LAWSON OLIVEIRA LIMA

A HYBRID GNN-HEURISTIC APPROACH FOR UNSUPERVISED PARTITIONING OF FEATURELESS GRAPHS

Undergraduate Thesis submitted to the Electrical Engineering Course of the Center of Technology of the Federal University of Ceará, as a partial requirement for obtaining the Bachelor Degree in Electrical Engineering.

Advisor: Prof. Dr. Arthur Plínio de Souza Braga

Co-advisor: Prof. Dr. Amauri Holanda de Souza Júnior

FORTALEZA

2025

LAWSON OLIVEIRA LIMA

A HYBRID GNN-HEURISTIC APPROACH FOR UNSUPERVISED PARTITIONING OF FEATURELESS GRAPHS

Undergraduate Thesis submitted to the Electrical Engineering Course of the Center of Technology of the Federal University of Ceará, as a partial requirement for obtaining the Bachelor Degree in Electrical Engineering.

Approved on: 30th July 2025

EXAMINATION BOARD

Prof. Dr. Arthur Plínio de Souza Braga (Advisor)
Federal University of Ceará (UFC)

Prof. Dr. Amauri Holanda de Souza Júnior (Co-advisor)
Federal Institute of Science and Technology (IFCE)

Prof. Dr. Diego Parente Paiva Mesquita
Getulio Vargas Foundation (FGV)

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

L698h Lima, Lawson Oliveira.
A Hybrid GNN-Heuristic approach for unsupervised partitioning of featureless graphs / Lawson Oliveira Lima. – 2025.
111 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2025.
Orientação: Prof. Dr. Arthur Plínio de Souza Braga.
Coorientação: Prof. Dr. Amauri Holanda de Souza Júnior.

1. Graph partitioning. 2. Graph neural networks. 3. Unsupervised learning. 4. Finite element method. 5. Spectral methods. I. Título.

CDD 621.3

Dedico este trabalho à minha família, que me ensinou o valor da perseverança. Aos meus amigos, que com leveza e alegria, tornaram cada etapa desta jornada mais significativa. E aos meus professores, cujo conhecimento e orientação foram a minha bússola.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to everyone who, in some way, contributed to the completion of this work and to my academic and personal development.

A special thanks to Professor Frédéric Magoulès, whose vision and initial proposal during my double degree at CentraleSupélec planted the seed that gave rise to this research. His inspiration and trust were fundamental to the development of this project.

I am particularly grateful to Professor Amauri Holanda de Souza Junior for his patient guidance, steady direction, all the support he provided throughout the different stages of this work, and for the productive and enlightening conversations about Graph Neural Networks and Positional Encoders. His insights greatly enriched both the theoretical and practical foundations of this thesis. His contributions were essential to my growth as a researcher. My sincere thanks also go to Professor Arthur Plínio de Souza Braga and Professor Guilherme de Alencar Barreto for the support.

I extend my thanks to friends and colleagues, whose intellectual partnership was a constant source of motivation. A special and indispensable thanks to Vitor Domingues do Amaral, whose collaboration was vital in developing and implementing the partitioning heuristics. His technical discussions and thoughtful questions were crucial for validating the comparison methods used in this thesis. I also thank José Lucas de Melo Costa and Lucas Vitoriano de Queiroz Lira for the countless conversations about artificial intelligence and related fields, which always broadened my perspective and kept the flame of curiosity alive.

To all professors and friends who have been part of my journey, from the early days of my undergraduate studies to this moment, thank you. Each of you has left an important mark on my education, not only through the knowledge you shared but also through your example of dedication and passion for teaching.

Finally, my deepest and most heartfelt thanks go to my family. To my parents, Lauro Santiago de Lima and Maria Celi de Oliveira, for all their love, sacrifice, encouragement, and for being my safe harbor in every moment. To my brother, for his constant support and the friendship that has always strengthened me. Without you, none of this would have been possible. This achievement is, above all, ours.

“A man is like a fraction whose numerator is what he is and whose denominator is what he thinks of himself.”

(Liev Tolstoy)

ABSTRACT

Graph partitioning is a fundamental problem in parallel scientific computing, particularly important for efficiently distributing Finite Element Method (FEM) meshes across multiple processors or GPUs. In this context, the dual graph representation, where nodes correspond to mesh elements and edges capture topological adjacency, is widely adopted to minimize inter-partition communication. This work proposes an unsupervised learning framework based on Graph Neural Networks (GNNs) for partitioning dual graphs derived from FEM meshes. Our method integrates spectral graph theory with deep learning by employing Laplacian Eigenmaps as positional encoders. Specifically, we use the Fiedler vector (the eigenvector associated with the second smallest Laplacian eigenvalue) to provide global structural information, mitigating the limitations of purely local encoders and the issues inherent to sign-invariant architectures such as SignNet. The proposed model is built upon the Graph Isomorphism Network (GIN) architecture, chosen for its expressive power in distinguishing graph structures. It is trained end-to-end using a differentiable relaxation of the Min-Max Cut (MMCut) objective, which encourages balanced and well-separated partitions. To further enhance partition quality, we introduce a hybrid variant (GNN-FM) that refines GNN predictions using the linear-time Fiduccia-Mattheyses algorithm as a post-processing step. Experiments were conducted on a large-scale industrial dataset (Fusion 360), comprising thousands of triangular meshes, as well as on a separate set of unseen evaluation meshes provided by an industrial partner. Our approach was benchmarked against nine classical and modern algorithms, including METIS, Spectral Clustering, and recursive bisection methods. The GNN-FM model achieved highly competitive results, outperforming METIS in 4 out of 6 test meshes by reducing the number of cut edges, while maintaining a perfect balance ratio of 1,0. These results demonstrate that combining spectral encodings with unsupervised GNNs and heuristic refinement yields a powerful and scalable solution for graph partitioning in scientific computing scenarios.

Keywords: Graph Partitioning; Graph Neural Networks; Unsupervised Learning; Finite Element Method; Spectral Methods; Domain Decomposition.

RESUMO

O particionamento de grafos é um problema fundamental na computação científica paralela, particularmente importante para a distribuição eficiente de malhas do Método de Elementos Finitos (FEM) em vários processadores ou GPUs. Nesse contexto, a representação dual, em que os nós correspondem a elementos de malha e as arestas capturam a adjacência topológica, é amplamente adotada para minimizar a comunicação entre partições. Este trabalho propõe uma estrutura de aprendizagem não supervisionada baseada em redes neurais de grafos (GNNs) para particionar grafos derivados de malhas FEM. Nosso método integra a teoria espectral com a aprendizagem profunda, usando os autovetores do laplaciano como codificadores posicionais. Especificamente, usamos o vetor Fiedler (o vetor próprio associado ao segundo menor valor próprio do Laplaciano) para fornecer informações estruturais globais, atenuando as limitações dos codificadores puramente locais e os problemas inerentes às arquiteturas invariantes de sinal, como a SignNet. O modelo proposto foi desenvolvido com base na arquitetura Graph Isomorphism Network (GIN), escolhida por sua capacidade expressiva de distinguir estruturas de grafos. Ele é treinado usando um relaxamento diferenciável da função objetivo de corte mínimo-máximo (MMCut), que incentiva partições equilibradas e bem separadas. Para melhorar ainda mais a qualidade da partição, apresentamos uma variante híbrida (GNN-FM) que refina as previsões da GNN usando o algoritmo Fiduccia-Mattheyses de tempo linear como uma etapa de pós-processamento. Os experimentos foram realizados em um conjunto de dados de grande escala (Fusion 360), composto por milhares de malhas triangulares, bem como em um conjunto separado de malhas fornecidas por um parceiro industrial. Nossa abordagem foi comparada a nove algoritmos clássicos e modernos, incluindo METIS, Spectral Clustering e métodos de bissecção. O modelo GNN-FM obteve bons resultados, superando o METIS em 4 das 6 malhas de teste, reduzindo o número de arestas cortadas e mantendo um perfeito balanceamento de 1,00. Esses resultados demonstram que a combinação de codificações espectrais com GNNs não supervisionadas e refinamento heurístico produz uma solução avançada e dimensionável para o particionamento de grafos em cenários de computação científica.

Palavras-chave: Particionamento de Grafos; Redes Neurais em Grafos; Aprendizado Não Supervisionado; Método dos Elementos Finitos; Métodos Espectrais; Decomposição de Domínio.

LIST OF FIGURES

Figure 1 – Triangular mesh partitioned into two balanced subdomains. In this dual graph view, each element represents a triangle. The partitioning minimizes boundary edges, which are critical in distributed computing as they represent communication between processing units. 24

Figure 2 – Multilevel partitioning strategy. From left to right: (1) Coarsening of the original graph; (2) Initial partitioning of the coarse graph; (3) Projection and refinement of the partition on the original graph. 29

Figure 3 – Visual comparison between the primal and dual graph representations. On the left, the primal graph connects mesh vertices (red) that share an edge. On the right, the dual graph connects element centroids (blue) that share a common edge. 32

Figure 4 – Computational diagram of a single neuron in a feedforward neural network. The neuron receives inputs from the previous layer $h_i^{(l-1)}$, multiplies each by its corresponding weight $W_{ji}^{(l)}$, adds a bias term $b_j^{(l)}$, and applies a nonlinear activation function $\sigma(\cdot)$ to produce the output $h_j^{(l)}$. This operation enables the network to learn complex, non-linear mappings from input to output. . . 41

Figure 5 – Common activation functions used in neural networks. Each function maps input x to the output $f(x)$, introducing nonlinearity into the model. Sigmoid and Tanh are bounded and smooth but prone to vanishing gradients. ReLU is piecewise linear and computationally efficient. GELU and SiLU provide smooth transitions and are favored in modern architectures like Transformers.. 43

Figure 6 – Visualization of the gradient descent optimization process. The red arrows show successive updates of the parameter θ as it follows the negative gradient direction to minimize the loss function $\mathscr{L}(\theta) = \theta^2$. 44

Figure 7 – Bias–Variance Trade-off illustrated through polynomial regression. Degree 1 underfits the data (high bias), degree 15 overfits the noise (high variance), while degree 4 achieves a good trade-off, capturing the underlying pattern with better generalization. 49

Figure 8 – Multi-Layer Perceptron (MLP) with input x_i, hidden activations h_j, and outputs y_k. Each neuron is fully connected to the previous layer. 52

Figure 9 – CNN pipeline with flattening: input is convolved into a feature map, pooled, flattened into a vector, and passed to a fully connected layer. 53
Figure 10 – Unrolled Recurrent Neural Network (RNN) across time steps. The hidden state h_t is updated recurrently based on the previous state h_{t-1} and current input x_t, and optionally used to produce an output y_t. 54
Figure 11 – Transformer Encoder Block with residual (dashed) connections added before each Add & Norm step. These help stabilize training and preserve input information. 56
Figure 12 – Message passing in a GNN: Neighbor features $h_u^{(l-1)}$ are aggregated to form $a_v^{(l)}$, then combined with $h_v^{(l-1)}$ to produce $h_v^{(l)}$. 59
Figure 13 – Cosine similarity between node embeddings produced by each positional encoder. Higher similarity for nearby nodes and more separation for distant ones suggest that the encoder captures distance-aware structure. 67
Figure 14 – Cosine similarity between structural embeddings extracted by RW-based encoders under different topologies. Diagonal encodings emphasize node identity, while sum-based encodings capture global accessibility. 68
Figure 15 – Examples of triangular meshes from the Fusion 360 dataset used in our experiments. 81
Figure 16 – Evaluation meshes provided by Transvalor S.A., used to test the model on unseen data. 82
Figure 17 – Overview of the proposed architecture. 83
Figure 18 – Evolution of training and validation losses. 92
Figure 19 – Visual comparison of partitioning results across algorithms and test meshes. Each row corresponds to a partitioning method, in the following top-down order: *GNN*, *GNN-FM*, *METIS*, *Spectral K-Means*, *Spectral Mean*, *Spectral Median*, *Kernighan-Lin*, *Fiduccia-Mattheyses*, *Hamiltonian Cycle*, *Coordinate* and *Inertial*. Each column represents a different Blocker mesh, indexed as Blocker_1 to Blocker_6. 96

LIST OF TABLES

Table 1 – Comparison of selected GNN architectures. 63
Table 2 – Asymptotic space–time cost per forward pass. 64
Table 3 – Statistical summary of the processed Fusion 360 dataset (primal vs dual graphs). 81
Table 4 – Structural statistics of the six evaluation meshes from Transvalor S.A. 82
Table 5 – Comparison of cut and balance (cut/bal) for each algorithm across all blocker meshes. Best algorithm per column is in bold. 93
Table 6 – Summary of model architecture and training setup. 113

LIST OF ALGORITHMS

1 Spectral Clustering K-Means 74

2 Spectral Bisection Mean/Median 74

3 Kernighan–Lin 75

4 Fiduccia–Mattheyses 76

5 Space Filling Hamiltonian Cycle 78

6 Coordinate 79

7 Inertial 80

8 Spectral GNN Partitioning 83

LIST OF ABBREVIATIONS AND ACRONYMS

Adam	Adaptive Moment Estimation
BasisNet	Basis-Invariant Neural Network
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
FEM	Finite Element Method
FFN	Feed-Forward Network
FM	Fiduccia–Mattheyses
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GELU	Gaussian Error Linear Unit
GIN	Graph Isomorphism Network
GNN	Graph Neural Network
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
GraphSAGE	Graph Sample and Aggregate
GraRep	Graph Representation with Global Structural Information
GRU	Gated Recurrent Unit
JK	Jumping Knowledge
K/L	Kernighan–Lin
KAN	Kolmogorov–Arnold Network
LaPE	Laplacian Positional Encoding
LLM	Large Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
MMCut	Min-Max Cut
NAG	Nesterov Accelerated Gradient
NCut	Normalized Cut
NP	Nondeterministic Polynomial time

NPO	Nondeterministic Polynomial time Optimization
PDE	Partial Differential Equation
PE	Positional Encoding
ProNE	Proximity Network Embedding
QR	QR Decomposition
RBFN	Radial Basis Function Network
RCut	Ratio Cut
ReLU	Rectified Linear Unit
ResNet	Residual Network
RNN	Recurrent Neural Network
RWPE	Random Walk Positional Encoding
RWSE	Random Walk Structural Encoding
SGD	Stochastic Gradient Descent
SignNet	Sign-Invariant Neural Network
SiLU	Sigmoid Linear Unit
SVD	Singular Value Decomposition
T5	Text-To-Text Transfer Transformer
TIN	Triangulated Irregular Network
ViT	Vision Transformer
WL	Weisfeiler–Leman

LIST OF SYMBOLS

$G = (V, E)$	Graph with set of vertices V and edges E
$\mathbb{R}$	Ensemble of real numbers
v_i, v_j	Nodes (vertices) in the graph
$e_{i,j}$	Edge between nodes v_i and v_j
$\omega(e_{i,j})$	Weight of edge $e_{i,j}$
S_k	Subset (partition) k of the node set
$\bar{S}_k$	Complement of partition S_k
$\text{Sim}(X, Y)$	Sum of weights of edges connecting sets X and Y
$\text{Cut}(S_1, ..., S_K)$	Total weight of inter-partition edges
$\text{Bal}(S_1, ..., S_K)$	Balance ratio between largest and smallest partitions
$\text{Vol}(X)$	Volume of set X: sum of degrees of nodes in X
$\text{Card}(X)$	Cardinality (number of nodes) of set X
$\text{NCut}(S_1, ..., S_K)$	Normalized cut metric
$\text{RCut}(S_1, ..., S_K)$	Ratio cut metric
$\text{MMCut}(S_1, ..., S_K)$	Min-max cut metric
λ_k	k-th smallest eigenvalue of the graph Laplacian
L	Combinatorial graph Laplacian: $L = D - A$
L_{sym}	Symmetric normalized Laplacian
L_{rw}	Random walk Laplacian
D	Degree matrix
A	Adjacency matrix
$h^{(l)}$	Activation at layer l in a neural network
$W^{(l)}$	Weight matrix at layer l
$b^{(l)}$	Bias vector at layer l
σ	Non-linear activation function
θ	Trainable parameters of the model

$\mathscr{L}(\theta)$	Loss function
η	Learning rate
Q, K, V	Query, Key, and Value matrices in attention mechanism
d_k	Dimensionality of key vectors
$\alpha_{v,u}$	Attention coefficient from node v to node u
$h_v^{(l)}$	Hidden state of node v at layer l
$a_v^{(l)}$	Aggregated message at node v at layer l
$\mathscr{N}(v)$	Neighborhood of node v
ε	Sign-flip or basis symmetry parameter
$f_{\text{aggregate}}, f_{\text{update}}$	Aggregation and update functions in GNNs
$H^{(l)}$	Feature matrix at layer l
$\hat{f}(x)$	Model prediction at input x
$f(x)$	True function generating the data
$\mathbb{E}[\cdot]$	Expected value operator
σ^2	Irreducible noise variance
m_t, v_t	First and second moment estimates in Adam optimizer
$\hat{m}_t, \hat{v}_t$	Bias-corrected moment estimates
β_1, β_2	Decay rates for Adam optimizer
$\Phi(x)$	Gaussian cumulative distribution function
$\text{FFN}(x)$	Feed-forward network applied to each token in Transformer
x_t, h_t, y_t	Input, hidden state, and output at time step t in RNN
x_i, h_j, y_k	Neurons in input, hidden, and output layers of MLP
$\psi_{q,p}, \Phi_q$	Univariate functions in Kolmogorov–Arnold representation
H_{pos}	Positional embedding matrix
H_{struct}	Structural embedding matrix
M	Proximity matrix used in ProNE
U_k, Δ_k, V_k	Truncated SVD components of M
W	Transition matrix: $W = D^{-1}A$

$\tilde{M}^p, M^p$	Multi-hop proximity matrices used in GraRep
τ	Negative sampling ratio
$P_{E,j}$	Negative sample distribution for node j
F_{struct}	Structural feature matrix from random walk encoders
$\phi(v)$	Shared function applied to eigenvectors in SignNet
V_i	Orthonormal basis for eigenspace i
Q_i	Orthogonal matrix for basis transformation
$V_i V_i^T$	Orthogonal projector used for basis-invariance in BasisNet
N	Total number of nodes/vertices in the graph
K	Total number of partitions/clusters
$\|V\|, \|E\|$	Cardinality of the vertex and edge sets, respectively
Ω	Set of elements in a mesh
Ω_k	Subdomain or partition k
Y	Hard assignment matrix $Y \in \{0,1\}^{N \times K}$
$\hat{Y}$	Soft (probabilistic) assignment matrix $\hat{Y} \in [0,1]^{N \times K}$
Z	Final aggregated node embedding matrix
P	Positional embedding matrix
I	Inertia matrix for geometric partitioning
$c_{i,j}$	Cost/weight of the edge between nodes i and j
c_{e_k}	Centroid of mesh element e_k
$\mathbf{1}$	Vector of all ones
$\langle \cdot, \cdot \rangle$	Inner product of two vectors
$\odot$	Element-wise (Hadamard) product
$\mathscr{L}_{XCut}$	Generalized differentiable cut-based loss function
$\mathscr{L}_{Bal}$	Differentiable partition balance loss function
Γ	Vector of cluster volumes or sizes
α, β	Hyperparameters for weighting loss terms
D_k	Cost (or gain) associated with node k in K/L algorithm

$g(i)$	Gain of moving cell i in the FM algorithm
$\mathfrak{T}$	Set of elements (e.g., triangles) in a mesh triangulation
I_N	Identity matrix of size N
$\tilde{A}, \tilde{D}$	Adjacency and degree matrices with self-loops
d	Vector of node degrees
y_k	Indicator vector for cluster k
E_k, I_k	External and internal costs for a node in the KL algorithm

CONTENTS

1 INTRODUCTION 23
1.1 Introduction 23
1.2 Objectives 25
1.2.1 General Objectives 25
1.2.2 Specific Objectives 26
1.3 Work Structure 26
1.4 Chapter Summary and Forward Look 27
2 THEORETICAL FOUNDATION 28
2.1 Introduction 28
2.2 Bibliographic Review 29
2.2.1 Classical Methods 29
2.2.2 Learning-Based Methods 30
2.2.3 Advances in Positional and Structural Encodings 30
2.3 Graph Theory 31
2.3.1 Primal and Dual Graph Representations 31
2.3.2 Spectral Theory 32
2.3.2.1 Laplacian Matrices *33*
2.3.2.2 Algebraic Connectivity and the Fiedler Vector *33*
2.3.2.3 Laplacian Eigenmaps and Positional Encodings *34*
2.3.2.4 Partitioning *34*
2.4 Neural Networks 37
2.4.1 Overview 37
2.4.1.1 Origin and Evolution *37*
2.4.1.2 Motivation and Intuition *38*
2.4.1.3 Universal Approximation Theorem *39*
2.4.1.4 Kolmogorov–Arnold Networks *40*
2.4.1.5 Other Expressive Architectures *41*
2.4.1.6 General Mathematical Formalism *41*
2.4.1.7 Optimization and Gradient Descent Algorithms *43*
2.4.1.7.1 Stochastic Gradient Descent (SGD) 44
2.4.1.7.2 Momentum 45

2.4.1.7.3 Adam (Adaptive Moment Estimation) . . . 46
2.4.1.8 Generalization, Overfitting, and the Bias–Variance Tradeoff . . . *47*
2.4.1.8.1 Training, Validation, and Test Errors . . . 48
2.4.1.8.2 Bias–Variance Decomposition . . . 48
2.4.1.8.3 Strategies to Improve Generalization and Training . . . 49
2.4.2 Deep Neural Architectures . . . 50
2.4.2.1 Multi-Layer Perceptron (MLP) . . . *51*
2.4.2.2 Convolutional Neural Networks (CNN) . . . *52*
2.4.2.3 Recurrent Neural Networks (RNN) . . . *54*
2.4.2.4 Transformer . . . *55*
2.4.2.5 Graph Neural Network . . . *58*
2.4.2.5.1 Message Passing . . . 58
2.4.2.5.2 Graph Convolutional Network . . . 59
2.4.2.5.3 GraphSAGE . . . 60
2.4.2.5.4 Graph Attention Network . . . 61
2.4.2.5.5 Graph Isomorphism Network . . . 62
2.4.2.5.6 Expressivity and Complexity of GNNs . . . 63
2.4.3 Positional and Structural Encoder . . . 64
2.4.3.1 Non-Learnable Positional Embeddings . . . *64*
2.4.3.1.1 Laplacian Eigenvectors . . . 65
2.4.3.1.2 ProNE . . . 65
2.4.3.1.3 GraRep . . . 66
2.4.3.2 Non Learnable Structural Embeddings . . . *67*
2.4.3.2.1 Random Walk . . . 67
2.4.3.3 Learnable Positional Embeddings . . . *68*
2.4.3.3.1 SignNet and BasisNet . . . 69
2.5 Chapter Summary and Forward Look . . . 70
3 METHODOLOGY . . . 72
3.1 Introduction . . . 72
3.2 Heuristics . . . 73
3.2.1 Heuristics for Topologicals Partitionings . . . 73
3.2.1.1 Spectral Clustering and Bisection . . . *73*

3.2.1.2 Kernighan–Lin . . . 74
3.2.1.3 Fiduccia-Mattheyses . . . 76
3.2.1.4 Space Filling Hamiltonian Cycle . . . 77
3.2.2 Heuristics for Spatial Partitionings . . . 78
3.2.2.1 Coordinate . . . 78
3.2.2.2 Inertial . . . 79
3.3 Dataset . . . 80
3.3.1 Fusion 360 . . . 80
3.3.2 Evaluation Set . . . 81
3.4 Architecture . . . 82
3.4.1 Positional and Structural Embedder . . . 83
3.4.2 Feature Embedder . . . 84
3.4.3 Feature Aggregator . . . 85
3.4.4 Classifier . . . 86
3.5 Differentiable Loss Formulation . . . 86
3.5.1 Hard NCut Formulation . . . 87
3.5.2 Soft Relaxation NCut Formulation . . . 88
3.5.3 Generalized Partition Loss . . . 88
3.6 Implementation Details . . . 89
3.7 Chapter Summary and Forward Look . . . 90
4 EXPERIMENTS . . . 91
4.1 Experimental Setup . . . 91
4.2 Neural Network Training . . . 92
4.3 Quantitative Analysis . . . 92
4.4 Qualitative Analysis . . . 94
4.5 Discussion of Model Design and Observations . . . 95
4.5.1 Effectiveness of the MMCut Loss Function . . . 95
4.5.2 Laplacian Eigenmaps as Positional Encoders . . . 95
4.5.3 Comparison of GNN Backbones . . . 97
4.5.4 Limitations of Alternative Encoders . . . 97
4.6 Chapter Summary and Forward Look . . . 98
5 CONCLUSION . . . 100

5.1 **Summary of Work and Contributions** 100
5.2 **Synwork of Key Findings** 101
5.3 **Limitations of the Study** 102
5.4 **Future Work** 103
REFERENCES 104
APPENDICES 113
APPENDIX A – Hyperparameters 113

1 INTRODUCTION

1.1 Introduction

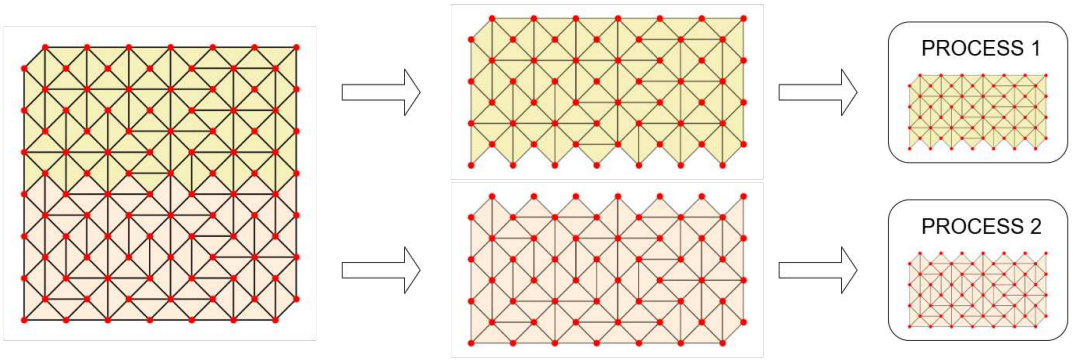
Graph partitioning is a fundamental technique in modern scientific computing and machine learning. Its importance spans applications such as combinatorial optimization, molecular analysis, image segmentation, and, notably, the simulation of physical systems through the Finite Element Method (FEM). FEM is a powerful numerical technique used to solve partial differential equations (PDEs) by discretizing a physical domain into a mesh of simple elements, typically triangles in 2D or tetrahedra in 3D over which the governing equations can be approximated locally. Each element contributes to the global system matrix, whose assembly and solution require significant computational effort.

To efficiently simulate large-scale physical systems, especially in three dimensions, the mesh must be partitioned across multiple processors or GPUs. This enables distributed computation, where each processing unit handles a subset of the mesh. However, poor partitioning can result in high inter-process communication costs and imbalanced workloads, undermining the advantages of parallelism. Effective partitioning aims to achieve three properties: (i) minimize the number of edges crossing between partitions (cut size), (ii) ensure each partition is approximately equal in size (load balance), and (iii) preserve internal connectivity of each partition.

In this context, the mesh can be naturally represented in two equivalent but complementary ways: as a *primal graph* or as a *dual graph*. In the **primal graph**, the vertices correspond to the mesh nodes (geometric points), and the edges represent the mesh elements (e.g., triangles). This representation is useful in finite element formulations where degrees of freedom are defined at nodes, and interpolation functions span adjacent elements. Alternatively, in the **dual graph**, which is particularly relevant for partitioning, each vertex represents a mesh element (e.g., a triangle), and an edge exists between two vertices if their corresponding elements share a common edge. This captures topological adjacency between elements and provides a compact and effective way to encode communication patterns in distributed simulations. Partitioning this dual graph results in element-based partitions, where minimizing edge cuts translates to reducing inter-partition communication.

Graph partitioning methods aim to divide this graph into K disjoint subsets that satisfy the following criteria: low edge cut (communication), load balance (equal number of

elements), and connectivity within each subdomain. Figure 1 shows an example of a triangular mesh partitioned into two balanced subdomains. The elements in each partition are shaded in different colors. Solid black lines represent internal edges within each subdomain and in the boundary between partitions, which correspond to inter-partition communications when each subdomain is assigned to a different processor or GPU in a parallel FEM simulation.



Source: Image from the author (2025)

Figure 1 – Triangular mesh partitioned into two balanced subdomains. In this dual graph view, each element represents a triangle. The partitioning minimizes boundary edges, which are critical in distributed computing as they represent communication between processing units.

Traditional partitioning algorithms include spectral methods based on the Fiedler vector (FIEDLER, 1973; MCSHERRY, 2001), as well as multilevel strategies used in tools like METIS and HMETIS (KARYPIS, 1998; KARYPIS; KUMAR, 1996; KARYPIS; KUMAR, 1998). These methods often rely on coarsening, recursive bisection, and refinement steps such as Kernighan–Lin (KERNIGHAN; LIN, 1970) or Fiduccia–Mattheyses (FIDUCCIA; MATTHEYSES, 1982) to improve partition quality. While effective, these approaches are often hand-crafted and lack the ability to adapt to specific structural or application-driven constraints.

Recently, Graph Neural Networks (GNNs) have emerged as a powerful framework for learning over graph-structured data (KIPF; WELLING, 2016). GNNs operate by applying message-passing layers, where each node aggregates information from its neighbors to update its representation. This iterative process captures both local and global patterns in the graph. Architectures such as GraphSAGE (HAMILTON *et al.*, 2017), Graph Attention Networks (GAT) (VELIČKOVIĆ *et al.*, 2018), and Graph Isomorphism Networks (GIN) (XU *et al.*, 2019) have demonstrated impressive performance across tasks like node classification, link prediction, and graph regression.

These advances have inspired several learning-based approaches for graph parti-

tioning (NAZI *et al.*, 2019; BIANCHI *et al.*, 2020), which optimize neural networks to predict node-to-partition assignments while minimizing edge cuts and maintaining balance. However, many real-world graphs, such as FEM meshes, contain limited or no node features. To address this, models such as (GATTI *et al.*, 2022a) approximate the Fiedler vector and use it as an input signal, while others leverage reinforcement learning (GATTI *et al.*, 2022b) to explore partitioning policies. Additionally, spectral encodings such as Laplacian Eigenmaps (BELKIN; NIYOGI, 2003) and sign-invariant neural representations like SignNet and BasisNet (LIM *et al.*, 2022) have been proposed to capture structural information regardless of feature availability.

In this work, we introduce an unsupervised GNN framework to produce balanced and low-cost partitions in graphs derived from FEM meshes, even in the absence of node features. Our method combines Laplacian-based positional encoders with message-passing layers and a model architecture designed to minimize cut cost while preserving balance. Unlike traditional heuristics, our framework integrates deep learning with spectral geometry to generalize across different graph families, a critical property for parallel FEM simulation and other distributed graph-based computations.

To summarize, the main contributions of this work are:

- The development of an unsupervised GNN architecture that uses spectral encodings to operate effectively on featureless FEM graphs;
- A hybrid strategy (GNN-FM) that refines the GNN's predictions using the Fiduccia–Mattheyses heuristic in linear time;
- A comprehensive evaluation of the proposed method against nine classical algorithms, including METIS, using industrial-scale and unseen 3D FEM meshes.
- The use of GNNs with spectral embeddings enables generalization across graphs

1.2 Objectives

1.2.1 General Objectives

Develop an unsupervised framework based on Graph Neural Networks (GNNs) for the problem of graph partitioning in the context of finite element meshes, aiming to generate balanced and efficient divisions that minimize edge cuts and are suitable for parallel and distributed computing.

1.2.2 Specific Objectives

- Study classical partitioning metrics such as Cut, Normalized Cut, Ratio Cut, and Min-Max Cut, and analyze their theoretical properties.
- Investigate message-passing GNN architectures (GCN, GAT, GraphSAGE, GIN) in the context of inductive partitioning.
- Integrate positional encoders (e.g., Laplacian eigenvectors, SignNet, BasisNet) to enrich node representations in graphs with few or no features.
- Propose a neural architecture capable of learning node-to-partition assignments in an unsupervised setting using differentiable loss functions.
- Evaluate the proposed method across benchmark graphs and finite element meshes, comparing it with classical partitioners such as METIS, spectral clustering, and FM-based algorithms.
- Analyze the trade-offs between cut minimization and partition balance, and discuss limitations regarding partition connectivity.

1.3 Work Structure

This work is organized as follows:

- **Chapter 1** introduces the graph partitioning problem, highlighting its importance in the context of finite element simulations and distributed computing. It also motivates the use of Graph Neural Networks (GNNs) as a learning-based approach to partition scientific meshes.
- **Chapter 2** presents the theoretical background necessary to understand the proposed method. It covers graph representations of meshes, classical and modern partitioning algorithms, spectral theory, partitioning quality metrics (e.g., NCut, RCut, MMCut), neural network fundamentals, and architectural models such as MLPs, CNNs, RNNs, GNNs, and Transformers, including their mathematical formulations and inductive biases.
- **Chapter 3** details the methodology, including the proposed architecture, the loss function design and the algorithm of baseline heuristics. It also describes the implementation setup, highlighting the use of some libraries.
- **Chapter 4** presents the experimental protocol and results on various graphs, comparing the proposed method with traditional partitioning algorithms in terms of cut quality and

partition balance.

- **Chapter 5** concludes the work, summarizing the main findings, discussing limitations, such as the lack of strict connectivity guarantees and outlining possible future research directions, including enforcing contiguity and extending to k-way partitioning using recursive bisection.

1.4 Chapter Summary and Forward Look

The introduction provided a comprehensive overview of the graph partitioning problem in the context of parallel finite element simulations. It highlighted the importance of distributing FEM meshes across multiple processors while minimizing inter-partition communication and maintaining balanced workloads. It outlined the limitations of classical heuristics when applied to featureless graphs and motivated the use of Graph Neural Networks (GNNs) for adaptive, data-driven partitioning.

Additionally, the section clarified the central research question: how to construct an unsupervised GNN-based framework that performs competitively without access to node features. The objectives, both general and specific, were articulated to guide the design of the architecture, the selection of metrics like Min-Max Cut (MMCut), and the integration of spectral encodings such as the Fiedler vector.

By framing the work in terms of its challenges, motivations, and expected contributions, this section sets the stage for what follows. The next part provides the theoretical and mathematical underpinnings required to understand the design choices, such as Laplacian operators, positional encoding strategies, and neural network foundations, which are critical to both the model architecture and its loss function formulation.

2 THEORETICAL FOUNDATION

2.1 Introduction

This chapter presents the theoretical principles underlying the proposed framework, unifying concepts from finite element analysis, graph theory, spectral methods, deep learning, and distributed computing. The convergence of these disciplines is critical for solving large-scale problems defined over irregular domains, such as computational meshes in scientific simulations.

We begin by introducing how computational meshes can be represented as graphs through either a **primal** or **dual** formulation (GEUZAINE; REMACLE, 2009). In the primal graph, nodes represent mesh vertices and edges connect physically adjacent nodes. In contrast, the dual graph treats each mesh element as a node and connects elements that share a common interface. This graph abstraction enables the use of graph-theoretic algorithms for mesh manipulation, analysis, and partitioning.

We then examine spectral graph theory and its central role in graph partitioning. In particular, we analyze the properties of graph Laplacians, their eigenvalues and eigenvectors, and how these relate to key concepts such as algebraic connectivity and the Fiedler vector. These spectral components form the basis for various partitioning criteria, including cut, normalized cut, ratio cut, and min-max cut, and provide a mathematical foundation for positional encodings (BELKIN; NIYOGI, 2003) used in Graph Neural Networks (GNNs)

Finally, we delve into the evolution of neural networks and their extension to non-Euclidean domains via Graph Neural Networks (GNNs). From classical architectures such as Multi-Layer Perceptrons and Convolutional Neural Networks to modern Graph Isomorphism Networks and attention-based models (KIPF; WELLING, 2016; XU *et al.*, 2019; VELIČKOVIĆ *et al.*, 2018), we highlight how these models leverage the structure of data to learn representations. In particular, we emphasize their growing use in scientific computing, including applications like mesh segmentation, load balancing, and connectivity-aware partitioning (NAZI *et al.*, 2019).

This chapter thus establishes the theoretical and algorithmic context for the framework proposed in this work, which integrates geometric priors from finite element meshes, spectral theory, and learning-based models into a unified solution for graph partitioning on scientific domains.

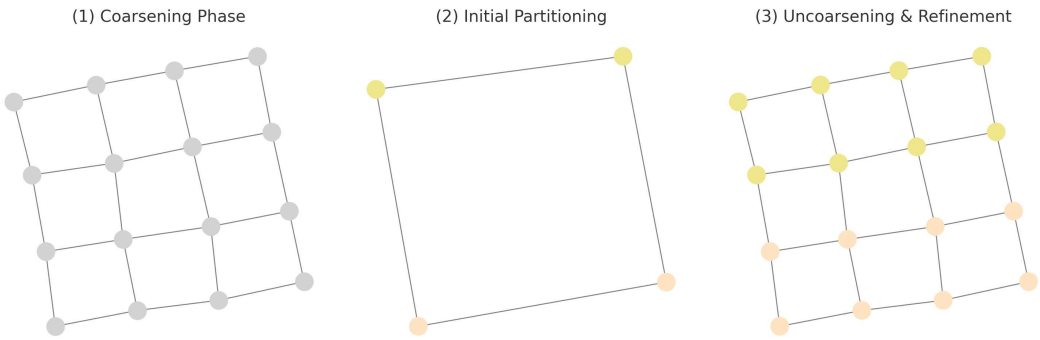
2.2 Bibliographic Review

Over the past decades, the graph partitioning problem has been extensively studied, resulting in a wide range of algorithms with varying levels of scalability, complexity, and theoretical guarantees. These methods can be broadly categorized into classical combinatorial or spectral approaches, and more recently, learning-based models.

2.2.1 Classical Methods

Spectral partitioning methods were among the first to establish theoretical foundations for the problem. In particular, the Fiedler vector, the eigenvector associated with the second smallest eigenvalue of the graph Laplacian, is used to bisect graphs based on sign or magnitude (FIEDLER, 1973; MCSHERRY, 2001). Although computationally expensive, these approaches offer strong guarantees in terms of partition quality and are deeply connected to graph connectivity.

Multilevel methods represent the state of the art in scalable partitioning. Algorithms such as METIS, PMETIS, and HMETIS adopt a coarse-to-fine strategy to address the computational challenges of large-scale graphs (KARYPIS, 1998). As illustrated in Figure 2, the multilevel pipeline consists of three main stages. First, during the *coarsening phase*, the original graph is iteratively collapsed into a smaller graph, aggregating nearby vertices to form coarse representations. This phase significantly reduces the problem size. Then, in the *initial partitioning* step, a fast (and potentially suboptimal) partitioning algorithm is applied to the coarsened graph. Finally, the solution is projected back to the original graph through *uncoarsening and refinement*, where local optimization techniques refine the partition while preserving the overall structure.



Source: Image from the author (2025)

Figure 2 – Multilevel partitioning strategy. From left to right: (1) Coarsening of the original graph; (2) Initial partitioning of the coarse graph; (3) Projection and refinement of the partition on the original graph.

These methods are widely used in scientific computing due to their robustness and performance, particularly in mesh-based applications such as finite element simulations. Refinement heuristics such as Kernighan–Lin (KERNIGHAN; LIN, 1970) and Fiduccia–Mattheyses (FIDUCCIA; MATTHEYSES, 1982) further improve existing partitions by iteratively swapping vertices across partitions to reduce edge cuts. These heuristics are often embedded within the uncoarsening phase of multilevel frameworks.

2.2.2 Learning-Based Methods

With the emergence of deep learning, several models have been proposed to perform graph partitioning in a learnable and generalizable way. The GAP framework (NAZI *et al.*, 2019) introduced a supervised GNN-based model that learns to predict soft partition assignments using differentiable versions of cut and balance metrics. Mincut Pooling (BIANCHI *et al.*, 2020) proposes a layer-wise pooling operation that implicitly partitions the graph via a cut-optimization loss.

Recent work has also explored deep spectral embedding (GATTI *et al.*, 2022a), where node embeddings are learned to be spectrally separable, as well as reinforcement learning-based partitioning (GATTI *et al.*, 2022b), which formulates the task as a sequential decision-making problem.

Additionally, the survey by Bronstein et al. (BRONSTEIN *et al.*, 2021) provides a comprehensive overview of geometric deep learning, contextualizing the evolution of GNNs from the perspective of symmetries, topologies, and domains such as grids, manifolds, and graphs. This work highlights both theoretical underpinnings and practical architectures relevant to mesh-based computations.

2.2.3 Advances in Positional and Structural Encodings

A longstanding challenge for graph-based models is the absence of an inherent notion of position. Classical **Laplacian Eigenmaps** (BELKIN; NIYOGI, 2003) address this by projecting each vertex onto the first non-trivial eigenvectors of the (normalized) graph Laplacian, so that nearby vertices in the graph remain close in the embedding space. While effective, these eigenvectors are sign-ambiguous and suffer from basis indeterminacy in the presence of repeated eigenvalues, which can destabilize downstream learning.

Recent work has proposed more scalable, spectrum-inspired positional encoders.

GraRep (CAO *et al.*, 2015) factorises multi-step transition matrices, capturing progressively global structural information while remaining agnostic to node features. On the other hand, **ProNE** (ZHANG *et al.*, 2019) accelerates spectral filtering via randomized SVD followed by Chebyshev polynomial propagation, producing high-quality embeddings in near-linear time. Both techniques provide fixed, purely positional features that can be concatenated with learnable representations inside a GNN.

Complementary to position, structural encoders aim to characterise a node's role or type in the graph. Random-walk statistics, for example, the distribution of t-step return probabilities, form the basis of the **Random-Walk Structural Encoding (RWSE)** introduced in (DWIVEDI *et al.*, 2022). Because these signatures depend only on the walk dynamics, they are invariant to graph isomorphisms while being sensitive to meso-scale regularities such as core–periphery structure.

Finally, **SignNet** and **BasisNet** (LIM *et al.*, 2022) learn to aggregate sets of eigenvectors in a way that is provably invariant to both arbitrary sign flips and rotations within degenerate eigenspaces. These models yield stable positional encodings that pair naturally with the fixed embeddings above, further boosting robustness on mesh-like graphs where eigenvalue multiplicities are common.

Taken together, modern positional (Laplacian Eigenmaps, ProNE, GraRep) and structural (RWSE) encoders provide complementary, hand-crafted priors that can be fused or fine-tuned inside message-passing networks, giving rise to state-of-the-art performance on graphs that lack rich node attributes.

2.3 Graph Theory

2.3.1 Primal and Dual Graph Representations

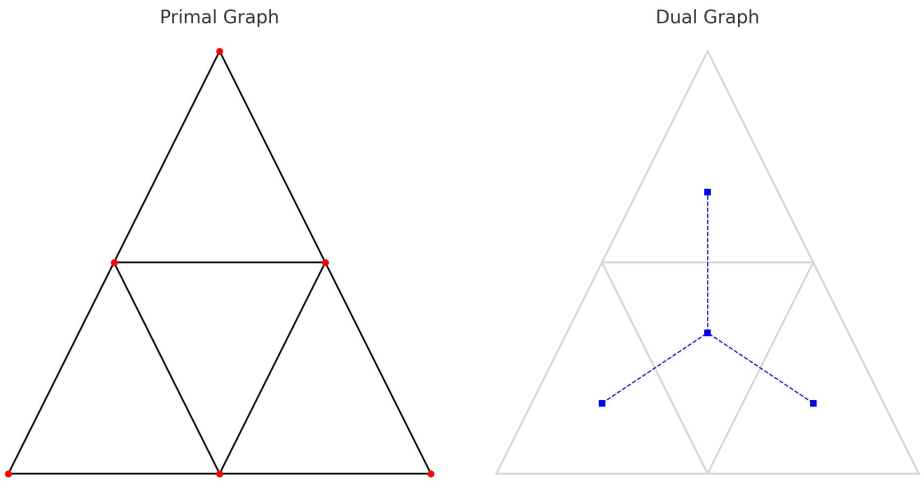
In the context of mesh-based simulations, graphs arise naturally as abstractions of the underlying spatial discretization. Two distinct but complementary representations are commonly employed:

- **Primal graph**: Each node corresponds to a vertex (or degree of freedom) of the original mesh, and edges represent direct physical connections (edges) between mesh vertices. This representation is widely used in Graph Neural Networks (GNNs) as it aligns with node-based features and physical quantities.

- **Dual graph**: Each node represents a mesh element (e.g., triangle or tetrahedron), and an edge is formed between two nodes if the corresponding elements share a common face or edge. This representation is commonly used in mesh partitioning and domain decomposition methods.

Let $\mathscr{T}$ be a triangulation of the domain Ω. Define the primal graph $G_P = (V_P, E_P)$ such that V_P is the set of mesh vertices $v_i \in \Omega$ and $(v_i, v_j) \in E_P$ if v_i and v_j are connected by an edge in some element $T_k \in \mathscr{T}$. Conversely, the dual graph $G_D = (V_D, E_D)$ takes V_D as the set of elements $T_k \in \mathscr{T}$ and places an edge between T_i and T_j if they share a common face (3D) or edge (2D).

These representations allow mesh-related problems to be framed as graph problems, where partitioning, message passing, and load balancing correspond to topological operations on nodes and edges. The dual graph is especially suitable for partitioning in distributed finite element methods, while the primal graph is more suitable for modeling physical interactions in numerical solvers and neural architectures.



Source: Image from the author (2025)

Figure 3 – Visual comparison between the primal and dual graph representations. On the left, the primal graph connects mesh vertices (red) that share an edge. On the right, the dual graph connects element centroids (blue) that share a common edge.

2.3.2 Spectral Theory

Spectral theory plays a central role in modern graph partitioning algorithms. It bridges discrete graph properties with continuous optimization through the eigenstructure of graph Laplacians. This section delves into the mathematical underpinnings of spectral partitioning, detailing the role of Laplacian eigenvalues, the Fiedler vector, and their connection to

partitioning metrics and positional encodings.

2.3.2.1 Laplacian Matrices

Given an undirected graph $G = (V, E)$ with n vertices, the combinatorial Laplacian matrix $L \in \mathbb{R}^{n \times n}$ is defined as:

$$L = D - A, \tag{2.1}$$

where D is the degree matrix (a diagonal matrix where $D_{ii} = \deg(v_i)$), and A is the adjacency matrix.

There are two important normalized variants of the Laplacian:

- **Symmetric normalized Laplacian:**

  $$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}, \tag{2.2}$$

  which is symmetric and has real eigenvalues.

- **Random walk Laplacian:**

  $$L_{rw} = D^{-1} L = I - D^{-1} A, \tag{2.3}$$

  which governs diffusion processes and random walks on graphs.

All three Laplacians are positive semi-definite, and their eigenvalues carry structural information about the graph. In particular, the smallest eigenvalue is always $\lambda_0 = 0$, with the corresponding eigenvector being the constant vector $\mathbf{1}$. The multiplicity of the zero eigenvalue equals the number of connected components in the graph. The **random walk Laplacian** is particularly relevant in algorithms based on diffusion dynamics and Markov chains. One of the most influential applications of this operator is the **PageRank** algorithm (PAGE *et al.*, 1999), originally developed by Google to rank web pages. PageRank models a random surfer that either follows a hyperlink with a certain probability or teleports to a random node, resulting in a stationary distribution over the graph. This steady-state distribution captures the importance or centrality of each node based on global connectivity patterns and is computed using a stochastic matrix derived from $D^{-1}A$.

2.3.2.2 Algebraic Connectivity and the Fiedler Vector

The second smallest eigenvalue λ_1 of the Laplacian is known as the *algebraic connectivity* (FIEDLER, 1973). A larger λ_1 indicates a more connected graph. The corresponding eigenvector is called the **Fiedler Vector**, which is instrumental in spectral partitioning:

- The sign or magnitude of Fiedler vector entries is used to bipartition the graph: nodes are assigned to different subsets based on whether their value is above or below a threshold (typically the median)(POTHEN *et al.*, 1990).
- This yields partitions that minimize the edge cut while preserving balanced sizes, particularly effective for planar graphs or meshes.

2.3.2.3 Laplacian Eigenmaps and Positional Encodings

Laplacian Eigenmaps (BELKIN; NIYOGI, 2003) are a nonlinear dimensionality reduction technique that leverages the eigenvectors of the graph Laplacian to map nodes to a lower-dimensional Euclidean space. The intuition is to preserve local neighborhood structures in the embedding:

- Nodes that are strongly connected (i.e., have high edge weight) are embedded close to each other.
- The first few non-trivial eigenvectors of L_{sym} (excluding the constant eigenvector) are used as coordinates in the low-dimensional space.

This method has become foundational for positional encodings in Graph Neural Networks (GNNs), where eigenvectors serve as structural features that capture the graph topology (DWIVEDI *et al.*, 2022; LIM *et al.*, 2022). These embeddings are permutation-invariant and reflect both local and global connectivity patterns.

2.3.2.4 Partitioning

Graph partitioning is the problem of dividing a graph $G = (V, E)$ into K disjoint subsets $S_1, ..., S_K$ such that $V = \bigcup_{k=1}^{K} S_k$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. This task is fundamental in applications such as distributed computing, mesh simulation, and data clustering.

The main objectives of graph partitioning algorithms are:

- **Minimize the edge cut**, i.e., the total weight of the edges crossing between partitions;
- **Balance** the size of the partitions, ensuring that each S_k has approximately the same number of vertices;
- **Maximize internal connectivity**, ensuring that each partition forms a connected subgraph.

Several metrics can be minimized to perform graph partitioning, the most basic being

the cut-edge, which measures the total weight of edges that are severed by the partitioning:

$$\text{Cut}(S_1,...,S_K) = \sum_{k=1}^{K} \text{Sim}(S_k, \bar{S}_k) \tag{2.4}$$

where

$$\text{Sim}(X,Y) = \sum_{e_{i,j} \in E \cap (X \times Y)} \omega(e_{i,j}), \tag{2.5}$$

$\omega(e_{i,j})$ is the weight of the edge (v_i, v_j) and X, Y are the partitions S_k.

However, a desirable property in many applications such as mesh partitioning or parallel computing is that the partitions are balanced in size. The balance metric measures how evenly nodes are distributed among the partitions:

$$\text{Bal}(S_1,...,S_K) = \frac{\max_{k \in \{1,...,K\}} |S_k|}{\min_{k \in \{1,...,K\}} |S_k|} \tag{2.6}$$

To address this, normalized versions of the cut metric are used. These metrics incorporate information about the size or volume of each partition to discourage trivial solutions with very small clusters. There are several metrics that can be minimized to perform graph partitioning, the most common being cut-edge, normalized cut-edge, ratio cut-edge and minmax cut-edge. However, a very desirable characteristic in several cases such as mesh partitioning is that the partitions have similar sizes, which is not satisfied by some of these metrics, so, as in (DING *et al.*, 2001), we analyze the case of a random graph.

$$\text{NCut}(S_1,...,S_K) = \sum_{k=1}^{K} \frac{\text{Sim}(S_k, \bar{S}_k)}{\text{Vol}(S_k)} \tag{2.7}$$

$$\text{RCut}(S_1,...,S_K) = \sum_{k=1}^{K} \frac{\text{Sim}(S_k, \bar{S}_k)}{\text{Card}(S_k)} \tag{2.8}$$

$$\text{MMCut}(S_1,...,S_K) = \sum_{k=1}^{K} \frac{\text{Sim}(S_k, \bar{S}_k)}{\text{Sim}(S_k, S_k)} \tag{2.9}$$

where

$$\text{Vol}(X) = \sum_{e_{i,j} \in E \cap (X \times V)} \omega(e_{i,j}), \quad \text{Card}(X) = |X| \tag{2.10}$$

These normalized metrics aim to produce partitions that are not only weakly connected to the rest of the graph but also internally cohesive and balanced, as discussed in (DING *et al.*, 2001).

Theorem 1. *Let $G = (V,E)$ be a random graph with N nodes and E edges such that $0 < p \leq 1$ is the probability of two nodes being connected. Minimizing the cut-edge favors skewed cuts, NCut and RCut have no size preferences, and MMCut favors balanced partitions.*

Proof. All nodes have the same degree $p(N-1)$. We compute each metric accordingly:

$$\text{Cut}(S_1,...,S_K) = p\sum_{k=1}^{K}|S_k|(N-|S_k|) \tag{2.11}$$

$$\text{NCut}(S_1,...,S_K) = \sum_{k=1}^{K}\frac{p|S_k|(N-|S_k|)}{p|S_k|(N-1)} = \sum_{k=1}^{K}\frac{N-|S_k|}{N-1} = \frac{(K-1)N}{N-1} \tag{2.12}$$

$$\text{RCut}(S_1,...,S_K) = \sum_{k=1}^{K}\frac{p|S_k|(N-|S_k|)}{|S_k|} = \sum_{k=1}^{K}p(N-|S_k|) = pN(K-1) \tag{2.13}$$

$$\text{MMCut}(S_1,...,S_K) = \sum_{k=1}^{K}\frac{p|S_k|(N-|S_k|)}{p|S_k||S_k|} = \sum_{k=1}^{K}\frac{N-|S_k|}{|S_k|} = N\left(\sum_{k=1}^{K}\frac{1}{|S_k|}\right) - K \tag{2.14}$$

Minimizing each of the above metrics, we get that Cut results in $S_1 = N-K+1$ and $S_i = 1$ for $i \neq 1$, while NCut and RCut show no preference in size and MMCut requires $S_1 \approx S_2 \approx \cdots \approx S_K$. □

In order to perform an optimization problem, it is important that the function has a lower bound. We can see from the equations of the metrics that they are positive, which already guarantees a lower bound. However, we can go further and find a more specific value. In this context, (GU *et al.*, 2001; CHAN *et al.*, 1994) proposed some algorithms for partitioning using these metrics, and obtained the following lower bounds, where λ_k denotes the k-th smallest eigenvalue of the Laplacian matrix:

$$\sum_{k=1}^{K}\lambda_k \leq \min \text{NCut}(S_1,...,S_K) \tag{2.15}$$

$$\sum_{k=1}^{K}\lambda_k \leq \min \text{RCut}(S_1,...,S_K) \tag{2.16}$$

$$\frac{k^2}{\sum_{k=1}^{K}\lambda_k} - k \leq \min \text{MMCut}(S_1,...,S_K) \tag{2.17}$$

Such algorithms work well because the eigenvectors represent the local and global connectivity of the graph, so the smaller eigenvalues represent more connected components, making it possible to create partitions that are strongly internally connected and weakly connected to each other.

2.4 Neural Networks

2.4.1 Overview

Artificial Neural Networks (ANNs) are computational models inspired by the interconnected structure and dynamics of biological neurons in the brain. These models are built from simple processing units (neurons) that, when composed in layered architectures, can learn to represent complex patterns and relationships in data. Neural networks have evolved significantly over the past decades and now serve as the backbone of modern artificial intelligence, enabling transformative advances in fields such as vision, language, robotics, and scientific modeling.

2.4.1.1 Origin and Evolution

The origin of neural networks can be traced back to the 1940s with the McCulloch-Pitts model, a simplified abstraction of a biological neuron. However, a major milestone came with the development of the Perceptron by Frank Rosenblatt in 1958 (ROSENBLATT, 1958). The Perceptron introduced the concept of learning a linear decision boundary and was able to solve simple pattern recognition tasks.

The revival of neural networks occurred in the 1980s with the introduction of the backpropagation algorithm (RUMELHART *et al.*, 1986), which allowed training of multi-layer perceptrons (MLPs) through gradient descent. This breakthrough showed that non-linear functions could be learned by deep hierarchical compositions of neurons, though training remained slow and data-hungry.

Despite theoretical progress, neural networks struggled in practice due to several limitations: insufficient computational power, lack of large labeled datasets, and unstable training dynamics (e.g., vanishing/exploding gradients). These issues delayed the widespread adoption of deep learning.

The modern deep learning revolution began in 2012 with the success of AlexNet, a deep convolutional neural network that drastically outperformed traditional methods in the

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (KRIZHEVSKY *et al.*, 2017). Key factors contributing to this breakthrough included:

- The use of GPUs for efficient parallel computation of large models.
- The availability of large-scale datasets like ImageNet, containing millions of annotated images.
- Advances in regularization techniques such as dropout and data augmentation.
- Improved software frameworks like TensorFlow (ABADI *et al.*, 2016) and PyTorch (PASZKE *et al.*, 2019), which facilitated rapid experimentation.

Since then, deep learning has made extraordinary strides across domains:

- In computer vision, models like ResNet (HE *et al.*, 2015a), EfficientNet (TAN; LE, 2019), and Vision Transformers (DOSOVITSKIY *et al.*, 2020) have achieved near-human or even superhuman performance in classification, detection, and segmentation.
- In natural language processing, the development of Transformer architectures (VASWANI *et al.*, 2017) and large language models (LLMs) like BERT (DEVLIN *et al.*, 2018) and GPT (RADFORD *et al.*, 2019) has enabled machines to generate and reason over human language with unprecedented fluency.
- In robotics and control, end-to-end models learn control policies directly from sensory data (LEVINE *et al.*, 2016).
- In computational biology, AlphaFold (JUMPER *et al.*, 2021) used deep learning to solve the protein folding problem, a grand challenge in structural biology.
- In materials science, neural models are used to discover novel compounds and predict physical properties (BUTLER *et al.*, 2018).

This trajectory of progress continues, with deep learning now being extended to irregular domains such as graphs, manifolds, and scientific simulations.

2.4.1.2 Motivation and Intuition

The power of neural networks lies in their ability to learn from data without the need for manual feature engineering. A key theoretical result that supports their flexibility is the Universal Approximation Theorem (HORNIK *et al.*, 1989), which asserts that a feedforward neural network with a single hidden layer containing a sufficient number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$, given an appropriate activation function.

This ability to approximate arbitrary nonlinear mappings makes neural networks pow-

erful tools for a wide variety of supervised and unsupervised learning tasks. Their performance is further enhanced by the composition of multiple layers, enabling hierarchical feature extraction, from low-level patterns in early layers (e.g., edges in images) to high-level abstractions in deeper layers (e.g., object categories, semantics, or symbolic reasoning).

The success of neural networks also stems from their compatibility with efficient optimization techniques (e.g., stochastic gradient descent) and their scalability to large datasets and distributed computing environments. This has made them the model of choice in applications ranging from real-time speech translation to scientific simulations at exascale.

2.4.1.3 Universal Approximation Theorem

It states that a feedforward network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, given suitable activation functions.

Theorem 2 (Universal Approximation)**.** *Let σ be a nonconstant, bounded, and continuous activation function. Then, for any continuous function f on a compact subset of $\mathbb{R}^n$ and any $\varepsilon > 0$, there exists a neural network $\phi(x) = \sum_{i=1}^{N} \alpha_i \sigma(w_i^T x + b_i)$ such that $\sup_x |f(x) - \phi(x)| < \varepsilon$.*

This remarkable result, originally proved by Cybenko in 1989 (CYBENKO, 1989), and extended by Hornik et al. (HORNIK *et al.*, 1989), provides theoretical justification for using neural networks in a wide variety of applications.

Although the theorem guarantees approximation in the L^∞ norm for continuous functions on compact sets, it does not provide a bound on the number of neurons required, nor on the sample complexity. Approximation in L^p norms for $p < \infty$ involves integration and may result in lower or more tractable requirements, but it remains sensitive to high-dimensional inputs.

The **curse of dimensionality** remains a significant challenge: as the dimensionality of the input space increases, the volume of the space grows exponentially, requiring exponentially more training samples to cover the space adequately. Consequently, neural networks must rely on architectural inductive biases (e.g., convolutional structure, weight sharing) and regularization strategies.

2.4.1.4 Kolmogorov–Arnold Networks

The Kolmogorov–Arnold representation theorem is a fundamental result in approximation theory, stating that any multivariate continuous function can be expressed as a finite composition of continuous univariate functions and additions. It highlights that, in theory, multivariate function approximation can be reduced to the approximation of simpler one-dimensional components.

Theorem 3 (Kolmogorov–Arnold Representation)**.** *For every continuous function* $f : [0,1]^n \to \mathbb{R}$*, there exist continuous univariate functions* $\Phi_q : \mathbb{R} \to \mathbb{R}$ *and* $\psi_{q,p} : \mathbb{R} \to \mathbb{R}$ *such that:*

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{q,p}(x_p) \right) \tag{2.18}$$

While theoretically powerful, this result has long been considered impractical for machine learning due to the potentially pathological nature of the inner functions $\psi_{q,p}$ and outer functions Φ_q, which can be non-smooth or highly irregular, rendering them unsuitable for gradient-based training or generalization in real-world data.

However, recent developments have revisited this representation with optimism. In particular, Kolmogorov–Arnold Networks (KANs) (LIU *et al.*, 2025) propose a neural architecture that directly builds upon this theorem. Unlike conventional multilayer perceptrons (MLPs) that place fixed nonlinearities at nodes, KANs associate learnable univariate activation functions with edges and implement summations at the nodes. Each "weight" is replaced by a function, often parameterized as a spline.

This structural change enables KANs to decompose high-dimensional functions into compositions of one-dimensional transformations, aligning precisely with the Kolmogorov–Arnold theorem. Empirical results demonstrate that KANs achieve superior performance in regression tasks, often with fewer parameters than MLPs, and exhibit strong interpretability and generalization properties, especially when the target functions have compositional or symbolic structure.

Thus, although the original theorem was not directly usable in standard learning frameworks, its recent reinterpretation through KANs revives its relevance and provides a novel, mathematically grounded alternative to conventional neural architectures.

2.4.1.5 *Other Expressive Architectures*

- **Boltzmann Machines:** Stochastic neural networks that learn a probability distribution over inputs. Weights are learned using contrastive divergence (HINTON, 2002).
- **Radial Basis Function Networks (RBFNs):** Use radial basis functions as activation functions and are especially suited for interpolation.

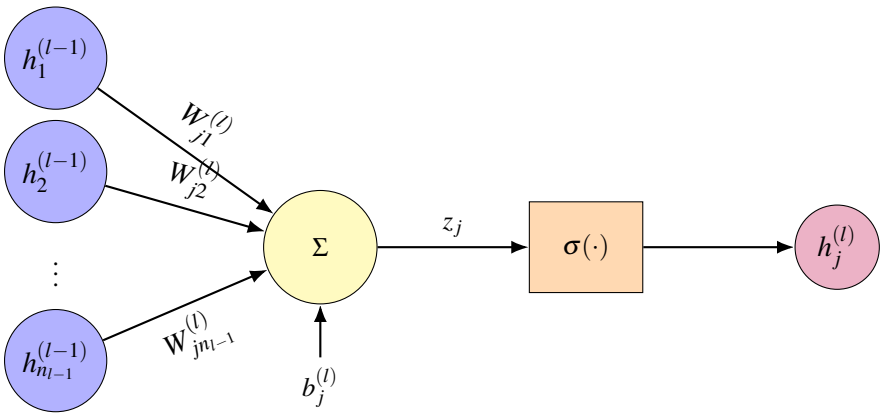
2.4.1.6 *General Mathematical Formalism*

To visually explain the operation of a single neuron in a feedforward layer, we refer to Figure 4, which depicts the computational process behind the update rule. In a feedforward neural network, each layer l performs an affine transformation followed by a nonlinear activation function. The output of layer l is computed as:

$$h^{(l)} = \sigma\left(W^{(l)}h^{(l-1)} + b^{(l)}\right) \tag{2.19}$$

where

- $h^{(l-1)} \in \mathbb{R}^{n_{l-1}}$ is the input to layer l,
- $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix,
- $b^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector,
- $\sigma(\cdot)$ is a nonlinear activation function applied element-wise,
- $h^{(l)} \in \mathbb{R}^{n_l}$ is the resulting activation.



Source: Image from the author (2025)

Figure 4 – Computational diagram of a single neuron in a feedforward neural network. The neuron receives inputs from the previous layer $h_i^{(l-1)}$, multiplies each by its corresponding weight $W_{ji}^{(l)}$, adds a bias term $b_j^{(l)}$, and applies a nonlinear activation function $\sigma(\cdot)$ to produce the output $h_j^{(l)}$. This operation enables the network to learn complex, non-linear mappings from input to output. .

The choice of activation function σ plays a critical role in introducing nonlinearity into the network. Some commonly used activation functions include:

- **Sigmoid:**

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.20}$$

  Maps input to the interval $(0,1)$. However, it suffers from the vanishing gradient problem for large input magnitudes (GLOROT; BENGIO, 2010).

- **Hyperbolic Tangent (Tanh):**

$$\sigma(x) = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.21}$$

  Outputs values in $(-1,1)$ and is zero-centered, providing advantages over sigmoid in many training scenarios.

- **Rectified Linear Unit (ReLU):**

$$\sigma(x) = \max(0,x) \tag{2.22}$$

  ReLU is simple and computationally efficient. It helps mitigate the vanishing gradient problem but can suffer from "dying neurons" (NAIR; HINTON, 2010).

- **Gaussian Error Linear Unit (GELU):**

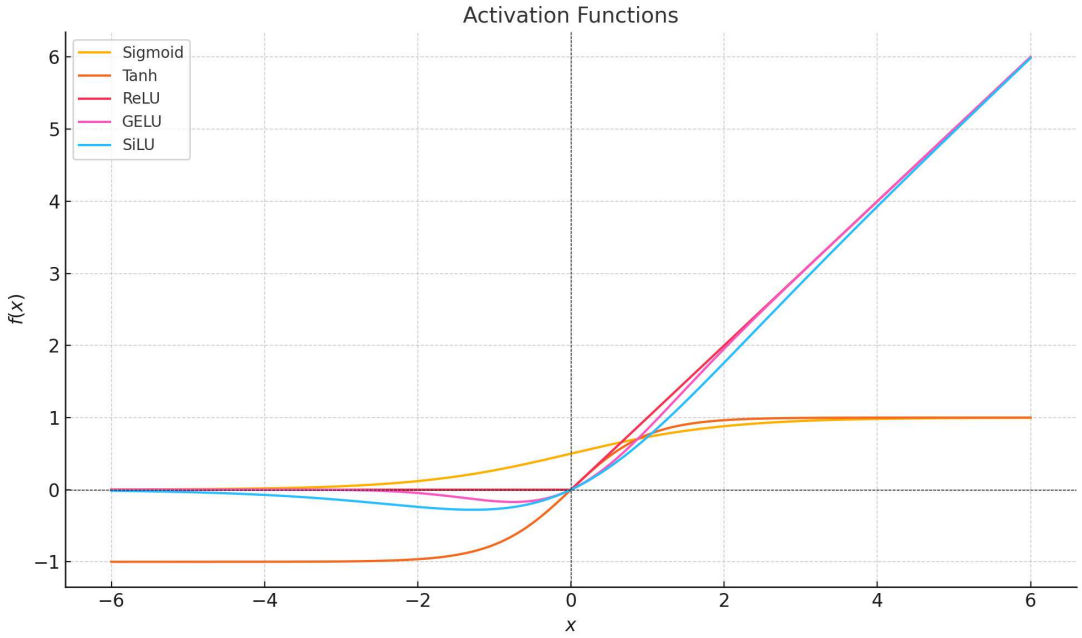
$$\sigma(x) = x \cdot \Phi(x), \quad \text{where } \Phi(x) = \frac{1}{2}\left[1+\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{2.23}$$

  GELU (HENDRYCKS; GIMPEL, 2016) smoothly blends properties of ReLU and sigmoid and is widely used in transformer-based architectures like BERT.

- **Sigmoid Linear Unit (SiLU) or Swish:**

$$\sigma(x) = x \cdot \frac{1}{1+e^{-x}} \tag{2.24}$$

  Introduced in (RAMACHANDRAN *et al.*, 2018), SiLU is differentiable and smooth, often outperforming ReLU in deep learning benchmarks.



Source: Image from the author (2025)

Figure 5 – Common activation functions used in neural networks. Each function maps input x to the output $f(x)$, introducing nonlinearity into the model. Sigmoid and Tanh are bounded and smooth but prone to vanishing gradients. ReLU is piecewise linear and computationally efficient. GELU and SiLU provide smooth transitions and are favored in modern architectures like Transformers..

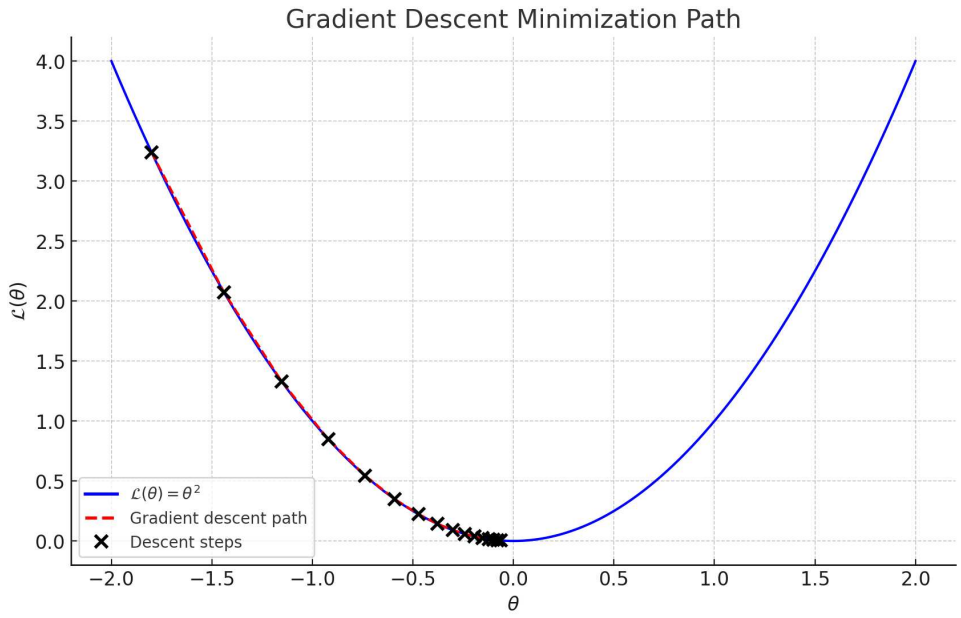
The Figure 5 shows each one the activation functions described above, its choice significantly affects the convergence behavior, gradient propagation, and generalization capacity of deep neural networks. Smooth activations such as GELU and SiLU are preferred in modern architectures due to their better optimization properties.

2.4.1.7 Optimization and Gradient Descent Algorithms

Artificial neural networks are trained by minimizing a loss function $\mathscr{L}(\theta)$, which quantifies the discrepancy between the model predictions and the ground truth. The optimization is typically performed using variants of gradient descent, such as SGD, Momentum, or Adam. These are iterative algorithms that update the parameters θ in the direction of the negative gradient of the loss:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathscr{L}(\theta) \tag{2.25}$$

where $\eta > 0$ is the learning rate, which controls the step size of the update.



Source: Image from the author (2025)

Figure 6 – Visualization of the gradient descent optimization process. The red arrows show successive updates of the parameter θ as it follows the negative gradient direction to minimize the loss function $\mathscr{L}(\theta) = \theta^2$. .

As illustrated in Figure 6, the optimization process starts from an initial value of θ and iteratively moves downhill along the slope of the loss function. The red arrows indicate the direction and magnitude of the parameter updates, while the black crosses mark the updated positions of θ at each step. Over time, the parameter converges toward the minimum of the loss function, where the gradient vanishes and the update becomes negligible.

2.4.1.7.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a fundamental optimization algorithm used to train neural networks by minimizing a loss function $\mathscr{L}(\theta)$, where θ denotes the model parameters. The key idea is to iteratively update the parameters in the direction of the negative gradient, which points toward the steepest descent of the loss surface.

Unlike traditional (batch) gradient descent, which computes gradients over the entire training dataset, SGD estimates the gradient using a randomly selected mini-batch of training samples at each iteration. This yields the update rule:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathscr{L}_{\text{mini-batch}}(\theta) \tag{2.26}$$

where

- $\eta > 0$ is the learning rate,

- $\mathscr{L}_{\text{mini-batch}}$ is the empirical loss over a subset of the training data.

The stochastic nature of SGD introduces noise into the gradient estimates, which, although it may make the updates less stable, provides several important advantages. First, by operating on small mini-batches rather than the entire dataset, SGD significantly reduces the computational cost per iteration, making it scalable to large datasets. Additionally, the inherent randomness helps the optimizer escape poor local minima and saddle points (BOTTOU, 2010), a crucial property in highly non-convex loss landscapes common in deep learning. This stochasticity also promotes faster exploration of the parameter space during the early stages of training, potentially leading to better solutions.

However, despite these strengths, vanilla SGD suffers from some notable limitations. It may converge slowly, particularly in regions of the loss surface with varying curvature, and it often oscillates in narrow valleys or becomes inefficient in flat regions, delaying convergence. To mitigate these issues, various improvements have been developed, including the use of momentum (or its more effective variant, Nesterov Accelerated Gradient (NESTEROV, 1983; SUTSKEVER *et al.*, 2013)), learning rate schedules, and adaptive gradient methods such as RMSProp, Adam (KINGMA; BA, 2015), and its regularized variant AdamW (LOSHCHILOV; HUTTER, 2019). These approaches aim to stabilize training, accelerate convergence, and improve generalization across different deep learning applications.

2.4.1.7.2 Momentum

The Momentum method augments SGD by introducing a concept analogous to inertia in physics. Rather than using only the current gradient to update the parameters, it accumulates a running average of past gradients, thereby smoothing the trajectory of updates. The momentum update rule introduces a velocity vector v_t, which is an exponentially weighted average of past gradients:

$$v_t = \beta v_{t-1} + (1-\beta)\nabla_\theta \mathscr{L}(\theta) \tag{2.27}$$

$$\theta \leftarrow \theta - \eta v_t \tag{2.28}$$

where

- $\beta \in [0,1)$ is the momentum coefficient, typically set between 0.9 and 0.99,
- v_t accumulates historical gradients, favoring directions with consistent updates.

Momentum improves the efficiency of gradient descent by incorporating past gra-

dients into the current update, thereby smoothing the optimization trajectory. Specifically, it helps suppress oscillations in narrow valleys of the loss landscape by damping updates that are perpendicular to the direction of descent, and it accelerates convergence along long, shallow slopes where gradients remain consistent. A well-known extension of this idea is Nesterov Accelerated Gradient (NAG), which enhances the basic momentum method by computing the gradient not at the current position, but at a projected future position based on the accumulated momentum. This anticipatory approach leads to more informed and responsive updates, often resulting in faster and more stable convergence. NAG was originally proposed in the context of convex optimization (NESTEROV, 1983), and its effectiveness in training deep neural networks was later demonstrated in (SUTSKEVER *et al.*, 2013).

2.4.1.7.3 Adam (Adaptive Moment Estimation)

Adam (KINGMA; BA, 2015) is one of the most widely used optimization algorithms in deep learning, renowned for its robustness, ease of tuning, and strong empirical performance across a broad range of tasks. It integrates key ideas from both Momentum and RMSProp by maintaining per-parameter learning rates that are adapted based on estimates of the first and second moments of the gradients. Specifically, Adam keeps track of two exponentially decaying averages at each iteration: m_t, the first moment estimate (i.e., the mean of the gradients), and v_t, the second moment estimate (i.e., the uncentered variance or squared gradients). These moving averages enable the optimizer to adjust the magnitude of each update dynamically, allowing for more stable and efficient learning, particularly in the presence of sparse or noisy gradients. The resulting updates are bias-corrected to compensate for the initialization of the moving averages at zero, ensuring proper scaling early in training. The combination of adaptive step sizes and momentum makes Adam well-suited for a wide variety of deep learning applications and has established it as a default choice in many practical settings.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathscr{L}(\theta) \tag{2.29}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla_\theta \mathscr{L}(\theta)\right)^2 \tag{2.30}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.31}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2.32}$$

$$\theta \leftarrow \theta - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{2.33}$$

where

- $\beta_1, \beta_2 \in [0, 1)$ are decay rates for the first and second moments (typically $\beta_1 = 0.9$, $\beta_2 = 0.999$),
- $\hat{m}_t, \hat{v}_t$ are bias-corrected estimates,
- $\varepsilon \approx 10^{-8}$ is a small constant for numerical stability.

This method offers several advantages that make it a popular choice in deep learning. One of its key strengths is adaptivity: it adjusts the learning rate individually for each parameter based on the estimated uncertainty derived from the magnitude of past gradients. This allows parameters with high variance to receive smaller updates, while more stable parameters can update more aggressively. Adam is also known for its robustness, often performing well across a wide range of hyperparameter choices with minimal tuning. Additionally, it is particularly effective for sparse gradients, making it well-suited for applications such as natural language processing, where many parameters receive infrequent or small updates. However, despite these advantages, Adam has been observed to sometimes yield worse generalization performance than SGD with momentum, especially in computer vision tasks like image classification. To address this, variants such as AdamW (LOSHCHILOV; HUTTER, 2019) have been introduced. AdamW decouples weight decay from the gradient-based parameter updates, providing better regularization and improving generalization in large-scale deep learning settings.

2.4.1.8 Generalization, Overfitting, and the Bias–Variance Tradeoff

In supervised learning, the goal is not only to fit a model to training data but also to ensure that it generalizes well to unseen data. This introduces the challenge of balancing underfitting and overfitting.

- **Underfiting:** A model is said to underfit when it is too simplistic to capture the underlying structure of the data. Underfitting typically results in both high training and test error. It often occurs when the model lacks capacity, such as in linear models applied to non-linear data, or when training is stopped prematurely.
- **Overfitting:** Overfitting happens when a model fits the training data too well, including its noise and fluctuations, failing to generalize to new data. This results in low training error but high test error. Overfitting is exacerbated in high-capacity models (e.g., deep networks) and small datasets, especially in the absence of regularization.

2.4.1.8.1 Training, Validation, and Test Errors

The performance of a learning algorithm is typically assessed using three distinct datasets, each serving a specific role in the development and evaluation process. The training set is used to compute gradients and iteratively update the model parameters via optimization algorithms such as SGD or Adam. The validation set is reserved for tuning hyperparameters, such as learning rate, regularization strength, or architecture size, and for monitoring how well the model generalizes during training. Finally, the test set is held out entirely during model development and used only for final evaluation to simulate performance on truly unseen data.

An effective model should exhibit low error on both the training and validation sets, indicating it has captured meaningful patterns without overfitting. When there is a significant gap between the training and validation errors, particularly if the training error is low and the validation error is high,it suggests that the model is overfitting to the training data. This behavior is commonly diagnosed by plotting training and validation losses across epochs. In a well-generalizing model, both curves decrease and stabilize in tandem; in contrast, a divergence between them signals that the model is memorizing rather than learning, which undermines its predictive power on new data.

2.4.1.8.2 Bias–Variance Decomposition

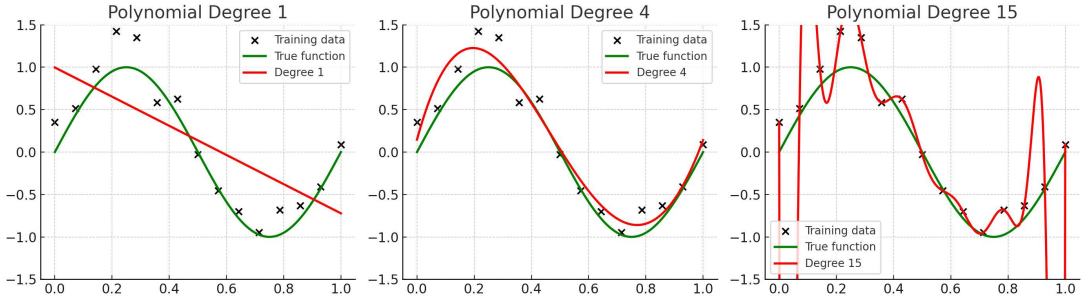
Let $y = f(x) + \varepsilon$ be the true data-generating process, where $f(x)$ is the underlying function and ε is zero-mean noise with variance σ^2.

The expected prediction error of a learning algorithm at a point x is:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \underbrace{(\mathbb{E}[\hat{f}(x)] - f(x))^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible error}} \tag{2.34}$$

where

- $\hat{f}(x)$ is the model's prediction.
- $\mathbb{E}[\cdot]$ denotes expectation taken over different training sets.
- Bias^2 measures the error introduced by approximating the true function $f(x)$ with a simplified model.
- Variance quantifies how much $\hat{f}(x)$ fluctuates across different training datasets.
- σ^2 is the irreducible error caused by noise in the data, which cannot be eliminated by any model.



Source: Image from the author (2025)

Figure 7 – Bias–Variance Trade-off illustrated through polynomial regression. Degree 1 underfits the data (high bias), degree 15 overfits the noise (high variance), while degree 4 achieves a good trade-off, capturing the underlying pattern with better generalization.

To illustrate the bias–variance trade-off, consider fitting polynomials of varying degrees to noisy samples of the function $f(x) = \sin(2\pi x)$. A low-degree polynomial (e.g., degree 1) underfits the data, failing to capture the underlying structure and resulting in high bias. Conversely, a high-degree polynomial (e.g., degree 15) fits the training data too closely, including noise, which leads to high variance and poor generalization. A moderately complex model (e.g., degree 4) achieves a better balance, capturing the essential pattern without overfitting. This trade-off is visually demonstrated in Figure 7, which shows how model complexity affects the fit and generalization behavior.

2.4.1.8.3 Strategies to Improve Generalization and Training

To combat overfitting, enhance generalization in deep learning models and accelerate the training, especially in settings with limited data or high variance, several complementary strategies can be employed:

- **Regularization:** Penalizing large weights via L1 (lasso) or L2 (ridge) regularization helps to constrain the model capacity. Dropout (SRIVASTAVA *et al.*, 2014) stochastically disables neurons during training, reducing co-adaptation. Data augmentation synthetically expands the training distribution to improve robustness.
- **Early Stopping:** Monitoring validation loss during training and halting optimization when performance stagnates helps prevent overfitting.
- **Cross-Validation:** Partitioning the dataset into multiple training and validation folds allows for a more reliable estimate of generalization error and hyperparameter selection.
- **Ensemble Methods:** Aggregating predictions from multiple models (e.g., bagging, boosting) reduces variance and improves predictive accuracy by leveraging model diversity.

- **Normalization Techniques:** Normalization mitigates internal covariate shift and accelerates convergence. Several techniques are widely adopted:
  - **Batch Normalization (BN)** (IOFFE; SZEGEDY, 2015): Normalizes feature activations across the mini-batch and is effective in MLPs and CNNs.
  - **Layer Normalization (LN)** (BA *et al.*, 2016): Normalizes across feature dimensions per sample and is better suited for recurrent and transformer-based architectures.
  - **Group Normalization (GN)** (WU; HE, 2018): Divides features into groups and normalizes within each, making it effective even with small batch sizes.
- **Graph-Specific Normalization:** For Graph Neural Networks (GNNs), specialized normalization schemes address the irregular structure and over-smoothing issues:
  - **PairNorm** (ZHAO; AKOGLU, 2020): Preserves pairwise distances to mitigate over-smoothing in deep GNNs.
  - **GraphNorm** (CAI *et al.*, 2021): Normalizes features across all nodes within a graph, stabilizing training in multi-graph datasets.
  - **NodeNorm and EdgeNorm** (ZHOU *et al.*, 2020b): Normalize features within local node or edge neighborhoods, enhancing learning stability in heterogeneous or dynamic graphs.

2.4.2 Deep Neural Architectures

Modern deep learning is powered by a diverse array of neural architectures, each designed to handle specific types of data and to exploit particular structural or statistical properties, known as inductive biases. These biases guide the learning process by embedding assumptions about the nature of the data directly into the network architecture. For instance, Multi-Layer Perceptrons (MLPs) assume no spatial or temporal structure and are therefore fully connected and permutation-invariant. They serve as the most general-purpose model and are mathematically simple yet powerful universal approximators. However, MLPs do not scale efficiently to high-dimensional inputs such as images or sequences without incorporating additional structure.

To overcome these limitations, more specialized architectures have been developed. Convolutional Neural Networks (CNNs) introduce spatial locality and weight sharing as inductive biases, making them highly effective for grid-like data such as images or audio spectrograms. CNNs exploit the hierarchical nature of patterns such as edges, textures, and shapes by stacking convolutional layers that progressively capture increasingly abstract features. Their translation

invariance and parameter efficiency have led to widespread success in computer vision and beyond.

For sequential data, such as time series, speech, or natural language, Recurrent Neural Networks (RNNs) are more appropriate. RNNs process data one element at a time and maintain a hidden state that carries temporal information from previous steps, allowing them to capture dependencies across time. Despite their success, classical RNNs suffer from vanishing and exploding gradients, limiting their ability to model long-term dependencies. Solutions such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) were introduced to mitigate these issues, introducing gating mechanisms that selectively retain or discard information. These advances extended the applicability of RNNs to complex language modeling, machine translation, and sequential decision-making.

These core models MLPs, CNNs, and RNNs, serve as the architectural backbone of deep learning. They form the conceptual and mathematical foundation for more advanced paradigms like Graph Neural Networks (GNNs), which generalize convolution and message-passing operations to arbitrary graph-structured data, and Transformers, which replace recurrence with self-attention mechanisms to capture global context more efficiently. Understanding the mathematical principles and design motivations behind these fundamental architectures is essential for appreciating how deep learning adapts to various data modalities and computational challenges.

2.4.2.1 Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron is the canonical example of a feedforward neural network. It consists of a sequence of fully connected layers, each applying an affine transformation followed by a nonlinear activation function.

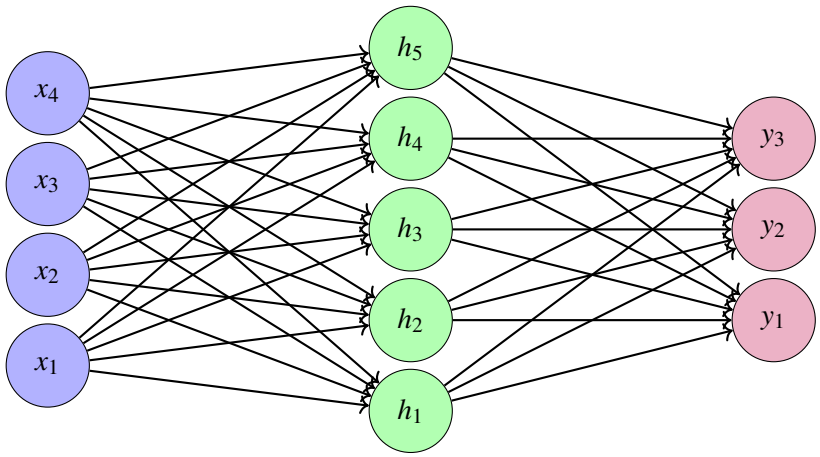
Figure 8 illustrates the MLP structure with one hidden layer. Each neuron in a given layer is fully connected to all neurons in the previous layer, reflecting the dense connectivity pattern that defines this architecture. Formally, given an input vector $x \in \mathbb{R}^{n_0}$, the forward pass through an MLP with L layers is computed recursively as:

$$h^{(l)} = \sigma\left(W^{(l)} h^{(l-1)} + b^{(l)}\right), \quad l = 1, \ldots, L \tag{2.35}$$

with $h^{(0)} = x$, where

- $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix for layer l,

- $b^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector,
- $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU, sigmoid),
- $h^{(l)} \in \mathbb{R}^{n_l}$ is the output (activation) at layer l.



Source: Image from the author (2025)

Figure 8 – Multi-Layer Perceptron (MLP) with input x_i, hidden activations h_j, and outputs y_k. Each neuron is fully connected to the previous layer.

MLPs have no inherent notion of spatial locality or sequential structure: every neuron in a layer is connected to all neurons in the previous layer. This makes MLPs general-purpose approximators (HORNIK *et al.*, 1989), but inefficient or ineffective when dealing with data that exhibits topological or geometric structure, such as images, sequences, or graphs.

Despite their simplicity, MLPs have seen renewed interest through architectural variations such as residual MLPs (TOLSTIKHIN *et al.*, 2021) and Kolmogorov–Arnold Networks (LIU *et al.*, 2025). These models have shown that, under proper initialization, normalization, and regularization, pure MLPs can match or even outperform more structured models in certain settings. However, standard MLPs remain limited in their ability to exploit locality or hierarchical features without architectural modifications or external preprocessing.

2.4.2.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks were developed to address the limitations of MLPs when processing structured data like images, audio signals, and spatially correlated inputs (LECUN *et al.*, 1998). They incorporate three critical inductive biases: **locality**, **weight sharing**, and **spatial hierarchy**.

The core operation of a CNN layer is the convolution:

$$h^{(l)} = \sigma(K^{(l)} * h^{(l-1)} + b^{(l)}) \tag{2.36}$$

Here:

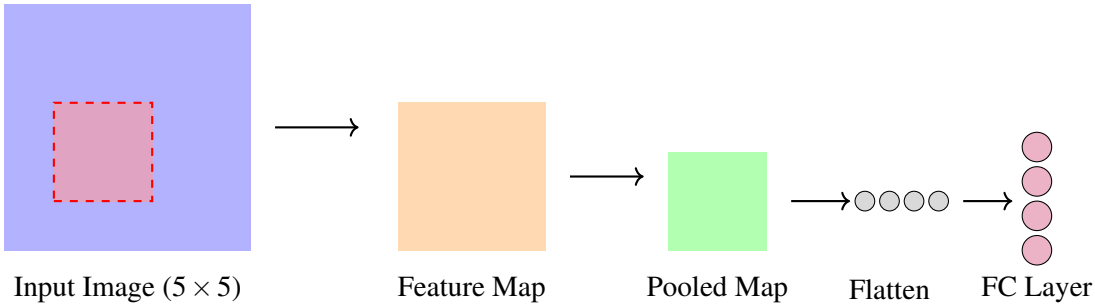
- $K^{(l)}$ denotes a set of convolutional filters (kernels) with spatially limited receptive fields,
- $*$ represents the convolution operation (e.g., 2D or 1D convolution),
- $h^{(l-1)}$ is the input feature map, and
- σ is an activation function (typically ReLU).

In the 2D case, for an input feature map $X \in \mathbb{R}^{H \times W}$ and a kernel $K \in \mathbb{R}^{m \times n}$, the convolution is defined as:

$$(K * X)[i, j] = \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} K[u, v] \cdot X[i-u, j-v] \tag{2.37}$$

where

- $X[i, j]$ is the input value at spatial position (i, j),
- $K[u, v]$ is the kernel weight at offset (u, v),
- The operation slides the kernel over the input to compute local weighted sums.



Source: Image from the author (2025)

Figure 9 – CNN pipeline with flattening: input is convolved into a feature map, pooled, flattened into a vector, and passed to a fully connected layer.

This operation applies the same filter across spatial locations, effectively capturing local patterns regardless of their absolute position. Pooling operations such as max pooling or average pooling are used to reduce spatial dimensions and introduce invariance to small translations. A typical CNN pipeline, illustrated in Figure 9, consists of:

- **Convolution layers:** Extract local features via learned kernels.
- **Activation functions:** Apply nonlinearity, usually ReLU.
- **Pooling layers:** Downsample spatial dimensions, capturing invariance.

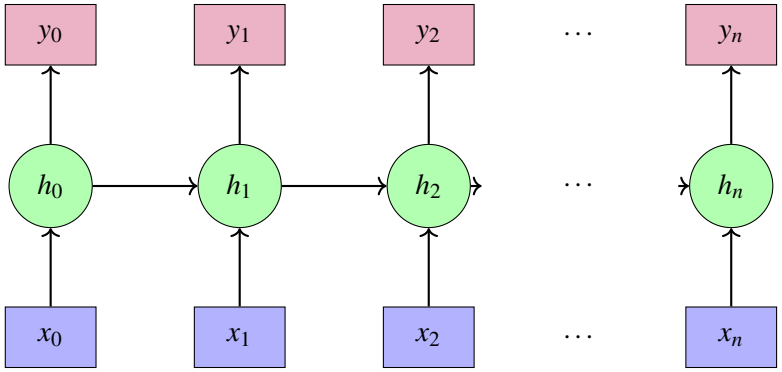
- **Flattening:** Convert 2D feature maps into 1D vectors.
- **Fully-connected layers:** Act as final decision stages (classification, regression).

CNNs have proven remarkably effective for vision tasks, leading to milestone architectures such as LeNet-5 (LECUN *et al.*, 1998), AlexNet (KRIZHEVSKY *et al.*, 2017), VGG (SIMONYAN; ZISSERMAN, 2014), ResNet (HE *et al.*, 2015a), and EfficientNet (TAN; LE, 2019). Their capacity to extract multi-scale hierarchical features makes them suitable for applications ranging from medical imaging to self-driving cars.

However, CNNs assume **Euclidean grid topology**, which restricts their generalizability to irregular domains such as graphs or meshes. This limitation motivated the development of generalized convolution operations in non-Euclidean spaces, such as those used in graph convolutional networks (GCNs) (KIPF; WELLING, 2016) and mesh-based CNNs (HANOCKA *et al.*, 2019).

2.4.2.3 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a class of neural architectures designed to model sequential and temporal data. Unlike feedforward networks, RNNs maintain a hidden state that evolves over time, allowing the network to retain memory of past inputs. This makes RNNs especially suitable for tasks such as time series prediction, natural language modeling, and speech recognition (ELMAN, 1990).



Source: Image from the author (2025)

Figure 10 – Unrolled Recurrent Neural Network (RNN) across time steps. The hidden state h_t is updated recurrently based on the previous state h_{t-1} and current input x_t, and optionally used to produce an output y_t.

The core idea is to apply the same transformation at each time step t, as we can see at Figure 10, updating a hidden state $h_t \in \mathbb{R}^d$ based on the current input $x_t \in \mathbb{R}^n$ and the previous

hidden state h_{t-1}:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{2.38}$$

where

- $W_h \in \mathbb{R}^{d \times d}$ is the recurrent weight matrix,
- $W_x \in \mathbb{R}^{d \times n}$ is the input weight matrix,
- $b \in \mathbb{R}^d$ is the bias vector,
- σ is a non-linear activation function, typically `tanh` or `ReLU`.

At each step the output can optionally be computed from the hidden state via an output transformation $y_t = W_y h_t + b_y$. The parameters (W_x, W_h, W_y, b, b_y) are shared across all time steps. Despite their conceptual simplicity, RNNs suffer from significant limitations during training due to the vanishing and exploding gradient problems (BENGIO *et al.*, 1994). As gradients are backpropagated through many time steps, they may shrink or explode exponentially, making it difficult to learn long-range dependencies.

To address these issues, gated architectures such as the Long Short-Term Memory (LSTM) (HOCHREITER; SCHMIDHUBER, 1997) and Gated Recurrent Unit (GRU) (CHO *et al.*, 2014) were introduced. These models use gating mechanisms to control the flow of information:

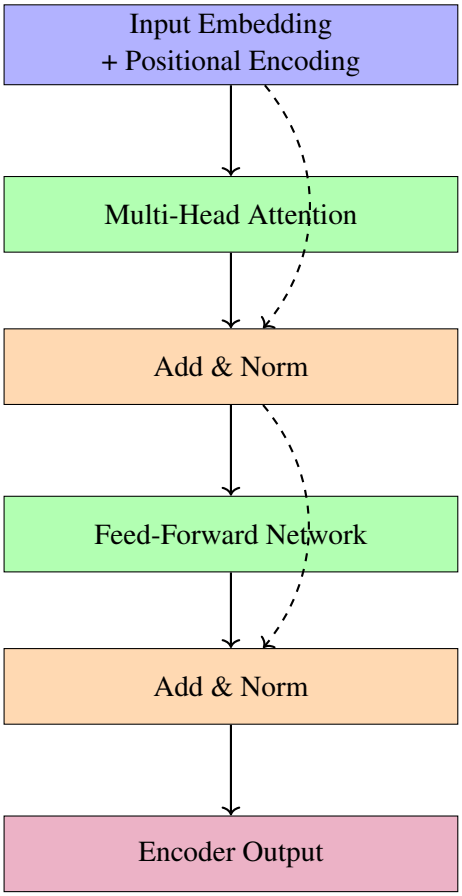
- **LSTM:** Introduces cell states and gates (input, forget, and output) that allow the model to preserve information over long sequences.
- **GRU:** A simplified variant with fewer gates (reset and update), providing similar performance with lower computational cost.

These advancements significantly improved the modeling of sequential data and became foundational in early sequence-to-sequence models, particularly in machine translation and speech synthesis.

2.4.2.4 Transformer

The Transformer architecture (VASWANI *et al.*, 2017) revolutionized sequence modeling by completely removing recurrence and convolutions. Instead, it is built entirely on a mechanism known as **self-attention**, which allows each token in the input sequence to attend to all other tokens simultaneously. This design enables highly parallelizable computation and the modeling of long-range dependencies, two properties that have made Transformers the

foundation of modern large language models (LLMs), such as BERT (DEVLIN *et al.*, 2018) and GPT (RADFORD *et al.*, 2019).



Source: Image from the author (2025)

Figure 11 – Transformer Encoder Block with residual (dashed) connections added before each Add & Norm step. These help stabilize training and preserve input information.

Figure 11 illustrates a standard Transformer encoder block. The input tokens are first embedded and combined with **positional encodings** to preserve the sequential order, crucial for models that are otherwise permutation-invariant. These encodings can be either sinusoidal, as originally proposed (VASWANI *et al.*, 2017), or learned embeddings (GEHRING *et al.*, 2017).

The enriched input is then passed through the first core component: the **Multi-Head Self-Attention** mechanism (blue block in the figure). This operation computes dependencies between all token pairs using the *scaled dot-product attention* formula:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.39}$$

Here, Q, K, and V are the query, key, and value matrices derived from the input sequence. The attention scores are computed by taking the dot product between queries and keys, scaled by the dimensionality d_k, and then applied to weight the values accordingly.

To allow the model to jointly capture information from different representation subspaces, the Transformer uses **multi-head attention**, which consists of multiple parallel attention layers (or "heads"):

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O \tag{2.40}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.41}$$

Each head has its own learned projections W_i^Q, W_i^K, W_i^V, and their outputs are concatenated and linearly transformed via W^O, a learnable linear projection matrix. Following the attention mechanism, the architecture includes an **Add & Norm** sublayer, which performs two key functions:

- **Residual Connection:** adds the input of the sublayer to its output, helping gradient flow and preserving low-level features.
- **Layer Normalization:** standardizes the result to stabilize training.

This operation is shown in Figure 11 as a dashed skip connection feeding directly into the Add & Norm block. The next component is a **position-wise Feed-Forward Network**, which applies the same two-layer MLP independently to each token:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

This increases the representational capacity of the model by transforming each token's features through a non-linear projection. It is followed again by an Add & Norm block that applies the same residual and normalization steps. The output of this encoder block can either be passed to the next encoder layer in a stack or, in a sequence-to-sequence setting, sent to a decoder block for autoregressive generation or translation.

Each of these sublayers multi-head attention, feed-forward network, normalization, and residual connections plays a critical role in the Transformer's ability to model long-range dependencies efficiently and at scale. Due this formulation, they are now the de facto architecture in natural language processing and beyond, some of the key milestone includes:

- **BERT** (DEVLIN *et al.*, 2018): A bi-directional encoder trained via masked language modeling.
- **GPT** (RADFORD *et al.*, 2019): An autoregressive decoder trained for text generation.
- **T5** (RAFFEL *et al.*, 2020), **LLama** (TOUVRON *et al.*, 2023), **GPT-4**: Recent LLMs built entirely on Transformer backbones.

Transformers have also expanded into vision as (ViT) Visual Transformer (DOSOVITSKIY *et al.*, 2020), audio, and graph data, making them one of the most general and successful neural architectures in the deep learning era.

2.4.2.5 Graph Neural Network

Graph Neural Networks (GNNs) are a class of deep learning architectures specifically designed to operate on graph-structured data (YE *et al.*, 2022). Unlike traditional neural networks that process inputs with fixed structures (such as sequences or grids), GNNs are capable of modeling arbitrary relational structures, where entities are represented as nodes and their interactions as edges. These models integrate both node features and graph topology to learn rich and expressive representations, supporting tasks such as node classification, link prediction, and graph-level classification (ZHOU *et al.*, 2020a).

At the core of GNNs lies the principle of information propagation across the graph, whereby node embeddings are iteratively updated by aggregating features from their neighbors. This process is typically implemented through a message passing scheme (GILMER *et al.*, 2017), which defines how information is exchanged and combined over the graph structure.

While recent advances such as Graph Transformers (KREUZER *et al.*, 2021; YING *et al.*, 2021) have demonstrated strong performance in several tasks by incorporating global attention mechanisms, they come with a significantly higher computational cost, particularly in terms of memory and runtime complexity. Due to the need for scalability in large industrial graphs, such as those derived from FEM simulations, this work focuses on message passing-based GNNs, which offer a favorable trade-off between expressiveness and efficiency.

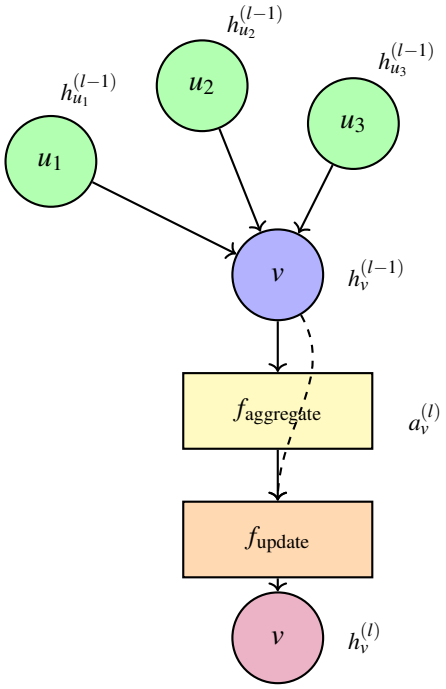
2.4.2.5.1 Message Passing

Let $G = (V, E)$ be a graph with a node features matrix $H \in \mathbb{R}^{N \times F}$. Where, N is the number of nodes and F is the number of features. Here, V and E represent the vertices and edges, and $\mathcal{N}(v)$ denotes the neighbors of a node v. A l-th message passing layer can be described by the following equations:

$$a_v^{(l)} = f_{\text{aggregate}}^{(l)}(\{h_u^{(l-1)} : \forall u \in \mathcal{N}(v)\}) \tag{2.42}$$

$$h_v^{(l)} = f_{\text{update}}^{(l)}(h_v^{(l-1)}, a_v^{(l)}) \tag{2.43}$$

$$H^{(l)} = \left[(h_1^{(l)})^T \quad (h_2^{(l)})^T \quad \cdots \quad (h_n^{(l)})^T\right]^T \tag{2.44}$$



Source: Image from the author (2025)

Figure 12 – Message passing in a GNN: Neighbor features $h_u^{(l-1)}$ are aggregated to form $a_v^{(l)}$, then combined with $h_v^{(l-1)}$ to produce $h_v^{(l)}$.

The function $f_{\text{aggregate}}$ takes the features of the neighbors of a node v and aggregates them to obtain a representation of the entire neighborhood. This process is illustrated in Figure 12, where each neighboring node u_i sends its representation $h_{u_i}^{(l-1)}$ to the central node v. These messages are combined into an aggregated vector $a_v^{(l)}$ via $f_{\text{aggregate}}$, which is then merged with the node's own previous state $h_v^{(l-1)}$ using f_{update} to compute the new representation $h_v^{(l)}$.

It is worth noting that multiple levels of depth can be considered in the neighborhood by stacking GNN layers, allowing information to propagate from distant nodes. The functions $f_{\text{aggregate}}$ and f_{update} can take various forms, such as summation, averaging, attention mechanisms, MLPs (multi-layer perceptrons), or RNNs (Recurrent Neural Networks). However, their design choices are directly linked to the model's expressiveness and generalization capacity, as discussed in (XU *et al.*, 2019). The primary goal of this message passing framework is to learn local filters that enable either inductive or transductive learning over graphs.

2.4.2.5.2 Graph Convolutional Network

Convolutional neural networks (CNNs) are highly effective in image processing due to their ability to leverage the regular structure inherent in images. However, graphs, in general, do not possess this regular structure, which necessitates a more generalized convolution

operation. In line with the concept introduced in (KIPF; WELLING, 2016), let g_Θ be a filter parameterized by $\Theta \in \mathbb{R}^{H_{in} \times H_{out}}$ in the Fourier domain. By employing the inverse Fourier transform and decomposing it using truncated Chebyshev polynomials to the second order, we obtain the generalized graph convolution:

$$g_\Theta * H^{(l-1)} \approx (I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) H^{(l-1)} \Theta. \tag{2.45}$$

As $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ has eigenvalues within the range [0, 2], it is prone to vanishing or exploding gradients. To alleviate this, the author proposes a renormalization step:

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \tag{2.46}$$

where $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

In the case where g_Θ is a single layer perceptron, $f_{\text{aggregate}}$ is a simple mean aggregator defined as a local spectral convolution and f_{update} is the composition $\sigma \circ g_\Theta$. The equation for updating the features becomes:

$$H^{(l)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l-1)} W^{(l-1)}) \tag{2.47}$$

Where σ is a non-linear activation function, W represents the neural network used to represent the filters, and H denotes the features.

2.4.2.5.3 GraphSAGE

First introduced in (HAMILTON *et al.*, 2017), GraphSAGE aims to generate embeddings by sampling and aggregating features from a node's local neighborhood. It differentiates itself from GCN as an inductive method, while the original GCN is transductive. In other words, GraphSAGE learns a function responsible for creating embeddings that can be applied to a newly added node in the graph, whereas GCNs learn to embed existing nodes, which cannot be directly applied to a newly added node. Following the approach outlined in the aforementioned paper, the function $f_{\text{aggregate}}$ takes a fixed-size sample with repetition from the neighbors of node v and performs an operation using a single layer perceptron, LSTM, mean, or max. On the other hand, f_{update} is a composition of σ and a structure similar to possible options for f_{update}. For instance, the author found that combining mean and LSTM yields excellent performance, leading to the

following rule for updating node features:

$$a_v^{(l)} = \text{mean}(\{h_u^{(l-1)} : \forall u \in \mathcal{N}(v)\}) \tag{2.48}$$

$$h_v^{(l)} = \sigma(\text{LSTM}(\text{concatenate}(h_v^{(l-1)}, a_v^{(l)}))) \tag{2.49}$$

$$H^{(l)} = \left[(h_1^{(l)})^T \quad (h_2^{(l)})^T \quad \cdots \quad (h_n^{(l)})^T\right]^T \tag{2.50}$$

To manage computational costs, the author employs a fixed-size sample strategy from the neighbors, such as selecting a sample of size 3 from $\mathcal{N}(v)$.

2.4.2.5.4 Graph Attention Network

In the paper (VELIČKOVIĆ *et al.*, 2018), the authors propose a novel structure for message passing using the attention mechanism, which has gained significant prominence, particularly with the use of transformers in natural language processing, following the publication of (VASWANI *et al.*, 2017). In the context of NLP, this mechanism is employed to focus on specific portions of a sequence, capturing dependencies between terms. It assigns weights or scores to each element, indicating their importance in the context of the input being used. Furthermore, it enables the model to handle inputs of varying sizes, thus reducing the limitations of fixed-length representations. In the case of graphs, it operates similarly. For instance, given a node v with features h, it allows us to concentrate on the most relevant elements of the neighborhood and utilize them to update h. Consequently, this mechanism facilitates better generalization of the model to unseen graphs during training, making it valuable for inductive tasks like GraphSAGE. Using the structure proposed in (VELIČKOVIĆ *et al.*, 2018), an attention layer can be defined as follows. First, we define a linear transformation of the features using a matrix $W \in \mathbb{R}^{F^{(l-1)} \times F^l}$ where $F^{(l)}$ represents the number of features in $H^{(l)}$. Next, we establish an attention mechanism shared by the function att: $\mathbb{R}^{F^{(l-1)}} \times \mathbb{R}^{F^{(l-1)}} \to \mathbb{R}$ which could be a single-layer perceptron. As these coefficients represent the degrees of importance for each node, we can use a LeakyReLU activation function to introduce non-linearity and normalize them using softmax. Thus, we have:

$$\alpha_{v,u} = \text{softmax}_u(\text{leakyReLU}(\text{att}(W^{(l)} h_v^{(l-1)}, W h_u^{(l-1)}))) \tag{2.51}$$

With this, the self-attention mechanism can be expressed as follows:

$$a_v^{(l)} = \sum_{u \in N(v)} \alpha_{v,u} W^{(l)} h_u^{(l-1)} \tag{2.52}$$

$$h_v^{(l)} = \sigma(\text{sum}(\alpha_{v,v} W^{(l)} h_u^{(l-1)}, a_v^{(l)})) \tag{2.53}$$

$$H^{(l)} = \left[(h_1^{(l)})^T \quad (h_2^{(l)})^T \quad \cdots \quad (h_n^{(l)})^T \right]^T \tag{2.54}$$

Following the same idea as in (VASWANI *et al.*, 2017), the multi-head attention of size K is defined by multiple applications of attention and concatenation at the end. For $k = 1, ..., K$:

$$h_v^{(k,l)} = \sigma(\text{sum}(\alpha_{v,v} W^{(k,l)} h_u^{(l-1)}, a_v^{(k,l)})) \tag{2.55}$$

$$h_v^{(l)} = \text{concatenate}_k(h_v^{(k,l)}) \tag{2.56}$$

In classification tasks, when performing the multi-head attention in the last layer, we can replace concatenation by mean and apply the activation function afterwards.

2.4.2.5.5 Graph Isomorphism Network

The paper (XU *et al.*, 2019) demonstrated that the generalization capacity of certain structures is limited. Surprisingly, two different graphs can result in the same embedding, which should only occur if the graphs are topologically isomorphic and have the same features. This issue arises because the functions $f_{\text{aggregate}}$ and f_{update} are not injective in certain architectures, such as GCNs and GraphSAGE. The author sought to address this problem using the Weisfeiler-Lehman test, which checks for graph isomorphism. To accomplish this, the author showed that an architecture following the following rule has the same expressive power as the WL test when it comes to distinguishing graphs. In other words, a GNN with the following architecture is as powerful as the 1-WL test:

$$h_v^{(l)} = \phi^{(l)}((1 + \varepsilon^{(l)}) f^{(l)}(h_v^{(l-1)}) + \sum_{u \in N(v)} f^{(l)}(h_u^{(l-1)})) \tag{2.57}$$

Here, ε is a fixed parameter that can also be learned, and ϕ and f are injective functions. Setting $\varepsilon = 0$ and using an MLP to learn the composition $\phi \circ f$, we obtain:

$$h_v^{(l)} = \text{MLP}^{(l)}(\sum_{u \in v \cup N} h_u^{(l-1)}) \tag{2.58}$$

$$H^{(l)} = \left[(h_1^{(l)})^T \quad (h_2^{(l)})^T \quad \cdots \quad (h_n^{(l)})^T \right]^T \tag{2.59}$$

This structure is justified in the paper. In summary, the use of an MLP is due to its universal approximation capability, allowing it to learn any injective function. Furthermore, the use of summation as the aggregation function is because it can capture maximum information from the neighborhood according to the theory of multisets. Lastly, the author shows that due to the utilization of only a single layer perceptron and aggregators like mean and max, GCNs and GraphSAGE do not generalize well for certain types of graphs.

2.4.2.5.6 Expressivity and Complexity of GNNs

The expressiveness of Graph Neural Networks (GNNs) plays a central role in determining their ability to capture and distinguish the underlying structure of graph data. This expressiveness is theoretically bounded by the Weisfeiler-Lehman (WL) test, a widely adopted heuristic for graph isomorphism. Xu et al. (XU *et al.*, 2019) showed that common architectures such as GCNs and GATs are no more powerful than the 1-WL test, which iteratively updates node features based on neighborhood aggregation. In response to this limitation, the Graph Isomorphism Network (GIN) was proposed to match the full discriminative power of 1-WL, thereby offering improved performance in tasks that require detailed structural awareness, such as partitioning dual graphs of FEM meshes.

In parallel to expressiveness, another critical dimension of GNN design lies in the choice between inductive and transductive learning paradigms. Inductive models, such as GraphSAGE and GIN, are built to generalize to unseen graphs and nodes, making them well-suited for scientific and industrial applications where new mesh instances are constantly encountered. Conversely, models like GCN and GAT are often deployed in transductive settings, where the graph structure remains fixed during training and inference, as in citation networks or semi-supervised classification tasks. Although adaptable to inductive use cases, these models are traditionally optimized for scenarios where the global graph context is available.

Table 1 – Comparison of selected GNN architectures.

Architecture	**Inductive**	**Transductive**	**Attention**	**Aggregation Type**
GCN (KIPF; WELLING, 2016)	Yes	Yes	No	Mean
GraphSAGE (HAMILTON *et al.*, 2017)	Yes	No	No	Mean / Pool / LSTM
GAT (VELIČKOVIĆ *et al.*, 2018)	Yes	Yes	Yes	Weighted sum
GIN (XU *et al.*, 2019)	Yes	No	No	Sum

Source: Table from the author (2025)

Another key practical factor is the memory and compute footprint of each layer,

especially when targeting very large graphs or real-time inference. Table 2 summarises the asymptotic space–time cost of four representative architectures under the standard sparse-adjacency assumption:

- **GCN / GIN** incur one sparse–dense multiply per layer $\big(\mathcal{O}(Ed)\big)$ plus a dense feature transform $\big(\mathcal{O}(Nd^2)\big)$. Hence they scale linearly with the number of edges and layers, but quadratically with the hidden width through the d^2 parameter term.
- **GraphSAGE** samples a fixed fan-out p neighbours per node and layer. With a mini-batch of b target nodes, both memory and compute depend on $b\,p^L$ rather than the full N or E, making it well suited to web-scale graphs or low-latency serving when p and L are modest.
- **GATv2** attaches a per-edge attention score for each of the h heads, yielding a cost of $\mathcal{O}\big(Lh\,(Ed+Nd^2)\big)$. The linear edge term is now multiplied by h, and the quadratic d^2 factor remains; wide layers or many heads therefore dominate the budget and can limit scalability.

In short, the computational bottleneck shifts from *edges* (GCN/GIN) to the *sampled fan-out* (GraphSAGE) or the *feature–head product* hd^2 (GATv2), emphasising the need to match the encoder to the deployment constraints.

Table 2 – Asymptotic space–time cost per forward pass.

Model	**Spatial Complexity**	**Time Complexity**
GCN (KIPF; WELLING, 2016)	$\mathcal{O}(Nd+E+Ld^2)$	$\mathcal{O}\big(L\,(Ed+Nd^2)\big)$
GraphSAGE (HAMILTON *et al.*, 2017)	$\mathcal{O}(b\,p^L d+Ld^2)$	$\mathcal{O}\big(Lb\,p^L(d+d^2)\big)$
GIN (XU *et al.*, 2019)	$\mathcal{O}(Nd+E+Ld^2)$	$\mathcal{O}\big(L\,(Ed+Nd^2)\big)$
GATv2 (BRODY *et al.*, 2021)	$\mathcal{O}(Nd+E+Lhd^2)$	$\mathcal{O}\big(Lh\,(Ed+Nd^2)\big)$

Source: Table from the author (2025)

Where L is the number of layers, b the batch size of the subgraph, p the fixed number of neighbors, h is the number of attention heads, $d = d_{in} = d_{out}$ is the embedding dimension, N the number of nodes, and E the number of edges.

2.4.3 Positional and Structural Encoder

2.4.3.1 Non-Learnable Positional Embeddings

A key challenge in graph-based models like GNNs is the absence of inherent positional information. Unlike sequences or images, graphs do not possess a natural order or spatial grid. **Positional encoders** aim to inject topological or distance-aware information into node

features, enabling GNNs to better exploit the structure of the graph.

2.4.3.1.1 Laplacian Eigenvectors

First introduced in (BELKIN; NIYOGI, 2003), the eigenvectors of the normalized Laplacian matrix can serve as powerful position-aware features. This approach generalizes the sinusoidal encodings used in transformers (VASWANI *et al.*, 2017), since text data can be represented as chain graphs, for which sine and cosine are eigenvectors (DWIVEDI *et al.*, 2023). The key property of this encoding is that it captures both local and global structural information: nodes close in the graph tend to have similar embeddings.
The symmetric normalized Laplacian is given by:

$$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}, \tag{2.60}$$

where A is the adjacency matrix and D the degree matrix. By computing the k smallest eigenvectors (excluding the trivial zero eigenvalue), we obtain a positional embedding $H_{pos} \in \mathbb{R}^{N \times k}$.

Despite the expressiveness of Laplacian embeddings, computing them can be expensive for large graphs. However, efficient approximations such as the Lanczos algorithm allow spectral decompositions in nearly linear time. One important drawback is signal ambiguity: each eigenvector can be flipped (sign ambiguity), leading to 2^k possible equivalent representations. In (LIM *et al.*, 2022), the authors proposed *SignNet*, a way to enforce sign-invariant architectures, mitigating this instability in training.

2.4.3.1.2 ProNE

Introduced in (ZHANG *et al.*, 2019), ProNE is a fast and scalable technique for positional encoding that improves upon traditional methods like DeepWalk (PEROZZI *et al.*, 2014), Node2Vec (GROVER; LESKOVEC, 2016), LINE (TANG *et al.*, 2015), and GraRep (CAO *et al.*, 2015). It operates in two stages: first, it performs a truncated singular value decomposition (SVD) on a proximity matrix M, followed by a spectral propagation phase.
The proximity matrix M is defined as:

$$M_{i,j} = \begin{cases} \ln p_{i,j} - \ln\left(\tau P_{E,j}\right) & ,(i,j) \in E \\ 0 & ,(i,j) \notin E \end{cases} \tag{2.61}$$

where $p_{i,j} = \frac{A_{i,j}}{D_{ii}}$, τ is the negative sample ratio, and $P_{E,j}$ represents negative samples for node j.

After performing SVD as $M \approx U_k \Delta_k V_k^\top$, the embedding matrix is computed as:

$$H_{pos} = U_k \Delta_k^{1/2}$$

Finally, a spectral propagation filter is applied to inject global information:

$$H_{pos} \leftarrow D^{-1}A(I_n - G)H_{pos} \tag{2.62}$$

The authors highlight that this post-processing step can improve embeddings from various methods.

2.4.3.1.3 GraRep

GraRep (CAO *et al.*, 2015) generates positional embeddings by capturing multi-hop relationships in the graph. Starting with the transition matrix $W = D^{-1}A$, it leverages powers of W to model paths of increasing lengths. For a given path length p, the proximity matrix is defined as:

$$\begin{aligned} \tilde{M}^p_{i,j} &= \log\left(\frac{W^p_{i,j}}{\sum_t W^p_{t,j}}\right) - \log(\tau) \\ M^p &= \max(\tilde{M}^p, 0) \end{aligned} \tag{2.63}$$

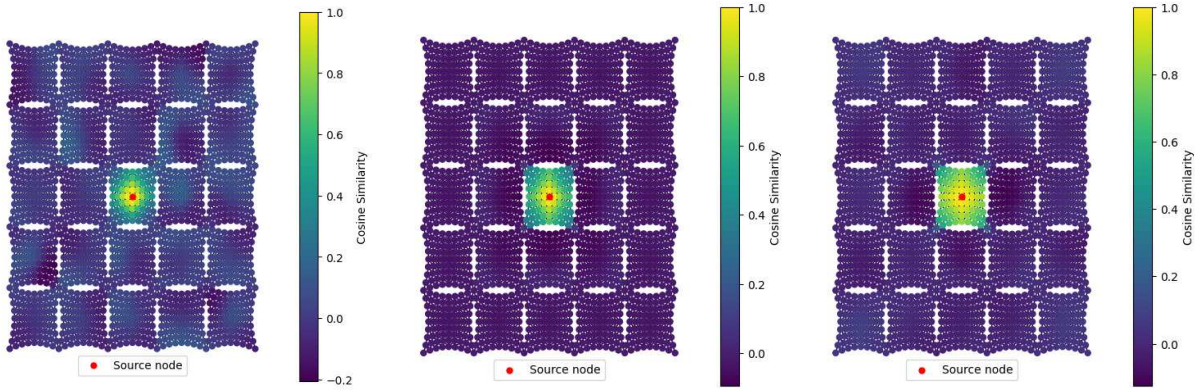
SVD is applied to each M^p yielding:

$$\tilde{H}^p_{pos} = U^p_k (\Delta^p_k)^{1/2}$$

and the final embedding is a concatenation across path lengths:

$$H_{pos} = \text{concat}_p(\tilde{H}^p_{pos})$$

To reduce the dimensionality of H_{pos}, an autoencoder is applied before feeding it to downstream models. Like ProNE, GraRep can also benefit from the spectral propagation defined in equation 2.62.



LaPE ProNE GraRep

Source: Image from the author (2025)

Figure 13 – Cosine similarity between node embeddings produced by each positional encoder. Higher similarity for nearby nodes and more separation for distant ones suggest that the encoder captures distance-aware structure.

2.4.3.2 Non Learnable Structural Embeddings

2.4.3.2.1 Random Walk

While positional encoders focus on embedding spatial or spectral positions of nodes, **structural encoders** aim to characterize the *role* or *function* of each node within the graph's topology. In other words, structural encoders distinguish nodes not by where they are, but by how they are connected, capturing patterns like centrality, clustering, or neighborhood distributions.

A widely adopted method in structural encoding is the use of **random walks**. These encoders compute node features based on probabilistic transitions over the graph, revealing how accessible a node is from others over paths of varying lengths. One of the most commonly used formulations is based on powers of the transition matrix $W = D^{-1}A$, where A is the adjacency matrix and D the degree matrix.

Two variations are frequently employed to extract node-level features:

1. **Diagonal encoding:** This method encodes the probability of returning to the same node after p steps of a random walk, where p is the desired embedding dimension. Specifically:

$$F_{struct} = \text{concat}_p(\text{diag}(W^p)), \tag{2.64}$$

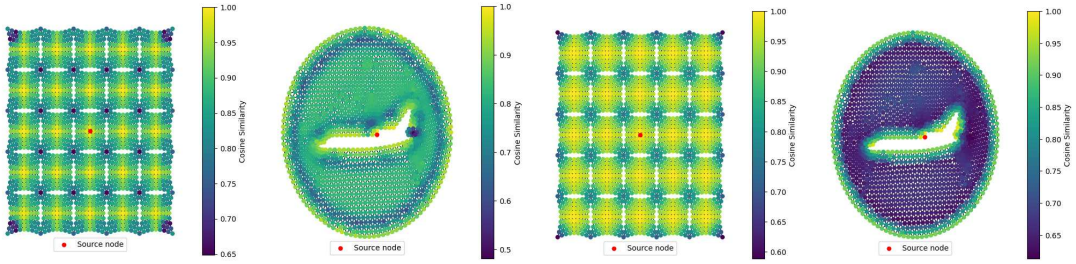
where each component of the feature vector reflects the self-return probability at various walk lengths.

2. **Row-sum encoding:** Instead of focusing on return probabilities, this approach encodes the overall accessibility of other nodes from node i through p-step walks, where p is the desired embedding dimension:

$$F_{struct} = \text{concat}_p \left(\sum_j W^p_{i,:} \right), \tag{2.65}$$

capturing broader diffusion-like properties of the graph structure.

These techniques have been successfully applied in works such as (DWIVEDI *et al.*, 2023; DWIVEDI *et al.*, 2022; LI *et al.*, 2020), where structural encodings are used either standalone or in combination with positional ones, providing GNNs with richer inductive biases.



RWSE Diagonal (Quad) | RWSE Diagonal (Falcon) | RWSE Sum (Quad) | RWSE Sum (Falcon)

Source: Image from the author (2025)

Figure 14 – Cosine similarity between structural embeddings extracted by RW-based encoders under different topologies. Diagonal encodings emphasize node identity, while sum-based encodings capture global accessibility.

2.4.3.3 Learnable Positional Embeddings

While non-learnable positional encodings such as Laplacian eigenvectors or proximity-based methods provide strong geometric priors, they often suffer from issues like ambiguity (e.g., sign flips, basis invariance) and lack task-adaptivity. Learnable positional encoders aim to overcome these limitations by allowing neural networks to extract and refine positional information directly from the graph structure in a data-driven manner. These methods not only improve adaptability across tasks and datasets but can also enhance generalization by learning invariances instead of relying on handcrafted positional features. Among the most notable contributions in this direction are SignNet and BasisNet (LIM *et al.*, 2022), which propose universal architectures that process Laplacian eigenvectors while respecting the intrinsic symmetries of the eigenspace.

2.4.3.3.1 SignNet and BasisNet

The work by (LIM *et al.*, 2022) introduces SignNet and BasisNet, neural network architectures specifically designed to address the inherent symmetries in eigenvectors when used for representation learning. Eigenvectors possess two main symmetries:

1. **Sign Flips:** If v is an eigenvector, then $-v$ is also an eigenvector with the same eigenvalue; this ambiguity has been noted in various fields (BRO *et al.*, 2008). This means for k eigenvectors, there are 2^k possible sign combinations, which can make learning challenging if not handled properly, often addressed by prior works with methods like random sign flipping (KREUZER *et al.*, 2021; DWIVEDI *et al.*, 2023).
2. **Basis Symmetries:** For eigenvalues with a multiplicity greater than one (i.e., higher-dimensional eigenspaces), the choice of basis eigenvectors is not unique: any orthogonal transformation of a valid basis results in another valid basis (OVSJANIKOV *et al.*, 2008; KRAFT; PROCESI, 1996). This is a significant issue, as the paper notes that such higher-dimensional eigenspaces are common in real-world datasets (e.g., 64% of molecule graphs in the ZINC dataset exhibit this property, as shown in Appendix C.2 of (LIM *et al.*, 2022) using data from (IRWIN *et al.*, 2012; DWIVEDI *et al.*, 2023)).

To tackle these issues, SignNet is proposed to create representations that are invariant to sign flips. A common strategy to achieve this, as presented in their work, is to process an eigenvector v_i using a shared function for both v_i and $-v_i$, and then combine the outputs, for example, as $\phi(v_i) + \phi(-v_i)$ (this core idea is from (LIM *et al.*, 2022), with potentially implemented by architectures like DeepSets (ZAHEER *et al.*, 2017)). This ensures that the output remains the same regardless of the input eigenvector's sign.

For the more general case of basis invariance in higher-dimensional eigenspaces, BasisNet is introduced by (LIM *et al.*, 2022). If V_i is an orthonormal basis for a d_i-dimensional eigenspace, any other basis for the same eigenspace can be written as $V_i Q_i$ where Q_i is an orthogonal matrix. BasisNet achieves invariance by operating on a quantity that is unaffected by this transformation, such as the orthogonal projector $\tilde{V}_i V_i^T$ (the use of VV^T is justified by invariant theory (KRAFT; PROCESI, 1996), and its properties as a projector are standard; the BasisNet architecture itself is from (LIM *et al.*, 2022), using Invariant Graph Networks (MARON *et al.*, 2019) for components).

The authors in (LIM *et al.*, 2022) prove that under certain conditions, these networks are universal, capable of approximating any continuous function of eigenvectors that respects

these invariances. When applied to Laplacian eigenvectors for graph representation learning, SignNet and BasisNet are shown to be more expressive than previous spectral methods. They can subsume spectral graph convolutions and generalize existing graph positional encodings, such as those based on heat kernels (e.g., (FELDMAN *et al.*, 2023)) and random walks (e.g., (DWIVEDI *et al.*, 2022)), as special cases (LIM *et al.*, 2022). The pipeline involves computing Laplacian eigenvectors, processing them through SignNet (or BasisNet) to obtain invariant positional encodings, and then concatenating these with existing node features as input to a downstream graph neural network or Transformer model.

The additional computational cost of SignNet/BasisNet depends on the complexity of the neural networks ϕ and ρ used within their architecture. For tasks like graph regression on the ZINC dataset (IRWIN *et al.*, 2012), SignNet introduces a modest increase in training time per iteration. The precomputation of eigenvectors is done once and is generally efficient for datasets of typical size. The universality of these networks can theoretically require high-order tensors for certain equivariant functions, but practical implementations (SignNet and BasisNet using 2-IGNs) use efficient components like elementwise MLPs, DeepSets, or GNNs for ϕ and ρ, making them computationally tractable while still offering significant expressive power. However, memory for element-wise operations (or more generally, features and parameters within ϕ and ρ) when working with very large graphs remains a consideration.

2.5 Chapter Summary and Forward Look

This section established the mathematical and algorithmic foundations essential for the development of the hybrid GNN-heuristic framework. Beginning with the modeling of FEM meshes as graphs, it introduced both primal and dual graph representations, explaining how dual graphs are particularly suited for partitioning due to their encoding of element adjacencies.

The spectral theory component introduced Laplacian matrices and their eigenvalues, especially the Fiedler vector, which plays a crucial role in capturing global structural information for partitioning tasks. Partition quality metrics such as Cut, Normalized Cut (NCut), Ratio Cut (RCut), and Min-Max Cut (MMCut) were explored both conceptually and analytically, showing how different objectives impact partition balance and interconnectivity.

We also covered deep learning principles, including feedforward networks, convolutional architectures, and the progression toward Graph Neural Networks. It emphasized how GNNs extend traditional architectures to graph domains using message passing. Importantly,

it introduced key models like GCN, GAT, GraphSAGE, and GIN, explaining their relative expressivity and relevance to featureless graphs.

Finally, various positional and structural encoding strategies were reviewed, including Laplacian Eigenmaps, Random Walk-based encoders, and the sign-invariant models SignNet and BasisNet. These provide critical input representations when raw node features are absent.

Together, these concepts form the intellectual and computational toolkit used in the methodology that follows. In the next section, this theoretical grounding is put into practice: positional encodings are embedded into a GIN-based GNN architecture, and the MMCut objective is incorporated into a differentiable training framework.

3 METHODOLOGY

3.1 Introduction

This chapter presents the methodological framework proposed for the unsupervised graph partitioning of Finite Element Method (FEM) meshes using Graph Neural Networks (GNNs). The central objective is to assign elements of a mesh, represented as nodes in a graph to partitions in a balanced and communication-efficient manner, without pre-labeled data.

FEM meshes can be abstracted either through **primal** or **dual** graph representations. In the primal formulation, each vertex corresponds to a geometric point in the mesh, and edges represent physical adjacency. In the dual formulation, each vertex represents a mesh element (e.g., triangle or tetrahedron), and edges indicate shared interfaces between elements. While both representations are valid, this work focuses exclusively on **dual graphs**, as they are more directly associated with inter-element communication in distributed simulations. Nevertheless, the proposed method does not rely on a specific topological structure and remains applicable to primal graphs or arbitrary domains, since it learns from spectral and positional information rather than handcrafted topological features.

The approach combines components from spectral graph theory, deep learning, and combinatorial optimization. Specifically, the proposed architecture employs a stack of message-passing layers guided by positional encoders such as Laplacian Eigenmaps, SignNet, or BasisNet. These encodings allow the network to extract expressive node representations even in the absence of node features, a common scenario in scientific computing.

The architecture consists of four main modules:

1. **Positional and Structural Embedder**: computes the k smallest Laplacian eigenvectors (or alternative structural signals) and embeds them through an MLP.
2. **Feature Embedder**: applies GNN layers to locally propagate information across the graph topology.
3. **Feature Aggregator**: merges outputs from positional and message-passing stages using pooling or sequence-based aggregation (e.g., BiLSTM + Attention).
4. **Classifier**: predicts soft partition assignments for each node via an MLP followed by a softmax.

To ensure high-quality partitions, the training process minimizes a differentiable loss function that combines two main terms: a generalized edge-cut cost ($\mathscr{L}_{\text{XCut}}$), adaptable to

metrics like NCut, RCut, or MMCut; and a balance term ($\mathscr{L}_{\text{Bal}}$), which penalizes imbalance between partition sizes.

Finally, we apply the Fiduccia–Mattheyses (FM) algorithm at inference time. This smoothing step refines the partitioning output while preserving the overall structure.

The next sections detail the heuristics used for baseline comparisons, the dataset, the design of the GNN architecture, the loss functions, and the implementation details.

3.2 Heuristics

3.2.1 Heuristics for Topologicals Partitionings

Topological algorithms use only the connectivity information of the graph and are thus applicable to a wider range of problems where geometric data is unavailable or not meaningful.

3.2.1.1 Spectral Clustering and Bisection

The spectral clustering algorithm, as proposed in (NG *et al.*, 2001) is a powerful technique that clusters points using the eigenvectors of a matrix derived from the data. It is particularly effective at identifying non-convex clusters where methods like K-Means would typically fail, as demonstrated by the authors. The algorithm achieves this by transforming the data into a new, often lower-dimensional space (derived from the top k eigenvectors of a normalized Laplacian matrix), where cluster membership becomes more apparent and amenable to simpler clustering algorithms like K-Means. The theoretical analysis provided in their work shows that the method's success is linked to the "eigengap", the difference between the k-th and (k + 1)-th eigenvalues of the Laplacian. A large eigengap indicates that the subspace spanned by the first k eigenvectors is stable, which corresponds to situations where each of the k true clusters is internally cohesive and well-separated from the others.

- **Time Complexity:** Computing the first k eigenvectors using the Lanczos algorithm on a sparse Laplacian takes $\mathscr{O}(k(|V|+|E|))$ time, assuming good convergence and sparsity. The K-Means step adds $\mathscr{O}(k|V|d)$, where $d = k$.
- **Space Complexity:** The algorithm requires $\mathscr{O}(|V|+|E|+k|V|)$ space for storing the graph, Laplacian, and eigenvectors.

Algorithm 1 Spectral Clustering K-Means

1: **Input:** A graph $G = (V, E)$ with vertices V and edges E
2: Compute the Laplacian matrix: $L(G) \leftarrow D - \text{Adj}(G)$
3: Compute the first k eigenvectors of $L(G)$ associated with the smallest k eigenvalues: $\{v_i\}_{i=1}^{k}$
4: Construct the matrix: $M \leftarrow [v_1|v_2|\ldots|v_k]$ where each v_i is a column of M
5: Apply the K-Means clustering algorithm to the rows of M: $\{\Omega_i\}_{i=1}^{k} \leftarrow \text{K-Means}(M, k)$
6: Each row corresponds to an element, and it is labeled by a cluster
7: **Output:** Subdomains $\{\Omega_k\}$

For the special case of bisection ($k = 2$), a simpler and computationally cheaper approach is available through the Fiedler vector, which is the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix. This method avoids the need for K-Means by partitioning the graph based on a scalar threshold, typically the mean or median(POTHEN *et al.*, 1990) of the Fiedler vector values. This strategy is particularly effective for mesh-like graphs where geometric locality correlates with spectral smoothness.

- **Time Complexity:** Computing the Fiedler vector via Lanczos requires $\mathcal{O}(|V| + |E|)$ per iteration, typically converging in $\mathcal{O}(k)$ iterations.
- **Space Complexity:** $\mathcal{O}(|V| + |E|)$, assuming the Laplacian is sparse and only one eigenvector is stored.

Algorithm 2 Spectral Bisection Mean/Median

1: **Input:** A graph $G = (V, E)$ with vertices V and edges E
2: Compute the Laplacian matrix: $L(G) \leftarrow D - \text{Adj}(G)$
3: Compute the Laplacian matrix: $L(G) \leftarrow D - \text{Adj}(G)$
4: Compute the first eigenvector of $L(G)$ associated with the smallest eigenvalue: v
5: Sort the elements of Ω using the fiedler vector v, $\Pi \leftarrow \text{sort}(\Omega, v)$
6: Split Π in Ω_1 and Ω_2 using the mean or median of v
7: **Output:** Subdomains Ω_1 and Ω_2

This bisection approach is a cornerstone of multilevel and recursive partitioning algorithms due to its simplicity and effectiveness. It balances partition sizes while minimizing edge cuts, especially in FEM-derived dual graphs with strong spatial coherence.

3.2.1.2 Kernighan–Lin

This algorithm, first proposed by its namesakes in 1970 (KERNIGHAN; LIN, 1970), is a widely recognized iterative heuristic for refining an existing partition to reduce the cut-size. It does not create a partition from scratch, but rather improves an existing one, making it a

valuable tool to combine with faster, more global methods. In each pass, the algorithm evaluates all possible swaps of pairs of unlocked nodes between two partitions. It identifies the pair swap that produces the greatest gain (reduction in cut size). The nodes considered for swapping are then typically locked for the remainder of the pass to prevent them from being moved multiple times. Crucially, the algorithm continues this process of identifying best potential swaps and their gains, even accepting sequences of swaps that might temporarily increase the cut size (i.e., have negative gain), in an effort to navigate out of local minima. At the end of the pass, only the prefix of performed hypothetical swaps that yielded the best cumulative cut-size improvement is actually executed as the result of that pass.

- **Time Complexity:** A single pass of the original K/L algorithm has a complexity of $\mathcal{O}(|V|^2 \log |V|)$, or $\mathcal{O}(|V|^2)$ for sparse graphs.
- **Space Complexity:** Storing the graph and gain information requires $\mathcal{O}(|V|^2)$ or $\mathcal{O}(|E|)$ space.

Algorithm 3 Kernighan–Lin

Input: A graph $G = (V,E)$ with vertices V and edges E
Initialize two partitions: $\Omega_1, \Omega_2 \leftarrow$ Initial partition (e.g., random, linear, etc.)
Compute the cost for each node $e_k \in G(V,E)$: $D_k \leftarrow E_k - I_k$ where E_k is the external cost and I_k is the internal cost
Create an empty set V to store possible swaps and their corresponding gains: $(e_i, e_j, g_{e_i,e_j}) \in V$
Set: $U_1 \leftarrow \Omega_1, U_2 \leftarrow \Omega_2$ as the sets of unlocked elements
while $U_1 \neq \emptyset$ and $U_2 \neq \emptyset$ **do**
    Find a pair (e_i, e_j) with $e_i \in U_1$ and $e_j \in U_2$ that maximizes the gain:
    $G(e_i, e_j) = D_{e_i} + D_{e_j} - 2c_{e_i,e_j}$
    Add $(e_i, e_j, G(e_i, e_j))$ to set V
    Remove e_i from U_1 and e_j from U_2
    Update costs D_k for all $k \in U_1 \cup U_2$
end while
Find the prefix k^* that maximizes the cumulative gain: $k^* = \arg\max_k \sum_{i=1}^{k} g_i$
if $\sum_{i=1}^{k^*} g_i > 0$ **then**
    Swap the first k^* pairs $(e_{s1}, e_{s2}) \in V$ between partitions Ω_1 and Ω_2
    Go to step 2
else
    The algorithm terminates
end if
Output: Subdomains Ω_1 and Ω_2

3.2.1.3 Fiduccia-Mattheyses

The Fiduccia–Mattheyses (FM) algorithm is a linear-time refinement of the K/L method, introduced by its namesakes in (FIDUCCIA; MATTHEYSES, 1982). The critical improvements are that it moves single nodes (or cells) instead of swapping pairs, and it uses an efficient bucket-based data structure to store nodes according to their gain values. This allows the best node to be moved to be identified in essentially constant time, and updating the gains of its neighbors is also a constant-time operation for each neighbor affected by the move. These optimizations reduce the complexity of one pass of the algorithm to $O(|\mathscr{E}|)$ (or $O(P)$ where P is the total number of pins/net connections, which is proportional to the number of edges in many graph representations). The algorithm is also designed to maintain or improve the balance between partitions, typically by selecting which block to move a cell from based on a balance criterion and only allowing moves that satisfy this criterion.

- **Time Complexity:** One pass of the FM algorithm runs in $\mathscr{O}(|E|)$, which is linear in the number of edges.
- **Space Complexity:** The data structures, including the bucket lists, require space proportional to the graph size, $\mathscr{O}(|V|+|E|)$.

Algorithm 4 Fiduccia–Mattheyses

Input: Graph $G = (V, E)$; initial partition into two blocks Ω_1 and Ω_2
while a pass improves the cutset size **do**
    Compute gain $g(i)$ for each free cell i
    Initialize bucket lists for blocks Ω_1 and Ω_2
    Save initial partition and cutset as the best seen
    while there are free cells **do**
        Determine which block to move from, respecting balance constraint
        Select the free cell with maximum gain from the corresponding bucket
        Move the base cell to the opposite block
        Lock the base cell for the rest of the pass; remove from bucket
        for all nets connected to the base cell **do**
            Update gains for affected free neighbor cells
            Update bucket positions for changed cells
        end for
        if current partition improves the best cutset **then**
            Save it
        end if
    end while
    Set the best partition found in this pass as the starting point for the next pass
    Unlock all cells for the next pass
end while
Output: Subdomains Ω_1 and Ω_2

3.2.1.4 Space Filling Hamiltonian Cycle

Finding a Hamiltonian cycle that visits every node in a general graph is an NP-complete problem (COOK, 1971), and finding the longest one is NPO-complete, meaning it is unlikely to have an efficient, constant-ratio approximation algorithm (WU *et al.*, 1998). Even for the specific case of finding a cycle through the standard edge-adjacent dual of a triangulated irregular network (TIN), the problem remains NP-complete (ARKIN *et al.*, 1996).

However, the algorithm described in (Bartholdi III; GOLDSMAN, 2004) provides a constructive proof that a Hamiltonian cycle is guaranteed to exist and can be found efficiently if one considers the vertex-adjacency dual of a TIN, where triangles are connected if they share at least one vertex (1-adj) or an edge (2-adj). The algorithm works by iteratively growing a cycle, it begins with a small initial cycle of three edge-adjacent triangles and adds one new triangle at a time until all triangles are included. The core of the method is a detailed case-based analysis (Cases I, II, and III) that determines how to splice a new triangle into the cycle based on its adjacency to its new neighbors, ensuring the cycle remains Hamiltonian at every step.

This method provides a way to sequence all elements of a mesh into a single continuous path, which can then be partitioned into contiguous subdomains

- **Time Complexity:** The constructive algorithm runs in linear time with respect to the number of triangles, $\mathcal{O}(|V|)$, as each of the insertion steps takes constant time with appropriate data structures (Bartholdi III; GOLDSMAN, 2004).
- **Space Complexity:** Requires storing the mesh's adjacency information and the cycle itself, which is $\mathcal{O}(|V|)$.

Algorithm 5 Space Filling Hamiltonian Cycle

Input: A triangular mesh $G = (V, E)$ with vertices V and edges E
Let Ω be the set of elements
Select an initial curve $C = \{e_1, e_2, e_3\}$ such that the elements (e_1, e_2) and (e_2, e_3) share a common edge
Let $N_\Omega \leftarrow |\Omega|$ be the total number of elements in the mesh
Mark elements e_1, e_2, e_3 as visited
Let $N_C \leftarrow \{e \in \Omega \setminus C \mid e \text{ shares an edge with any } e_i \in C\}$
while $|C| < N_\Omega$ **do**
    Select an element $s \in N_C$
    Find $t \in C$ such that t shares an edge with s
    Append s to the curve C in order relative to t
    Mark element s as visited
    Update N_C with unvisited neighbors of s
end while
$\Omega_k \leftarrow C[\lfloor \frac{(k-1)N}{K_{\text{total}}} \rfloor, \text{if } k = K_{\text{total}} \text{ then } N \text{ else } \lfloor \frac{kN}{K_{\text{total}}} \rfloor]$
Output: Subdomains $\{\Omega_k\}$

3.2.2 Heuristics for Spatial Partitionings

Geometric partitioning algorithms leverage the spatial coordinates of a graph's vertices. These methods are typically fast but are only applicable for problems where such geometric information is available and meaningful, such as in finite element meshes. They generally assume that vertices which are close in space are also connected by a short path in the graph.

3.2.2.1 Coordinate

This is the simplest coordinate-based method. It partitions the vertices by finding a hyperplane orthogonal to a chosen coordinate axis (e.g., the xaxis) that divides the set of points into two equal halves (SIMON, 1991). This is done by calculating the median of the vertex coordinates along that axis. When used recursively to create more than two subdomains, it is common to alternate the bisection axis (e.g., x, then y, then z) to produce partitions with better aspect ratios and hopefully a lower cut-size. The main weakness of this method is its dependence on the orientation of the coordinate system; a rotation of the mesh can result in a significantly different partition.

- **Time Complexity:** The dominant operation is sorting the vertices along the chosen axis, which has a complexity of $\mathcal{O}(|V| \log |V|)$.

- **Space Complexity:** The algorithm requires storing the coordinates of the vertices, leading to a space complexity of $\mathcal{O}(|V|)$.

Algorithm 6 Coordinate

1: **Input:** A triangular mesh $G = (V, E)$ with vertices V and edges E.
2: Let Ω be the set of elements
3: Let dim $\leftarrow$ spatial dimension (2D or 3D)
4: Let $N_\Omega \leftarrow$ number of elements in the mesh Ω
5: Compute centroids of all elements c_{e_k} for all $e_k \in \Omega$
6: Determine the longest axis of the mesh
7: Sort elements along the longest axis $\Pi \leftarrow \text{sort}(c_{e_k}[\text{axis}])$
8: Split Π in Ω_1 and Ω_2 using the median of the sorted elements
9: **Output:** Subdomains Ω_1 and Ω_2

3.2.2.2 Inertial

This method, described in works like (POTHEN, 1997), improves upon coordinate bisection by finding a cutting plane based on the intrinsic geometry of the point distribution, making it coordinate-system independent. The idea is to identify the axis of minimal rotational inertia for the set of vertices and partition the graph with a plane orthogonal to this axis. This axis tends to align with the overall shape of the mesh, which can lead to a smaller cut-size. The algorithm first computes the center of mass of the vertices. It then constructs an inertia matrix and finds the eigenvector corresponding to its smallest eigenvalue. This eigenvector defines the direction of the axis of minimal inertia, which is used to define the cutting plane.

- **Time Complexity:** The most expensive step is sorting the N vertices based on their projection onto the inertial axis, which is $\mathcal{O}(|V| \log |V|)$. The computation of the center of mass and the inertia matrix takes $\mathcal{O}(|V|)$ time, and the eigendecomposition of the small (2×2 or 3×3) inertia matrix is a constant-time operation.
- **Space Complexity:** Requires storing vertex coordinates, making it $\mathcal{O}(|V|)$.

Algorithm 7 Inertial

Input: A triangular mesh $G = (V, E)$ with vertices V and edges E.
Let Ω be the set of elements
Let $N_\Omega \leftarrow$ number of elements in the mesh Ω
Compute the centroid of each element c_{e_k} for all $e_k \in \Omega$
Compute the global mesh centroid $c_\Omega \leftarrow \frac{1}{N_\Omega} \sum_{k=1}^{N_\Omega} c_{e_k}$
Shift the coordinates for each element $\Delta_k \leftarrow c_{e_k} - c_\Omega$
Initialize the inertia matrix $I \leftarrow 0_{\text{dim}\times\text{dim}}$
for each element $e_k \in \Omega$ **do**
    Update inertia matrix $I \leftarrow I + \Delta_k^\top \Delta_k$
end for
Compute eigenvalues λ and eigenvectors v of the inertia matrix I
Let $v_{\min}$ be the eigenvector associated with the smallest eigenvalue
Normalize the eigenvector $v_{\min} \leftarrow \frac{v_{\min}}{\|v_{\min}\|}$
Compute signed distances of e_k to the hyperplane orthogonal to $v_{\min}$, $d_k \leftarrow \langle v_{\min}, \Delta_k \rangle$
Sort the elements of Ω using the signed distances d, $\Pi \leftarrow \text{sort}(\Omega, d)$
Split Π in Ω_1 and Ω_2 using the median of the sorted elements
Output: Subdomains Ω_1 and Ω_2

3.3 Dataset

This study uses two complementary datasets to train and evaluate the proposed method. We use the **Fusion 360 Gallery Part Segmentation** dataset (WILLIS *et al.*, 2021) for training, validation, and internal testing, and a subset of unseen industrial meshes from **Transvalor S.A** for evaluation.

3.3.1 Fusion 360

We adopt version 1,0 of the **Fusion 360 Gallery Part Segmentation** dataset, a large-scale collection of 3D CAD models derived from the Autodesk Fusion 360 platform. It contains **35 858** manifold surface meshes representing individual mechanical parts. Each mesh is annotated with semantic part labels and was originally authored by human users via parametric CAD modeling.

To ensure consistency in graph construction, we filter out non-contiguous meshes (i.e., those composed of disconnected components). This filtering results in a total of **35 805** contiguous meshes used in our pipeline.

We partition these meshes into three disjoint subsets: **30 000** for training, **3 000** for validation, and **2 805** for internal testing. The remaining meshes are reserved for further experiments.

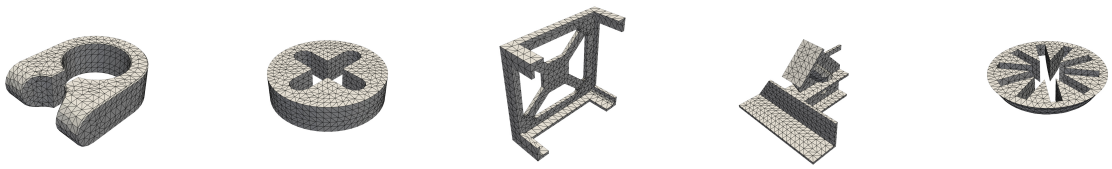
Table 3 summarizes key statistics for both the primal and dual graphs derived from these meshes. The primal graph is constructed from mesh vertices with edges based on physical adjacency, while the dual graph treats each element (e.g., triangle) as a node and connects them by shared faces.

Table 3 – Statistical summary of the processed Fusion 360 dataset (primal vs dual graphs).

Metric	**Primal Graph**	**Dual Graph**
Total meshes	35 805	35 805
Avg. nodes per mesh	1 010,71	2 020,84
Std. dev. of node count	26,12	51,89
Min / Max node count	804 / 1 168	1 668 / 2 332
Avg. edges per mesh	6 062,51	6 062,51
Std. dev. of edge count	155,68	155,68
Min / Max edge count	5 004 / 6 996	5 004 / 6 996
Avg. degree per node	12,00	6,00

Source: Table from the author (2025)

Figure 15 shows examples of meshes from the dataset, illustrating the geometric and topological variety across different mechanical parts.



Source: Image from the author (2025)
Figure 15 – Examples of triangular meshes from the Fusion 360 dataset used in our experiments.

The Fusion 360 dataset has been widely adopted in recent literature. Originally introduced in (WILLIS *et al.*, 2021), it has been used in follow-up works that explore mesh segmentation algorithms, geometric deep learning techniques, and hybrid representation frameworks. For instance, (FERGUSON *et al.*, 2024) proposed a topology-agnostic Graph U-Net for scalar field prediction on unstructured meshes, and (UWIMANA *et al.*, 2025) presented a hybrid model combining B-rep and mesh data for semantic segmentation tasks.

3.3.2 Evaluation Set

To evaluate the generalization ability of our model, we curated a set of **six unseen meshes** provided by **Transvalor S.A.**, a company specializing in simulation software for industrial forming processes. These internal meshes were selected to test the model's performance

on out-of-distribution data with realistic topologies and element complexities not present in the Fusion 360 training set.

All meshes are contiguous and vary significantly in resolution and geometry. They represent real-world industrial parts, making them a valuable benchmark for assessing robustness in practical scenarios.

Table 4 reports the node and edge counts for both the primal and dual graph representations of each mesh.

Table 4 – Structural statistics of the six evaluation meshes from Transvalor S.A.

Mesh	**Primal Graph**		**Dual Graph**	
	# Nodes	# Edges	# Nodes	# Edges
Blocker_1	3 692	22 140	7 380	22 140
Blocker_2	1 612	9 660	3 220	9 660
Blocker_3	1 822	10 920	3 640	10 920
Blocker_4	8 986	53 904	17 968	53 904
Blocker_5	9 215	55 278	18 426	55 278
Blocker_6	6 617	39 690	13 230	39 690

Source: Table from the author (2025)

Figure 16 illustrates these six meshes, highlighting their geometric variability and realism.

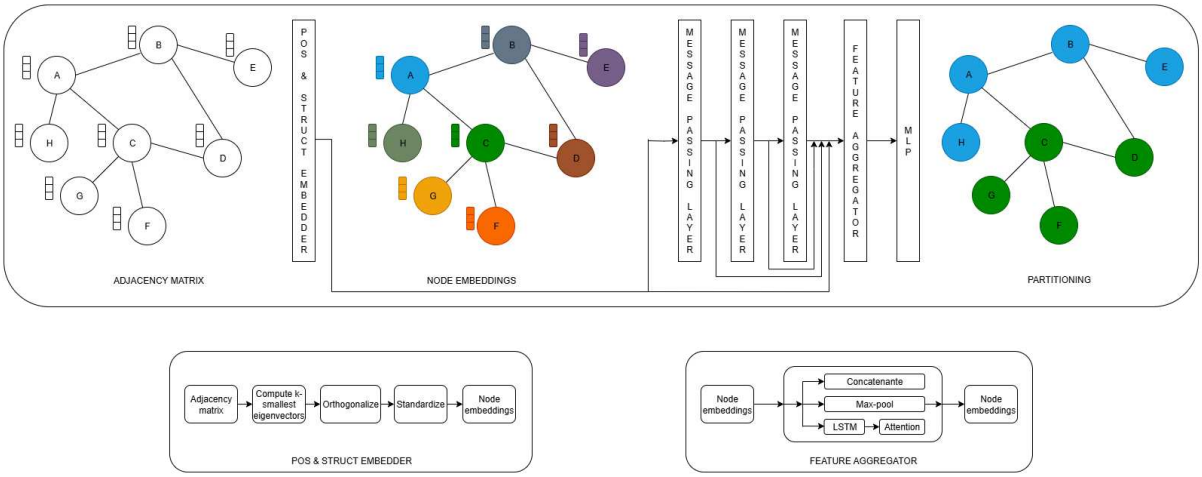
Source: Image from the author (2025)
Figure 16 – Evaluation meshes provided by Transvalor S.A., used to test the model on unseen data.

These meshes are used exclusively during inference to assess the partitioning model's robustness. Quantitative results and visual comparisons are presented in Section 4, demonstrating the method's performance on real-world, industrial-grade geometries.

3.4 Architecture

The architecture proposed in this work, illustrated in the Figure 17, is composed of four main modules: (i) Positional and Structural Embedder, (ii) Feature Embedder, (iii) Feature Aggregator, and (iv) Classifier. The model is designed to perform node-level partitioning in

arbitrary graphs, particularly dual graphs derived from FEM meshes, where nodes represent mesh elements and edges capture shared interfaces.



Source: Image from the author (2025)
Figure 17 – Overview of the proposed architecture.

This pipeline transforms raw adjacency information into meaningful partition predictions in an end-to-end differentiable manner, suitable for integration with gradient-based optimization schemes. The overall process is summarized in Algorithm 8.

Algorithm 8 Spectral GNN Partitioning

1: **Input:** A triangular mesh $G = (V, E)$ with vertices V and edges E.
2: Compute positional embeddings $P \in \mathbb{R}^{|V| \times d}$ using LaPE or alternative encoder
3: Apply message-passing layers to obtain features F_i at each layer
4: Aggregate P and $\{F_i\}$ using chosen strategy to get Z
5: Pass Z through an MLP followed by softmax to obtain $\hat{Y} \in \mathbb{R}^{|V| \times K}$
6: Assign each node to the cluster with the highest predicted probability: $Y_i = \arg\max_k \hat{Y}_{i,k}$
7: (Optional) Apply FM algorithm on Y to refine partitions
8: **Output:** Subdomains $\{\Omega_k\}$

3.4.1 Positional and Structural Embedder

To address the lack of discriminative node features in scientific graphs, we employ positional encoders derived from spectral methods. This is particularly crucial in domains like mesh partitioning, where raw input graphs, such as dual graphs from FEM discretizations, typically lack meaningful node attributes, normally has only spatial data. Without adequate node-level features, classical message-passing Graph Neural Networks (GNNs) suffer with the problem that structurally symmetric nodes may receive identical embeddings, limiting the

model's ability to learn expressive representations.

To mitigate this, we leverage positional and structural encoders that enrich the graph with additional signals reflecting its topology. Specifically, we compute the k smallest eigenvectors of the graph Laplacian using the Lanczos algorithm, which capture global structural properties of the graph. These eigenvectors are orthogonalized using QR decomposition, normalized to have zero mean and unit variance, and then passed through a Multi-Layer Perceptron (MLP) to produce the final embeddings.

The inclusion of these encoders effectively breaks graph symmetries and allows the network to distinguish nodes based on their position within the global structure. This is especially beneficial for partitioning tasks, where subtle differences in connectivity patterns can imply different subdomain assignments. In addition to classical Laplacian Eigenmaps, positional encoders such as SignNet and BasisNet (LIM *et al.*, 2022) can be used. These models are invariant to basis rotation and designed to overcome the limitations of standard spectral encodings by learning invariant functions over sets of eigenvectors.

Beyond spectral methods, alternative structural encoders can also be employed. For instance, Prone (ZHANG *et al.*, 2019) is an efficient embedding method that leverages proximity-aware neighborhoods to generate high-quality structural representations. GraRep (CAO *et al.*, 2015) extends the idea of skip-gram models by capturing high-order proximities through matrix factorization. Similarly, random walk-based approaches as described in the previous section can be used to generate embeddings by concatening the embedding vector of each step of the diffusion process. While these methods are originally unsupervised and task-agnostic, their output can be adapted as initial node features or concatenated with spectral encodings to enrich the learned representations.

The availability of diverse encoders allows the model to adapt to various graph topologies, enabling better generalization across domains and more robust partitioning performance in the absence of handcrafted features.

3.4.2 Feature Embedder

This module is responsible for learning localized representations by propagating information across the graph topology through message passing. It consists of a stack of L Graph Neural Network (GNN) layers, where each layer updates the representation of a node by aggregating information from its neighbors. The choice of GNN layer is modular and can

be instantiated with architectures such as GCN, GAT, GraphSAGE, or GIN, depending on the desired trade-off between expressivity and computational cost.

Formally, each layer computes an embedding $F_i \in \mathbb{R}^{|V| \times d}$ for all nodes, where $i \in \{1, \ldots, L\}$ denotes the layer index and d is the embedding dimension. The output F_i captures structural information at the i-hop neighborhood scale. To enhance model expressiveness and facilitate training, we employ two key architectural strategies:

- **Residual Connections**: Inspired by ResNets (HE *et al.*, 2015b), skip connections are added between layers to combat the vanishing gradient problem and reduce the risk of oversmoothing, a phenomenon where node representations become indistinguishable across the graph as depth increases.
- **Jumping Knowledge (JK) Networks**: To aggregate multi-scale information, we adopt the Jumping Knowledge framework (XU *et al.*, 2018), which combines intermediate outputs $\{F_i\}_{i=1}^{L}$ using operations such as max-pooling, mean, or concatenation. This strategy allows the final representation to flexibly incorporate both local and global context.

The stack outputs a set of representations at different receptive fields, which are then passed to the Feature Aggregator module. This architecture allows the model to adaptively focus on relevant structural scales during training, which is essential in graphs with irregular connectivity patterns, such as those derived from FEM meshes.

3.4.3 Feature Aggregator

The outputs from the Positional and Structural Embedder and the Feature Embedder are combined in the Feature Aggregator module to produce a unified node representation. This step is essential for fusing global structural information (captured by positional encoders) with local topological features (learned through message passing), enabling the model to make partitioning decisions that are both globally coherent and locally consistent.

The aggregation strategy used in this module is flexible and can be adapted to the task or dataset. Simple strategies such as **concatenation** or **element-wise max-pooling** can be used to merge the embeddings, providing computational efficiency and interpretability. For more expressive fusion, we can employ sequence-based architectures, such as a **Bidirectional Long Short-Term Memory (BiLSTM)** network followed by an **attention mechanism** (BAHDANAU *et al.*, 2015). In this configuration, each node's embedding sequence, typically the outputs from multiple message-passing layers or from different encoder streams is processed by the BiLSTM

to model contextual dependencies, and the attention layer then assigns importance weights to each embedding in the sequence.

This flexible aggregation scheme yields a final node embedding matrix $Z \in \mathbb{R}^{|V| \times d}$, where each row Z_i represents a fused, context-aware representation of node i. These embeddings are then used by the downstream classifier to predict partition assignments.

3.4.4 Classifier

The final node representations $Z \in \mathbb{R}^{|V| \times d}$, produced by the Feature Aggregator, are passed through a feedforward neural network (MLP) that maps each embedding to a K-dimensional output space, where K is the target number of partitions. This transformation is followed by a softmax activation, yielding a probabilistic assignment for each node:

$$\hat{Y} = \text{softmax}(\text{MLP}(Z)) \in \mathbb{R}^{|V| \times K}$$

Each row $\hat{Y}_{i,:}$ represents the predicted categorical distribution over the K clusters for node i, encoding the model's confidence in assigning node i to each partition.

This soft assignment allows the model to be trained using differentiable loss functions while enabling flexibility for downstream refinement, such as discrete projection or post-processing with partitioning heuristics. In practice, hard assignments can be obtained by taking the argmax over each row, producing a deterministic label $Y_i = \arg\max_k \hat{Y}_{i,k}$ for each node, which is adopted at inference time.

3.5 Differentiable Loss Formulation

The goal is to divide a graph into disjoint clusters while minimizing the number of edges crossing between them and ensuring that clusters remain balanced. Classical formulations such as Normalized Cut (NCut), Ratio Cut (RCut), and Min-Max Cut (MMCut) express this goal through discrete, combinatorial objectives based on hard node assignments. However, these objectives are inherently non-differentiable and thus incompatible with gradient-based learning methods.

To address this, we adopt continuous and differentiable relaxations of these metrics. The output of the model is a soft assignment matrix $\hat{Y} \in [0,1]^{N \times K}$, where each entry $\hat{Y}_{i,k}$ represents the probability that node i is assigned to cluster k. This probabilistic representation enables end-to-end optimization using gradient descent while approximating classical objectives.

Different cut-based metrics can be unified under this framework by modifying the normalization term in the objective. As an illustrative example, we derive the differentiable relaxation of the NCut metric.

3.5.1 Hard NCut Formulation

The Normalized Cut (NCut) is a classical graph partitioning objective that penalizes inter-cluster edges while promoting balance among partitions. Let $G = (V, E)$ be an undirected graph with $N = |V|$ nodes, $A \in \mathbb{R}^{N \times N}$ the adjacency matrix and $d \in \mathbb{R}^{N \times 1}$ the degree vector, where $d_i = \sum_{j=1}^{N} A_{ij}$.

Let $Y \in \{0,1\}^{N \times K}$ be the hard assignment matrix, where $Y_{i,k} = 1$ if node i belongs to cluster k, and 0 otherwise. Let $y_k = Y_{:,k} \in \{0,1\}^N$ denote the indicator vector for cluster k.

The cut and volume terms defined directly in matrix form are:

$$\text{cut}_k(Y) = \sum_{i,j} A_{ij} Y_{i,k} (1 - Y_{j,k}), \tag{3.1}$$

$$\text{vol}_k(Y) = \sum_{i} d_i Y_{i,k} = d^\top Y_{:,k}. \tag{3.2}$$

Which leads to NCut expressed as:

$$\text{NCut}(Y) = \sum_{k=1}^{K} \frac{\text{cut}_k(Y)}{\text{vol}_k(Y)} = \sum_{k=1}^{K} \frac{\sum_{i,j} A_{ij} Y_{i,k} (1 - Y_{j,k})}{d^\top Y_{:,k}}. \tag{3.3}$$

This formulation penalizes connections between nodes that are likely to belong to different clusters, while encouraging connected nodes to stay together.

Let us define $\Gamma = d^\top Y \in \mathbb{R}^{1 \times K}$, which collects the volumes of all clusters. With this, the numerator for each cluster k can be rewritten as:

$$\sum_{i,j} A_{ij} Y_{i,k} (1 - Y_{j,k}) = Y_{:,k}^\top A (\mathbf{1} - Y_{:,k}).$$

Thus, the overall loss becomes:

$$\text{NCut}(Y) = \sum_{k=1}^{K} \frac{Y_{:,k}^\top A (\mathbf{1} - Y_{:,k})}{\Gamma_k}. \tag{3.4}$$

Expanding this expression using matrix operations and denoting $\oslash$ as column-wise division, we observe that:

$$\sum_{i,j} A_{ij} \sum_{k=1}^{K} \frac{Y_{i,k}}{\Gamma_k} (1 - Y_{j,k}) = \sum_{i,j} A_{ij} (Y_i \oslash \Gamma)(1 - Y_j)^\top,$$

Finally, we sum over all elements and use $\odot$ to element-wise multiplication and obtain

$$\text{NCut}(Y) = \sum \left((Y \oslash \Gamma)(1-Y)^\top \odot A \right). \tag{3.5}$$

3.5.2 Soft Relaxation NCut Formulation

To make the Normalized Cut objective compatible with gradient-based optimization, we relax the hard cluster assignment matrix $Y \in \{0,1\}^{N \times K}$ into a soft assignment matrix $\hat{Y} \in [0,1]^{N \times K}$, where each row $\hat{Y}_{i,:}$ represents a probability distribution over the K clusters. This relaxation transforms the discrete partitioning problem into a smooth, differentiable objective.

Following the same principles as in the hard case we get:

$$\mathscr{L}_{\text{NCut}}(\hat{Y}) = \sum \left((\hat{Y} \oslash \hat{\Gamma})(1-\hat{Y})^\top \odot A \right). \tag{3.6}$$

3.5.3 Generalized Partition Loss

While we have detailed the differentiable relaxation of the Normalized Cut (NCut) loss, the same framework applies to other classical partitioning objectives such as Ratio Cut (RCut) and Min-Max Cut (MMCut). These objectives differ only in the definition of the denominator term $\hat{\Gamma}$, which encodes the notion of volume, cardinality, or internal edge mass, respectively.

In general, the relaxed cut-based loss can be written as:

$$\mathscr{L}_{XCut}(\hat{Y}) = \sum \left((\hat{Y} \oslash \hat{\Gamma})(1-\hat{Y})^\top \odot A \right), \tag{3.7}$$

where $\hat{\Gamma} \in \mathbb{R}^{1 \times K}$ is a cluster-specific normalization vector:

- For NCut: $\hat{\Gamma} = d^\top \hat{Y}$
- For RCut: $\hat{\Gamma} = \sum_{i=1}^{N} \hat{Y}_{i,:}$
- For MMCut: $\hat{\Gamma} = diag(\hat{Y}^\top A \hat{Y})$

In parallel, we introduce a simple balancing term to ensure partition sizes are approximately uniform:

$$\mathscr{L}_{Bal}(\hat{Y}) = \frac{\max(\hat{\Gamma}) - \min(\hat{\Gamma})}{\min(\hat{\Gamma})}, \tag{3.8}$$

where $\hat{\Gamma}$ can be chosen to reflect the same cluster volume used in the cut term (e.g., $\hat{\Gamma} = d^\top \hat{Y}$ for NCut).

Finally, the complete differentiable loss function used during training is a weighted sum of both objectives:

$$\mathscr{L}(\hat{Y}) = \alpha \mathscr{L}_{XCut}(\hat{Y}) + \beta \mathscr{L}_{Bal}(\hat{Y}), \tag{3.9}$$

where α and β are hyperparameters (typically set to 1,0) that balance the trade-off between minimizing the cut and achieving balanced clusters.

3.6 Implementation Details

The proposed framework was implemented in Python, integrating several specialized libraries to support sparse matrix operations, neural network modeling, and mesh visualization:

- **PyTorch** (PASZKE *et al.*, 2019) and **PyTorch Geometric** (FEY; LENSSEN, 2019): Utilized for building and training the Graph Neural Network, including message-passing layers, differentiable objectives, and optimization routines.
- **SciPy Sparse** (VIRTANEN *et al.*, 2020): Enabled efficient manipulation of large sparse matrices, especially for constructing adjacency and Laplacian matrices.
- **PyVista** (SULLIVAN; KASZYNSKI, 2019): Used for visualization of the finite element meshes and their respective primal and dual graph representations.

All experiments were conducted on a single NVIDIA A100 GPU with 40 GB of memory. The total training time for the proposed architecture was approximately 12 hours. To improve training efficiency and stability, the positional embeddings derived from Laplacian eigenvectors were precomputed before training and kept fixed during the optimization process. This reduced the computational cost and allowed for consistent geometric encodings across epochs.

In addition to the GNN-based method, all classical heuristics, such as Coordinate Bisection, Inertial Bisection, Spectral Clustering, Kernighan–Lin, Fiduccia–Mattheyses, and the Space Filling Hamiltonian Cycle were manually implemented by the author. Each implementation adheres to the computational complexity of the original algorithm, as described in the respective publications. Whenever possible, parallelism and vectorized operations were employed using **NumPy** (HARRIS *et al.*, 2020) to enhance performance and scalability.

3.7 Chapter Summary and Forward Look

This part translated the theoretical principles into an implementable framework by detailing the architecture, training strategy, and refinement steps of the proposed hybrid GNN-heuristic model. It began by defining the spectrum of baseline heuristics, both topological (e.g., spectral clustering, Kernighan–Lin, Fiduccia–Mattheyses) and geometric (e.g., coordinate-based and inertial methods). These serve as classical comparators for the GNN approach and also inform the design of the hybrid strategy.

The dataset subsection introduced Fusion 360 and industrial meshes from Transvalor S.A., noting their size, variability, and lack of node features, making them ideal test cases for evaluating the model's inductive and generalization capabilities.

A modular Graph Neural Network (GNN) architecture was then constructed with distinct embedding stages. First, positional and structural encoders, such as Laplacian Eigenmaps and Random Walk Structural Encodings (RWSE), were used to generate the input signals. Next, a Graph Isomorphism Network based message-passing network aggregates local structural information. Finally, a feature aggregator and classifier produces the soft partition assignments.

The learning objective was defined using a soft relaxation of the cut and balance metrics, optimized via a differentiable loss that balances edge cut minimization and partition equality, enabling unsupervised learning for the model. At inference time, the output can be refined using the linear-time Fiduccia–Mattheyses algorithm (GNN-FM variant), closing the loop between deep learning and classical heuristics.

This methodological structure directly supports the upcoming experiments. In the next section, the architecture will be benchmarked against classical algorithms across various meshes. The effects of each component (loss function, encoders and heuristics) will be evaluated in isolation and in combination, confirming the utility of design decisions made here.

4 EXPERIMENTS

This chapter presents a comprehensive evaluation of the proposed unsupervised Graph Neural Network (GNN) based partitioning models against a suite of established heuristic and spectral methods. We assess the performance of each algorithm on a dataset of six blocker meshes, focusing on the quality of the 2-way partitions they produce. The evaluation is conducted through both quantitative metrics, as shown in Table 5, and qualitative visual analysis, depicted in Figure 19, aiming to highlight the strengths and weaknesses of each approach.

4.1 Experimental Setup

The experiments were conducted on a test set of six 3D Finite Element Method (FEM) meshes provided by Transvalor S.A., designated as Blocker_1 through Blocker_6. These meshes were selected to evaluate generalization to out-of-distribution geometries with varying topological complexity. The primary task was graph bisection ($k = 2$) on the dual graph representation of each mesh, a key operation in parallel FEM solvers and domain decomposition pipelines.

The performance of our proposed models, a baseline GNN and a refined variant using the **Fiduccia-Mattheyses** (FM) algorithm as a post-processing step (GNN-FM), was benchmarked against nine classical and modern partitioning algorithms:

- **Topological/Spectral Methods:** METIS, Spectral Clustering with K-Means (NG *et al.*, 2001), Spectral Bisection via Fiedler Vector (mean/median threshold), Fiduccia–Mattheyses (FM), Kernighan–Lin (KL) and Hamiltonian Cycle.
- **Geometric/Coordinate-Based Methods:** Coordinate and Inertial Bisection.

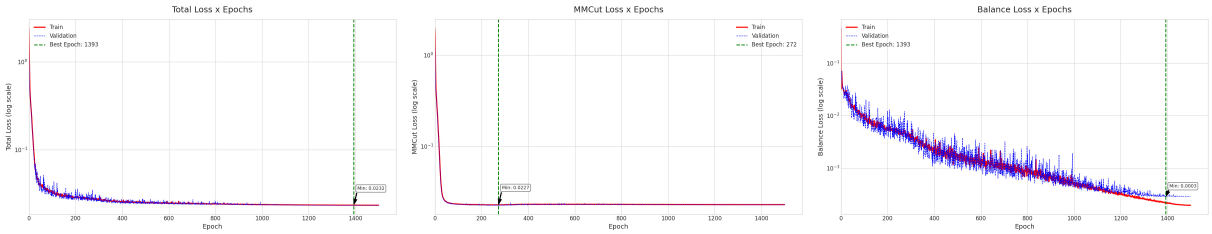
All methods were evaluated using two standard metrics:

- **Edge Cut** (Equation 2.4): The number of edges crossing between partitions. Lower values correspond to reduced inter-processor communication.
- **Balance (bal)** (Equation 2.6): The ratio between the sizes of the largest and smallest partitions. The optimal value is 1,00, indicating perfectly balanced partitions.

Our GNN models, based on a **Graph Isomorphism Network (GIN)** backbone, were trained to optimize a differentiable version of the **Min-Max Cut (MMCut)** objective, combined with a balance regularization term. This choice was motivated by theoretical analysis showing that MMCut favors both quality and balanced partitions, unlike other common metrics. The model was trained for 1 500 epochs using the AdamW optimizer and a cosine learning rate scheduler on a single NVIDIA A100 GPU, the Appendix A details the hyperparameters setting

4.2 Neural Network Training

The training and validation loss curves, depicted in Figure 18, illustrate a stable and effective learning process over 1 500 epochs. The total loss is composed of two main terms: the MMCut loss, which measures partition quality, and the balance loss, which enforces similarly sized partitions. The MMCut loss converges rapidly within the first 100 epochs and remains low, reaching the optimum at 272 epochs, indicating the model quickly learns to minimize edge cuts. In contrast, the balance loss shows a slower, yet steady, decrease throughout the training, demonstrating continuous refinement towards perfectly balanced partitions and reaching its minimum at 1 393 epochs.



Source: Image from the author (2025)
Figure 18 – Evolution of training and validation losses.

The training and validation curves for all loss components remain closely aligned, confirming that the model generalizes well and does not exhibit overfitting.

4.3 Quantitative Analysis

The quantitative results for all algorithms across the six evaluation meshes are summarized in Table 5. The data presents the edge cut and balance (cut / bal) for each method, with the best-performing algorithm for each mesh highlighted in bold.

Table 5 – Comparison of cut and balance (cut/bal) for each algorithm across all blocker meshes. Best algorithm per column is in bold.

Algorithm	Blocker_1	Blocker_2	Blocker_3	Blocker_4	Blocker_5	Blocker_6
gnn	83 / 1,0005	**57 / 1,0012**	**66 / 1,0000**	**98 / 1,0004**	142 / 1,0007	140 / 1,0003
gnn-fm	**80 / 1,0000**	**58 / 1,0000**	**66 / 1,0000**	**98 / 1,0000**	143 / 1,0000	139 / 1,0000
metis	**75 / 1,0016**	65 / 1,0012	69 / 1,0011	106 / 1,0004	**110 / 1,0002**	**96 / 1,0009**
spectral kmeans	86 / 1,5466	51 / 2,5038	57 / 2,1902	112 / 1,0107	90 / 1,7901	123 / 1,7649
spectral mean	113 / 1,2398	62 / 1,5394	69 / 1,4186	117 / 1,0056	147 / 1,4591	159 / 1,0000
spectral median	104 / 1,0000	64 / 1,0000	78 / 1,0000	112 / 1,0000	173 / 1,0000	159 / 1,0000
kernighan-lin	178 / 1,0131	83 / 1,0138	122 / 1,0290	246 / 1,0018	269 / 1,0180	215 / 1,0097
fiduccia-mattheyses	164 / 1,0000	102 / 1,0000	114 / 1,0000	202 / 1,0000	235 / 1,0000	235 / 1,0000
hamiltonian cycle	176 / 1,0000	128 / 1,0000	100 / 1,0000	202 / 1,0000	275 / 1,0000	271 / 1,0000
coordinate	100 / 1,0000	70 / 1,0000	82 / 1,0000	120 / 1,0000	175 / 1,0000	**93 / 1,0000**
inertial	90 / 1,0000	84 / 1,0000	102 / 1,0000	124 / 1,0000	183 / 1,0000	143 / 1,0000

Source: Table from the author (2025)

The results clearly indicate that our proposed **GNN-FM** model and the state-of-the-art **METIS** are the top-performing algorithms. They consistently produce the lowest or near-lowest edge cuts while maintaining near-perfect balance. For instance, GNN-FM achieves the best cut on Blocker_1, Blocker_2, Blocker_3, and Blocker_4, while METIS excels on Blocker_5 and Blocker_6. The integration of the FM algorithm as a refinement step provides a noticeable improvement over the baseline GNN, reducing the cut size and improving the balance, but without increase the complexity, because FM algorithm is linear.

Regarding spectral approaches, **Spectral K-Means** demonstrates a strong cut score in isolated cases, such as Blocker_2 (cut = 51), but this comes at the expense of extremely poor balance (e.g., bal = 2,50). This imbalance occurs because clustering in the embedded eigenvector space lacks explicit constraints on partition size, making it unreliable for practical applications where distributing the workload evenly is critical.

On the other hand, the **Spectral Bisection** techniques based on the Fiedler vector, splitting nodes by the mean or median of their spectral coordinate, exhibit more robust balance properties. In particular, the **Median** variant achieves perfect balance (bal = 1,00) across all meshes, a result of the symmetric distribution of eigenvector values. However, it still performs suboptimally in terms of edge cut compared to GNN-FM and METIS, especially on Blocker_5 and Blocker_6. The **Mean** variant, while slightly better at minimizing cut in some cases, introduces significant imbalance due to sensitivity to skewed eigenvector distributions. This sensitivity is visible in Blocker_2, where balance drops to 1,54, and in Blocker_5, with a ratio of 1,46.

Classical iterative heuristics like **Kernighan-Lin** and **Fiduccia-Mattheyses**, when

used alone and initialized with random assignments, generate significantly higher edge cuts, often 2-3 times larger than those from GNN-FM and METIS, despite achieving good balance.

In addition, **Hamiltonian Cycle** follows a topological traversal that tends to produce contiguous yet highly irregular partitions with large surface areas, which makes it ineffective at minimizing inter-processor communication. Similarly, geometric methods like **Coordinate** and **Inertial** bisection, based solely on spatial projections, also yield poorly results, showing that relying purely on geometric information is not enough for high-quality partitioning.

4.4 Qualitative Analysis

A visual comparison of the partitions generated by each algorithm is provided in Figure 19. This qualitative view complements the numerical data and offers insights into the geometric properties of the resulting subdomains.

The visual results strongly align with the quantitative findings:

- **GNN, GNN-FM, and METIS** (Rows 1-3) consistently produce high-quality partitions. The boundaries are smooth and well-defined, and the subdomains are contiguous and compact. This is the desired outcome for applications like FEM, as it minimizes the surface area-to-volume ratio, thereby reducing inter-processor communication costs.
- **Spectral K-Means** (Row 4) produces connected partitions, despite not explicitly enforcing connectivity. This is likely due to the smoothness of the spectral embedding, which clusters nodes with similar structural roles. However, some imbalance can still be observed across partitions, as k-Means clustering in the spectral space does not inherently optimize for partition sizes or balance.
- **Spectral Mean and Spectral Median** (Rows 5–6) produce partitions with better balance and contiguity than Spectral K-Means. The **Median-based** method, in particular, often results in visually cleaner and more balanced splits due to its symmetry-preserving threshold. However, both methods still underperform in cut quality. The **Mean-based** variant shows skewed partitions in some meshes (e.g., Blocker_2), likely due to asymmetric Fiedler vector distributions.
- **Kernighan-Lin and Fiduccia-Mattheyses** (Rows 7-8) produce non-contiguous partitions with boundaries visibly more irregular and less optimal than those from the GNN-based methods and METIS.
- The **Hamiltonian Cycle** (Row 9) produces snake-like partitions that, while contiguous,

have a very high surface area, making them inefficient for minimizing communication.

- The **geometric heuristics** (Rows 10-11) reveal their simplistic nature. **Coordinate** and **Inertial** bisection create simple planar cuts, which are oblivious to the mesh's underlying topology and lead to suboptimal results.

4.5 Discussion of Model Design and Observations

The strong performance of our GNN-based approach is attributable to several key design choices and observations made during development.

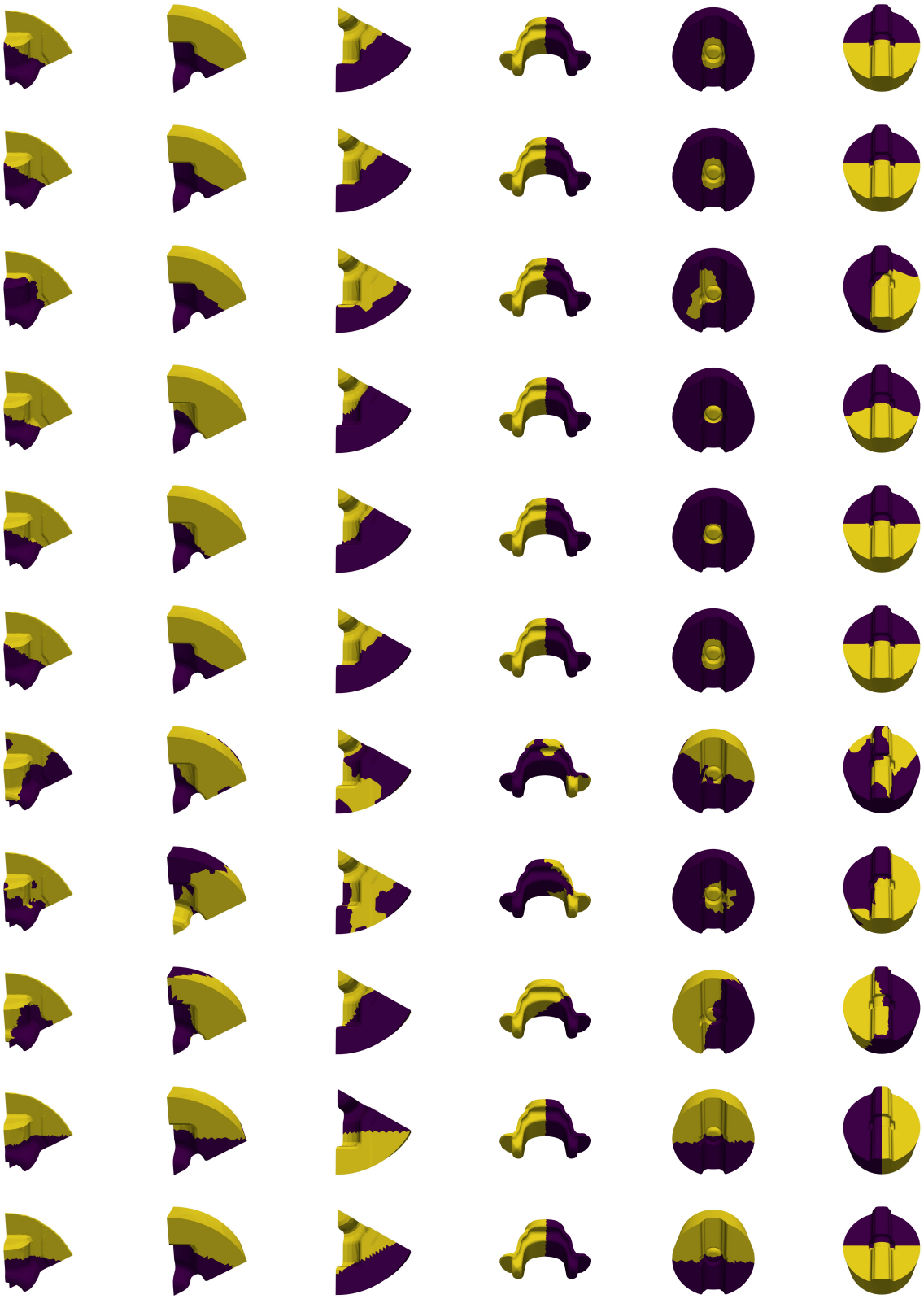
4.5.1 Effectiveness of the MMCut Loss Function

The choice of the **MMCut** loss function was critical for our primary experiments. As demonstrated by the theoretical analysis in Section 2.3.2.4, MMCut is the only one of the common cut-based metrics that inherently promotes balanced partitions on random graphs, whereas metrics like the standard Cut can favor skewed solutions. For the bisection task ($K = 2$) central to this work, our experiments validated this advantage, with the MMCut objective consistently yielding the high-quality, balanced partitions seen in Table 5.

However, we also observed that the differentiable relaxation of these combinatorial objectives has its limits. In exploratory tests with a higher number of partitions (e.g., $K > 16$), all three losses (**MMCut**, **NCut**, and **RCut**) tended to produce skewed or collapsed partitions. This suggests that while MMCut is theoretically more robust, a combination of numerical instabilities and the challenges of optimizing in a high-dimensional probabilistic space cause all three relaxed objectives to struggle with finding balanced solutions when K is large. Therefore, MMCut was the most effective choice for the bisection scenarios presented, but scaling these differentiable losses to many-way partitioning remains an open challenge.

4.5.2 Laplacian Eigenmaps as Positional Encoders

The most effective positional encoding was achieved using **Laplacian Eigenmaps (LaPE)**. For the bisection task ($K = 2$), we verified that using a single eigenvector, specifically the **Fiedler vector**, provided results as effective as using two or more eigenvectors. This is consistent with spectral graph theory, which shows that the Fiedler vector offers a natural basis for near-optimal bipartitioning. This finding enabled a more efficient and simplified model,



Source: Image from the author (2025)

Figure 19 – Visual comparison of partitioning results across algorithms and test meshes. Each row corresponds to a partitioning method, in the following top-down order: *GNN*, *GNN-FM*, *METIS*, *Spectral K-Means*, *Spectral Mean*, *Spectral Median*, *Kernighan-Lin*, *Fiduccia-Mattheyses*, *Hamiltonian Cycle*, *Coordinate* and *Inertial*. Each column represents a different Blocker mesh, indexed as Blocker_1 to Blocker_6.

relying on a single positional feature without compromising performance. Furthermore, we observed that in higher dimensions (i.e., when using multiple eigenvectors), attempts to mitigate sign ambiguity by flipping eigenvector directions did not significantly improve the final results, as the model converged to similar solutions regardless of sign configuration.

4.5.3 Comparison of GNN Backbones

During the development of the Feature Embedder module, we evaluated several GNN architectures to serve as the message-passing backbone. Our findings indicate that the choice of architecture presents a significant trade-off between partition quality, training time, and memory consumption.

We found that the **Graph Isomorphism Network (GIN)** produced the best results, achieving the lowest edge cuts for a given level of balance, which is why it was selected for our final model configuration. The **Graph Attention Network v2 (GATv2)** produced partitions of nearly the same quality as GIN. However, the attention mechanism, while powerful, introduced a considerable increase in training time and memory overhead, making it less efficient and scalable for our application.

On the other hand, simpler architectures like **GraphSAGE** and **Graph Convolutional Networks (GCNs)** were faster to train and had a much lower memory footprint. Despite this efficiency, the quality of the partitions they produced was not as good, resulting in higher edge cuts compared to GIN. This observation aligns with the theoretical understanding that GIN possesses greater expressive power than GCN and GraphSAGE, which can be limited in their ability to distinguish different graph structures. Ultimately, GIN offered the best compromise, delivering superior partition quality without the prohibitive computational cost of attention-based models.

4.5.4 Limitations of Alternative Encoders

Our experiments also highlighted the shortcomings of other encoding strategies, reinforcing our choice of LaPE.

- **SignNet:** This architecture fundamentally struggles with graph partitioning due to its sign-invariance property. For bisection ($K = 2$), the model learns from the magnitude of the Fiedler vector's components but is invariant to their sign. Since the sign is important to identify clusters, this invariance leads to an inability to distinguish the two partitions

correctly, resulting in non-contiguous regions. This problem was visualized to be even more pronounced for **k-way partitioning**. In this scenario, nodes are embedded in a multi-dimensional space using several eigenvectors. SignNet's invariance applies to each eigenvector independently, effectively making a node's spectral coordinate indistinguishable from its reflections across any axis. This ambiguity shatters the clusters in the spectral space, leading to visually fragmented partitions where a single partition consists of multiple, disconnected islands scattered across the mesh. This makes SignNet unsuitable for generating the contiguous subdomains required in FEM.

- **Structural Encoders:** Encoders based purely on local structure (e.g., **Random Walk Structural Encoders** or RWSE) also performed poorly. These methods capture local connectivity patterns but lack information about the global shape of the graph. This global perspective, which is naturally captured by the low-frequency eigenvectors of the graph Laplacian, is essential for finding globally optimal cuts in complex geometries.

4.6 Chapter Summary and Forward Look

This section rigorously evaluated the proposed GNN and GNN-FM frameworks across a diverse set of graphs, including industrial and unseen meshes. The evaluation began with a detailed experimental setup, specifying the hardware used, training batch size, optimizer, learning rate scheduling, and regularization strategies such as dropout and gradient clipping.

Model training followed an unsupervised regime using the differentiable Min-Max Cut (MMCut) objective. Loss curves were tracked across epochs to monitor convergence and generalization. The thoroughness of the training process and evaluation protocol ensures that the model's performance is not the result of overfitting or favorable initialization, but a product of principled design.

Quantitative analysis revealed that the models consistently outperformed or matched classical partitioning tools like METIS across most industrial meshes, achieving lower edge cuts and near-perfect balance. In 4 out of 6 test meshes, GNN-FM delivered better partitions in terms of communication cost, demonstrating its real-world applicability.

Qualitative visualizations supported these findings, showing visually cleaner and more contiguous partitions. Additional discussion addressed model behavior under different encoder settings (e.g., SignNet vs Laplacian Eigenmaps) and explored architectural sensitivity.

These findings validate the methodological choices and highlight the interplay be-

tween spectral encodings and GNN expressivity for partitioning. The results set the stage for the final section, which reflects on the contributions and opens avenues for future work, such as enforcing contiguity, scaling to multi-way partitions, and integrating other optimization techniques.

5 CONCLUSION

This work presented a novel, unsupervised framework for graph partitioning, specifically tailored for Finite Element Method (FEM) meshes, leveraging the power of Graph Neural Networks (GNNs). This final chapter synthesizes the work, summarizes the key findings, acknowledges the limitations of the proposed method, and outlines promising avenues for future research.

5.1 Summary of Work and Contributions

The primary objective of this research was to develop and evaluate an unsupervised learning framework capable of partitioning large-scale graphs, such as those derived from FEM simulations, into balanced and communication-efficient subdomains. The motivation stems from the critical need for effective domain decomposition in parallel scientific computing, where partitioning quality directly impacts performance by minimizing inter-processor communication (edge cuts) and ensuring even computational workloads (balance).

To this end, we proposed a modular, end-to-end GNN architecture composed of four main stages: a Positional and Structural Embedder, a Feature Embedder, a Feature Aggregator, and a Classifier. The key contributions of this work are:

- **An Unsupervised GNN Framework:** We designed a complete pipeline that learns to partition graphs without requiring any labeled data. This was achieved by formulating a differentiable loss function based on a relaxation of the **Min-Max Cut (MMCut)** objective (DING *et al.*, 2001), which allows the model to be trained directly on the structural properties of the graphs themselves.
- **Integration of Spectral and GNN Methods:** The framework effectively bridges classical spectral methods with modern deep learning. We utilized **Laplacian Eigenmaps** as positional encoders to provide the GNN with a global understanding of the graph topology, overcoming the common issue of featureless nodes in scientific meshes (BELKIN; NIYOGI, 2003). The GNN backbone, a **Graph Isomorphism Network (GIN)**, was chosen for its high expressive power in learning local structural patterns (XU *et al.*, 2019).
- **Hybrid Learning and Heuristic Refinement:** We introduced a hybrid approach that combines the predictive power of the GNN with the refinement capabilities of classical heuristics. By applying the linear-time **Fiduccia-Mattheyses (FM)** algorithm (FIDUC-

CIA; MATTHEYSES, 1982) as a post-processing step, we demonstrated a significant improvement in partition quality with negligible computational overhead.

- **Comprehensive Evaluation:** The proposed model was rigorously benchmarked against nine established algorithms, including the industry-standard **METIS** (KARYPIS, 1998), on a challenging set of out-of-distribution industrial meshes. This evaluation provided a clear and detailed comparison of the model's performance in terms of both quantitative metrics and qualitative partition characteristics.

This work successfully demonstrates that a GNN-based approach can learn the complex, non-linear patterns required for high-quality graph partitioning, achieving results that are competitive with state-of-the-art specialized tools.

5.2 Synwork of Key Findings

Our experimental results, detailed in Chapter 4, led to several important findings that validate our methodological choices.

- **Competitive Performance:** The proposed **GNN-FM** model demonstrated performance highly competitive with METIS. Across the six industrial "Blocker" meshes, our method consistently produced partitions with low edge cuts and near-perfect balance, even outperforming METIS on four of the six test cases. It significantly surpassed all other classical and geometric heuristics.
- **Superiority of MMCut for Bisection:** The choice of the **MMCut** loss function proved critical. As predicted by theory (DING *et al.*, 2001), it excelled at producing balanced partitions for the bisection task ($K = 2$), a clear advantage over other cut-based objectives that can favor skewed or trivial solutions.
- **Validation of Architectural Choices:** Our ablation studies and comparisons confirmed our design decisions. The **GIN** backbone offered the best trade-off between expressive power and computational efficiency, outperforming GCN and GraphSAGE in quality and GATv2 in training time and memory overhead. Furthermore, using just the first non-trivial Laplacian eigenvector (**the Fiedler vector** (FIEDLER, 1973)) as a positional encoding was sufficient and optimal for the bisection task, affirming classical spectral partitioning theory.
- **Explained Failure Modes of Alternatives:** The research successfully identified why certain advanced encoders were unsuitable for this task. **SignNet** (LIM *et al.*, 2022),

despite its theoretical appeal, produced non-contiguous partitions due to its sign-invariance property, which fundamentally conflicts with the goal of separating a graph into distinct halves. Similarly, purely local **structural encoders** (e.g., RWSE) failed to capture the necessary global information for effective partitioning.

In essence, the findings confirm that a carefully designed GNN, enriched with global spectral information and guided by a theoretically sound loss function, can effectively learn to solve the complex combinatorial problem of graph partitioning.

5.3 Limitations of the Study

Despite the promising results, this study has several limitations that must be acknowledged.

- **Lack of Contiguity Guarantee:** The primary limitation of the proposed GNN model is that it does not formally guarantee that the resulting partitions are contiguous. Because the final assignments are determined by a softmax classifier on a per-node basis, it is possible, though infrequent in our tests, for the model to create fragmented subdomains.
- **Scalability Challenges for Many-Way Partitioning:** While the differentiable loss functions performed well for bisection ($K = 2$), our exploratory experiments revealed that they suffer from numerical instabilities and a tendency to produce collapsed or skewed partitions when the number of target partitions (K) becomes large (e.g., $K > 16$). This indicates that the single-shot, relaxed-objective approach does not scale well for many-way partitioning, requiring the use of a recursive bisection to perform k-partitioning.
- **Pre-computation of Laplacian Eigenvectors:** The model's reliance on Laplacian Positional Encoder requires a pre-computation step to solve for the eigenvectors of the graph Laplacian. While the Lanczos algorithm is efficient, this spectral decomposition can still be a computational bottleneck for extremely large graphs when compared to non-spectral heuristic methods.
- **Dependence on Heuristic Refinement:** The best-performing model, GNN-FM, relied on a classical heuristic for final refinement. This indicates that while the GNN provides an excellent initial partition, it may not always capture the finest-grained local optima, which iterative refinement algorithms are designed to find.

5.4 Future Work

The findings and limitations of this work open up several exciting directions for future research.

- **Enforcing Partition Contiguity:** A key avenue for improvement is to incorporate a mechanism that explicitly enforces partition contiguity. This could be achieved by introducing a differentiable loss term that penalizes disconnected components within each partition or by exploring structured prediction models that can be layered on top of the GNN to ensure connected outputs.
- **Recursive Bisection for k-Way Partitioning:** To address the scalability issue with many-way partitioning, a promising approach is to apply the trained bisection model recursively. This hierarchical strategy, which is successfully employed by classical tools like METIS, would leverage our model's proven strength in the $K = 2$ case and is a natural extension of the current work.
- **End-to-End Learnable Encoders:** Instead of relying on fixed, pre-computed Laplacian eigenvectors, future work could explore end-to-end learnable positional and structural encodings. This would remove the spectral pre-computation bottleneck and could allow the model to learn task-specific embeddings that are more effective than the general-purpose LaPE.
- **Generalization Across Diverse Graph Domains:** The current model was trained on CAD-based FEM meshes. Evaluating and fine-tuning its performance on a wider range of graph types, such as social networks, citation graphs, or molecular structures, would test the limits of its generalization and could lead to a more universally applicable partitioning tool.
- **Deeper Integration of Learning and Heuristics:** Rather than using a heuristic as a simple post-processing step, future models could explore a deeper integration. For example, a GNN could be used to provide a learned policy within a reinforcement learning framework to guide the moves of an iterative refinement algorithm like Fiduccia-Mattheyses or Kernighan-Lin, potentially leading to superior results.

REFERENCES

ABADI, M.; BARHAM, P.; CHEN, J.; CHEN, Z.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; IRVING, G.; ISARD, M.; KUDLUR, M.; LEVENBERG, J.; MONGA, R.; MOORE, S.; MURRAY, D. G.; STEINER, B.; TUCKER, P.; VASUDEVAN, V.; WARDEN, P.; WICKE, M.; YU, Y.; ZHENG, X. Tensorflow: a system for large-scale machine learning. USENIX Association, USA, p. 265–283, 2016.

ARKIN, E. M.; HELD, M.; MITCHELL, J. S. B.; SKIENA, S. S. Hamiltonian triangulations for fast rendering. **The Visual Computer**, v. 12, n. 9, p. 429–444, 1996. ISSN 1432-2315. Available at: <https://doi.org/10.1007/BF01782475>.

BA, J. L.; KIROS, J. R.; HINTON, G. E. **Layer Normalization**. 2016. Available at: <https://arxiv.org/abs/1607.06450>.

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. In: **International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2015.

Bartholdi III, J. J.; GOLDSMAN, P. The vertex-adjacency dual of a triangulated irregular network has a hamiltonian cycle. **Operations Research Letters**, v. 32, n. 4, p. 304–308, 2004. ISSN 0167-6377. Available at: <https://www.sciencedirect.com/science/article/pii/S0167637703001536>.

BELKIN, M.; NIYOGI, P. Laplacian eigenmaps for dimensionality reduction and data representation. **Neural Computation**, v. 15, n. 6, p. 1373–1396, 2003.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE Transactions on Neural Networks**, v. 5, n. 2, p. 157–166, 1994.

BIANCHI, F. M.; GRATTAROLA, D.; ALIPPI, C. **Mincut Pooling in Graph Neural Networks**. 2020. Available at: <https://openreview.net/forum?id=BkxfshNYwB>.

BIEWALD, L. **Experiment Tracking with Weights and Biases**. 2020. Software available from wandb.com. Available at: <https://www.wandb.com/>.

BOTTOU, L. Large-scale machine learning with stochastic gradient descent. In: LECHEVALLIER, Y.; SAPORTA, G. (Ed.). **Proceedings of COMPSTAT'2010**. Heidelberg: Physica-Verlag HD, 2010. p. 177–186.

BRO, R.; ACAR, E.; KOLDA, T. G. Resolving the sign ambiguity in the singular value decomposition. **Journal of Chemometrics**, v. 22, n. 2, p. 135–140, 2008. Available at: <https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/abs/10.1002/cem.1122>.

BRODY, S.; ALON, U.; YAHAV, E. How attentive are graph attention networks? **CoRR**, abs/2105.14491, 2021. Available at: <https://arxiv.org/abs/2105.14491>.

BRONSTEIN, M. M.; BRUNA, J.; COHEN, T.; VELICKOVIC, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. **CoRR**, abs/2104.13478, 2021. Available at: <https://arxiv.org/abs/2104.13478>.

BUTLER, K. T.; DAVIES, D. W.; CARTWRIGHT, H.; ISAYEV, O.; WALSH, A. Machine learning for molecular and materials science. **Nature**, Nature Publishing Group, v. 559, n. 7715, p. 547–555, 2018. ISSN 1476-4687. Available at: <https://doi.org/10.1038/s41586-018-0337-2>.

CAI, J.; LUO, B.; HUANG, J. Graphnorm: A principled approach to accelerating graph neural network training. In: **Proceedings of the 39th International Conference on Machine Learning (ICML)**. [s.n.], 2021. p. 1209–1218. Available at: <https://arxiv.org/abs/2009.03294>.

CAO, S.; LU, W.; XU, Q. Grarep: Learning graph representations with global structural information. In: **Proceedings of the 24th ACM International on Conference on Information and Knowledge Management**. New York, NY, USA: Association for Computing Machinery, 2015. (CIKM '15), p. 891–900. ISBN 9781450337946. Available at: <https://doi.org/10.1145/2806416.2806512>.

CHAN, P.; SCHLAG, M.; ZIEN, J. Spectral k-way ratio-cut partitioning and clustering. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 13, n. 9, p. 1088–1096, 1994.

CHO, K.; MERRIENBOER, B. van; GÜLÇEHRE, Ç.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. **CoRR**, abs/1406.1078, 2014. Available at: <http://arxiv.org/abs/1406.1078>.

COOK, S. A. The complexity of theorem-proving procedures. In: **Proceedings of the Third Annual ACM Symposium on Theory of Computing**. New York, NY, USA: Association for Computing Machinery, 1971. (STOC '71), p. 151–158. ISBN 9781450374644. Available at: <https://doi.org/10.1145/800157.805047>.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals and Systems**, Springer, v. 2, n. 4, p. 303–314, 1989. ISSN 1435-568X. Available at: <https://doi.org/10.1007/BF02551274>.

DEVLIN, J.; CHANG, M.; LEE, K.; TOUTANOVA, K. BERT: pre-training of deep bidirectional transformers for language understanding. **CoRR**, abs/1810.04805, 2018. Available at: <http://arxiv.org/abs/1810.04805>.

DING, C.; HE, X.; ZHA, H.; GU, M.; SIMON, H. A min-max cut algorithm for graph partitioning and data clustering. In: **Proceedings 2001 IEEE International Conference on Data Mining**. [S.l.: s.n.], 2001. p. 107–114.

DOSOVITSKIY, A.; BEYER, L.; KOLESNIKOV, A.; WEISSENBORN, D.; ZHAI, X.; UNTERTHINER, T.; DEHGHANI, M.; MINDERER, M.; HEIGOLD, G.; GELLY, S.; USZKOREIT, J.; HOULSBY, N. An image is worth 16x16 words: Transformers for image recognition at scale. **CoRR**, abs/2010.11929, 2020. Available at: <https://arxiv.org/abs/2010.11929>.

DWIVEDI, V. P.; JOSHI, C. K.; LUU, A. T.; LAURENT, T.; BENGIO, Y.; BRESSON, X. Benchmarking graph neural networks. **J. Mach. Learn. Res.**, JMLR.org, v. 24, n. 1, Jan. 2023. ISSN 1532-4435.

DWIVEDI, V. P.; LUU, A. T.; LAURENT, T.; BENGIO, Y.; BRESSON, X. Graph neural networks with learnable structural and positional representations. In: **International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2022.

ELMAN, J. L. Finding structure in time. **Cognitive Science**, v. 14, n. 2, p. 179–211, 1990. ISSN 0364-0213. Available at: <https://www.sciencedirect.com/science/article/pii/036402139090002E>.

FELDMAN, O.; BOYARSKI, A.; FELDMAN, S.; KOGAN, D.; MENDELSON, A.; BASKIN, C. Weisfeiler and leman go infinite: Spectral and combinatorial pre-colorings. **Transactions on Machine Learning Research**, 2023. ISSN 2835-8856. Available at: <https://openreview.net/forum?id=YJDqQSAuB6>.

FERGUSON, K.; CHEN, Y. hsuan; CHEN, Y.; GILLMAN, A.; HARDIN, J.; KARA, L. B. **Topology-Agnostic Graph U-Nets for Scalar Field Prediction on Unstructured Meshes**. 2024. Available at: <https://arxiv.org/abs/2410.06406>.

FEY, M.; LENSSEN, J. E. Fast graph representation learning with PyTorch Geometric. In: **ICLR Workshop on Representation Learning on Graphs and Manifolds**. [S.l.: s.n.], 2019.

FIDUCCIA, C.; MATTHEYSES, R. A linear-time heuristic for improving network partitions. In: **19th Design Automation Conference**. [S.l.: s.n.], 1982. p. 175–181.

FIEDLER, M. Algebraic connectivity of graphs. **Czechoslovak Mathematical Journal**, Institute of Mathematics, Academy of Sciences of the Czech Republic, v. 23, n. 2, p. 298–305, 1973. Available at: <http://eudml.org/doc/12723>.

GATTI, A.; HU, Z.; SMIDT, T.; NG, E. G.; GHYSELS, P. Deep learning and spectral embedding for graph partitioning. In: **Proceedings of the 2022 SIAM Conference on Parallel Processing for Scientific Computing (PP)**. [s.n.], 2022. p. 25–36. Available at: <https://epubs.siam.org/doi/abs/10.1137/1.9781611977141.3>.

GATTI, A.; HU, Z.; SMIDT, T.; NG, E. G.; GHYSELS, P. Graph partitioning and sparse matrix ordering using reinforcement learning and graph neural networks. **J. Mach. Learn. Res.**, JMLR.org, v. 23, n. 1, Jan. 2022. ISSN 1532-4435.

GEHRING, J.; AULI, M.; GRANGIER, D.; YARATS, D.; DAUPHIN, Y. N. Convolutional sequence to sequence learning. In: **Proceedings of the 34th International Conference on Machine Learning - Volume 70**. [S.l.]: JMLR.org, 2017. (ICML'17), p. 1243–1252.

GEUZAINE, C.; REMACLE, J.-F. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. **International Journal for Numerical Methods in Engineering**, v. 79, n. 11, p. 1309–1331, 2009. Available at: <https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579>.

GILMER, J.; SCHOENHOLZ, S. S.; RILEY, P. F.; VINYALS, O.; DAHL, G. E. Neural message passing for quantum chemistry. In: **Proceedings of the 34th International Conference on Machine Learning - Volume 70**. [S.l.]: JMLR.org, 2017. (ICML'17), p. 1263–1272.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: **International Conference on Artificial Intelligence and Statistics**. [s.n.], 2010. Available at: <https://api.semanticscholar.org/CorpusID:5575601>.

GROVER, A.; LESKOVEC, J. node2vec: Scalable feature learning for networks. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2016. (KDD '16), p. 855–864. ISBN 9781450342322. Available at: <https://doi.org/10.1145/2939672.2939754>.

GU, M.; ZHA, H.; DING, C.; HE, X.; SIMON, H. D. **Spectral relaxation models and structure analysis for k-way graph clustering and bi-clustering**. [S.l.], 2001.

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

HANOCKA, R.; HERTZ, A.; FISH, N.; GIRYES, R.; FLEISHMAN, S.; COHEN-OR, D. Meshcnn: a network with an edge. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 4, Jul. 2019. ISSN 0730-0301. Available at: <https://doi.org/10.1145/3306346.3322959>.

HARRIS, C. R.; MILLMAN, K. J.; WALT, S. J. van der; GOMMERS, R.; VIRTANEN, P.; COURNAPEAU, D.; WIESER, E.; TAYLOR, J.; BERG, S.; SMITH, N. J.; KERN, R.; PICUS, M.; HOYER, S.; KERKWIJK, M. H. van; BRETT, M.; HALDANE, A.; RíO, J. F. del; WIEBE, M.; PETERSON, P.; GéRARD-MARCHANT, P.; SHEPPARD, K.; REDDY, T.; WECKESSER, W.; ABBASI, H.; GOHLKE, C.; OLIPHANT, T. E. Array programming with numpy. **Nature**, v. 585, n. 7825, p. 357–362, 2020. ISSN 1476-4687. Available at: <https://doi.org/10.1038/s41586-020-2649-2>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 770–778, 2015. Available at: <https://api.semanticscholar.org/CorpusID:206594692>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. **IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 770–778, 2015. Available at: <https://api.semanticscholar.org/CorpusID:206594692>.

HENDRYCKS, D.; GIMPEL, K. Gaussian error linear units (gelus). **arXiv: Learning**, 2016. Available at: <https://api.semanticscholar.org/CorpusID:125617073>.

HINTON, G. E. Training products of experts by minimizing contrastive divergence. **Neural Computation**, v. 14, n. 8, p. 1771–1800, 2002.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Networks**, Pergamon, v. 2, n. 5, p. 359–366, 1989. ISSN 0893-6080. Available at: <https://www.sciencedirect.com/science/article/pii/0893608089900208>.

IOFFE, S.; SZEGEDY, C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: **Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37**. [S.l.]: JMLR.org, 2015. (ICML'15), p. 448–456.

IRWIN, J. J.; STERLING, T.; MYSINGER, M. M.; BOLSTAD, E. S.; COLEMAN, R. G. ZINC: A free tool to discover chemistry for biology. **Journal of Chemical Information and Modeling**, American Chemical Society, v. 52, n. 7, p. 1757–1768, 2012. ISSN 1549-9596. Available at: <https://doi.org/10.1021/ci3001277>.

JUMPER, J.; EVANS, R.; PRITZEL, A.; GREEN, T.; FIGURNOV, M.; RONNEBERGER, O.; TUNYASUVUNAKOOL, K.; BATES, R.; ŽíDEK, A.; POTAPENKO, A.; BRIDGLAND, A.; MEYER, C.; KOHL, S. A. A.; BALLARD, A. J.; COWIE, A.; ROMERA-PAREDES,

B.; NIKOLOV, S.; JAIN, R.; ADLER, J.; BACK, T.; PETERSEN, S.; REIMAN, D.; CLANCY, E.; ZIELINSKI, M.; STEINEGGER, M.; PACHOLSKA, M.; BERGHAMMER, T.; BODENSTEIN, S.; SILVER, D.; VINYALS, O.; SENIOR, A. W.; KAVUKCUOGLU, K.; KOHLI, P.; HASSABIS, D. Highly accurate protein structure prediction with alphafold. **Nature**, Nature Publishing Group, v. 596, n. 7873, p. 583–589, 2021. ISSN 1476-4687. Available at: <https://doi.org/10.1038/s41586-021-03819-2>.

KARYPIS, G. **METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 5.0**. [S.l.], 1998.

KARYPIS, G.; KUMAR, V. Parallel multilevel k-way partitioning scheme for irregular graphs. In: **Proceedings of the 1996 ACM/IEEE Conference on Supercomputing**. USA: IEEE Computer Society, 1996. (Supercomputing '96), p. 35–es. ISBN 0897918541. Available at: <https://doi.org/10.1145/369028.369103>.

KARYPIS, G.; KUMAR, V. **hMETIS: A Hypergraph Partitioning Package Version 1.5.3**. [S.l.], 1998.

KERNIGHAN, B. W.; LIN, S. An efficient heuristic procedure for partitioning graphs. **The Bell System Technical Journal**, v. 49, n. 2, p. 291–307, 1970.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: **International Conference on Learning Representations (ICLR)**. arXiv, 2015. Accepted as a conference paper at ICLR 2015. Available at: <https://arxiv.org/abs/1412.6980>.

KIPF, T.; WELLING, M. Semi-supervised classification with graph convolutional networks. **ArXiv**, abs/1609.02907, 2016. Available at: <https://api.semanticscholar.org/CorpusID:3144218>.

KRAFT, H.; PROCESI, C. **Classical invariant theory: a primer**. 1996. Available at: <https://api.semanticscholar.org/CorpusID:116057187>.

KREUZER, D.; BEAINI, D.; HAMILTON, W. L.; LéTOURNEAU, V.; TOSSOU, P. Rethinking graph transformers with spectral attention. In: **Proceedings of the 35th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2021. (NIPS '21). ISBN 9781713845393.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 60, n. 6, p. 84–90, May 2017. ISSN 0001-0782. Available at: <https://doi.org/10.1145/3065386>.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998.

LEVINE, S.; FINN, C.; DARRELL, T.; ABBEEL, P. End-to-end training of deep visuomotor policies. **J. Mach. Learn. Res.**, JMLR.org, v. 17, n. 1, p. 1334–1373, Jan. 2016. ISSN 1532-4435.

LI, P.; WANG, Y.; WANG, H.; LESKOVEC, J. Distance encoding: design provably more powerful neural networks for graph representation learning. In: **Proceedings of the 34th**

International Conference on Neural Information Processing Systems. Red Hook, NY, USA: Curran Associates Inc., 2020. (NIPS '20). ISBN 9781713829546.

LIM, D.; ROBINSON, J. D.; ZHAO, L.; SMIDT, T.; SRA, S.; MARON, H.; JEGELKA, S. Sign and basis invariant networks for spectral graph representation learning. In: **ICLR 2022 Workshop on Geometrical and Topological Representation Learning**. [s.n.], 2022. Available at: <https://openreview.net/forum?id=BlM64by6gc>.

LIU, Z.; WANG, Y.; VAIDYA, S.; RUEHLE, F.; HALVERSON, J.; SOLJACIC, M.; HOU, T. Y.; TEGMARK, M. KAN: Kolmogorov–arnold networks. In: **International Conference on Learning Representations**. [s.n.], 2025. Available at: <https://openreview.net/forum?id=Ozo7qJ5vZi>.

LOSHCHILOV, I.; HUTTER, F. Decoupled weight decay regularization. **International Conference on Learning Representations (ICLR)**, 2019. Available at: <https://openreview.net/forum?id=Bkg6RiCqY7>.

MARON, H.; BEN-HAMU, H.; SHAMIR, N.; LIPMAN, Y. Invariant and equivariant graph networks. In: **International Conference on Learning Representations**. [s.n.], 2019. Available at: <https://openreview.net/forum?id=Syx72jC9tm>.

MCSHERRY, F. Spectral partitioning of random graphs. In: **Proceedings 42nd IEEE Symposium on Foundations of Computer Science**. [S.l.: s.n.], 2001. p. 529–537.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: **Proceedings of the 27th International Conference on International Conference on Machine Learning**. Madison, WI, USA: Omnipress, 2010. (ICML'10), p. 807–814. ISBN 9781605589077.

NAZI, A.; HANG, W.; GOLDIE, A.; RAVI, S.; MIRHOSEINI, A. GAP: generalizable approximate graph partitioning framework. **CoRR**, abs/1903.00614, 2019. Available at: <http://arxiv.org/abs/1903.00614>.

NESTEROV, Y. A method for solving the convex programming problem with convergence rate $o(1/k^2)$.**Proceedings of the USSR Academy of Sciences**, *v.* 269, *p*.543 − −547, 1983.

NG, A. Y.; JORDAN, M. I.; WEISS, Y. On spectral clustering: analysis and an algorithm. In: **Proceedings of the 15th International Conference on Neural Information Processing Systems: Natural and Synthetic**. Cambridge, MA, USA: MIT Press, 2001. (NIPS'01), p. 849–856.

OVSJANIKOV, M.; SUN, J.; GUIBAS, L. Global intrinsic symmetries of shapes. In: **Proceedings of the Symposium on Geometry Processing**. Goslar, DEU: Eurographics Association, 2008. (SGP '08), p. 1341–1348.

PAGE, L.; BRIN, S.; MOTWANI, R.; WINOGRAD, T. The pagerank citation ranking : Bringing order to the web. In: **The Web Conference**. [s.n.], 1999. Available at: <https://api.semanticscholar.org/CorpusID:1508503>.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KöPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.;

BAI, J.; CHINTALA, S. Pytorch: an imperative style, high-performance deep learning library. In: ____. **Proceedings of the 33rd International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2019.

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

POTHEN, A. **Graph Partitioning Algorithms with Applications to Scientific Computing**. USA, 1997.

POTHEN, A.; SIMON, H. D.; LIOU, K.-P. Partitioning sparse matrices with eigenvectors of graphs. **SIAM Journal on Matrix Analysis and Applications**, v. 11, n. 3, p. 430–452, 1990. Available at: <https://doi.org/10.1137/0611030>.

RADFORD, A.; WU, J.; CHILD, R. *et al.* Language models are unsupervised multitask learners. **OpenAI Blog**, v. 1, n. 8, p. 9, 2019.

RAFFEL, C.; SHAZEER, N.; ROBERTS, A.; LEE, K.; NARANG, S.; MATENA, M.; ZHOU, Y.; LI, W.; LIU, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. **J. Mach. Learn. Res.**, JMLR.org, v. 21, n. 1, Jan. 2020. ISSN 1532-4435.

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. Searching for activation functions. **ArXiv**, abs/1710.05941, 2018. Available at: <https://api.semanticscholar.org/CorpusID:10919244>.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, v. 65 6, p. 386–408, 1958. Available at: <https://api.semanticscholar.org/CorpusID:12781225>.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

SIMON, H. Partitioning of unstructured problems for parallel processing. **Computing Systems in Engineering**, v. 2, n. 2, p. 135–148, 1991. ISSN 0956-0521. Parallel Methods on Large-scale Structural Analysis and Physics Applications. Available at: <https://www.sciencedirect.com/science/article/pii/095605219190014V>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **CoRR**, abs/1409.1556, 2014. Available at: <https://api.semanticscholar.org/CorpusID:14124313>.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. **J. Mach. Learn. Res.**, JMLR.org, v. 15, n. 1, p. 1929–1958, Jan. 2014. ISSN 1532-4435.

SULLIVAN, C.; KASZYNSKI, A. **PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)**. The Open Journal, 2019. 1450 p. <https://doi.org/10.5281/zenodo.3406468>. Version 0.25.3. Available at: <https://doi.org/10.21105/joss.01450>.

SUTSKEVER, I.; MARTENS, J.; DAHL, G.; HINTON, G. On the importance of initialization and momentum in deep learning. In: **Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28**. [S.l.]: JMLR.org, 2013. (ICML'13), p. III–1139–III–1147.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. **ArXiv**, abs/1905.11946, 2019. Available at: <https://api.semanticscholar.org/CorpusID:167217261>.

TANG, J.; QU, M.; WANG, M.; ZHANG, M.; YAN, J.; MEI, Q. Line: Large-scale information network embedding. In: **Proceedings of the 24th International Conference on World Wide Web**. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2015. (WWW '15), p. 1067–1077. ISBN 9781450334693. Available at: <https://doi.org/10.1145/2736277.2741093>.

TOLSTIKHIN, I.; HOULSBY, N.; KOLESNIKOV, A.; BEYER, L.; ZHAI, X.; UNTERTHINER, T.; YUNG, J.; STEINER, A.; KEYSERS, D.; USZKOREIT, J.; LUCIC, M.; DOSOVITSKIY, A. Mlp-mixer: an all-mlp architecture for vision. In: **Proceedings of the 35th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2021. (NIPS '21). ISBN 9781713845393.

TOUVRON, H.; LAVRIL, T.; IZACARD, G.; MARTINET, X.; LACHAUX, M.-A.; LACROIX, T.; ROZIÈRE, B.; GOYAL, N.; HAMBRO, E.; AZHAR, F.; RODRIGUEZ, A.; JOULIN, A.; GRAVE, E.; LAMPLE, G. Llama: Open and efficient foundation language models. **ArXiv**, abs/2302.13971, 2023. Available at: <https://api.semanticscholar.org/CorpusID:257219404>.

UWIMANA, C.; ZHOU, S.; YANG, L.; LI, Z.; MUTAGISHA, N.; NIYONGABO, E.; ZHOU, B. Segmentation of cad models using hybrid representation. **Virtual Reality Intelligent Hardware**, v. 7, n. 2, p. 188–202, 2025. ISSN 2096-5796. Available at: <https://www.sciencedirect.com/science/article/pii/S2096579625000014>.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L. u.; POLOSUKHIN, I. Attention is all you need. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2017. v. 30.

VELIČKOVIĆ, P.; CUCURULL, G.; CASANOVA, A.; ROMERO, A.; LIÒ, P.; BENGIO, Y. **Graph Attention Networks**. 2018. Available at: <https://openreview.net/forum?id=rJXMpikCZ>.

VIRTANEN, P.; GOMMERS, R.; OLIPHANT, T. E.; HABERLAND, M.; REDDY, T.; COURNAPEAU, D.; BUROVSKI, E.; PETERSON, P.; WECKESSER, W.; BRIGHT, J. *et al.* Scipy 1.0: Fundamental algorithms for scientific computing in python. **Nature Methods**, Nature Publishing Group, v. 17, n. 3, p. 261–272, 2020.

WILLIS, K. D. D.; PU, Y.; LUO, J.; CHU, H.; DU, T.; LAMBOURNE, J. G.; SOLAR-LEZAMA, A.; MATUSIK, W. Fusion 360 gallery: a dataset and environment for programmatic cad construction from human design sequences. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 40, n. 4, Jul. 2021. ISSN 0730-0301. Available at: <https://doi.org/10.1145/3450626.3459818>.

WU, Q.; CHAO, K.; LEE, R. The npo-completeness of the longest hamiltonian cycle problem. **Information Processing Letters**, v. 65, n. 3, p. 119–123, 1998. ISSN 0020-0190. Available at: <https://www.sciencedirect.com/science/article/pii/S0020019098000039>.

WU, Y.; HE, K. Group normalization. In: **Proceedings of the European Conference on Computer Vision (ECCV)**. Berlin, Heidelberg: Springer-Verlag, 2018. p. 3–19. ISBN 978-3-030-01260-1. Available at: <https://doi.org/10.1007/978-3-030-01261-8_1>.

XU, K.; HU, W.; LESKOVEC, J.; JEGELKA, S. How powerful are graph neural networks? In: **International Conference on Learning Representations**. [s.n.], 2019. Available at: <https://openreview.net/forum?id=ryGs6iA5Km>.

XU, K.; LI, C.; TIAN, Y.; SONOBE, T.; KAWARABAYASHI, K.-i.; JEGELKA, S. Representation learning on graphs with jumping knowledge networks. In: DY, J.; KRAUSE, A. (Ed.). **Proceedings of the 35th International Conference on Machine Learning**. PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 5453–5462. Available at: <https://proceedings.mlr.press/v80/xu18c.html>.

YE, Z.; KUMAR, Y. J.; SING, G. O.; SONG, F.; WANG, J. A comprehensive survey of graph neural networks for knowledge graphs. **IEEE Access**, v. 10, p. 75729–75741, 2022.

YING, C.; CAI, T.; LUO, S.; ZHENG, S.; KE, G.; HE, D.; SHEN, Y.; LIU, T.-Y. Do transformers really perform bad for graph representation? In: **Proceedings of the 35th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2021. (NIPS '21). ISBN 9781713845393.

ZAHEER, M.; KOTTUR, S.; RAVANBHAKHSH, S.; PóCZOS, B.; SALAKHUTDINOV, R.; SMOLA, A. J. Deep sets. In: **Proceedings of the 31st International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 3394–3404. ISBN 9781510860964.

ZHANG, J.; DONG, Y.; WANG, Y.; TANG, J.; DING, M. Prone: fast and scalable network representation learning. In: **Proceedings of the 28th International Joint Conference on Artificial Intelligence**. [S.l.]: AAAI Press, 2019. (IJCAI'19), p. 4278–4284. ISBN 9780999241141.

ZHAO, L.; AKOGLU, L. Pairnorm: Tackling oversmoothing in gnns. In: **International Conference on Learning Representations (ICLR)**. [s.n.], 2020. Available at: <https://arxiv.org/abs/1909.12223>.

ZHOU, J.; CUI, G.; HU, S.; ZHANG, Z.; YANG, C.; LIU, Z.; WANG, L.; LI, C.; SUN, M. Graph neural networks: A review of methods and applications. **AI Open**, v. 1, p. 57–81, 2020. ISSN 2666-6510. Available at: <https://www.sciencedirect.com/science/article/pii/S2666651021000012>.

ZHOU, K.; CUI, P.; YANG, S. Towards deeper graph neural networks with differentiable group normalization. In: **Advances in Neural Information Processing Systems (NeurIPS)**. [s.n.], 2020. Available at: <https://arxiv.org/abs/2006.06972>.

APPENDIX A – HYPERPARAMETERS

The proposed model is based on a Graph Neural Network architecture, specifically using the Graph Isomorphism Network (GIN) as the backbone. It includes five encoder layers with GELU activation, layer normalization, and a final softmax output layer. Laplacian eigenvectors are used as positional encodings to provide geometric context for message passing.

The training pipeline relies on gradient descent optimization with a cosine learning rate scheduler and standard regularization techniques such as weight decay and gradient clipping. The loss function is a differentiable approximation of the Min-Max Cut (MMCut) objective, combined with a balance regularization term. Training was conducted on a single GPU using mini-batches of size 128, and all experiments were tracked using *Weights & Biases* (BIEWALD, 2020).

Table 6 – Summary of model architecture and training setup.

(a) Model Configuration

Hyperparameter	**Value**
Architecture	GNNModel
Base GNN	GIN
Activation Function	GELU
Normalization	Layer Norm
Dropout Rate	0,0
Hidden Dimension	32
Output Dimension	2
Encoder Layers	5
JK Mode	max
Last Layer	softmax
Positional Encoding	Laplacian
Positional Features	1
Graph Features	0

(b) Training Setup

Hyperparameter	**Value**
Loss Functions	Expected MMCut, Balance
Loss Weights	1,0, 1,0
Learning Rate	1×10^{-3}
Weight Decay	1×10^{-6}
Scheduler	Cosine
Epochs	1 500
Batch Size	128
Grad Clipping	1,0
Precision	32-bit
Accelerator	GPU
Devices	1
Workers	8
Prefetch Factor	16
Pin Memory	True
Shuffle	True
Logger	Weight & Biases

Source: Table from the author (2025)