These tasks involve analyzing the overall structure and properties of the graph as a whole. Graph-level predictions encompass tasks like graph classification, graph regression, and others.

- **Problem Definition**: Given a set of graphs $\{G_1, G_2, \ldots, G_n\}$, where each graph $G_i = (V_i, E_i, \mathbf{X}_i)$ has its own set of nodes, edges, and features, the goal

is to predict an outcome $y_i$ for the entire graph $G_i$. This task can take various forms, such as graph classification, where the goal is to assign a discrete label to the graph, or graph regression, where a continuous value is predicted for the graph.

- **Applications**: Graph-level predictions are highly applicable in domains such as bioinformatics, where graph classification can be used to predict molecular properties based on molecular structures [68]. In chemistry, graph classification helps classify chemical compounds based on their molecular graph [69]. Additionally, in social networks, graph-level predictions can help determine the type or behavior of entire sub-networks or communities [70].
- **Methodology**: For graph-level tasks, node embeddings are first learned through message passing in a Graph Neural Network (GNN). Afterward, a *graph pooling* or *readout* operation is applied to aggregate node-level information into a single global graph representation $\mathbf{h}_G$. In pooling operations:

$$\mathbf{h}_G = \mathrm{POOL}_{\mathrm{op}} \left( \mathbf{h}_v \mid v \in V \right), \tag{3.8}$$

where op can be *sum*, *mean*, or *max* depending on the pooling method.

Once the global graph representation $\mathbf{h}_G$ is obtained, it is typically passed into a classifier (e.g., a fully connected neural network) or a regressor to predict the desired graph-level attribute or label.

## 3.2 Types of Convolutions in GNNs

Graph Neural Networks (GNNs) can apply various convolutional or message-passing mechanisms to learn from graph-structured data. These mechanisms define how information is propagated between nodes and are broadly classified into two categories: spectral-based and spatial-based convolutions [71]. Each type has its unique mathematical foundation and approach to aggregating information from neighboring nodes.

### 3.2.1 Spectral-Based Convolutions

Spectral-based convolutions operate in the frequency domain of the graph by leveraging the graph Laplacian matrix. They are defined using the eigenvalues and eigenvectors of the Laplacian to represent the graph in the Fourier domain, capturing global graph properties. Several spectral based convolutions are provided below.

**Graph Fourier Transform** [72]: For an undirected graph, the Laplacian $L$ is defined as $L = D - A$, where $D$ is the degree matrix and $A$ is the adjacency matrix. The graph Laplacian can be decomposed as $L = U\Lambda U^T$, with $U$ as the matrix of eigenvectors and $\Lambda$ as the diagonal matrix of eigenvalues. The graph Fourier transform of a signal $\mathbf{x}$ is given by:

$$\hat{\mathbf{x}} = U^T\mathbf{x},$$

and the inverse by

$$\mathbf{x} = U\hat{\mathbf{x}}.$$

**Spectral Graph Convolutions** [73]: A spectral convolution on a graph is the multiplication of the graph signal $\mathbf{x}$ with a filter $g_\theta$ in the spectral domain:

$$g_\theta * \mathbf{x} = U g_\theta(\Lambda)U^T\mathbf{x}.$$

Although effective, this approach is computationally expensive for large graphs due to the need for eigendecomposition.

**Chebyshev Polynomials and Localized Filters** [73]: To address scalability, Chebyshev polynomials $T_k(\Lambda)$ approximate the spectral filters, enabling localized and more efficient convolutions:

$$g_\theta * \mathbf{x} \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})\mathbf{x}.$$

This formulation enables the convolution to focus on local neighborhoods and reduces the computational cost for large graphs.

### 3.2.2 Spatial-Based Convolutions

Spatial-based convolutions directly operate in the graph domain by aggregating information from a node's local neighborhood. These methods apply message-passing techniques, where each node gathers information from its neighbors and updates its features. Unlike spectral methods, spatial convolutions do not rely on eigendecomposition, making them more scalable and efficient for large graphs. Several spatial based convolutions are provided below

**Message-Passing Framework** [74]: In spatial-based methods, each node $v$ updates its features $\mathbf{h}_v$ by aggregating information from its neighbors $N(v)$ of function (3.4). The aggregation function can vary, using operations like mean, sum, or attention mechanisms.

**GraphSAGE** [75]: GraphSAGE improves efficiency by sampling a fixed number of neighbors and applying aggregation functions like mean, pooling, or LSTM. This allows scalable feature updates.

**Graph Attention Networks (GATs)** [76]: GATs use attention mechanisms to assign different importance to each neighbouring node. The attention score between nodes $i$ and $j$ is computed and normalized, allowing adaptive aggregation based on the learned importance of neighbours. This attention-based aggregation makes GATs flexible in focusing on the most relevant information.

## 3.3 Topology Adaptive Graph Convolutions (TAGConv)

Topology Adaptive Graph Convolutions (TAGConv) are a class of graph convolutional methods that unify both spectral and spatial (vertex) domains for efficient learning on graphs [77]. Unlike traditional graph convolutional networks (GCNs), which either rely on fixed filters or operate purely in the spectral domain, TAGConv dynamically adapts its filters to the local topology of the graph. This makes it particularly useful for learning complex representations in heterogeneous and irregular graph structures [78, 79, 80].

### 3.3.1 Overview of TAGConv

TAGConv focuses on the idea that the local topology of the graph can vary widely across different regions. For example, nodes in densely connected regions may require different convolutional filters compared to nodes in sparsely connected regions. TAGConv incorporates this flexibility by adapting its convolutional filters to the local neighborhood structure of each node [77].

The main idea behind TAGConv is to apply polynomial filters over the graph Laplacian, which allows the filters to be adaptive to the node's local topology. By using polynomials of the adjacency matrix $\mathbf{A}$, the convolution can capture multi-hop information and adapt to varying topological structures. The TAGConv operation can be thought of as combining information from both the spectral domain (eigenvectors of the graph Laplacian) and the spatial domain (node neighborhoods).

### 3.3.2 Unifying Spectral and Spatial Domains

TAGConv combines the benefits of spectral and spatial-based graph convolutions by leveraging the polynomial approximation of the graph Laplacian. In spectral methods, graph convolutions are defined in terms of the eigenvectors of the Laplacian $\mathbf{L}$. However, directly using the spectral approach is computationally expensive and often impractical for large graphs, as it involves eigenvalue decomposition.

TAGConv circumvents this problem by approximating the spectral filters using polynomials of the adjacency matrix. The key idea is that the eigenvectors of $\mathbf{L}$ (which capture global graph structure) can be approximated using local neighborhoods of nodes. The adjacency matrix powers $\mathbf{A}^j$ encode multi-hop information that corresponds to different scales in the graph, similar to the spectral approach but computationally more efficient.

Thus, TAGConv achieves a unification of spectral and spatial approaches:

- **Spectral Perspective**: The polynomial filters approximate spectral graph convolutions without the need for expensive spectral decomposition.
- **Spatial Perspective**: The convolution operates directly in the vertex domain by considering local neighborhood structures through powers of the adjacency matrix.

### 3.3.3 Mathematical Formulation of Topology Adaptive Graph Convolutions

Topology Adaptive Graph Convolutions (TAGC) are designed to adaptively learn the structure of the graph and apply convolution operations that respect the underlying topology [77]. In this section, we mathematically formalize the key components of this approach, including the convolution operation, filter design, and the adaptability of the filter.

#### Graph Convolution Operation

In graph signal processing, the convolution operation extends from traditional grid-based data (like images) to graphs, where the structure is defined by nodes and edges [52]. Given a graph $G = (V, E)$ with an adjacency matrix $\mathbf{A}$, the convolution operation at layer $l$, input channel $c$, and output feature $f$ is defined as:

$$\mathbf{G}_{c,f}^{(l)}\mathbf{x}_c^{(l)} = \sum_{k=0}^{K} g_{c,f,k}^{(l)}\mathbf{A}^k\mathbf{x}_c^{(l)} \tag{3.9}$$

where:

- $\mathbf{G}_{c,f}^{(l)}$ is the polynomial filter for input channel $c$ and output feature $f$ at layer $l$,
- $g_{c,f,k}^{(l)}$ are the coefficients of the polynomial filter, which are learnable parameters that adapt to the graph structure,
- $\mathbf{A}$ is the adjacency matrix representing the connections between nodes,
- $\mathbf{x}_c^{(l)}$ is the input feature vector for channel $c$ at layer $l$,
- $K$ is the maximum filter size, which defines how many hops (or neighbours) around each node are considered.

Equation (3.9) shows that the graph convolution is computed by applying a weighted sum of powers of the adjacency matrix $\mathbf{A}$ to the input features. The powers of $\mathbf{A}$ (i.e., $\mathbf{A}^k$) allow the model to aggregate information from $k$-hop neighbours, where $k$ controls the extent of the neighbourhood to be considered in the filtering process.

**Filter Design**

The design of the filter $\mathbf{G}_{c,f}^{(l)}$ as a polynomial of the adjacency matrix is central to adapting the convolution to arbitrary graph structures. By expressing the filter as:

$$\mathbf{G}_{c,f}^{(l)} = \sum_{k=0}^{K} g_{c,f,k}^{(l)} \mathbf{A}^k, \tag{3.10}$$

we ensure that the convolution operation is shift-invariant and respects the graph's topology. The term $\mathbf{A}^k$ considers nodes that are $k$-hops away from each other, making the filter sensitive to both local and global information in the graph. The coefficients $g_{c,f,k}^{(l)}$ act as learnable parameters that adapt the filter to the specific graph structure during training, effectively learning how to weigh neighbours at different distances (up to $K$-hops).

**Adaptability of the Filter**

One of the key aspects of topology-adaptive graph convolutions is their ability to adapt the filtering process to the structure of the graph. The adaptability is introduced through the combination of the learnable polynomial coefficients $g_{c,f,k}^{(l)}$ and the ability to model relationships at different distances in the graph via $\mathbf{A}^k$. This allows the model to dynamically capture both local interactions (via smaller powers of $\mathbf{A}$) and long-range dependencies (via larger powers of $\mathbf{A}$).

At each layer $l$, the output feature map for node $i$ in feature channel $f$, derived from functions (3.9) and (3.10) is given by:

$$y_f^{(l)}(i) = \sum_{c=1}^{C^{(l)}} \sum_{k=1}^{K^{(l)}} \sum_{j \in \tilde{j}} g_{c,f,k}^{(l)} \omega(p_{k,j,i}) x_c^{(l)}(j) + b_f \cdot \mathbf{1}_{N^{(l)}} \tag{3.11}$$

from where:

- $y_f^{(l)}(i)$ is the output feature for node $i$ in feature channel $f$ at layer $l$,
- $C^{(l)}$ is the number of input channels at layer $l$,
- $K^{(l)}$ is the number of filters,
- $\tilde{j}$ denotes the set of nodes $j$ that are reachable from node $i$ through $k$-hop paths,
- $g_{c,f,k}^{(l)}$ represents the weights of the $k$-th filter for the $c$-th channel and $f$-th output feature,
- $\omega(p_{k,j,i})$ represents the correlation strength between nodes $j$ and $i$ for the $k$-th path,
- $x_c^{(l)}(j)$ is the input feature for node $j$ in the $c$-th channel at layer $l$,
- $b_f$ is a bias term applied to the output feature, and
- $\mathbf{1}_{N^{(l)}}$ is a vector of ones with length equal to the number of nodes $N^{(l)}$ in the graph.

The function (3.11) reflects the adaptability of the convolution operation by weighting the neighbours based on both the learned filter coefficients $g_{c,f,k}^{(l)}$ and the correlation strength $\omega(p_{k,j,i})$ between nodes. The adaptive filter is not only sensitive to the graph's structure but can also dynamically adjust the strength of interactions between nodes depending on their topological relationship.

### 3.3.4 Advantages of TAGConv

TAGConv brings several advantages to graph learning:

- **Adaptability to Local Topology**: The filters dynamically adapt to the local structure of the graph, making them highly effective in heterogeneous graphs where node degree and neighbourhood size vary significantly [81].
- **Efficient Multi-Hop Aggregation**: By using powers of the adjacency matrix, TAGConv efficiently captures information from multiple hops in the graph without the need for manually specifying neighbourhood sizes [82].

- **Unification of Spectral and Spatial Domains**: TAGConv combines the interpretability of spectral graph convolution methods with the computational efficiency of spatial methods [78].
- **Scalability**: The polynomial approximation enables efficient computation, making TAGConv scalable to large graphs without requiring expensive eigenvalue decompositions [83].

### 3.3.5 Comparison with Other Graph Convolutions

Compared to traditional GCNs, which use a first-order approximation of the graph Laplacian, TAGConv provides a more flexible and expressive framework by incorporating higher-order neighbourhoods. This allows TAGConv to better capture long-range dependencies and structural patterns in the graph. In comparison with Graph Attention Networks (GATs), which use attention mechanisms to assign importance to neighbours, TAGConv naturally adapts to the topology without requiring explicit attention weights [84]. While GATs may excel in scenarios where fine-grained neighbour importance is necessary, polynomial filter approach of TAGConv offers a more computationally efficient alternative, especially for deep graph networks.

# Chapter 4

# Structural Optimization and Data Generation

## 4.1 Structural optimization approach

The structural optimization technique used in this study is a connectivity level based discrete domain optimization that uses standard linear programming with interior point algorithm [22]. It involves generating a base mesh, constructing a ground structure, and formulating an optimization problem that minimizes the structure's volume under stress constraints. In this section, we provide the complete mathematical formulations, including the derivation of the objective function and constraints, as well as detailed explanations of the problem setup.

### 4.1.1 Base Mesh Generation

The base mesh defines the spatial layout for the optimization problem. It consists of nodes (points in the domain) and elements (connections between nodes). This mesh can either be structured (e.g., a grid) or unstructured, depending on the complexity of the problem domain.

**Node and Element Representation**: The nodes and elements in the domain are represented in matrix form, where NODE ([p] = (x, y)) defines the coordinates of the $p$-th node in 2D space and ELEM is a matrix where each row corresponds to an element and contains the indices of the nodes forming that element.

### 4.1.2 Ground Structure Generation

Given the base mesh, the ground structure is generated by adding potential truss members (bars) between nodes. These members represent possible connections that

can later be optimized for their cross-sectional area.

**Connectivity Matrix A**: Define a connectivity matrix **A** of size $N_n \times N_n$, where $N_n$ is the number of nodes.

$$\mathbf{A}[p, q] = 1 \quad \text{if nodes } p \text{ and } q \text{ are connected by a truss member, and } 0 \quad \text{otherwise.}$$

The ground structure can be created at different **connectivity levels**:

- **Level 1**: Connects all neighboring nodes.
- **Level 2**: Connects neighbors of neighbors, and so on.

After defining the connectivity, redundant or overlapping members are removed using a **collinearity check** to avoid unnecessary bars in the optimization problem. Additionally, **restriction zones** ensure that no members are placed in certain areas of the domain, such as holes or passive regions.

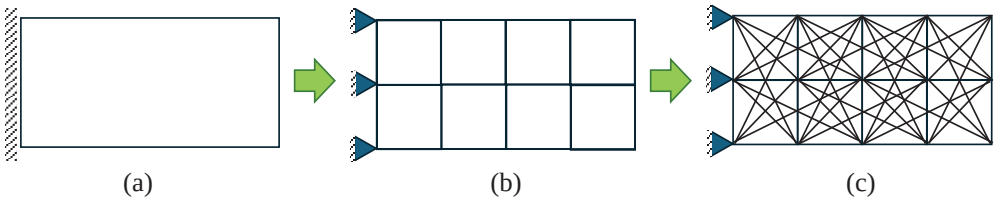



Figure 4.1: Base mesh (b) and ground structure generation (level 2 connectivity) (c) of a rectangular cantilever design domain (a)

### 4.1.3 Mathematical Formulation of the Optimization Problem

The primary goal of the ground structure-based optimization is to minimize the volume (or weight) of the truss structure, subject to equilibrium and stress constraints and this study uses a plastic formulation for the optimization problem.

The plastic formulation of the objective function for structural optimization differs from elastic formulations by focusing on equilibrium rather than compatibility or stress-strain relationships [22]. This method allows for more efficient optimization by treating the design as a linear programming problem, aiming to minimize the structure's volume while ensuring that the stress constraints are satisfied.

#### Objective Function

The objective of the plastic formulation is to minimize the total structural volume $V$, expressed as a function of the cross-sectional areas **a** of the members in the structure. The volume minimization can be formulated as:

$$\min_{\mathbf{a}} V = \mathbf{l}^T \mathbf{a} \tag{4.1}$$

where $\mathbf{l}$ is the vector of member lengths, and $\mathbf{a}$ is the vector of cross-sectional areas of each member. This formulation seeks to reduce the overall material usage while maintaining structural integrity under the applied loads.

**Equilibrium Constraints**

In plastic analysis, equilibrium is enforced without explicitly considering the material compatibility. The equilibrium constraint ensures that the internal forces within the structure balance the external loads applied to it. This constraint can be written as:

$$\mathbf{B}^T \mathbf{n} = \mathbf{f} \tag{4.2}$$

where $\mathbf{B}^T$ is the nodal equilibrium matrix, constructed from the directional cosines of the members, and $\mathbf{n}$ is the vector of internal axial forces for all members in the ground structure. The vector $\mathbf{f}$ represents the external nodal forces applied to the structure. These equilibrium constraints ensure that the structure is balanced and does not experience unbalanced forces.

**Stress Constraints**

In plastic formulation, stress constraints are imposed to ensure that the internal forces in each member do not exceed the allowable limits for tension $\sigma_T$ and compression $\sigma_C$. The constraints are expressed as:

$$-\sigma_C \mathbf{a}_i \leq \mathbf{n}_i \leq \sigma_T \mathbf{a}_i \quad \text{for} \quad i = 1, 2, \ldots, N_b \tag{4.3}$$

This ensures that the internal force $\mathbf{n}_i$ in each member is bounded by the allowable stresses in both tension and compression. In order to simplify the constraints in (4.3), slack variables $\mathbf{s}^+$ and $\mathbf{s}^-$ are introduced, converting the inequalities into equalities:

$$\mathbf{n}_i + \frac{2\sigma_0}{\sigma_C} \mathbf{s}_i^- = \sigma_T \mathbf{a}_i \tag{4.4}$$

$$-\mathbf{n}_i + \frac{2\sigma_0}{\sigma_T} \mathbf{s}_i^+ = \sigma_C \mathbf{a}_i \tag{4.5}$$

With the slack variables $\mathbf{s}^+$ and $\mathbf{s}^-$, the cross-sectional areas $\mathbf{a}_i$ and internal forces $\mathbf{n}_i$ can be derived from equations (4.4) and (4.5) as:

$$\mathbf{a}_i = \frac{\mathbf{s}_i^+}{\sigma_T} + \frac{\mathbf{s}_i^-}{\sigma_C} \tag{4.6}$$

$$\mathbf{n}_i = \mathbf{s}_i^+ - \mathbf{s}_i^- \tag{4.7}$$

This formulation allows a clearer representation of whether a member is in tension or compression, where only one of $\mathbf{s}_i^+$ or $\mathbf{s}_i^-$ will be non-zero.

**Final Optimization Problem Formulation**

With the introduction of slack variables, the optimization problem provided by is transformed into a linear programming problem through equations (4.1), (4.6), (4.7) along with (4.2). The final form of the plastic layout optimization problem can be written as:

$$\min_{\mathbf{s}^+,\mathbf{s}^-} V = \mathbf{l}^T \left( \frac{\mathbf{s}^+}{\sigma_T} + \frac{\mathbf{s}^-}{\sigma_C} \right) \tag{4.8}$$

$$\text{subject to: } \mathbf{B}^T(\mathbf{s}^+ - \mathbf{s}^-) = \mathbf{f} \tag{4.9}$$

$$\mathbf{s}_i^+, \mathbf{s}_i^- \geq 0 \tag{4.10}$$

In this formulation, the objective is to minimize the total volume $V$, with the constraint that the structure remains in equilibrium, as represented by $\mathbf{B}^T(\mathbf{s}^+ - \mathbf{s}^-) = \mathbf{f}$. Additionally, the stress constraints ensure that only one of the slack variables $\mathbf{s}_i^+$ or $\mathbf{s}_i^-$ is non-zero for each member, indicating whether the member is in tension or compression. The optimal solution typically involves a statically determinate structure, where only $N_{\text{dof}}$ non-zero variables remain, representing the most efficient use of material for the given loading conditions.

### 4.1.4 Linear Problem Solution

The above optimization problem provided by equations (4.8), (4.9) and (4.10) is a **Linear Programming (LP) problem**. Interior-point methods or simplex algorithms can be used to efficiently solve it for large-scale ground structures.

**Solution Properties:** The solution to the linear program yields the cross-sectional areas $a_i$ for each member. At the optimal solution, only a small subset of members will have non-zero areas, forming the final **optimal truss layout**. This layout is typically statically determinate, meaning the structure can be analyzed with basic equilibrium equations.

**Post-Processing:** After solving the optimization problem, redundant members (those with zero or negligible cross-sectional areas) and unnecessary nodes (those connected only to two collinear members) are removed to simplify the structure.

Figure 4.2 can be used to summarise this process

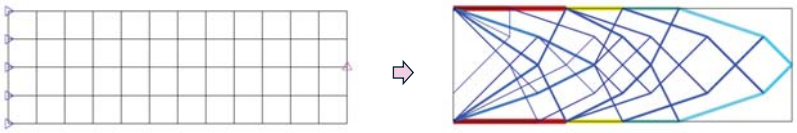



Figure 4.2: Base mesh plotted within the design space along with the final optimized structure

## 4.2 Input and output encoding

### 4.2.1 Input Feature Encoding and Edge Indexing

In the ground structure optimization process using a Graph Neural Network (GNN), the input feature tensor encodes key structural and load characteristics for each node in the design domain. These features include initial stresses, nodal coordinates, support conditions, load application data, and distance-based metrics. Each of these components contributes to a feature vector for every node $k$ in the system.

**Initial Stress Tensor $\mathbf{s}_k$** The initial stress $\mathbf{s}_k \in R^{1\times N_{\text{nd}}}$ of the members from node $k$ to every other node in the ground structure is calculated by solving the matrix equation of equilibrium (refer to Equation (2)). For a member between nodes $k$ and $k'$, the stress $\mathbf{s}_{k,k'}$ is expressed as:

$$\mathbf{s}_{k,k'} = \mathbf{s}^{+}_{k,k'} - \mathbf{s}^{-}_{k,k'}$$

where:

- $\mathbf{s}^{+}_{k,k'} > 0$ if the member is in tension,
- $\mathbf{s}^{-}_{k,k'} > 0$ if the member is in compression,
- $\mathbf{s}^{+}_{k,k'} \cdot \mathbf{s}^{-}_{k,k'} = 0$, meaning a member can only be either in tension or compression, not both simultaneously.

**Node Coordinates and Support Conditions** Each node $k$ in the system is assigned a 2D coordinate represented as:

$$(\mathbf{x}_k, \mathbf{y}_k) \in R^{1\times 2}$$

In addition, the support conditions in the $x$- and $y$-directions are denoted by binary values:

$$(\mathbf{S}_{k,x}, \mathbf{S}_{k,y}) \in \{0,1\}^{1\times 2}$$

where 1 indicates a fixed support, and 0 indicates no support in the corresponding direction.

**Load Application Data** For each node $k$, the external loads applied in the $x$- and $y$-directions are given by:

$$(\mathbf{L}_{k,x}, \mathbf{L}_{k,y}) \in R^{1\times 2}$$

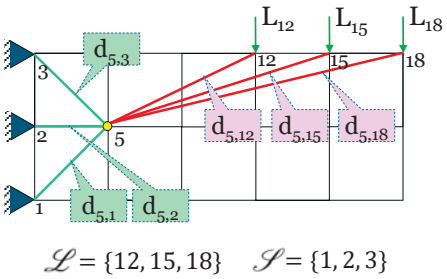



Figure 4.3: Distances from node $k = 5$ to load-applied nodes $\mathscr{L}$ and support-applied nodes $\mathscr{S}$ of an 18 node cantilever structure

**Distance-Based Metrics** Three distance-based metrics are included in the input tensor to capture the geometric relationship between nodes and load/support conditions:

1. **Cumulative Euclidean Distance to Load-Applied Nodes**:

$$\mathbf{D}_k = \sum_{i\in\mathscr{L}} \mathbf{d}_{k,i}$$

   where $\mathbf{d}_{k,i}$ is the Euclidean distance between node $k$ and load-applied node $i$ out of the load applied nodes in $\mathscr{L}$ (see Figure 4.3).

2. **Weighted Sum of Euclidean Distances to Load-Applied Nodes**:

$$\varphi_k = \sum_{i\in\mathscr{L}} \mathbf{d}_{k,i} \cdot \mathbf{L}_i$$

   where $\mathbf{L}_i$ is the magnitude of the load at node $i$.

3. **Cumulative Euclidean Distance to Support-Applied Nodes**:

$$\psi_k = \sum_{j \in \mathscr{S}} \mathbf{d}_{k,j}$$

where $\mathbf{d}_{k,j}$ is the Euclidean distance between node $k$ and support-applied node $j$ out of the load applied nodes in $\mathscr{S}$ (see Figure 4.3).

**Input Feature Vector** The input features for each node $k$ are combined into a feature vector:

$$\mathbf{X}_k = [\mathbf{s}_k, \mathbf{x}_k, \mathbf{y}_k, \mathbf{S}_{k,x}, \mathbf{S}_{k,y}, \mathbf{L}_{k,x}, \mathbf{L}_{k,y}, \mathbf{D}_k, \varphi_k, \psi_k] \in R^{1\times(N_{\text{nd}}+9)}$$

where:

- $\mathbf{s}_k \in R^{1\times N_{\text{nd}}}$ is the initial stress vector for node $k$,
- $\mathbf{x}_k, \mathbf{y}_k \in R^{1\times 2}$ are the node coordinates,
- $\mathbf{S}_{k,x}, \mathbf{S}_{k,y} \in \{0,1\}^{1\times 2}$ are the support conditions,
- $\mathbf{L}_{k,x}, \mathbf{L}_{k,y} \in R^{1\times 2}$ are the load application values,
- $\mathbf{D}_k, \boldsymbol{\varphi}_k, \boldsymbol{\psi}_k \in R$ are the cumulative distance metrics.

Thus, the input tensor $\mathbf{X}$ for all nodes is constructed as:

$$\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{N_{\text{nd}}}] \in R^{N_{\text{nd}}\times(N_{\text{nd}}+9)}$$

**Nodal Connectivity Tensor** The connectivity tensor $\mathbf{C}_k \in \{0,1\}^{1\times N_{\text{nd}}}$ for each node $k$ captures the structural connectivity within the design domain. Each element $c_{k,k'}$ indicates whether node $k$ is connected to node $k'$:

$$\mathbf{C}_k = [c_{k,1}, c_{k,2}, \ldots, c_{k,N_{\text{nd}}}] \quad \text{where} \quad c_{k,k'} = \begin{cases} 1 & \text{if nodes } k \text{ and } k' \text{ are connected,} \\ 0 & \text{otherwise.} \end{cases}$$

### 4.2.2 Output Embedding Tensor

The output embedding tensor $\mathbf{Y}$ captures the optimized structural outputs of the system in the form of nodal embeddings. Each node in the graph is associated with an output embedding that represents the optimized design characteristics, particularly the cross-sectional area of members between connected nodes.

**Output Tensor Definition** For node $k$, the output embedding is represented as a vector $\mathbf{y}_k \in R^{1\times N_{\text{nd}}}$, where each element $y_{k,k'}$ corresponds to the cross-sectional area of the member between nodes $k$ and $k'$.

Thus, the output embedding tensor for the entire system is:

$$\mathbf{Y} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,N_{\text{nd}}} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N_{\text{nd}}} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N_{\text{nd}},1} & y_{N_{\text{nd}},2} & \cdots & y_{N_{\text{nd}},N_{\text{nd}}} \end{bmatrix} \in R^{N_{\text{nd}}\times N_{\text{nd}}}$$

where $y_{k,k'}$ denotes the cross-sectional area of the member between nodes $k$ and $k'$ in the optimized structure.

Therefore, the full output embedding tensor $\mathbf{Y}$ for the optimized ground truth structure is:

$$\mathbf{Y} = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,N_{\text{nd}}} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N_{\text{nd}}} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N_{\text{nd}},1} & y_{N_{\text{nd}},2} & \cdots & y_{N_{\text{nd}},N_{\text{nd}}} \end{pmatrix}$$

## 4.3 Database Generation - Case Studies

In this study, we evaluate the performance of a data-driven model through practical applications involving two structural configurations: a cantilever truss structure and a simply supported truss structure. The model's effectiveness is assessed under varying conditions, specifically by altering the number of nodes and increasing the data sample sizes. The selected node counts are 24, 45, and 60, while the data sample sizes are 3,000, 6,000, and 12,000. During data generation for each case, the loading conditions, design domain dimensions, and load values are varied to explore different scenarios.

### 4.3.1 Case Study 1: Cantilever Truss Structure

In Case Study 1, we examine a cantilever truss problem under six distinct loading conditions (Figure 4.4), which are typical in structural applications. The total load values are varied within a range of 1 kN to 5 kN, using Latin hypercube sampling to ensure a well-distributed set of load values across the specified range. Latin hypercube sampling was chosen over a normal distribution to avoid clustering of data points around the mean, ensuring broader coverage of the possible values. Ten different design domain sizes were used in combination with the loading conditions

to generate the data sets of 3,000, 6,000, and 12,000 samples.

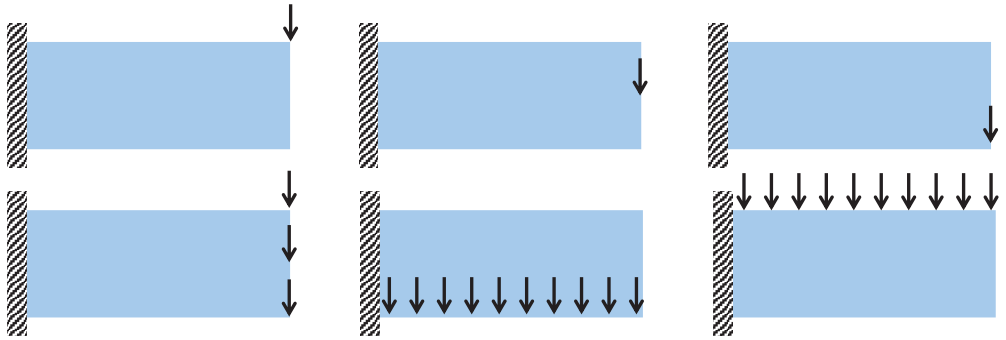


Figure 4.4: Loading conditions considered for the cantilever truss problem

### 4.3.2 Case Study 2: Simply Supported Truss Structure

Case Study 2 focuses on a simply supported truss structure. The structure features a pinned connection at one end and roller supports at the opposite end, simplifying the problem by reducing the number of nodes. As in Case Study 1, the loading conditions (Figure 4.5), design domain sizes, and load values were varied, with the same ranges and combinations applied to generate the respective data sets. This consistent approach allows for a comparative analysis between the two structural configurations.

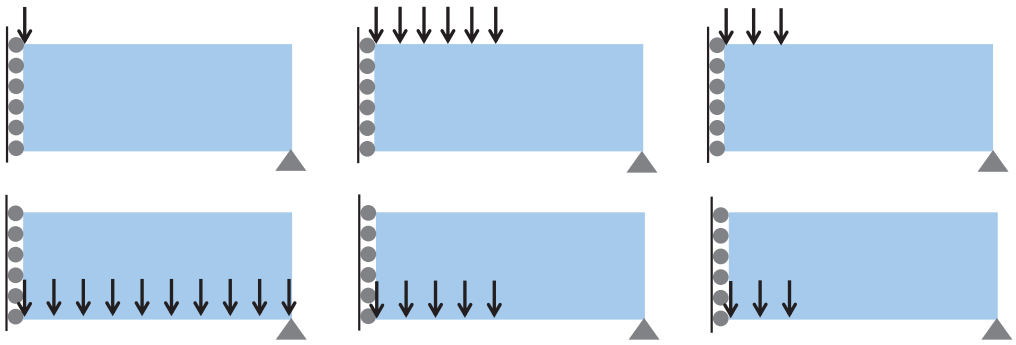


Figure 4.5: Loading conditions considered for the simply supported design problem

# Chapter 5

# Model Development and Evaluation

This section outlines the development of the Graph Neural Network (GNN) model employed for structural optimization, focusing on the model architecture, evaluation techniques, and validation strategies. The model is designed to predict both topology and size optimization for ground structures using a dataset of structural configurations. Each component of the model development process is discussed in detail below.

## 5.1 Model Architecture and Hyperparameters

The architecture of the GNN used in this study is designed to effectively capture the structural characteristics and optimize the layout of ground structures. Specifically, the model was constructed to handle input and output tensors that correspond to the structural features and outputs of the optimization task.

- **Input Layer**: The input layer of the GNN is designed to process the features of the input tensor $\mathbf{X}_k$, which encapsulates the nodal information for each node $k$ in the structure. In the case of a cantilever problem involving 60 nodes in the ground structure, the input layer is configured to handle 69 nodes. These nodes include features such as initial stresses, node coordinates, support conditions, load applications, and distance-based metrics, as outlined in Section 4.2.
- **Hidden Layers**: To capture the complex relationships and patterns inherent in structural optimization, the GNN incorporates two hidden layers:
    1. **First Hidden Layer**: This layer consists of 225 channels and utilizes the Rectified Linear Unit (ReLU) activation function, which introduces

non-linearity into the model [85]. The ReLU function allows the GNN to learn more complex and intricate features by enabling it to handle the non-linear relationships between input features and the structural outputs.

2. **Second Hidden Layer**: This layer contains 121 channels and employs a linear activation function. The linear activation facilitates the final transformations needed for accurate predictions of the optimized structural configurations. The combination of non-linear and linear layers enables the model to capture a broad range of patterns in the data [86].

- **Output Layer**: The output layer is structured to match the dimensionality of the output embedding tensor $\mathbf{y}_k$. For the cantilever problem involving 60 nodes in the ground structure, the output layer is configured with 60 nodes, aligning with the number of nodes in the graph. Each node in the output layer represents the optimized structural properties, such as the cross-sectional areas of the members connecting adjacent nodes.

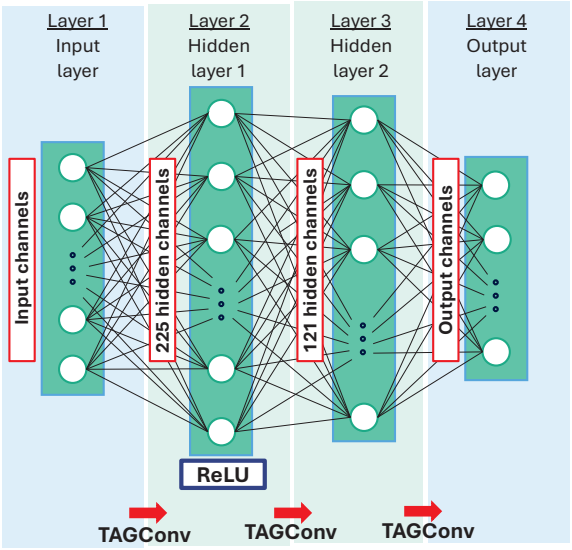



Figure 5.1: GNN model architecture

The model architecture is carefully designed to balance the complexity of the task while ensuring computational efficiency. The two-layer structure with appropriately chosen activation functions allows the GNN to effectively model the non-linear behavior of the structure, while the output layer provides accurate predictions for the optimized design.

The GNN model is trained using the Mean Squared Error (MSE) loss function, which measures the difference between the predicted and actual structural performance.

The MSE is calculated using equation (5.1), where $y_i$ represents the actual values, $\hat{y}_i$ denotes the predicted values, and $N$ is the total number of predictions.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{5.1}$$

To optimize the model parameters, the Adam optimizer [87] is employed with a learning rate of 0.001, chosen after extensive experimentation with different configurations to ensure optimal performance. Additionally, since the model aims to predict an undirected structure, two key constraints are applied to the output adjacency matrix. First, the matrix is enforced to be symmetric, ensuring that the relationships between nodes remain bidirectional. Second, the diagonal elements are set to zero, reflecting the absence of self-connections between nodes. These constraints help maintain the physical validity of the predicted structure. The training was accelerated using the CUDA platform, which leveraged the parallel processing capabilities of Graphics Processing Units (GPUs). This significantly reduced the computational time required for matrix multiplications and convolution operations essential for model training [88].

## 5.2 Data Prepossessing and Post-processing

### 5.2.1 Prepossessing before training

Once the structurally optimised trusses generated through the optimization technique explained in Section 4.1 and the tensor encodings are carried out as mentioned in Section 4.2, before using the data for model training and evaluation, the following preprocessing steps are done.

#### Min-Max Scaling

The use of MSE as the loss function in this study introduces a scaling issue due to the presence of both high-magnitude and smaller values in the data [89]. Larger values dominate the error calculation, diminishing the relative significance of smaller values. As a result, the model tends to prioritize accurate predictions for larger values while failing to predict smaller values with precision. This imbalance adversely affects the optimization process, particularly in terms of the resulting topology. To address this issue, the input data is scaled. Zeros are preserved as zeros, while non-zero values are scaled within the range of 500 to 5000. This normalization helps ensure a balanced influence of all values, regardless of their magnitude. The scaling transformation is defined as follows:

$$\mathbf{X}_{\text{scaled}} = \mathbf{X}_{\text{min}} + \frac{(\mathbf{X} - \mathbf{X}_{\text{orig-min}})}{(\mathbf{X}_{\text{orig-max}} - \mathbf{X}_{\text{orig-min}})} \times (\mathbf{X}_{\text{max}} - \mathbf{X}_{\text{min}}). \tag{5.2}$$

In this equation, $\mathbf{X}$ represents the original data, while $\mathbf{X}_{\text{orig-min}}$ and $\mathbf{X}_{\text{orig-max}}$ denote the minimum and maximum values of the original data, respectively. The terms $\mathbf{X}_{\text{min}}$ and $\mathbf{X}_{\text{max}}$ correspond to the desired scaling range, which in this case is set between 500 and 5000.

**Batching**

To improve memory efficiency during training, batching along with gradient accumulation is employed [90]. A batch size of 16, combined with 4 accumulation steps, was found to be the most efficient configuration. This effectively allows the model to train with an accumulated batch size of $16 \times 4 = 64$, providing a balance between memory usage and training performance.

**Data Splitting**

To evaluate the model on unseen data, the dataset is split into training and testing sets. A random split of 80% of the data for training and 20% for evaluation was used. This ensures that the model is trained on a majority of the data while still reserving a portion for testing and validation purposes.

### 5.2.2 Post processing after prediction

Once the model is trained, the following post-processing steps are applied during the prediction phase:

**Thresholding**

Since the model is trained as a node-level regressor using the MSE loss function, it may predict small non-zero values for locations where the ground truth is zero. These small values can be handled through thresholding, where values below a certain threshold are set to zero. Given that the data was scaled to the range of 500 to 5000 during preprocessing, a threshold of 100 is applied. This means all predicted values below 100 will be set to zero, removing unnecessary bars to obtain the final optimized out.

**Scaling Back**

After prediction, the scaled data must be transformed back to its original scale to obtain the actual cross-sectional values. This is achieved by inverting the scaling equation used during preprocessing (equation (5.2)). The inverse scaling equation is given by:

$$\mathbf{X}_{\text{orig}} = \mathbf{X}_{\text{orig-min}} + \frac{(\mathbf{X}_{\text{scaled}} - \mathbf{X}_{\text{min}})}{(\mathbf{X}_{\text{max}} - \mathbf{X}_{\text{min}})} \times (\mathbf{X}_{\text{orig-max}} - \mathbf{X}_{\text{orig-min}}) \tag{5.3}$$

This transformation returns the predicted values to their original scale, providing meaningful structural properties for evaluation.

## 5.3 Model Validation Techniques

Model validation is a crucial step in ensuring that the GNN generalizes effectively to unseen data. The dataset is divided into training and validation sets, with the model being trained on the training set and evaluated on the validation set. As outlined in the previous section, the dataset is split into these two subsets to assess the model performance on new, unseen data. During training, the loss value is recorded for each epoch, alongside the corresponding evaluation loss from the validation set. Ideally, the training and validation error values should converge, indicating that the model is neither overfitting nor underfitting [91]. If both losses show convergence, the model can be considered validated, demonstrating its ability to generalize to unseen data effectively.

## 5.4 Model Evaluation Techniques

Model evaluation is crucial for assessing the performance of the GNN in predicting optimal structural configurations. Several evaluation metrics are used to quantify the accuracy of the model's predictions in both topology and size optimization tasks.

### 5.4.1 Topology Optimization Evaluation

For topology optimization, the focus is on predicting the correct presence or absence of edges in the optimized structure. The following metrics are employed:

- **Precision**: Precision measures the proportion of true positive predictions (TP) out of all predicted positive instances (TP + FP), where TP refers to the count of edges correctly predicted as present, and FP refers to the count of edges incorrectly predicted as present. Precision is mathematically defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.4}$$

- **Recall**: Recall evaluates the proportion of true positive predictions (TP) out of all actual positive instances (TP + FN), where FN refers to the count of

edges incorrectly predicted as absent. Recall is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.5}$$

- **F1 Score**: The F1 score provides a harmonic mean of precision and recall, ensuring a balanced evaluation when both false positives and false negatives are present. It is computed as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.6}$$

- **Jaccard Similarity**: This metric quantifies the similarity between the predicted set of positive edges and the actual set of positive edges. It is calculated as:

$$\text{Jaccard Similarity} = \frac{TP}{TP + FP + FN} \tag{5.7}$$

  The Jaccard similarity is particularly useful in measuring how closely the predicted structure matches the ground truth structure.

### 5.4.2 Size Optimization Evaluation

For size optimization, where the goal is to predict the correct cross-sectional area of each member in the optimized structure, two error metrics are used:

- **Mean Absolute Percentage Error (MAPE)**: MAPE measures the average magnitude of errors between the predicted and actual values. It is expressed as:

$$\text{MAPE} = \frac{1}{|E_{\text{gt}}|} \sum_{i=1}^{|E_{\text{gt}}|} \left| \frac{A_i - P_i}{A_i} \right| \times 100 \tag{5.8}$$

  where $E_{\text{gt}}$ is the set of edges in the ground truth, $A_i$ is the actual value, and $P_i$ is the predicted value for edge $i$.

- **Symmetric Mean Absolute Percentage Error (SMAPE)**: SMAPE accounts for the relative errors in predictions by taking into consideration the sum of actual and predicted values. It is defined as:

$$\text{SMAPE} = \frac{1}{|E_{\text{gt}} \cup E_{\text{pred}}|} \sum_{i=1}^{|E_{\text{gt}} \cup E_{\text{pred}}|} \frac{|P_i - A_i|}{|P_i| + |A_i|} \times 100 \tag{5.9}$$

  This metric provides a more symmetric view of prediction accuracy, especially in cases where the actual and predicted values vary significantly.

# Chapter 6

# Results

This section presents the results from the training of the GNN model and the outcomes of case studies conducted on a cantilever structure under various load conditions. The performance of the GNN-based surrogate model is evaluated using metrics such as MSE, precision, recall, F1 score, Jaccard similarity, and error rates (MAPE, SMAPE). These results demonstrate the model's predictive capability for both topology and size optimization of ground structures.

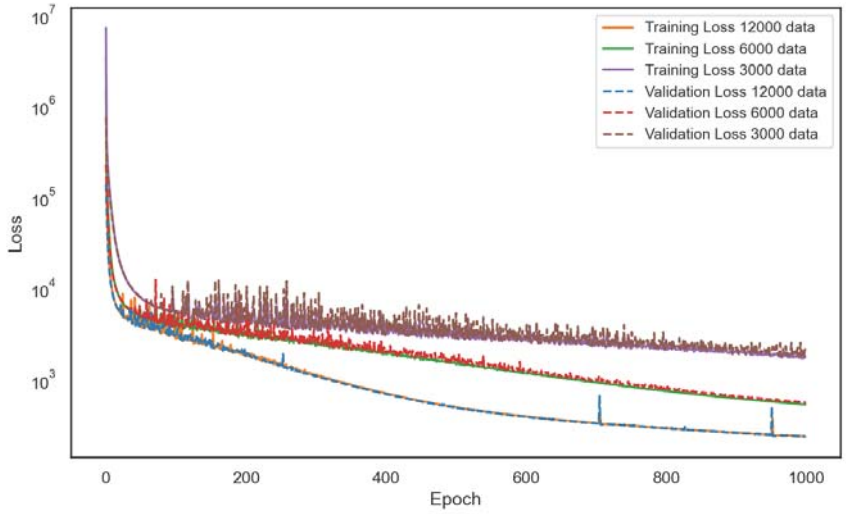
## 6.1 Training Results and Model Validation

The GNN model was developed to predict the topology and size-optimized configuration of structure when the design space, loading conditions and load values are provided. The training of the model carried out and provided are the results of the training process.

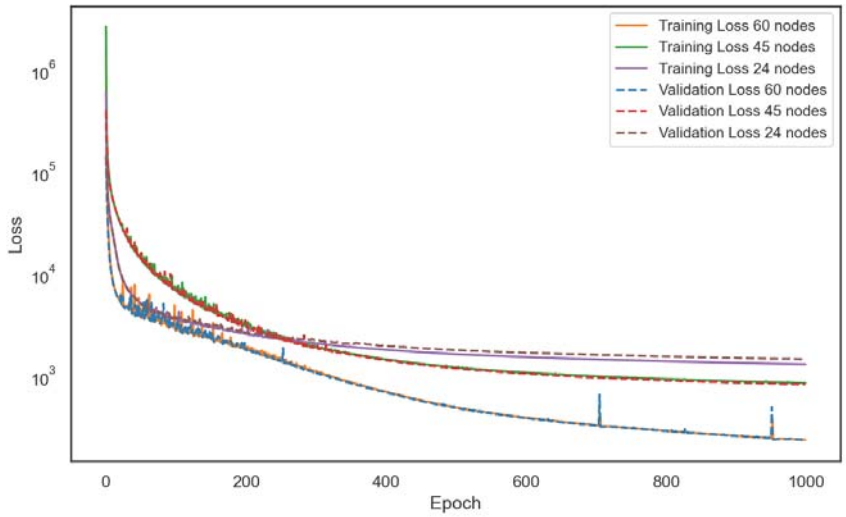
Since the data base was split into two parts for the training and evaluation, the evaluation data set was used for the validation purpose as well. Here, it is important to have error values close to each other so that the model otherwise the model is either over fitted or under fitted. This is checked in both case studies that the GNN model was applied for.

### 6.1.1 Case study 1

Based on the results presented in Figure 6.1, the model can be validated, as all models exhibit good convergence in both training and validation errors. The errors correspond to the mean squared error values calculated during the model training process. It should be noted that the post-processing steps, including thresholding and scaling back (as described in Section 5.2.2), are not considered in these error




(a)




(b)

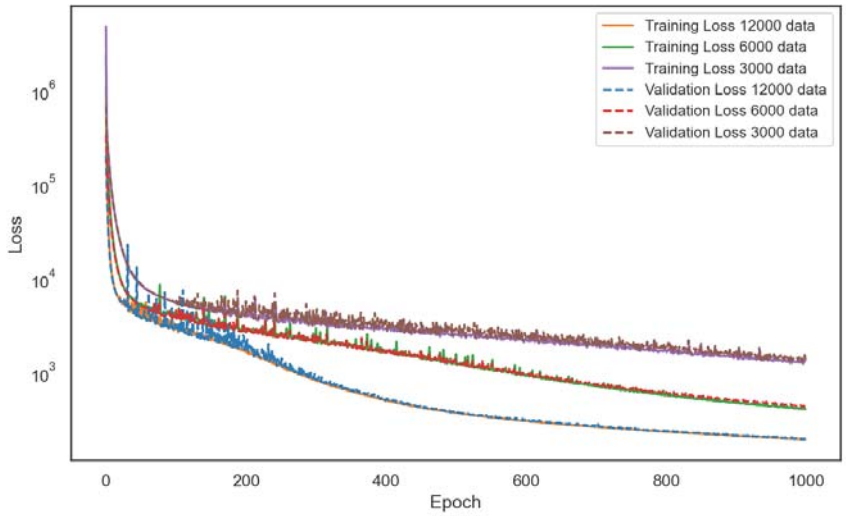
Figure 6.1: (a) Training and validation loss for (a) 60 nodes for 12 000, 6000 and 3000 data points and (b) 12 000 data points in 60, 45 and 24 nodes in the cantilever truss problem

calculations.

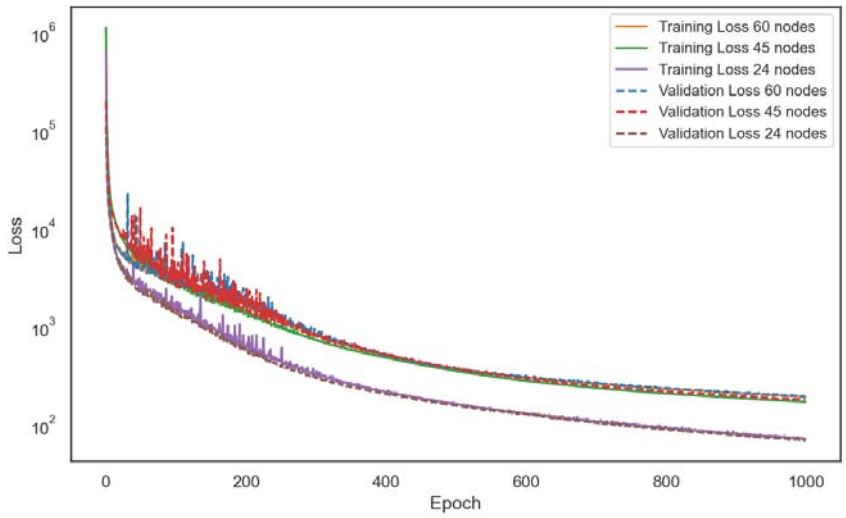
As shown in Figure 6.1a, the error decreases as the sample size increases. Similarly, when examining the variation in error values with respect to the number of nodes, as depicted in Figure 6.1b, the GNN model demonstrates improved training performance as the number of nodes increases. Several factors contribute to the success of the predictions, with the ability to capture the complexity of the problem being a critical one. Upon analyzing the error variations, the lower error values observed for the higher node cases might initially seem counterintuitive. However, a closer examination of the topologies in the ground-truth data reveals that the 45-node and

24-node structures of the cantilever problem exhibit greater variability in topology compared to the 60-node structures. This suggests that the complexity of the optimization problem, particularly with respect to topology variation as load values and design domain sizes change, is higher in cases with fewer nodes. This observation will be explored further in subsequent sections of the results.

### 6.1.2 Case study 2




(a)




(b)

Figure 6.2: (a) Training and validation loss for (a) 60 nodes for 12 000, 6000 and 3000 data points and (b) 12 000 data points in 60, 45 and 24 nodes in the simply supported truss problem

Similar to Case Study 1, the model can be validated as all models exhibited good convergence for both training and validation errors. As shown in Figure 6.2a, the

error decreases as the sample size increases. However, when examining the error variation with respect to the number of nodes, as illustrated in Figure 6.2b, the GNN model demonstrates improved training performance when the number of nodes decreases, contrary to the observations in Case Study 1. Analyzing the database reveals that the topology variations of the optimized structures in this case study are not as significant as those observed in Case Study 1. Consequently, the complexity of the optimization problem is closely linked to the number of nodes, with higher node counts corresponding to increased error values.

## 6.2 Prediction Capability Evaluation

In this section, the model evaluation is carried out using the error metrics explained in Section 5.4.

### 6.2.1 Case study 1

The mean errors, both overall and categorized by loading type (as shown in Figure 4.4), are presented in Tables 6.1, 6.2, and 6.3. Observing these results, we can see a consistent reduction in error values as the number of data points increases, aligning with the trend shown in Figure 6.1a. An important observation is that the recall values for structural predictions are perfect (i.e., equal to unity) across all cases, with the exception of structure 6 in the 24-node case and structure 1 in the 45-node case. In the 60-node case, the recall is consistently equal to unity for all structures. This improvement in performance as the number of nodes increases is consistent with the findings illustrated in Figure 6.2b. Achieving a perfect recall rate is critical, as it indicates that every member present in the ground truth is accurately identified in the predictions, ensuring comprehensive structural representation.

The precision of the predictions is notably high, with most values exceeding 0.9 and a few slightly lower, yet still equal or above 0.85. This reflects the strong predictive capabilities of the framework. Figures 6.3, 6.4, and 6.5 illustrate the best and worst predictions for load case 1 across different node numbers, trained with a sample size of 12,000. The best predictions were selected based on the structure with the lowest SMAPE value, all of which achieved a Jaccard index of 1, indicating a perfect match. For the worst predictions, the selection was made based on the structure with the lowest Jaccard index value. As depicted in the figures, the worst-predicted structure in the 45-node case has a Jaccard index of 0.7, whereas, in the 24-node case, even the worst prediction reached a Jaccard index of 1.

The accuracy of the model's predictions depends on how effectively the GNN cap-

tures the underlying optimization problem. As shown in Table 6.2, the error metrics indicate poor prediction performance for Structure 1, where a point load is applied at the centre of the free end. This poor prediction, combined with the outcomes for other structures, has resulted in higher error rates for the 45-node case compared to the 24-node and 60-node cases. The reason for this anomaly can be attributed to the variability within the database. In the dataset, the design parameters (i.e. loading condition, load value, and design domain size) are altered, leading to changes in topology and the cross-sectional area of the final structure. While the 45-node structures exhibit a high degree of topology variation, the GNN achieves a high recall rate for this case, although precision is significantly lower compared to the other cases. In contrast, the lower variability in topology observed in the 24-node and 60-node structures contributes to their lower error rates when compared to the 45-node case.

| | Metric | Overall | Structure 1 | Structure 2 | Structure 3 | Structure 4 | Structure 5 | Structure 6 |
|---|---|---|---|---|---|---|---|---|
| 12 000 | MSE testing | 1262.9 | 499.5 | 78.6 | 91.7 | 93.7 | 81.7 | 6258.7 |
| | RMSE testing | 35.5 | 22.3 | 8.9 | 9.6 | 9.7 | 9.0 | 79.1 |
| | NRMSE testing | 0.0071 | 0.0068 | 0.0018 | 0.0019 | 0.0051 | 0.0048 | 0.0263 |
| | R2 value | 0.9910 | 0.9969 | 0.9996 | 0.9996 | 0.9983 | 0.9985 | 0.9504 |
| | Precision | 0.9772 | 0.9824 | 0.9983 | 0.9920 | 1 | 0.9980 | 0.9436 |
| | Recall | 0.999 | 1 | 1 | 1 | 1 | 1 | 0.995 |
| | F1score | 0.9878 | 0.9911 | 0.9991 | 0.9960 | 1 | 0.9990 | 0.9689 |
| | Jaccard | 0.9758 | 0.9824 | 0.9983 | 0.9920 | 1 | 0.9980 | 0.9396 |
| | MAPE | 6.3 | 4.0 | 1.9 | 2.4 | 3.2 | 3.0 | 13.1 |
| | SMAPE | 10.9 | 7.6 | 2.2 | 4.1 | 3.3 | 3.5 | 24.2 |
| 6 000 | MSE testing | 1393.9 | 888.4 | 166.8 | 219.5 | 182.8 | 198.4 | 7437.2 |
| | RMSE testing | 37.3 | 29.8 | 12.9 | 14.8 | 13.5 | 14.1 | 86.2 |
| | NRMSE testing | 0.0075 | 0.0090 | 0.0026 | 0.0030 | 0.0072 | 0.0075 | 0.0287 |
| | R2 value | 0.9900 | 0.9941 | 0.9992 | 0.9990 | 0.9964 | 0.9963 | 0.9428 |
| | Precision | 0.9766 | 0.9795 | 0.9796 | 0.9884 | 0.9959 | 0.9976 | 0.9453 |
| | Recall | 0.999 | 1 | 1 | 1 | 1 | 1 | 0.996 |
| | F1score | 0.9877 | 0.9896 | 0.9897 | 0.9941 | 0.9980 | 0.9988 | 0.9702 |
| | Jaccard | 0.9757 | 0.9795 | 0.9796 | 0.9884 | 0.9959 | 0.9976 | 0.9421 |
| | MAPE | 7.6 | 5.5 | 3.0 | 3.7 | 4.9 | 4.7 | 15.7 |
| | SMAPE | 12.5 | 10.0 | 7.1 | 6.2 | 5.9 | 5.4 | 26.7 |
| 3 000 | MSE testing | 1962.9 | 1717.6 | 440.6 | 568.1 | 641.9 | 581.6 | 8272.6 |
| | RMSE testing | 44.3 | 41.4 | 21.0 | 23.8 | 25.3 | 24.1 | 91.0 |
| | NRMSE testing | 0.0089 | 0.0126 | 0.0042 | 0.0048 | 0.0134 | 0.0127 | 0.0305 |
| | R2 value | 0.9860 | 0.9896 | 0.9980 | 0.9973 | 0.9879 | 0.9894 | 0.9385 |
| | Precision | 0.9538 | 0.9545 | 0.9854 | 0.9703 | 0.9645 | 0.9780 | 0.9169 |
| | Recall | 0.999 | 1 | 1 | 1 | 1 | 1 | 0.998 |
| | F1score | 0.9761 | 0.9767 | 0.9926 | 0.9849 | 0.9819 | 0.9889 | 0.9558 |
| | Jaccard | 0.9533 | 0.9545 | 0.9854 | 0.9703 | 0.9645 | 0.9780 | 0.9154 |
| | MAPE | 11.7 | 8.7 | 5.6 | 6.7 | 10.0 | 9.1 | 19.9 |
| | SMAPE | 21.3 | 18.3 | 8.9 | 13.0 | 17.6 | 14.1 | 36.2 |

Table 6.1: Mean performance metrics for all testing data and for individual loading conditions for 24 node cantilever truss structure

| | Metric | Overall | Structure 1 | Structure 2 | Structure 3 | Structure 4 | Structure 5 | Structure 6 |
|---|---|---|---|---|---|---|---|---|
| 12 000 | MSE testing | 676.2 | 2348.3 | 146.1 | 131.1 | 223.9 | 380.9 | 816.6 |
| | RMSE testing | 26.0 | 48.5 | 12.1 | 11.5 | 15.0 | 19.5 | 28.6 |
| | NRMSE testing | 0.0052 | 0.0097 | 0.0025 | 0.0024 | 0.0077 | 0.0100 | 0.0066 |
| | R2 value | 0.9927 | 0.9797 | 0.9989 | 0.9990 | 0.9935 | 0.9891 | 0.9925 |
| | Precision | 0.9679 | 0.8660 | 0.9993 | 0.9988 | 0.9963 | 0.9795 | 0.9734 |
| | Recall | 0.99996 | 0.9997 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9837 | 0.9281 | 0.9996 | 0.9994 | 0.9981 | 0.9897 | 0.9865 |
| | Jaccard | 0.9678 | 0.8658 | 0.9993 | 0.9988 | 0.9963 | 0.9795 | 0.9734 |
| | MAPE | 7.2 | 11.5 | 4.5 | 4.1 | 6.5 | 7.4 | 8.2 |
| | SMAPE | 14.2 | 38.8 | 4.8 | 4.5 | 7.6 | 11.9 | 14.1 |
| 6 000 | MSE testing | 978.6 | 2771.6 | 330.1 | 340.8 | 480.2 | 691.7 | 1215.9 |
| | RMSE testing | 31.3 | 52.6 | 18.2 | 18.5 | 21.9 | 26.3 | 34.9 |
| | NRMSE testing | 0.0063 | 0.0105 | 0.0038 | 0.0038 | 0.0112 | 0.0134 | 0.0080 |
| | R2 value | 0.9891 | 0.9748 | 0.9973 | 0.9972 | 0.9863 | 0.9801 | 0.9889 |
| | Precision | 0.9553 | 0.8507 | 0.9813 | 0.9872 | 0.9796 | 0.9672 | 0.9728 |
| | Recall | 0.9998 | 0.999 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9770 | 0.9188 | 0.9906 | 0.9935 | 0.9897 | 0.9833 | 0.9862 |
| | Jaccard | 0.9551 | 0.8498 | 0.9813 | 0.9872 | 0.9796 | 0.9672 | 0.9728 |
| | MAPE | 10.9 | 15.2 | 7.2 | 7.4 | 9.8 | 11.4 | 12.4 |
| | SMAPE | 20.6 | 45.8 | 11.4 | 10.5 | 14.6 | 18.6 | 18.9 |
| 3 000 | MSE testing | 1807.1 | 3610.9 | 1292.0 | 848.8 | 1332.1 | 1556.7 | 2232.8 |
| | RMSE testing | 42.5 | 60.1 | 35.9 | 29.1 | 36.5 | 39.5 | 47.3 |
| | NRMSE testing | 0.0085 | 0.0120 | 0.0074 | 0.0060 | 0.0186 | 0.0201 | 0.0108 |
| | R2 value | 0.9804 | 0.9691 | 0.9902 | 0.9935 | 0.9616 | 0.9544 | 0.9793 |
| | Precision | 0.8491 | 0.7336 | 0.7508 | 0.8701 | 0.9181 | 0.8845 | 0.8962 |
| | Recall | 0.9998 | 0.999 | 1 | 1 | 1 | 1 | 0.9997 |
| | F1score | 0.9183 | 0.8460 | 0.8577 | 0.9305 | 0.9573 | 0.9387 | 0.9451 |
| | Jaccard | 0.8489 | 0.7331 | 0.7508 | 0.8701 | 0.9181 | 0.8845 | 0.8960 |
| | MAPE | 17.3 | 19.1 | 12.7 | 11.9 | 18.1 | 19.6 | 18.7 |
| | SMAPE | 47.4 | 70.8 | 60.8 | 37.4 | 36.0 | 43.2 | 40.5 |

Table 6.2: Mean performance metrics for all testing data and for individual loading conditions for 45 node cantilever truss structure

### 6.2.2 Case study 2

The overall and loading type-specific mean errors, as shown in Figure 4.5, are presented in Tables 6.4, 6.5, and 6.6. Similar to case study 1, we observe a reduction in error values as the number of data points increases, aligning with the trend depicted in Figure 6.2a. In the 24-node case, all structures exhibit a recall value of 1 (Table 6.4). In the 45-node case, all structures, except structures 3 and 6, also have perfect recall values, with those exceptions still showing values above 0.999. Consistent with case study 1, the 60-node case demonstrates perfect recall values for all structures. However, when examining the MSE, RMSE, and NRMSE values, an increasing trend is observed as the number of nodes increases, as seen in Tables 6.4, 6.5, and 6.6. This pattern mirrors the behavior shown in Figure 6.2b.

The precision of the predictions is notably high, with all values exceeding 0.9, which is an improvement over the cantilever problem where a few precision values fell below this threshold. Figures 6.6, 6.7, and 6.8 illustrate the best and worst predictions for load case 1 across different node counts, trained with a sample size of 12,000. The selection criteria for the best and worst predictions followed the same methodology

| | Metric | Overall | Structure 1 | Structure 2 | Structure 3 | Structure 4 | Structure 5 | Structure 6 |
|---|---|---|---|---|---|---|---|---|
| 12 000 | MSE testing | 108.8 | 102.8 | 153.5 | 98.6 | 69.9 | 62.1 | 159.4 |
| | RMSE testing | 10.4 | 10.1 | 12.4 | 9.9 | 8.4 | 7.9 | 12.6 |
| | NRMSE testing | 0.0021 | 0.0020 | 0.0025 | 0.0020 | 0.0034 | 0.0032 | 0.0025 |
| | R2 value | 0.9985 | 0.9989 | 0.9983 | 0.9989 | 0.9977 | 0.9979 | 0.9982 |
| | Precision | 0.9964 | 0.9927 | 0.9860 | 0.9980 | 0.9995 | 0.9993 | 1.0000 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9982 | 0.9963 | 0.9929 | 0.9990 | 0.9998 | 0.9997 | 1.0000 |
| | Jaccard | 0.9964 | 0.9927 | 0.9860 | 0.9980 | 0.9995 | 0.9993 | 1.0000 |
| | MAPE | 4.3 | 4.1 | 5.0 | 4.1 | 3.8 | 3.7 | 5.2 |
| | SMAPE | 5.2 | 5.6 | 7.9 | 4.6 | 4.1 | 3.9 | 5.6 |
| 6 000 | MSE testing | 310.1 | 315.5 | 417.3 | 286.4 | 181.8 | 203.5 | 464.5 |
| | RMSE testing | 17.6 | 17.8 | 20.4 | 16.9 | 13.5 | 14.3 | 21.6 |
| | NRMSE testing | 0.0035 | 0.0036 | 0.0041 | 0.0034 | 0.0055 | 0.0058 | 0.0043 |
| | R2 value | 0.9956 | 0.9966 | 0.9955 | 0.9969 | 0.9937 | 0.9930 | 0.9949 |
| | Precision | 0.9820 | 0.9511 | 0.9571 | 0.9851 | 0.9969 | 0.9979 | 0.9980 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9909 | 0.9750 | 0.9781 | 0.9925 | 0.9985 | 0.9989 | 0.9990 |
| | Jaccard | 0.9820 | 0.9511 | 0.9571 | 0.9851 | 0.9969 | 0.9979 | 0.9980 |
| | MAPE | 7.8 | 7.6 | 8.7 | 7.3 | 6.6 | 6.9 | 9.8 |
| | SMAPE | 11.8 | 17.4 | 17.4 | 10.6 | 7.6 | 7.8 | 11.1 |
| 3 000 | MSE testing | 1278.6 | 1326.2 | 1892.4 | 1233.6 | 875.8 | 691.3 | 1686.0 |
| | RMSE testing | 35.8 | 36.4 | 43.5 | 35.1 | 29.6 | 26.3 | 41.1 |
| | NRMSE testing | 0.0072 | 0.0073 | 0.0087 | 0.0071 | 0.0120 | 0.0107 | 0.0083 |
| | R2 value | 0.9818 | 0.9858 | 0.9787 | 0.9860 | 0.9699 | 0.9772 | 0.9822 |
| | Precision | 0.8915 | 0.7997 | 0.8403 | 0.8845 | 0.9355 | 0.9589 | 0.9177 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9426 | 0.8887 | 0.9132 | 0.9387 | 0.9667 | 0.9790 | 0.9571 |
| | Jaccard | 0.8915 | 0.7997 | 0.8403 | 0.8845 | 0.9355 | 0.9589 | 0.9177 |
| | MAPE | 17.5 | 16.1 | 21.3 | 17.0 | 16.5 | 14.5 | 20.1 |
| | SMAPE | 39.2 | 54.2 | 51.8 | 39.8 | 30.2 | 23.2 | 38.1 |

Table 6.3: Mean performance metrics for all testing data and for individual loading conditions for 60 node cantilever truss structure

outlined in Section 6.2.1. As shown in the figures, the Jaccard index is consistently equal to 1 for all cases, except for the 60-node structure, where it slightly deviates.

Unlike in Case Study 2, the variations in topology across the load cases are less pronounced, resulting in lower prediction errors. This is evident when analyzing the ground-truth optimized structures, reinforcing the conclusion that the complexity of the optimization problem is more strongly correlated with the number of nodes. However, for the 45-node case in load cases 2, 3, and 6, the SMAPE values are notably higher. These greater prediction errors can be attributed to significant topology variations observed in the ground truths in certain scenarios.

## 6.3 Optimization acceleration

In addition to achieving high prediction accuracy, the GNN-based framework has significantly enhanced the efficiency of the optimization process by accelerating prediction times by a factor of more than 20 6.7.

| | Metric | Overall | Structure 1 | Structure 2 | Structure 3 | Structure 4 | Structure 5 | Structure 6 |
|---|---|---|---|---|---|---|---|---|
| 12 000 | MSE testing | 27.6 | 22.4 | 33.8 | 20.1 | 11.3 | 38.3 | 38.7 |
| | RMSE testing | 5.3 | 4.7 | 5.8 | 4.5 | 3.4 | 6.2 | 6.2 |
| | NRMSE testing | 0.0011 | 0.0020 | 0.0025 | 0.0013 | 0.0007 | 0.0018 | 0.0012 |
| | R2 value | 0.9998 | 0.9996 | 0.9996 | 0.9998 | 0.9999 | 0.9997 | 0.9998 |
| | Precision | 0.9998 | 1.0000 | 0.9995 | 1.0000 | 0.9998 | 1.0000 | 0.9997 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9999 | 1.0000 | 0.9997 | 1.0000 | 0.9999 | 1.0000 | 0.9999 |
| | Jaccard | 0.9998 | 1.0000 | 0.9995 | 1.0000 | 0.9998 | 1.0000 | 0.9997 |
| | MAPE | 1.5 | 1.6 | 1.7 | 1.2 | 0.8 | 1.7 | 1.6 |
| | SMAPE | 1.6 | 1.6 | 1.9 | 1.3 | 0.9 | 1.8 | 1.7 |
| 6 000 | MSE testing | 75.5 | 54.7 | 88.3 | 49.0 | 37.6 | 129.7 | 85.0 |
| | RMSE testing | 8.7 | 7.4 | 9.4 | 7.0 | 6.1 | 11.4 | 9.2 |
| | NRMSE testing | 0.0017 | 0.0031 | 0.0040 | 0.0020 | 0.0012 | 0.0033 | 0.0018 |
| | R2 value | 0.9994 | 0.9989 | 0.9988 | 0.9996 | 0.9998 | 0.9988 | 0.9995 |
| | Precision | 0.9985 | 0.9984 | 0.9978 | 1.0000 | 0.9949 | 0.9995 | 0.9993 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9993 | 0.9992 | 0.9989 | 1.0000 | 0.9975 | 0.9998 | 0.9997 |
| | Jaccard | 0.9985 | 0.9984 | 0.9978 | 1.0000 | 0.9949 | 0.9995 | 0.9993 |
| | MAPE | 2.7 | 2.7 | 2.9 | 1.9 | 1.5 | 3.6 | 2.5 |
| | SMAPE | 3.0 | 3.1 | 3.5 | 2.0 | 2.5 | 3.9 | 2.8 |
| 3 000 | MSE testing | 264.3 | 274.1 | 185.7 | 186.0 | 207.7 | 387.9 | 360.1 |
| | RMSE testing | 16.3 | 16.6 | 13.6 | 13.6 | 14.4 | 19.7 | 19.0 |
| | NRMSE testing | 0.0033 | 0.0070 | 0.0059 | 0.0039 | 0.0029 | 0.0057 | 0.0038 |
| | R2 value | 0.9978 | 0.9949 | 0.9975 | 0.9985 | 0.9989 | 0.9962 | 0.9979 |
| | Precision | 0.9916 | 0.9902 | 0.9967 | 0.9993 | 0.9814 | 0.9978 | 0.9824 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9958 | 0.9951 | 0.9984 | 0.9996 | 0.9906 | 0.9989 | 0.9911 |
| | Jaccard | 0.9916 | 0.9902 | 0.9967 | 0.9993 | 0.9814 | 0.9978 | 0.9824 |
| | MAPE | 5.4 | 6.6 | 4.8 | 4.2 | 3.9 | 7.1 | 5.5 |
| | SMAPE | 7.4 | 8.9 | 5.6 | 4.5 | 7.7 | 8.1 | 9.2 |

Table 6.4: Mean performance metrics for all testing data and for individual loading conditions for 24 node simply supported truss structure

| | Metric | Overall | Structure 1 | Structure 2 | Structure 3 | Structure 4 | Structure 5 | Structure 6 |
|---|---|---|---|---|---|---|---|---|
| 12 000 | MSE testing | 93.3 | 110.5 | 103.6 | 75.3 | 88.0 | 63.0 | 118.5 |
| | RMSE testing | 9.7 | 10.5 | 10.2 | 8.7 | 9.4 | 7.9 | 10.9 |
| | NRMSE testing | 0.0019 | 0.0046 | 0.0042 | 0.0023 | 0.0019 | 0.0021 | 0.0022 |
| | R2 value | 0.9984 | 0.9958 | 0.9972 | 0.9988 | 0.9989 | 0.9989 | 0.9985 |
| | Precision | 0.9986 | 0.9996 | 1.0000 | 0.9976 | 0.9964 | 0.9989 | 0.9986 |
| | Recall | 0.9999 | 1 | 1 | 0.9997 | 1 | 1 | 0.9997 |
| | F1score | 0.9992 | 0.9998 | 1.0000 | 0.9986 | 0.9982 | 0.9995 | 0.9991 |
| | Jaccard | 0.9985 | 0.9996 | 1.0000 | 0.9972 | 0.9964 | 0.9989 | 0.9982 |
| | MAPE | 4.4 | 5.2 | 4.8 | 4.0 | 3.9 | 3.5 | 4.9 |
| | SMAPE | 4.9 | 5.6 | 5.1 | 4.7 | 4.7 | 3.8 | 5.6 |
| 6 000 | MSE testing | 207.7 | 217.9 | 149.5 | 195.0 | 186.1 | 214.4 | 292.1 |
| | RMSE testing | 14.4 | 14.8 | 12.2 | 14.0 | 13.6 | 14.6 | 17.1 |
| | NRMSE testing | 0.0029 | 0.0065 | 0.0051 | 0.0038 | 0.0027 | 0.0039 | 0.0034 |
| | R2 value | 0.9963 | 0.9920 | 0.9960 | 0.9965 | 0.9977 | 0.9962 | 0.9964 |
| | Precision | 0.9918 | 0.9959 | 0.9988 | 0.9776 | 0.9897 | 0.9974 | 0.9865 |
| | Recall | 0.9997 | 1 | 1 | 0.9989 | 1 | 1 | 0.9992 |
| | F1score | 0.9957 | 0.9979 | 0.9994 | 0.9881 | 0.9948 | 0.9987 | 0.9928 |
| | Jaccard | 0.9915 | 0.9959 | 0.9988 | 0.9765 | 0.9897 | 0.9974 | 0.9857 |
| | MAPE | 7.1 | 8.3 | 6.0 | 7.0 | 6.3 | 7.0 | 8.3 |
| | SMAPE | 9.3 | 9.6 | 6.5 | 11.9 | 8.8 | 8.1 | 11.8 |
| 3 000 | MSE testing | 613.7 | 704.1 | 560.8 | 445.2 | 616.2 | 682.1 | 676.3 |
| | RMSE testing | 24.8 | 26.5 | 23.7 | 21.1 | 24.8 | 26.1 | 26.0 |
| | NRMSE testing | 0.0050 | 0.0119 | 0.0098 | 0.0057 | 0.0050 | 0.0070 | 0.0052 |
| | R2 value | 0.9892 | 0.9730 | 0.9854 | 0.9929 | 0.9923 | 0.9871 | 0.9917 |
| | Precision | 0.9501 | 0.9658 | 0.9663 | 0.9344 | 0.9624 | 0.9273 | 0.9383 |
| | Recall | 0.9999 | 1 | 1 | 0.9997 | 1 | 1 | 0.9997 |
| | F1score | 0.9744 | 0.9826 | 0.9829 | 0.9659 | 0.9808 | 0.9623 | 0.9680 |
| | Jaccard | 0.9500 | 0.9658 | 0.9663 | 0.9341 | 0.9624 | 0.9273 | 0.9381 |
| | MAPE | 13.5 | 16.8 | 12.8 | 11.1 | 12.8 | 14.2 | 13.3 |
| | SMAPE | 24.7 | 25.6 | 20.8 | 24.6 | 21.6 | 29.7 | 26.6 |

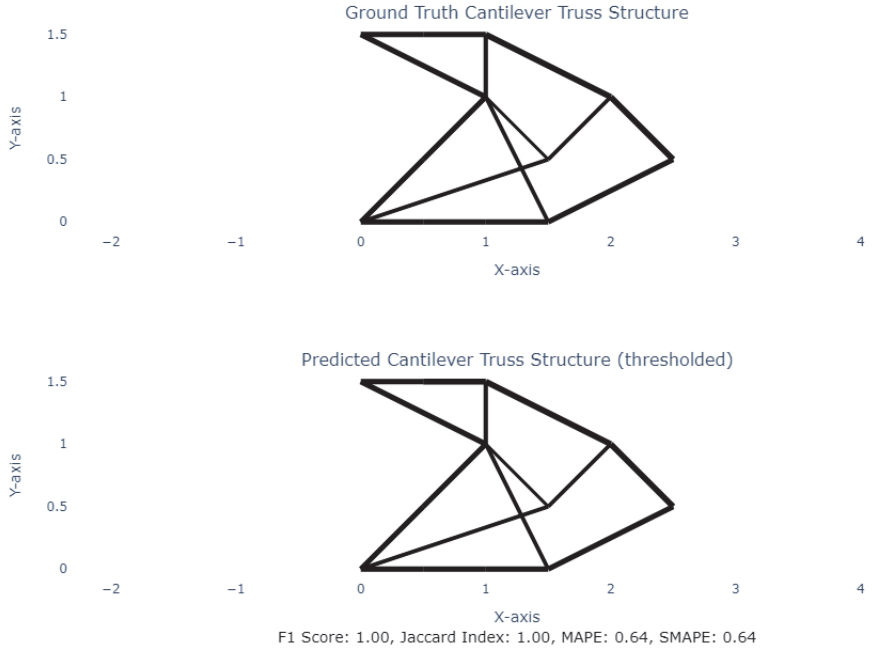
Table 6.5: Mean performance metrics for all testing data and for individual loading conditions for 45 node supported truss structure

| | Metric | Overall | Structure 1 | Structure 2 | Structure 3 | Structure 4 | Structure 5 | Structure 6 |
|---|---|---|---|---|---|---|---|---|
| 12 000 | MSE testing | 92.2 | 124.0 | 97.9 | 76.2 | 92.8 | 87.3 | 77.9 |
| | RMSE testing | 9.6 | 11.1 | 9.9 | 8.7 | 9.6 | 9.3 | 8.8 |
| | NRMSE testing | 0.0019 | 0.0039 | 0.0035 | 0.0020 | 0.0019 | 0.0022 | 0.0018 |
| | R2 value | 0.9985 | 0.9961 | 0.9976 | 0.9989 | 0.9988 | 0.9988 | 0.9990 |
| | Precision | 0.9972 | 0.9991 | 1.0000 | 0.9993 | 0.9922 | 0.9979 | 0.9935 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9986 | 0.9996 | 1.0000 | 0.9997 | 0.9961 | 0.9989 | 0.9968 |
| | Jaccard | 0.9972 | 0.9991 | 1.0000 | 0.9993 | 0.9922 | 0.9979 | 0.9935 |
| | MAPE | 4.3 | 5.8 | 4.6 | 3.9 | 4.1 | 4.0 | 3.6 |
| | SMAPE | 5.1 | 6.3 | 4.9 | 4.2 | 5.9 | 4.6 | 5.1 |
| 6 000 | MSE testing | 259.1 | 345.7 | 171.2 | 201.4 | 273.2 | 305.8 | 253.7 |
| | RMSE testing | 16.1 | 18.6 | 13.1 | 14.2 | 16.5 | 17.5 | 15.9 |
| | NRMSE testing | 0.0032 | 0.0065 | 0.0046 | 0.0033 | 0.0033 | 0.0041 | 0.0032 |
| | R2 value | 0.9957 | 0.9884 | 0.9961 | 0.9972 | 0.9963 | 0.9954 | 0.9968 |
| | Precision | 0.9831 | 0.9648 | 0.9996 | 0.9962 | 0.9702 | 0.9838 | 0.9766 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9915 | 0.9821 | 0.9998 | 0.9981 | 0.9849 | 0.9918 | 0.9882 |
| | Jaccard | 0.9831 | 0.9648 | 0.9996 | 0.9962 | 0.9702 | 0.9838 | 0.9766 |
| | MAPE | 7.8 | 10.4 | 6.4 | 6.8 | 7.6 | 8.4 | 7.1 |
| | SMAPE | 11.6 | 18.1 | 6.9 | 7.9 | 14.0 | 12.4 | 12.1 |
| 3 000 | MSE testing | 791.4 | 1189.8 | 550.2 | 587.4 | 896.8 | 689.0 | 816.3 |
| | RMSE testing | 28.1 | 34.5 | 23.5 | 24.2 | 29.9 | 26.2 | 28.6 |
| | NRMSE testing | 0.0056 | 0.0121 | 0.0082 | 0.0057 | 0.0060 | 0.0061 | 0.0057 |
| | R2 value | 0.9871 | 0.9612 | 0.9871 | 0.9920 | 0.9880 | 0.9901 | 0.9896 |
| | Precision | 0.9360 | 0.8764 | 0.9761 | 0.9674 | 0.9019 | 0.9525 | 0.9371 |
| | Recall | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | F1score | 0.9669 | 0.9341 | 0.9879 | 0.9834 | 0.9484 | 0.9757 | 0.9675 |
| | Jaccard | 0.9360 | 0.8764 | 0.9761 | 0.9674 | 0.9019 | 0.9525 | 0.9371 |
| | MAPE | 14.6 | 21.3 | 12.6 | 12.7 | 14.8 | 13.6 | 13.2 |
| | SMAPE | 28.3 | 46.9 | 18.6 | 19.8 | 35.0 | 24.3 | 26.4 |

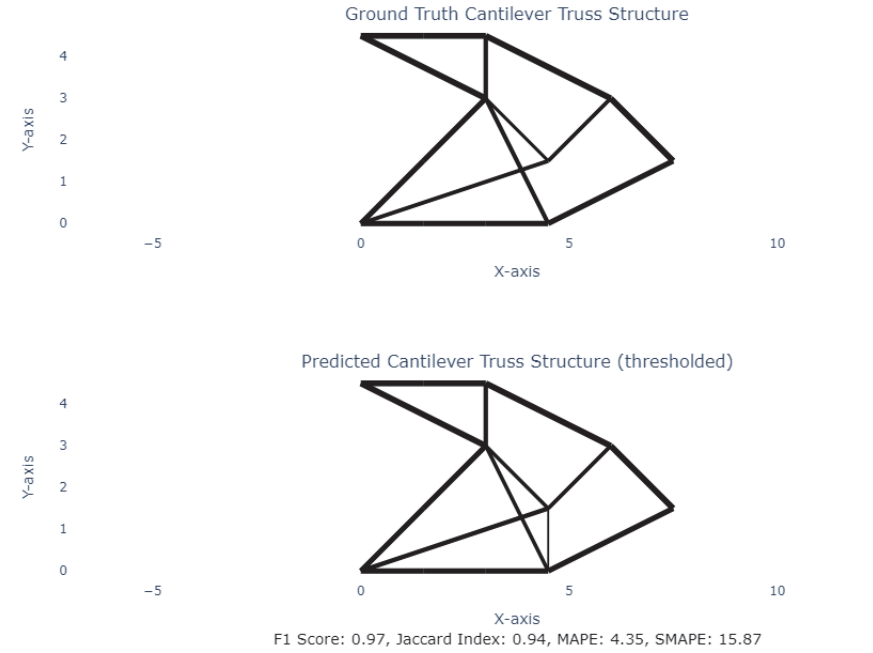
Table 6.6: Mean performance metrics for all testing data and for individual loading conditions for 60 node simply supported truss structure

| | Avg. time taken by the optimization algorithm (ms) | Avg. time taken by the GNN model (ms) | Time saved (%) | Acceleration factor |
|---|---|---|---|---|
| 24 nodes | 39.87 | 0.45 | 98.9 | 88.6 |
| 45 nodes | 43.31 | 1.52 | 96.5 | 28.5 |
| 60 nodes | 49.28 | 2.26 | 95.4 | 21.8 |

Table 6.7: Computational time savings for optimization with the use of GNNs

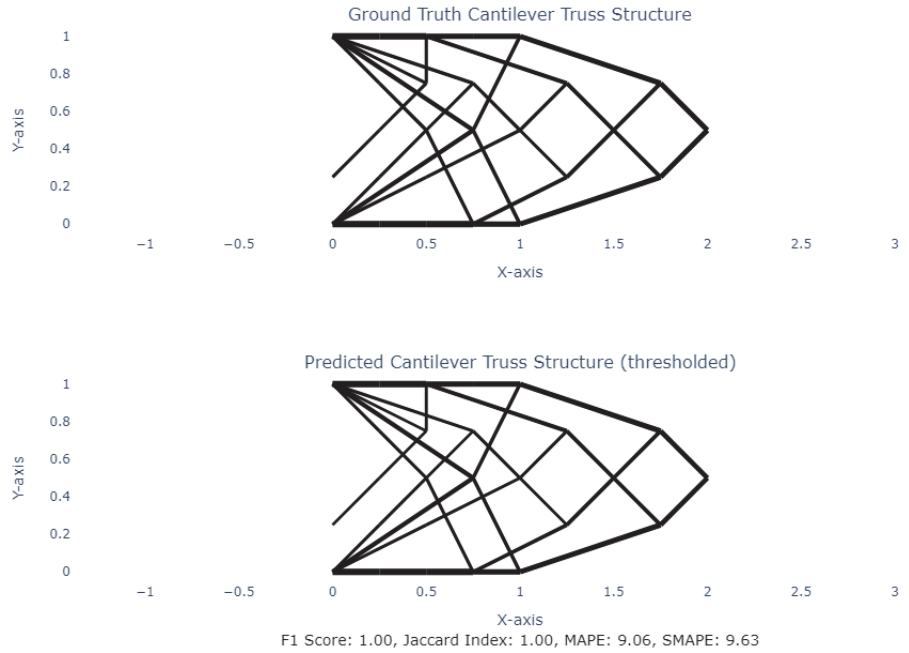



(a)

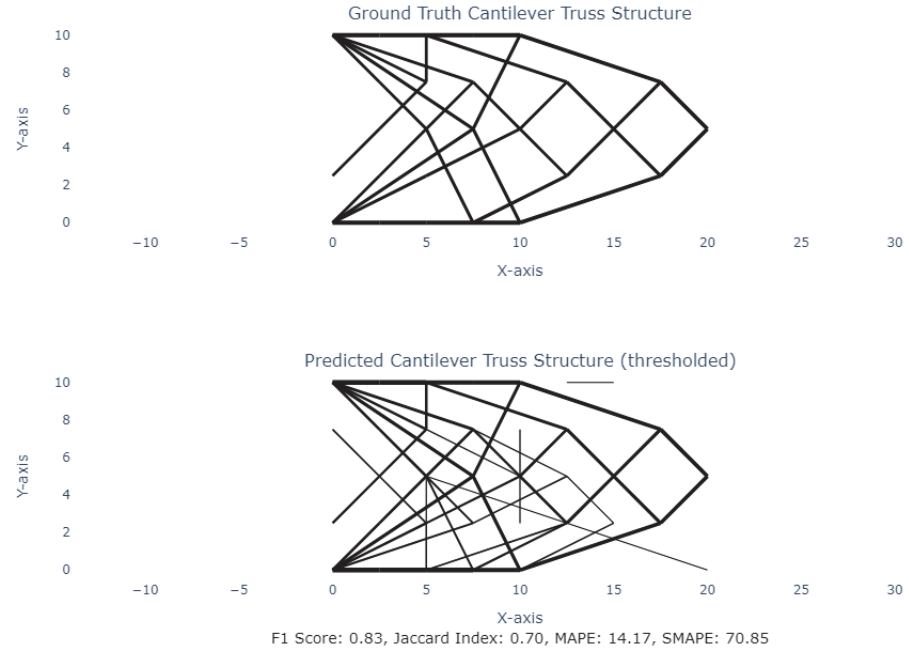



(b)

Figure 6.3: Prediction plots for the load case 1 of the cantilever problem with (a) lowest and (b) highest error in 24 nodes

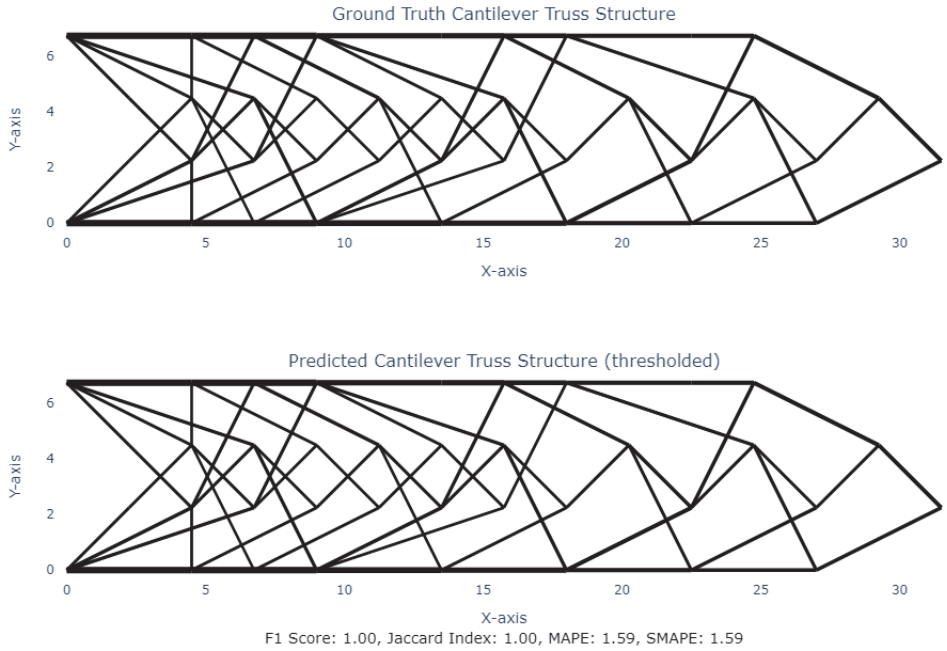



(a)

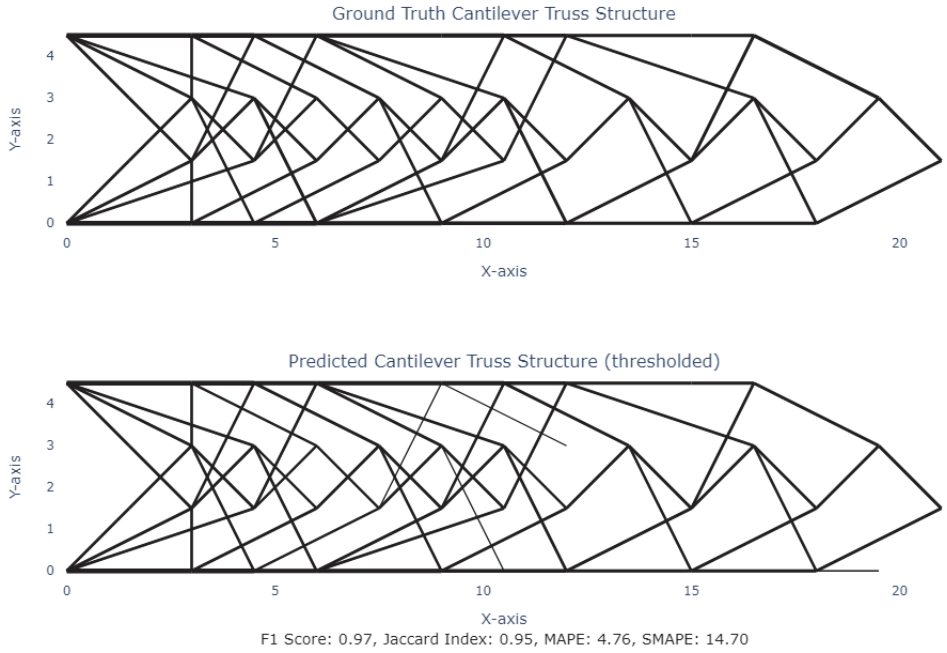



(b)

Figure 6.4: Prediction plots for the load case 1 of the cantilever problem with (a) lowest and (b) highest error in 45 nodes

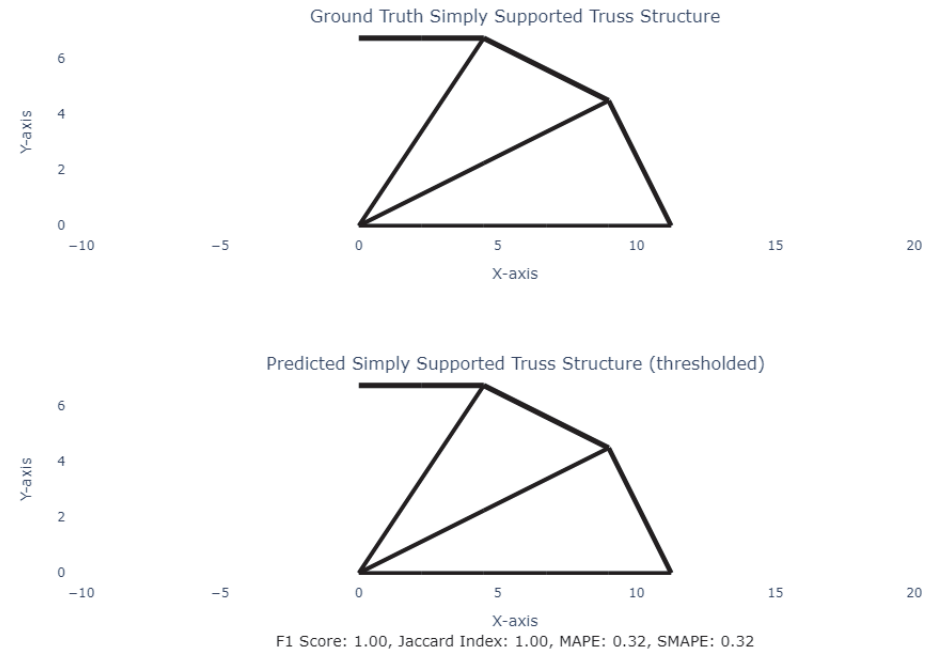



(a)

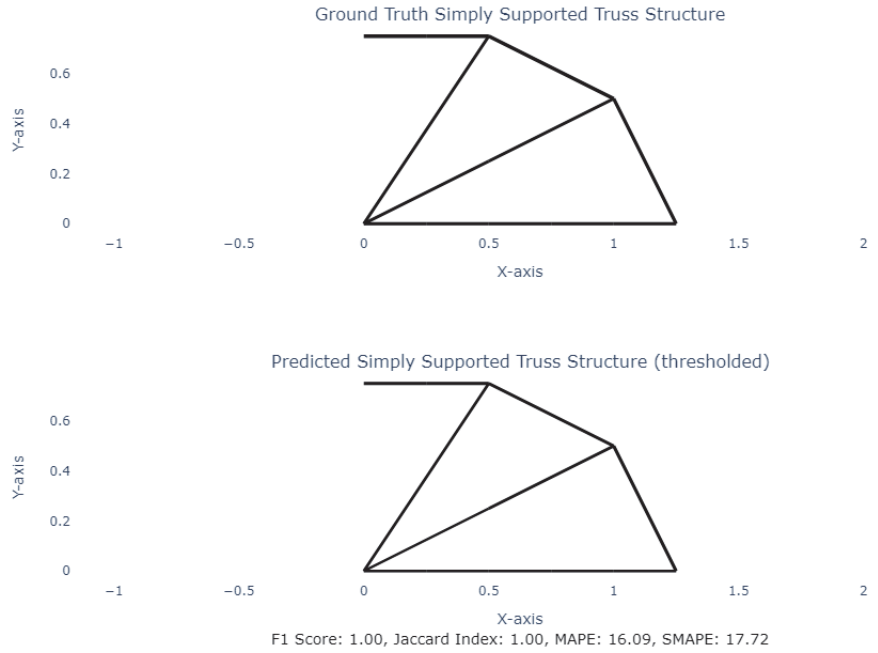



(b)

Figure 6.5: Prediction plots for the load case 1 of the cantilever problem with (a) lowest and (b) highest error in 60 nodes

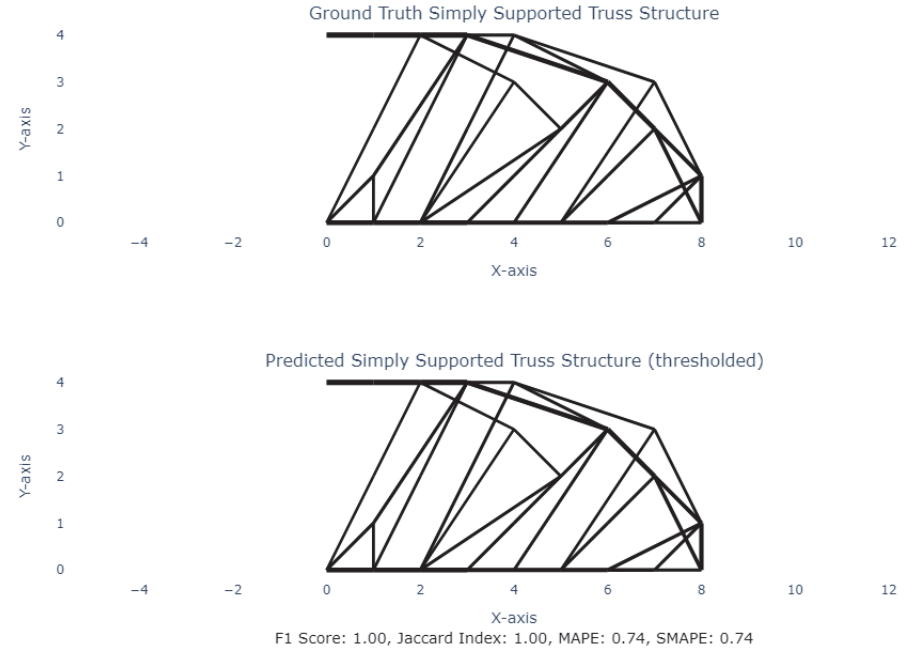



(a)

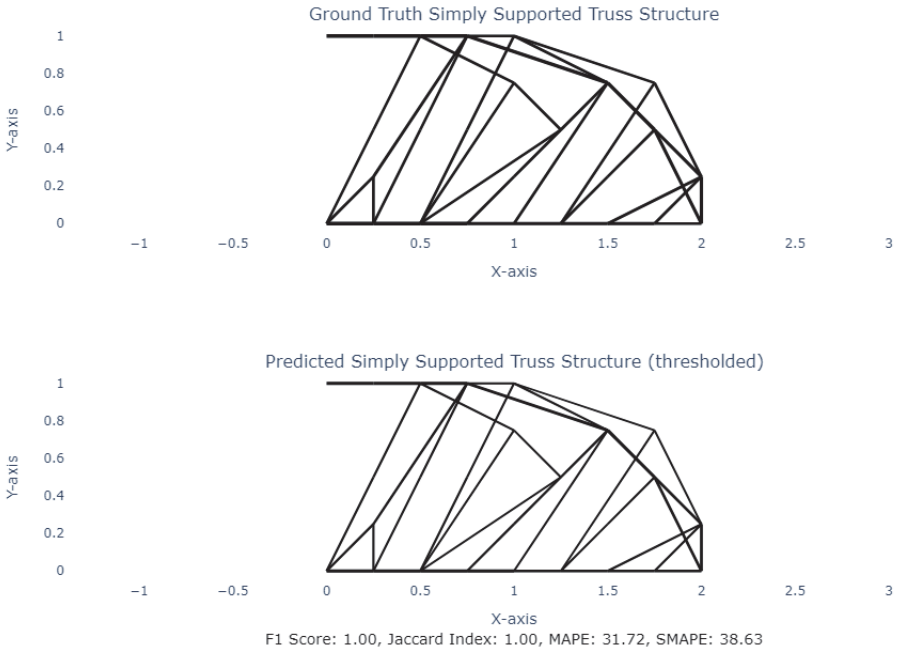



(b)

Figure 6.6: Prediction plots for the load case 1 of the simply supported problem with (a) lowest and (b) highest error in 24 nodes

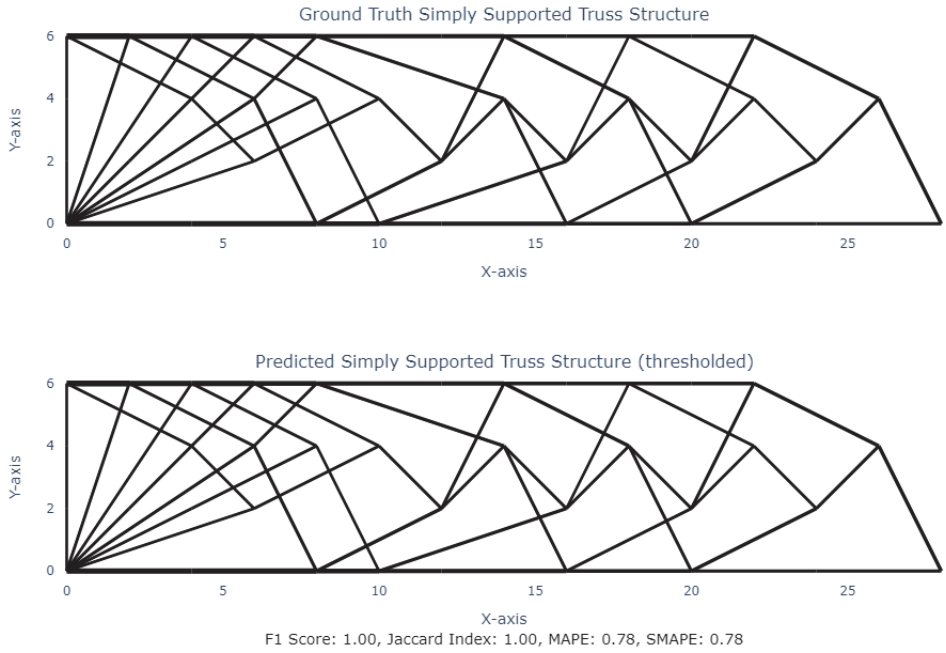



(a)

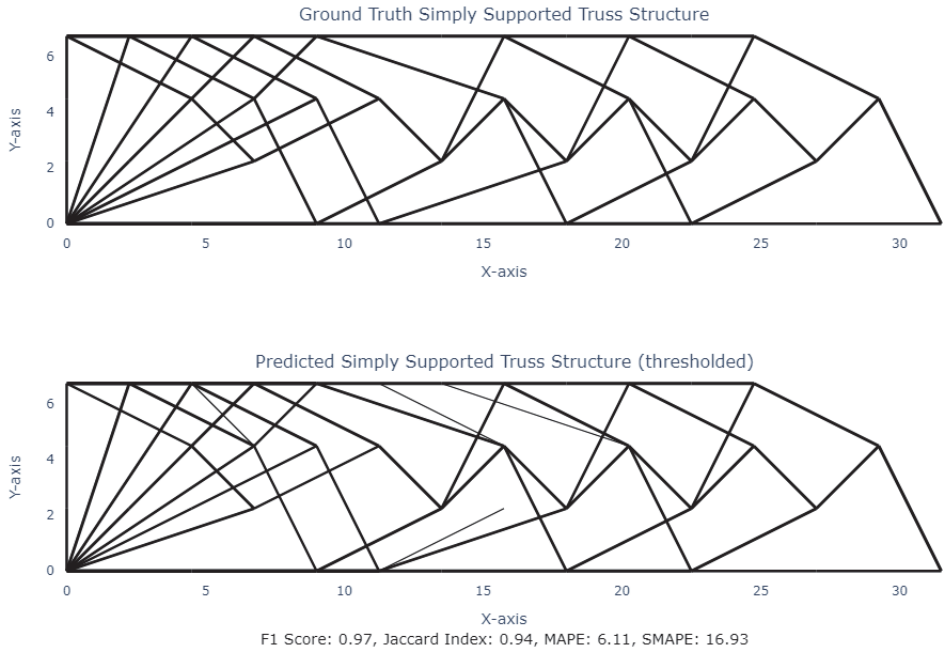



(b)

Figure 6.7: Prediction plots for the load case 1 of the simply supported problem with (a) lowest and (b) highest error in 45 nodes




(a)




(b)

Figure 6.8: Prediction plots for the load case 1 of the simply supported problem with (a) lowest and (b) highest error in 60 nodes

# Chapter 7

# Conclusion and Future Research

## 7.1 Conclusions

In this study, we introduced an efficient surrogate modeling approach for structural optimization by employing a variation of graph convolutions known as Topology Adaptive Graph Convolutions, integrated within a Graph Neural Network framework. This method replaces the traditionally computationally expensive optimization processes with a GNN-based model capable of predicting optimized structural layouts in real time, substantially reducing both computational time and resource requirements. The GNN model was trained on datasets generated by connectivity based ground structure optimization, encompassing both topological and size optimizations.

Through training on multiple load cases of a cantilever structure and a simply supported structure, the model demonstrated strong generalization capabilities, delivering highly accurate predictions. It showed minimal deviations from the ground truth, with MAPE and SMAPE values consistently below 5%, aside from a few exceptions. The high recall, precision, and Jaccard index values highlight the model's effectiveness in predicting structural topology, while the low MAPE and SMAPE values underscore its ability to optimize structure sizes efficiently. These metrics collectively showcase the model's robust performance in accurately approximating optimized structures while significantly reducing computational costs compared to existing optimization frameworks.

Since the outputs are provided in a graph format, with nodes representing joints and edges denoting members along with their corresponding cross-sectional areas, the predicted structures from the proposed framework are seamlessly utilized to construct 2D trusses with straight members. The cross-sectional data and topol-

ogy information generated by the framework provide all the necessary details for fabricators to assemble 2D trusses using members manufactured through formative techniques. Additionally, with the GNN framework achieving a reduction of over 95% in computation time compared to the baseline MATLAB-based optimization, real-time structural optimization becomes feasible. This efficiency enables practitioners to integrate the framework into daily workflows, making it a practical tool for routine structural design and optimization tasks.

## 7.2 Future Work

Building on the success of the GNN-based framework, the next stage involves incorporating additional design considerations, such as accounting for buckling and enhancing the robustness of the structures. Once these considerations are integrated, a simple application compatible with smartphones and computers could be developed. The ease of incorporating a trained machine learning model into such an application represents a significant step forward in making structural optimization accessible to everyday fabricators.

Despite the success of the GNN-based model, several opportunities exist for further improvement. One such enhancement involves integrating post-processing steps that remove mistakenly generated members, which could further improve the structural efficiency and performance of the optimized designs. This post-processing could involve filtering techniques or heuristics that detect and eliminate redundant or impractical connections in the predicted layouts, thereby streamlining the final design.

Due to the nodal number dependency of the input and output feature tensors, the computational demands during the training process significantly increase, limiting the model's scalability to node counts in the range of thousands. Expanding the model to handle higher numbers of nodes becomes challenging unless researchers have access to high-performance computing resources to train these large models. While the ML model effectively reduces computational costs once trained, this bottleneck poses a limitation for future research and the broader implementation of such models in practical applications.

Moreover, while this study focused on a cantilever and simply supported truss structure, the same surrogate modeling framework can be extended to other structural types, including multi-span or frame structures. Additionally, the model can be adapted for use in the 3D domain, which would open up a broader range of applications, from more complex building designs to aerospace structures. Exploring the model's capabilities in 3D optimization could potentially unlock more sophisticated designs.

# Bibliography


[1] W. Dillen, G. Lombaert, R. Mertens, H. Van Beurden, D. Jaspaert, and M. Schevenels, "Optimization in a realistic structural engineering context: Redesign of the market hall in ghent," *Engineering Structures*, vol. 228, p. 111473, 2021.

[2] Y. Wang and O. Sigmund, "Topology optimization of multi-material active structures to reduce energy consumption and carbon footprint," *Structural and Multidisciplinary Optimization*, vol. 67, no. 1, p. 5, 2024.

[3] E. Broniewicz and K. Dec, "Environmental impact of demolishing a steel structure design for disassembly," *Energies*, vol. 15, no. 19, 2022.

[4] W.-F. Chen and L. Duan, *Bridge engineering handbook. Superstructure Design.* Boca Raton: Taylor  Francis, 2nd ed. ed., 2014.

[5] A. Laurain, "A level set-based structural optimization code using fenics," *Structural and Multidisciplinary Optimization*, vol. 58, no. 3, pp. 1311–1334, 2018.

[6] D. Villalba, J. París, I. Couceiro, I. Colominas, and F. Navarrina, "Topology optimization of structures considering minimum weight and stress constraints by using the overweight approach," *Engineering Structures*, vol. 273, p. 115071, 2022.

[7] Z. Kang, J. He, L. Shi, and Z. Miao, "A method using successive iteration of analysis and design for large-scale topology optimization considering eigenfrequencies," *Computer Methods in Applied Mechanics and Engineering*, vol. 362, p. 112847, 2020.

[8] F. B. C. G. Florian Mitjana, Sonia Cafieri and F. Castanie, "Optimization of structures under buckling constraints using frame elements," *Engineering Optimization*, vol. 51, no. 1, pp. 140–159, 2019.

[9] V. Savsani, G. Tejani, and V. Patel, *Truss Optimization: A Metaheuristic Optimization Approach.* Springer Cham, 01 2024.

[10] A. Burkov, *Machine Learning Engineering.* True Positive Incorporated, 2020.

[11] P. Rodríguez-Belenguer, K. Kopańska, J. Llopis-Lorente, B. Trenor, J. Saiz, and M. Pastor, "Application of machine learning to improve the efficiency of electrophysiological simulations used for the prediction of drug-induced ventricular arrhythmia," *Computer Methods and Programs in Biomedicine*, vol. 230, p. 107345, 2023.

[12] P. Stolfi and F. Castiglione, "Emulating complex simulations by machine learning methods," *BMC Bioinformatics*, vol. 22, no. 14, p. 483, 2021.

[13] M. N.-A. Ahmad Shirvani and M. Y. Ha, "Machine learning-accelerated aerodynamic inverse design," *Engineering Applications of Computational Fluid Mechanics*, vol. 17, no. 1, p. 2237611, 2023.

[14] N. Peters, A. Wissink, and J. Ekaterinaris, "Machine learning-based surrogate modeling approaches for fixed-wing store separation," *Aerospace Science and Technology*, vol. 133, p. 108150, 2023.

[15] Z. B. Bouallègue, M. C. A. Clare, L. Magnusson, E. Gascón, M. Maier-Gerber, M. Janoušek, M. Rodwell, F. Pinault, J. S. Dramsch, S. T. K. Lang, B. Raoult, F. Rabier, M. Chevallier, I. Sandu, P. Dueben, M. Chantry, and F. Pappenberger, "The rise of data-driven weather forecasting: A first statistical assessment of machine learning–based weather forecasts in an operational-like context," *Bulletin of the American Meteorological Society*, vol. 105, no. 6, pp. E864 – E883, 2024.

[16] M. Chantry, S. Hatfield, P. Dueben, I. Polichtchouk, and T. Palmer, "Machine learning emulation of gravity wave drag in numerical weather forecasting," *Journal of Advances in Modeling Earth Systems*, vol. 13, no. 7, p. e2021MS002477, 2021. e2021MS002477 2021MS002477.

[17] N. Ariyasinghe and S. Herath, "Machine learning techniques for predictive modelling and uncertainty quantification of the mechanical properties of woven carbon fibre composites," *Materials Today Communications*, vol. 40, p. 109732, 2024.

[18] A. Gupta, A. Bhaduri, and L. Graham-Brady, "Accelerated multiscale mechanics modeling in a deep learning framework," *Mechanics of Materials*, vol. 184, p. 104709, 2023.

[19] L. S. Sarma, C. Mallikarachchi, and S. Herath, "Design-informed generative modelling of skeletal structures using structural optimization," *Computers Structures*, vol. 302, p. 107474, 2024.

[20] S. Riehl and P. Steinmann, "On structural shape optimization using an em-

bedding domain discretization technique," *International Journal for Numerical Methods in Engineering*, vol. 109, no. 9, pp. 1315–1343, 2017.

[21] Z. Kang and Y. Wang, "A nodal variable method of structural topology optimization based on shepard interpolant," *International Journal for Numerical Methods in Engineering*, vol. 90, no. 3, pp. 329–342, 2012.

[22] T. Zegard and G. H. Paulino, "Grand — ground structure based topology optimization for arbitrary 2d domains using matlab," *Structural and Multidisciplinary Optimization*, vol. 50, pp. 861–882, November 2014.

[23] X. Zhang, S. Maheshwari, A. S. Ramos, and G. H. Paulino, "Macroelement and macropatch approaches to structural topology optimization using the ground structure method," *Journal of Structural Engineering*, vol. 142, no. 11, p. 04016090, 2016.

[24] Y.-b. L. Ge Gao, Zhen-yu Liu and Y. feng Qiao, "A new method to generate the ground structure in truss topology optimization," *Engineering Optimization*, vol. 49, no. 2, pp. 235–251, 2017.

[25] M. S. Islam, A. Z. Nayem, and K. N. Hoque, "Finite element-based optimization procedure for an irregular domain with unstructured mesh," *Heliyon*, vol. 10, p. e25994, February 2024. doi: 10.1016/j.heliyon.2024.e25994.

[26] Gebremedhen, Hailu Shimels, Woldemicahel, Dereje Engida, and Hashim, Fakheruldin M., "Three-dimensional stress-based topology optimization using simp method," *Int. J. Simul. Multidisci. Des. Optim.*, vol. 10, p. A1, 2019.

[27] G.-C. Luh, C.-Y. Lin, and Y.-S. Lin, "A binary particle swarm optimization for continuum structural topology optimization," *Applied Soft Computing*, vol. 11, no. 2, pp. 2833–2844, 2011. The Impact of Soft Computing for the Progress of Artificial Intelligence.

[28] N. Cui, S. Huang, and X. Ding, "An improved strategy for genetic evolutionary structural optimization," *Advances in Civil Engineering*, vol. 2020, no. 1, p. 5924198, 2020.

[29] T. Xu, X. Lin, and Y. M. Xie, "Bi-directional evolutionary structural optimization with buckling constraints," *Structural and Multidisciplinary Optimization*, vol. 66, p. 67, March 2023.

[30] Z. C. Hongyou Cao, Xudong Qian and H. Zhu, "Enhanced particle swarm optimization for size and shape optimization of truss structures," *Engineering Optimization*, vol. 49, no. 11, pp. 1939–1956, 2017.

[31] S. G. S. M. Seyedpoor and S. R. Talebian, "An efficient structural optimisation algorithm using a hybrid version of particle swarm optimisation with simultaneous perturbation stochastic approximation," *Civil Engineering and Environmental Systems*, vol. 27, no. 4, pp. 295–313, 2010.

[32] S.-y. Chen, X.-f. Shui, D.-f. Li, and H. Huang, "Improved genetic algorithm with two-level approximation for truss optimization by using discrete shape variables," *Mathematical Problems in Engineering*, vol. 2015, no. 1, p. 521482, 2015.

[33] I. Delyová, P. Frankovský, J. Bocko, P. Trebuňa, J. Živčák, B. Schürger, and S. Janigová, "Sizing and topology optimization of trusses using genetic algorithm," *Materials*, vol. 14, no. 4, 2021.

[34] M. Fenton, C. McNally, J. Byrne, E. Hemberg, J. McDermott, and M. O'Neill, "Discrete planar truss optimization by node position variation using grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 577–589, 2016.

[35] V. F. Jean-Christophe Cuillière and A. Nana, "Automatic construction of structural cad models from 3d topology optimization," *Computer-Aided Design and Applications*, vol. 15, no. 1, pp. 107–121, 2018.

[36] N. Changizi and M. Jalalpour, "Stress-based topology optimization of steel-frame structures using members with standard cross sections: Gradient-based approach," *Journal of Structural Engineering*, vol. 143, no. 8, p. 04017078, 2017.

[37] J. Fu and H. Huang, "A structural discrete size and topology optimization method with extended approximation concepts," *Structural and Multidisciplinary Optimization*, vol. 65, no. 4, p. 116, 2022.

[38] W.-F. Du, Y.-Q. Wang, H. Wang, and Y.-N. Zhao, "Intelligent generation method for innovative structures of the main truss in a steel bridge," *Soft Computing*, vol. 27, no. 9, pp. 5587–5601, 2023.

[39] J. Seo and R. K. Kapania, "Development of deep convolutional neural network for structural topology optimization," *AIAA Journal*, vol. 61, no. 3, pp. 1366–1379, 2023.

[40] S. Zheng, L. Qiu, and F. Lan, "Tso-gcn: A graph convolutional network approach for real-time and generalizable truss structural optimization," *Applied Soft Computing*, vol. 134, p. 110015, 2023.

[41] Y. Wang, F. Shi, and B. Chen, "Real-time structure generation based on data-driven using machine learning," *Processes*, vol. 11, no. 3, 2023.

[42] M. Seo and S. Min, "Graph neural networks and implicit neural representation for near-optimal topology prediction over irregular design domains," *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106284, 2023.

[43] H. Jeong, J. Bai, C. Batuwatta-Gamage, C. Rathnayaka, Y. Zhou, and Y. Gu, "A physics-informed neural network-based topology optimization (pinnto) framework for structural optimization," *Engineering Structures*, vol. 278, p. 115484, 2023.

[44] H. T. Mai, J. Kang, and J. Lee, "A machine learning-based surrogate model for optimization of truss structures with geometrically nonlinear behavior," *Finite Elements in Analysis and Design*, vol. 196, p. 103572, 2021.

[45] H.-A. Pham, V.-H. Dang, T.-C. Vu, and B.-D. Nguyen, "An efficient k-nn-based rao optimization method for optimal discrete sizing of truss structures," *Applied Soft Computing*, vol. 154, p. 111373, 2024.

[46] T.-H. Nguyen and A.-T. Vu, "An efficient differential evolution for truss sizing optimization using adaboost classifier," *CMES - Computer Modeling in Engineering and Sciences*, vol. 134, no. 1, pp. 429–458, 2022.

[47] N. Nourian, M. El-Badry, and M. Jamshidi, "Design optimization of truss structures using a graph neural network-based surrogate model," *Algorithms*, vol. 16, no. 8, 2023.

[48] H. Ma, Y. Bian, Y. Rong, W. Huang, T. Xu, W. Xie, G. Ye, and J. Huang, "Cross-dependent graph neural networks for molecular property prediction," *Bioinformatics*, vol. 38, pp. 2003–2009, 01 2022.

[49] H. T. Mai, S. Lee, D. Kim, J. Lee, J. Kang, and J. Lee, "Optimum design of nonlinear structures via deep neural network-based parameterization framework," *European Journal of Mechanics - A/Solids*, vol. 98, p. 104869, 2023.

[50] R. Sedaghati, "Benchmark case studies in structural design optimization using the force method," *International Journal of Solids and Structures*, vol. 42, no. 21, pp. 5848–5871, 2005. PACAM VIII SPECIAL ISSUE.

[51] S. Zhu, M. Ohsaki, K. Hayashi, and X. Guo, "Machine-specified ground structures for topology optimization of binary trusses using graph embedding policy network," *Advances in Engineering Software*, vol. 159, p. 103032, 2021.

[52] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[53] L. Wu, P. Cui, J. Pei, L. Zhao, and L. Song, *Graph Neural Networks*, pp. 27–37. Singapore: Springer Nature Singapore, 2022.

[54] S. Min, Z. Gao, J. Peng, L. Wang, K. Qin, and B. Fang, "Stgsn — a spatial–temporal graph neural network framework for time-evolving social networks," *Knowledge-Based Systems*, vol. 214, p. 106746, 2021.

[55] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, WWW '19, (New York, NY, USA), p. 417–426, Association for Computing Machinery, 2019.

[56] K. X.-W. L. Na Hu, Dafang Zhang and M.-Y. Hsieh, "Graph learning-based spatial-temporal graph convolutional neural networks for traffic forecasting," *Connection Science*, vol. 34, no. 1, pp. 429–448, 2022.

[57] K. Guo, Y. Hu, Z. Qian, H. Liu, K. Zhang, Y. Sun, J. Gao, and B. Yin, "Optimized graph convolution recurrent neural network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 1138–1149, 2021.

[58] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer, and P. Friederich, "Graph neural networks for materials science and chemistry," *Communications Materials*, vol. 3, no. 1, p. 93, 2022.

[59] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[60] Y. Wu, H.-N. Dai, and H. Tang, "Graph neural networks for anomaly detection in industrial internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9214–9231, 2022.

[61] A. K. Awasthi, A. K. Garov, M. Sharma, and M. Sinha, "Gnn model based on node classification forecasting in social network," in *2023 International Conference on Artificial Intelligence and Smart Communication (AISC)*, pp. 1039–1043, 2023.

[62] X. Tang, Y. Liu, N. Shah, X. Shi, P. Mitra, and S. Wang, "Knowing your fate: Friendship, action and temporal explanations for user engagement prediction on social apps," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, (New York, NY, USA), p. 2269–2279, Association for Computing Machinery, 2020.

[63] I. Hoeronis and B. R. Trilaksono, "Node classification on the citation network

using graph neural network," *Inspiration: Jurnal Teknologi Informasi dan Komunikasi*, vol. 13, pp. 96–105, Jun. 2023.

[64] M. Yang and Z. Yang, "Dynamic heterogeneous graph learning: An adaptive research academic network," *IEEE Access*, vol. 12, pp. 74751–74761, 2024.

[65] M. Balakrishnan and G. T. V., "A neural network framework for predicting dynamic variations in heterogeneous social networks," *PLOS ONE*, vol. 15, pp. 1–21, 04 2020.

[66] M. Dossena, C. Irwin, and L. Portinale, "Graph-based recommendation using graph neural networks," *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1769–1774, 2022.

[67] G. Lv, Z. Hu, Y. Bi, and S. Zhang, "Learning unknown from correlations: Graph neural network for inter-novel-protein interaction prediction," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21* (Z.-H. Zhou, ed.), pp. 3677–3683, International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[68] S. Jiang and P. Balaprakash, "Graph neural network architecture search for molecular property prediction," *2020 IEEE International Conference on Big Data (Big Data)*, pp. 1346–1353, 2020.

[69] J. G. Rittig, Q. Gao, M. Dahmen, A. Mitsos, and A. M. Schweidtmann, *Graph Neural Networks for the Prediction of Molecular Structure–Property Relationships*, p. 159–181. Royal Society of Chemistry, Dec. 2023.

[70] K. Martinkus, P. A. Papp, B. Schesch, and R. Wattenhofer, "Agent-based graph neural networks," 2023.

[71] B. Muhammet, R. Guillaume, H. Pierre, G. Benoit, A. Sébastien, and P. Honeine, "When Spectral Domain Meets Spatial Domain in Graph Neural Networks," in *Thirty-seventh International Conference on Machine Learning (ICML 2020) - Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, (Vienna, Austria), July 2020.

[72] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs: Graph fourier transform," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6167–6170, 2013.

[73] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," 2017.

[74] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," 2017.

[75] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[76] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[77] J. Du, S. Zhang, G. Wu, J. M. F. Moura, and S. Kar, "Topology adaptive graph convolutional networks," 2018.

[78] J. Shi, M. Cheung, J. Du, and J. M. Moura, "Classification with vertex-based graph convolutional neural networks," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pp. 752–756, 2018.

[79] Y. Xiang, K. Fujimoto, J. Schneider, Y. Jia, D. Zhi, and C. Tao, "Network context matters: graph convolutional network model over social networks improves the detection of unknown HIV infections among young men who have sex with men," *Journal of the American Medical Informatics Association*, vol. 26, pp. 1263–1271, 06 2019.

[80] X. W.-D. L. J. W. Zhekun Huang, Haizhong Qian and L. Xie, "Graph neural network-based identification of ditch matching patterns across multi-scale geospatial data," *Geocarto International*, vol. 38, no. 1, p. 2294900, 2023.

[81] Y. Zou and J. Shu, "Heterogeneous network node classification based on graph neural networks," in *2023 3rd International Conference on Intelligent Communications and Computing (ICC)*, pp. 367–371, 2023.

[82] H. Wang, W. Li, X. Jin, K. Cho, H. Ji, J. Han, and M. D. Burke, "Chemical-reaction-aware molecule representation learning," 2021.

[83] S. Singh, A. Sharma, and V. K. Chauhan, "Gtagcn: Generalized topology adaptive graph convolutional networks," 2024.

[84] A. Alchihabi and Y. Guo, "Learning robust graph neural networks with limited supervision," in *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics* (F. Ruiz, J. Dy, and J.-W. van de Meent, eds.), vol. 206 of *Proceedings of Machine Learning Research*, pp. 8723–8733, PMLR, 25–27 Apr 2023.

[85] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2019.

[86] R. Helin, U. Indahl, O. Tomic, and K. H. Liland, "Non-linear shrinking of linear model errors," *Analytica Chimica Acta*, vol. 1258, p. 341147, 2023.

[87] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[88] D. Kirk, "Nvidia cuda software and gpu parallel computing architecture," in *Proceedings of the 6th International Symposium on Memory Management*, ISMM '07, (New York, NY, USA), p. 103–104, Association for Computing Machinery, 2007.

[89] A. Khakhar and J. Buckman, "Neural regression for scale-varying targets," 2023.

[90] X. Piao, D. Synn, J. Park, and J.-K. Kim, "Enabling large batch size training for dnn models beyond the memory limit while maintaining performance," *IEEE Access*, vol. PP, pp. 1–1, 01 2023.

[91] P. Charilaou and R. Battat, "Machine learning models and over-fitting considerations," *World Journal of Gastroenterology*, vol. 28, no. 5, pp. 605–607, 2022. ©The Author(s) 2022. Published by Baishideng Publishing Group Inc. All rights reserved.