SAPIENZA
UNIVERSITÀ DI ROMA

Deep Learning with Structured Data: Topology Inference and Higher-Order Modeling

Faculty of Information Engineering, Informatics, and Statistics

Dottorato di Ricerca in Data Science – XXXVII Ciclo

Candidate

Lev Telyatnikov

ID number 1902392

Thesis Advisor

Prof. Simone Scardapane

2023/2024

Thesis defended on 22 January 2025
in front of a Board of Examiners composed by:

Prof. Anna Monreale (chairman)

Prof. Silvia Salini

Prof. Ida D'attoma

Deep Learning with Structured Data: Topology Inference and Higher-Order Modeling

Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: lev.telyatnikovv@gmail.com

Abstract

Relational data –broadly defined as a set of data points and the relationships between them– plays a foundational role across diverse real-world applications, from social networks and biological systems to recommendation engines. For decades, graph representation has been the dominant paradigm for describing relational data, providing a framework that captures pairwise interactions between entities. In recent years, Graph Neural Networks (GNNs) have emerged as a powerful tool that enables the learning of complex patterns and dependencies within relational data.

GNNs have revolutionized relational data processing. However, traditional graph representation is inherently limited to encoding pairwise relations between entities. This is a fundamental constraint, as many real-world systems exhibit multi-way or higher-order interactions, where entities interact simultaneously in groups of three or more. These n-body interactions cannot be adequately captured by pairwise edges alone. Addressing this limitation requires moving beyond graphs to richer, more expressive frameworks. Recent advances in topological deep learning leverage concepts from algebraic topology—such as hypergraphs, simplicial complexes, and cell complexes—to enable higher-order network representation and modeling.

The pairwise or higher-order relationships –also referred to as topology– of relational data may present multiple challenges, such as missingness, redundancy, or even the absence of relations altogether. This last scenario, where the topology is entirely absent, opens up an intriguing possibility: any type of data may possess an underlying structure, even when it is not explicitly defined. Topology learning aims to infer underlying –in other words, latent– topology from the data itself.

This thesis contributes to methodological and practical aspects of relational data modeling, with a focus on the challenges of topology learning and higher-order network modeling. In this study, we address key challenges in these areas in multiple ways, including but not limited to: introducing novel methodologies for both graph and higher-order latent topology inference and demonstrating their effectiveness across various applications, including missing data imputation, network traffic compression, and goal-oriented semantic communication tasks. Introducing a novel conceptualization of homophily in higher-order networks; developing the MultiSet framework—capable of encompassing most current hypergraph neural network architectures—; creating the first benchmarking framework (TopoBenchmark); and proposing a new approach that combines sequence modeling with topological representations to enable efficient information propagation across different ranks.

Contents

List of Acronyms

CC	Cell Complex
CCC	Combinatorial Complex
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DCM	Differentiable Cell Complex Module
DGM	Differentiable Graph Module
EGG-GAE	EdGe Generation Graph AutoEncoder
GDL	Geometric Deep Learning
GINN	Graph Imputer Neural Network
GNN	Graph Neural Network
HG	Hypergraph
HNN	Hypergraph Neural Network
HOMP	Higher-Order Message Passing
ISP	Internet Service Providers
JSCC	Joint Source-Channel Coding
LGI	Latent Graph Inference
LGL	Latent Graph Learning
LTI	Latent Topology Inference
MAR	Missing at Random
MCAR	Missing Completely at Random
MDI	Missing Data Imputation
ML	Machine Learning
MLP	Multi-Layer Perceptron

MNAR	Missing Not at Random
MP	Message Passing
PIM	Polygon Inference Module
SC	Simplicial Complex
SNR	Signal-to-Noise Ratio
SSM	State Space Model
TB	TopoBenchmark
TDL	Topological Deep Learning
TNN	Topological Neural Network

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Section 1.1 introduces the field of relational data representation. Next, Section 1.2 outlines the key challenges addressed in this thesis. Finally, Section 1.3 summarizes the main contributions of this work and provides the manuscript's structure.

1.1 On Relation Data Representational Learning

Relational data is ubiquitous in the real world, broadly defined as a set of data points and the relationships between them, collectively referred to as a topology over the data. Today, the vast majority of relational data is represented using graph topology, which captures pairwise interactions between data entities. Graph-based data modeling typically relies on a Message Passing (MP) mechanism to aggregate local neighborhood information and learn efficient representations at the node, edge, or graph level. Neural networks designed to model graph data are known as Graph Neural Networks (GNNs). They were introduced in [180] and numerous variants have been developed since [139, 214, 95, 39].

GNNs have revolutionized relational data processing, facilitating the systematic modeling of data from various real-world applications such as social networks, biological systems, and recommendation systems [263, 232]. However, graphs are inherently limited to encoding pairwise relations. As a result, they struggle to capture the complex, higher-order interactions that naturally occur in many real-world systems. For instance, social dynamics often involve group relations among multiple individuals [23], ecological systems exhibit intricate many-body connections [144], and protein interactions frequently involve multiple atoms [150]. To effectively model such complex data, it is essential to move beyond traditional graph structures for relational data representation.

Recent advancements have demonstrated that algebraic topology concepts, such as simplicial complexes [33], cell complexes [30], and combinatorial complexes [92], provide a powerful framework for encoding multi-element relationships. These topological structures are effectively modeled through Higher-Order Message Passing (HOMP), which generalizes the graph MP mechanism to capture interactions within higher-order neighborhoods. Models that incorporate HOMP and learn from data represented over discrete topological domains are referred to as Topological Neural Networks (TNNs). Extending beyond pairwise interactions, this approach offers a more expressive and flexible framework for relational data modeling of complex systems. This methodology is one of the key branches of the rapidly emerging field known as Topological Deep Learning (TDL) [265, 159, 178], which aims to integrate algebraic topology concepts and topological data analysis tools with the adaptive learning capabilities of neural networks. Further details are provided in Chapter 2.

1.2 Challenges and Limitations

The landscape of neural network-based models has undergone a dramatic transformation over the past decade. This evolution has been primarily driven by two key factors: the increasing availability of high-quality, large-scale datasets and the rapid advancements in computational power. These developments have enabled researchers and practitioners to design and implement increasingly complex models capable of capturing intricate patterns and relationships within data.

Topological data representation learning has emerged as a powerful paradigm, achieving remarkable success across various real-world applications [126, 110, 14, 46, 170]. Despite its achievements, there are numerous challenges, particularly when it comes to modeling higher-order relations that go beyond pairwise interactions.

This section delves into the crucial challenges that this thesis addresses in the field of relational data modeling. The challenges we explore span two interconnected areas: *topology learning* and *higher-order network modeling*. By describing the key challenges that arise in these two areas and their causes, we establish the foundation for the novel approaches introduced in this thesis in Part II and III. The following paragraphs provide a brief overview of these challenges, which are explored in greater detail in Sections 1.2.1 and 1.2.2, respectively.

Topology learning, as elaborated in Section 1.2.1, aims to address imperfections in relational data topology. It is particularly useful in scenarios involving missing or redundant relational entries, and even in the extreme case of complete absence of structural information. The absence of topology can be framed from the perspective that any data possesses an underlying latent structure. For instance, in social network data, the connections between users form an implicit structure that can be modeled even when explicit relationships are missing.

Higher-order network modeling, discussed in depth in Section 1.2.2, faces challenges that have emerged due to the rapid progress in TDL research. The fast pace of development, combined with the diversity of higher-order topological structures, has overshadowed some fundamental aspects of the field. These include the scarcity of higher-order datasets and the need for standardization and broader accessibility.

1.2.1 Topology Learning

The process of collecting relational data, like any data collection endeavor, is inherently prone to imperfections. This issue becomes particularly crucial when dealing with structured data [238, 36], as the collection process can be compromised in both the gathering of data observations (data points) and the collection of relationships among these data points (topology). While both aspects present significant challenges, this work primarily focuses on the latter—the relationships that form the topology of the data.

For simplicity, let us provide examples in the context of graph data –although these problems are easily generalized to any other topological domain. The collected topology—in the graph case represented by the edge set—may exhibit various types of imperfections:

- **Missing topology entries:** Important connections are not captured (e.g., missing edges between nodes in graphs).
- **Redundant topology entries:** Unnecessary or erroneous connections are included (e.g., spurious edges in graphs).
- **Absence of topology:** In extreme cases, only the data points are available, with no information about their relationships (e.g., a set of nodes without any edges in graphs).

The challenges of missing or redundant topology have been long-standing issues in the field of graph data modeling. Numerous optimization solutions have been proposed to improve connectivity, such as link prediction algorithms and spectral methods [40, 15, 85]. These approaches aim to address issues like missing edges or remove noisy connections. However, the complete absence of topology—for instance, having no edge information in a graph—presents a more formidable challenge. The scenario of complete absence of topology allows an intriguing perspective: *any type of data can possess an underlying structure*. This implies that even when explicit relational information is unavailable, meaningful connections between data points can be inferred. This concept extends beyond graphs to any topological domain.

Recently, a series of works [117, 58, 205] have begun to investigate techniques for Latent Graph Inference (LGI), also known as Latent Graph Learning (LGL). The objective of these methods is to jointly learn the underlying graph structure and node representations with respect to a given predictive downstream task. The discovered graph structure is referred to as a latent graph. These approaches are particularly valuable in cases where only a point cloud of data is available or when the given graph is suboptimal for the downstream task.

Identifying and leveraging this latent topology potentially yields a broad range of applications. These include point cloud segmentation [225], disease prediction [54, 188], brain network modeling from MRI/fMRI scans [115, 167], building networks of patients for automatic and personalized diagnosis [54, 116], recommender systems [260], computer vision [9, 168], multi-view clustering [155], and brain connection representation [119]. Furthermore, applications extend to aerospace engineering and the prediction of Alzheimer's disease progression [117, 59].

1.2.2 Higher-Order Network Modeling

The field of TDL has experienced significant growth, largely driven by the adaptation and generalization of graph-based architectures to various higher-order domains: hypergraphs [71, 218], simplicial complexes [170, 67, 76], cell complexes [31, 86, 77], and combinatorial complexes [19, 68]. However, this prompt progress has simultaneously led to overshadowing some fundamental aspects of higher-order network modeling. In fact, only recently a few works [159, 160, 90] have unified the theory and notation of various TNNs. Its rapid development, coupled with the inherent variability of higher-order topological domains, has given rise to several critical challenges in the field. These include a lack of open software, standardization, and benchmarking. Indeed, these limitations are highlighted as key open challenges for TDL in a recent ICML 2024 position paper by Papamarkou et al. [157]. This section delves into two primary issues: the scarcity of higher-order datasets and the need for standardization and democratization in higher-order modeling. These challenges, which are addressed in various aspects throughout this dissertation, are discussed in depth in the following subsections.

Scarcity of Higher-Order Datasets

While algebraic topology provides well-defined concepts for describing higher-order relations, applying these concepts to real-world data collection is especially challenging. This limitation stems from several intertwined factors:

1. **Expertise requirements:** Collecting data in topological domains requires both extensive domain knowledge and expertise in algebraic topology, making annotation during the data collection phase particularly difficult.

2. **Domain selection complexity:** Given that there are multiple higher-order topologies capable of describing higher-order interactions, a prior decision on a particular higher-order domain must be made to perform data acquisition. Simultaneously, it remains an open question of which domain is more appropriate for describing various types of data.

As a result, most relational data are currently stored in the form of graph data, resulting in the scarcity of higher-order datasets. Indeed, the scarcity of such datasets hampers the development of reliable benchmarks for assessing (i) the utility of higher-order structures present in data and (ii) the validation and benchmarking of the progress in developing TNNs, thus potentially undermining trust in topological models among the broader research community and preventing the potential subsequent leap in relational data modeling.

Higher-Order Modeling Standardization and Democratization

The rapid advancement of TDL has led to several critical challenges in standardizing and democratizing higher-order network modeling:

1. **Lack of unified frameworks:** The diversity of topological domains and TNN architectures has resulted in a fragmented landscape of tools and methodologies, hindering reproducibility and accessibility.

2. **Theoretical and architectural gaps:** There is a growing need for stronger theoretical foundations and architectures specifically designed for higher-order networks, moving beyond adaptations of graph-based models.

3. **Limited accessibility:** The complexity of topological concepts and the lack of user-friendly tools restrict the adoption of TDL methods across various scientific domains.

Addressing these challenges is crucial for the advancement and broader impact of TDL. Standardization and democratization of higher-order modeling can significantly accelerate research progress, enable more robust and reproducible studies, and facilitate the application of these powerful techniques to a wider range of real-world problems. By making TDL more accessible and integrated with existing deep learning ecosystems, we can unlock its potential to revolutionize how we analyze and model complex systems across diverse fields.

1.3 Research Contributions and Structure of the Thesis

As stated in Section 1.2 this study focuses on different aspects of relational data modeling, with a particular focus on the challenges of *topology learning* and *higher-order network modeling*. In this thesis, we advance both the methodology and applications of relational data modeling. Specifically, in Part II of this thesis, we introduce novel methodologies for latent topology learning and their applications, demonstrating that leveraging latent topology leads to improved performance in downstream tasks across different settings, including Missing Data Imputation (MDI) in tabular data, network traffic compression, and goal-oriented semantic communications tasks. In Part II, Chapters 4 and 5, we introduce methodologies to learn higher-order data topology, thus directly addressing the scarcity of higher-order datasets by enabling inference of higher-order structures from existing data. Part III addresses key questions in higher-order network modeling, offering novel methodologies, frameworks, and theoretical contributions that aim to standardize and democratize TDL. Notably, Chapter 7 introduces a new conceptualization of homophily in higher-order networks and presents the MultiSet framework capable of encompassing most of the current Hypergraph Neural Network (HNN) architectures. Chapter 8 introduces TopoBenchmark (TB), the first open-source benchmarking framework for TDL, along with the results of a TDL challenge conducted using this framework. Finally, Chapter 9 presents a novel TNN architecture that integrates sequence modeling with topological representations, enabling efficient information propagation across different ranks.

A concise description of each chapter and its contributions is provided below. Note that parts of this thesis are adapted from material published or currently under review in journals and conferences, as summarized in **Table 1.1**. The advancements presented in this thesis offer foundational methodologies, tools, and techniques that push the boundaries of the field of relational data modeling and broaden its impact across various domains and real-world applications.

Part I is devoted to introducing the required background material.

- **Section 2.1** presents various topological domains for relational data modeling covering graphs, hypergraphs, simplicial complexes, and cell complexes. It also introduces the concept of lifting, which allows the mapping of one topological domain into another one.
- **Section 2.2** outlines the fundamental concepts underlying MP in topological domains. This includes discussions on k-cochain spaces for representing data, neighborhood functions for encoding relationships, and traditional MP on graphs.
- **Section 2.3** presents a generalized framework for MP on higher-order topological domains. It extends the concepts from graphs to more complex structures, providing a unified approach for modeling intricate multi-way interactions across various topological representations.

Part II presents the methodologies of latent topology learning and their applications.

Chapter 3 presents an end-to-end network architecture for learning latent graphs from tabular data using the EdGe Generation Graph AutoEncoder (EGG-GAE) module. This chapter introduces the methodology for optimizing latent graph tabular downstream task along with MDI task. Through extensive experiments, we demonstrate the state-of-the-art performance of the proposed method across various datasets, address scalability with a tabular graph mini-batching technique, and introduce learnable prototype nodes for more robust data representations.

Chapter 4 introduces Latent Topology Inference (LTI) for learning higher-order cell complexes, capable of modeling sparse and irregular topologies that capture multi-way interactions between data points. This chapter proposes the Differentiable Cell Complex Module (DCM), a learnable function designed to compute cell probabilities. By integrating it with cell complex MP layers and utilizing a scalable two-step inference procedure, we demonstrate significant performance improvements over state-of-the-art methods, particularly in scenarios where no input graph is available.

Chapter 5 introduces a novel TDL framework for lossy network traffic compression that detects complex patterns in traffic data and compresses them using TNNs. This chapter evaluates our approach on two real-world networking datasets, showing substantial improvements over GNN based architectures and standard autoencoders.

Chapter 6 presents a topology-driven approach for transformer-based deep Joint Source-Channel Coding (JSCC) in image processing aimed at improving communication efficiency and reducing computational latency. This chapter introduces a method that selectively merges tokens to optimize compression and transmission, focusing on task-relevant information. We demonstrate the effectiveness of this approach in both single and multiple-task scenarios, highlighting significant advantages in token-level compression compared to conventional techniques.

Part III addresses the challenges in higher-order network modeling, focusing on standardization, benchmarking, and novel architectural approaches.

Chapter 7 explores fundamental questions in HNNs through the lens of MP. It introduces a novel definition of homophily for hypergraphs, presents the MultiSet framework for incorporating hyperedge-dependent node representations, and investigates new mini-batch sampling strategies. Through extensive experiments, it provides insights into HNN architectures and hypergraph datasets, contributing to a better understanding of hypergraph representation learning.

Chapter 8 introduces TB, an open-source and modular benchmarking framework for TDL. TB addresses the challenges of dataset scarcity, standardization across topological domains, and comparative evaluation of diverse TNN architectures. It provides tools for generating topological datasets, standardizing input-output pipelines, and facilitating reproducible research in TDL.

Chapter 9 presents a novel approach combining sequence modeling with TDL. It adapts the Mamba State Space Model (SSM) for use with simplicial complexes, enabling efficient information propagation across different ranks. The chapter explores the effectiveness of this approach in comparison to state-of-the-art models for simplicial complexes and investigates efficient batching techniques for topological structures.

Part IV provides a summary of the key contributions of this thesis, along with potential directions for future research. The thesis is supported by four Appendices. In Appendices A and B, we offer additional experiments that complement Chapters 3 and 4, respectively. Appendix C includes proofs, extended experiments, and further analysis related to Chapter 7. Lastly, Appendix D outlines the full experimental details of the TopoBenchmark framework discussed in Chapter 8.

Table 1.1. Schematic overview of the research contributions related to the thesis

Part II	Chapter 3	Published at International Conference on Artificial Intelligence and Statistics 2023 (AISTATS) [200]
	Chapter 4	Published at International Conference on Learning Representations 2023 (ICLR) [17]
	Chapter 5	Published at the 2nd Graph Neural Networking Workshop 2023 (GNNet), ACM CoNEXT [25] Extended abstract and oral presentation at Learning on Graphs Conference 2023 (LoG) [24]
	Chapter 6	Currently under double-blind review at an A1 conference.
Part III	Chapter 7	Under review at Transactions on Machine Learning Research (TMLR) [201] Under review at Data-Centric Machine Learning Research (DMLR) [202].
	Chapter 8	Open-sourced library 'TopoBenchmark' on GitHub (80 stars at the time of writing.) Part of this chapter is published in the PMLR through the GRaM Workshop at ICML 2024 [26].
	Chapter 9	Under review at Learning on Graphs Conference 2024 [148]

Part I

Background Material

Chapter 2

Relational Data Modeling

Contents

Relational data modeling is a fundamental aspect of modern machine learning and data analysis, particularly in domains where complex relationships between entities play a crucial role. This chapter provides a comprehensive overview of the key concepts and techniques in relational data modeling, with a focus on topological approaches that capture intricate structural information.

We begin by exploring various topological domains, from the familiar terrain of graphs to more sophisticated structures such as hypergraphs, simplicial complexes, and cell complexes. These domains offer powerful frameworks for representing and analyzing complex relational data. Following this, we delve into the foundations of MP, a key mechanism for propagating information across these topological structures. We then extend these concepts to higher-order domains, introducing a general framework for HOMP that encompasses a wide range of relational data models.

Before we proceed, it's important to clarify some key terminology introduced in this chapter and used throughout this work:

- While a higher-order domain is not necessarily a topological space in the strict mathematical sense, it consists of higher-order cells that encode topological

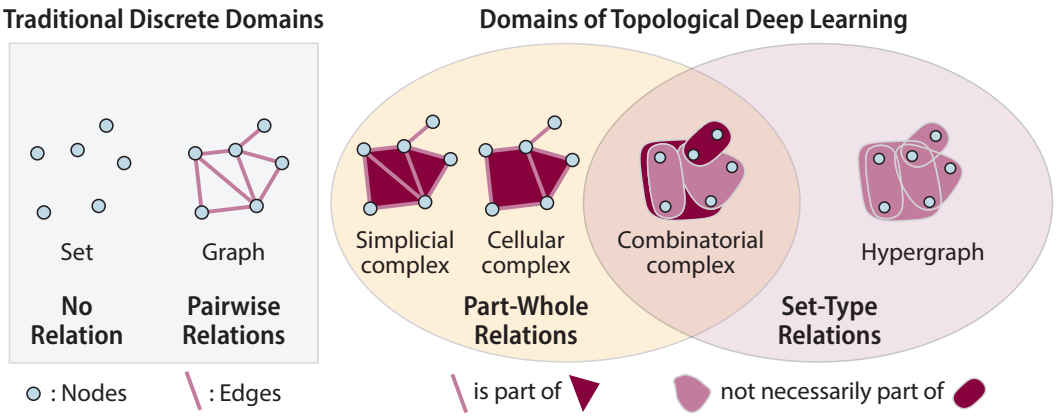


Figure 2.1. Representation of topological domains, reproduced from Papillon et al. [159] with the permission of the authors.

information. In this context, we use the terms 'higher-order domain' and 'topological domain' interchangeably.

- Similarly, we use 'higher-order neural network' and 'topological neural network' interchangeably when discussing learning models on these domains.
- A 'featured topological domain' refers to a topological domain where each element (e.g., node, edge, or higher-order cell) is associated with a feature vector. For simplicity, we often use 'topological domain' to encompass both the structural aspects and the associated features.

These conventions help us focus on the main ideas and their relationships without overloading the reader with overly technical details. Throughout this chapter, we provide definitions tailored for machine learning on structured data. For more rigorous mathematical treatments, particularly in the context of algebraic topology, we refer readers to Hatcher [97].

This chapter presents the principles of relational data modeling and introduces the fundamental tools required to address complex relational problems across a wide range of domains.

2.1 Topological Domains

Topological domains play a crucial role in relational data modeling, providing powerful frameworks for representing complex relationships and structures in data. This section introduces various topological domains, progressing from the fundamental concept of graphs to more sophisticated higher-order domains. Figure 2.1 provides a visual representation of the topological domains we will discuss, offering an overview of their structural differences and relationships.

We begin by formally defining the graph domain, which serves as a foundation for understanding more complex structures. From there, we explore higher-order topological domains, including hypergraphs, simplicial complexes, and cell complexes.

Each of these domains offers unique capabilities for capturing different types of relationships and hierarchies within data.

2.1.1 Graphs

Graphs are essential mathematical structures used to represent relationships between discrete entities, which capture pairwise interactions between data points.

Definition 2.1.1 Let $\mathcal{G} = (V, E)$ be a graph, with node set V and edge set E. A featured graph is a tuple $\mathcal{G}_F = (V, E, F_V, F_E)$, where $F_V : V \to \mathbb{R}^{d_v}$ is a function that maps each node to a feature vector in $\mathbb{R}^{d_v}$ and $F_E : E \to \mathbb{R}^{d_e}$ is a function that maps each edge to a feature vector in $\mathbb{R}^{d_e}$.

2.1.2 Hypergraphs

Hypergraphs (HGs) generalize traditional graphs by allowing edges, known as hyperedges, to connect any number of nodes. This flexibility enables hypergraphs to capture more complex relationships between entities than standard graphs, which only connect pairs of nodes. Hypergraphs exhibit set-type relationships that lack an explicit notion of hierarchy. Using these set-type relations makes them a powerful tool for representing relationships across a diverse range of complex systems.

Definition 2.1.2 (Hypergraph) A *hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where:

- $\mathcal{V}$ is a finite set of vertices,
- $\mathcal{E}$ is a set of hyperedges, where each hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$, i.e., $e \subseteq \mathcal{V}$, with no restriction on the cardinality of e (i.e., each hyperedge can connect any number of vertices).

A *featured hypergraph* is a tuple $\mathcal{H}_F = (\mathcal{V}, \mathcal{E}, F_{\mathcal{V}}, F_{\mathcal{E}})$, where:

- $F_{\mathcal{V}} : \mathcal{V} \to \mathbb{R}^{d_v}$ maps each vertex in $\mathcal{V}$ to a feature vector in $\mathbb{R}^{d_v}$,
- $F_{\mathcal{E}} : \mathcal{E} \to \mathbb{R}^{d_e}$ maps each hyperedge in $\mathcal{E}$ to a feature vector in $\mathbb{R}^{d_e}$.

2.1.3 Simplicial Complexes

Simplicial Complexes (SCs) extend graphs by incorporating hierarchical part-whole relationships through the multi-scale construction of cells. In this structure, nodes correspond to rank 0 cells, which can be combined to form edges (rank 1 cells). Edges can then be grouped to form faces (rank 2 cells), and faces can be combined to create volumes (rank 3 cells), continuing in this manner. Consequently, the faces of a simplicial complex are triangles, volumes are tetrahedrons, and higher-dimensional cells follow the same pattern. A key feature of simplicial complexes is their strict hierarchical structure, where each k-dimensional simplex is composed of $(k-1)$-dimensional simplices, reinforcing a strong sense of hierarchy across all levels.

Definition 2.1.3 (Simplicial complex) A *simplicial complex* is defined as a tuple $\mathcal{X} = (\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_K)$ consisting of finite sets, where:

- $\mathcal{X}_0$ is a set of vertices (or 0-cells),
- $\mathcal{X}_1$ is a set of edges (or 1-cells), with each edge $e \in \mathcal{X}_1$ being an unordered pair of vertices $e = \{u, v\}$ for $u, v \in \mathcal{X}_0$,
- $\mathcal{X}_2$ is a set of triangles (or 2-cells), with each triangle $\sigma \in \mathcal{X}_2$ consisting of an unordered set of three vertices $\sigma = \{u, v, w\}$, and its boundary being composed of three edges (1-cells) from $\mathcal{X}_1$,
- Each k-cell in $\mathcal{X}_k$ is an unordered set of $k+1$ vertices, and its boundary is composed of $(k-1)$-cells from $\mathcal{X}_{k-1}$.

A *featured simplicial complex* is a tuple $\mathcal{X}_F = (\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_K, F_{\mathcal{X}_0}, F_{\mathcal{X}_1}, \ldots, F_{\mathcal{X}_K})$, where, for each $0 \leq r \leq K$, $F_{\mathcal{X}_r} : \mathcal{X}_r \to \mathbb{R}^{d_r}$ maps each r-cell in $\mathcal{X}_r$ to a feature vector in $\mathbb{R}^{d_r}$.

2.1.4 Cell Complexes

Cell Complexes (CCs) provide a hierarchical interior-to-boundary structure, offering clear topological and geometric interpretations. Unlike simplicial complexes, cell complexes are not limited to simplexes; faces can involve more than three nodes, allowing for a more flexible representation. This increased flexibility grants cell complexes greater expressivity compared to simplicial complexes [30].

Definition 2.1.4 (Cell complex) A *cell complex* is a triplet $\mathcal{X} = (\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2)$ of finite ordered sets, where:

- $\mathcal{X}_0$ is a set of nodes (or 0-cells),
- $\mathcal{X}_1$ is a set of edges (or 1-cells), with each edge $e \in \mathcal{X}_1$ being an ordered pair of nodes $e = [u, v]$ for $u, v \in \mathcal{X}_0$,
- $\mathcal{X}_2$ is a set of faces (or 2-cells), with each face $\sigma \in \mathcal{X}_2$ being an ordered sequence of edges $\sigma = [e_1, e_2, \ldots, e_{m(\sigma)}]$ forming a closed path without self-intersections, where $m(\sigma) \geq 3$.

A *featured cell complex* is a tuple $\mathcal{X}_F = (\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2, F_{\mathcal{X}_0}, F_{\mathcal{X}_1}, F_{\mathcal{X}_2})$, where, for $0 \leq r \leq 2$, $F_{\mathcal{X}_r} : \mathcal{X}_r \to \mathbb{R}^{d_r}$ maps each r-cell in $\mathcal{X}_r$ to a feature vector in $\mathbb{R}^{d_r}$.

Note that another prominent topological domain in TDL literature is Combinatorial Complexes (CCCs), which generalizes both CCs and HGs, incorporating both part-whole and set-type relationships [88]. This domain is also depicted in Figure 2.1. However, the precise definition of CCCs is beyond the scope of this work.

2.1.5 Lifting

Once a featured graph and various higher-order domains have been defined, it becomes essential to introduce a procedure for mapping the featured graph onto a corresponding topological domain, a process known as lifting [90]. Definition 2.1.5 describes a lifting from a featured graph to featured cell complexes. However, lifting to other topological domains can be defined similarly.

Definition 2.1.5 (Lifting) Let $\mathcal{G}_F = (V, E, F_V, F_E)$ be a featured graph. A lifting of $\mathcal{G}_F$ is a triplet $(\mathcal{C}_F, \iota, L)$, where $\mathcal{C}_F = (\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2, F_{\mathcal{X}_0}, F_{\mathcal{X}_1}, F_{\mathcal{X}_2})$, is a featured cell complex, ι is an embedding map that maps the nodes and edges of $\mathcal{G}$ to cells in $\mathcal{X}$, and L is a collection of functions that map features of $\mathcal{G}_F$ to features of $\mathcal{C}_F$. The embedding map ι satisfies $\iota(v) = s_v$ for $v \in V$ with $s_v \in \mathcal{X}_0$ being a 0-cell corresponding to node v, and $\iota(e) = x_e$ for $e \in E$ with $x_e \in \mathcal{X}_1$ being a cell corresponding to edge e. The collection of functions in L consists of a lifting procedure for the higher-order cells in $\mathcal{X}$ that are not part of $\mathcal{G}$ and a lifting procedure for the feature maps $\{F_{\mathcal{X}^i}\}_{i=0}^2$ based on $\mathcal{G}_F$.

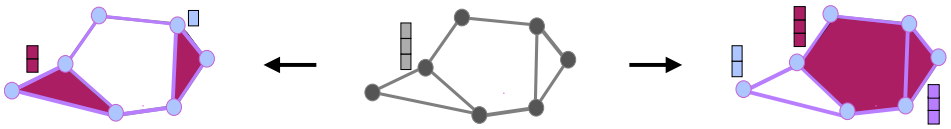


Figure 2.2. An illustration of lifting a featured graph $\mathcal{G}_F$ (center) to two different topological domains: a simplicial complex (left) and a cell complex (right). The lifting from $\mathcal{G}_F$ (center) to the featured cell complex $\mathcal{C}_F$ (right) is represented by the triplet $(\mathcal{C}_F, \iota, L)$, where ι is the embedding map that places the nodes and edges of $\mathcal{G}_F$ into the corresponding cells of $\mathcal{C}_F$, and L is the collection of functions that maps the features of $\mathcal{G}_F$ to the features of $\mathcal{C}_F$.

2.2 Foundations of Message Passing

2.2.1 Basic Concepts

K-Cochain Spaces

To represent data on topological domains, we first introduce the concept of k-cochain spaces. These spaces are essential for describing how data is structured and processed within higher-order topological domains:

Definition 2.2.1 (k-cochain spaces) Let $\mathcal{C}^k(\mathcal{X}, \mathbb{R}^d)$ be the $\mathbb{R}$-vector space of functions $\mathbf{H}_k : \mathcal{X}^k \to \mathbb{R}^d$ for a rank $k \in \mathbb{Z}_{\geq 0}$ and the data dimension d. $\mathcal{C}^k(\mathcal{X}, \mathbb{R}^d)$ is called the **k-cochain space**. Elements $\mathbf{H}_k$ in $\mathcal{C}^k(\mathcal{X}, \mathbb{R}^d)$ are called **k-cochains** or **k-signals**.

K-cochains represent the input topological data (feature vectors) defined on the topological domain. With this foundation for representing features over higher-order objects, we can now move on to encoding their relationships.

Neighborhood Functions

To formalize the relationships between elements in a topological domain, we introduce the concept of neighborhood functions:

Definition 2.2.2 (Neighborhood function) Let S be a nonempty set. A **neighborhood function** on S is a function $\mathcal{N} : S \to \mathcal{P}(\mathcal{P}(S))$ that assigns to each point x in S a nonempty collection $\mathcal{N}(x)$ of subsets of S. The elements of $\mathcal{N}(x)$ are called **neighborhoods** of x with respect to $\mathcal{N}$.

Here, $\mathcal{P}$ denotes the power set operator, where $\mathcal{P}(S)$ is the set of all subsets of S. Thus, $\mathcal{P}(\mathcal{P}(S))$ represents the set of all collections of subsets of S. This formulation enables the assignment of multiple, potentially overlapping neighborhoods to each point, providing the necessary flexibility to describe diverse neighborhood structures across various topological domains.

The flexibility of neighborhood functions is crucial for representing complex relationships in higher-order topological structures, where elements may exhibit multifaceted connections or interactions. By generalizing the concept of node neighborhoods from graphs to higher-order structures, these functions define how information propagates between different elements in the topological domain. This generalization forms the foundation for extending traditional graph-based algorithms to more complex topological spaces, enabling the modeling of sophisticated relational data.

2.2.2 Traditional Message Passing on Graphs

GNNs have emerged as a powerful class of models for processing graph-structured data. While numerous variations of GNN architectures exist [139, 214, 95, 39], at their core lies an iterative MP algorithm that propagates information between the nodes of the graph. This process can be understood in terms of the basic concepts we introduced earlier.

Formally, a graph is defined as a tuple of nodes and edges, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. In the context of k-cochain spaces introduced in Section 2.2, we can view node features as 0-cochains and edge features as 1-cochains. We denote by $h_k^t \in \mathbb{R}^d$ the hidden state of a node k at MP iteration t, which can be seen as an element of a 0-cochain space.

The neighborhood function for a graph, as per Definition 2.2.2, is typically defined as $N(k) = \{v \in \mathcal{V} \mid (k, v) \in \mathcal{E}\}$, representing the one-hop neighborhood of each node.

The MP process consists of three main steps:

1. **Message Generation**: Each node k receives messages from all the nodes in its neighborhood $N(k)$. Messages are generated by applying a message function $m(\cdot)$ to the hidden states of node pairs in the graph.

2. **Message Aggregation**: The generated messages are combined using a permutation invariant aggregation function $\oplus$, as shown in Equation 2.1. This aggregation can be seen as an application of the neighborhood function concept.

3. **Node Update**: An update function $u(\cdot)$ is used to compute a new hidden state for every node, as shown in Equation 2.2.

These steps are formalized in the following equations:

$$M_k^{t+1} = \bigoplus_{i \in N(k)} m(h_k^t, h_i^t), \tag{2.1}$$

$$h_k^{t+1} = u(h_k^t, M_k^{t+1}), \tag{2.2}$$

where $m(\cdot)$ and $u(\cdot)$ are differentiable functions and consequently may be implemented as neural networks.

This process can be seen as a specific instance of the more general HOMP framework we will introduce in the next section, applied to the case of graphs where we only have 0-cells (nodes) and 1-cells (edges).

2.3 Higher-Order Message Passing

HOMP generalizes information propagation techniques to complex topological domains such as hypergraphs, simplicial complexes, and cell complexes. This section introduces a formal framework for HOMP, building on the foundational concepts of k-cochain spaces and neighborhood functions defined earlier. By leveraging the rich relationships captured in these advanced topological representations, this unified approach enables modeling and analyzing intricate multi-way interactions across various topological structures, including both traditional graphs and more complex higher-order domains.

2.3.1 Extending Message Passing to Higher-Order Domains

The extension of MP to higher-order domains involves generalizing the concepts we have discussed for graphs to more complex topological structures. This generalization allows us to capture and process richer relational information that goes beyond pairwise interactions.

In higher-order domains, the notion of a "neighborhood" becomes more complex. Instead of just considering adjacent nodes, we now need to consider relationships between higher-dimensional cells (e.g., edges, faces, volumes). The neighborhood functions we defined earlier play a crucial role in formalizing these complex relationships.

2.3.2 Higher-Order Message Passing Framework

With k-cochain spaces providing a way to represent data and neighborhood functions defining relationships, we can now formally define the HOMP procedure. Let $\mathcal{X}$ be a topological domain, and let $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$ be a set of neighborhood functions defined on $\mathcal{X}$. Consider a cell x and another cell $y \in \mathcal{N}_k(x)$ for some $\mathcal{N}_k \in \mathcal{N}$. A message $m_{x,y}$ between cells x and y is a computation depending on these two cells or on the data they support. Let $\mathcal{N}(x)$ denote the multi-set $\{\!\{\mathcal{N}_1(x), \ldots, \mathcal{N}_n(x)\}\!\}$, and let $h_x^{(l)}$ represent the data supported on the cell x at layer l. Higher-Order Message Passing (HOMP) is defined as follows:

$$m_{x,y} = \alpha_{\mathcal{N}_k}(h_x^{(l)}, h_y^{(l)}), \tag{2.3}$$

$$m_x^k = \bigoplus_{y \in \mathcal{N}_k(x)} m_{x,y}, \quad 1 \leq k \leq n, \tag{2.4}$$

$$m_x = \bigotimes_{\mathcal{N}_k \in \mathcal{N}(x)} m_x^k, \tag{2.5}$$

$$h_x^{(l+1)} = \beta(h_x^{(l)}, m_x). \tag{2.6}$$

where $\bigoplus$ is a permutation-invariant aggregation function, which is referred to as intra-neighborhood aggregation of x, and $\bigotimes$, is an aggregation function called the inter-neighborhood aggregation of x. The functions $\alpha_{\mathcal{N}_k}$ and β are differentiable functions.

To summarize the HOMP process:

- **Message Generation**: $m_{x,y}$ is the message computed from x to y using the function $\alpha_{\mathcal{N}_k}$.
- **Message Aggregation (intra)**: m_x^k aggregates all messages from the neighbors y in the neighborhood $\mathcal{N}_k(x)$ using the intra-neighborhood function $\bigoplus$.
- **Message Aggregation (inter)**: m_x further aggregates these results across all neighborhoods $\mathcal{N}_k \in \mathcal{N}(x)$ using the inter-neighborhood function $\bigotimes$.
- **Cell Update**: $h_x^{(l+1)}$ updates the data on cell x by combining its current data $h_x^{(l)}$ with the aggregated message m_x using the function β.

This framework allows for rich information exchange across different dimensions and types of relationships in the topological domain, enabling the modeling of complex, multi-way interactions in various real-world systems.

Part II

Topology Learning: Methodologies and Their Applications

Chapter 3

EGG-GAE: scalable graph neural networks for tabular data imputation

Contents

The organization of this chapter is as follows. First, in Section 3.1, we describe and motivate the problem of MDI and the proposed solution. In Section 3.2, we formulate the problem, and in the following Section 3.3, we describe the EGG-GAE model that solves the MDI problem by leveraging the recent progress in latent topology learning. In Section 3.4, we provide extensive experiments supporting the proposed methodology. Section 3.5 reviews the related work. The chapter concludes with a final discussion in Section 3.6.

The content of this chapter is adapted from the material published in [200]

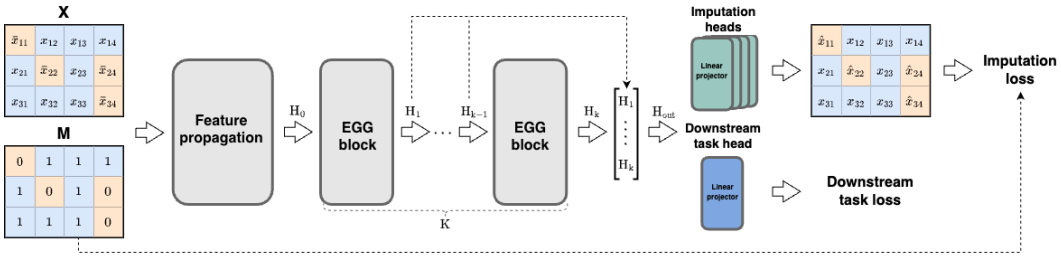


Figure 3.1. Pipeline: initially, the preprocessed batch X is encoded with a feature propagation block (Eq. equation 3.1), resulting in the initial representations H_0. These representations are then updated with a sequence of EGG-GAE blocks obtaining H_k, $k \in \{1, \dots, K\}$. The concatenation of $\{H_k\}_{k=1}^{K}$ gives the final representation H_{out}. It is used as the input for the subsequent heads: numerical imputation, categorical imputation, and downstream task one.

3.1 Introduction

Missing data imputation is a ubiquitous issue that arises in a variety of domains. Most supervised deep learning methods require complete datasets, but in the real world, datasets often suffer from incompleteness due to access problems or mistakes in data collection [253, 255, 128, 209]. Numerous fields thus require MDI methods to reconstruct a complete dataset, including biostatistics [143], epidemiology [193], and irregular time-series analysis [128].

Classically, the underlying mechanism giving rise to missing data is categorized into three types [176]. (1) If the probability of being missing is completely independent of the data, then the data are said to be Missing Completely at Random (MCAR). (2) Data is said to be Missing at Random (MAR) if the probability of being missing is the same only within the data-defined groups. (3) If the probability of missing data depends on both observed and unobserved variables, then such data are Missing Not at Random (MNAR).

Predictive approaches to MDI [27] can be categorized in two families: (i) building a global model for data imputation, or (ii) inferring the missing components employing similar data points to the one having missing values. The second class typically uses advanced k-NN strategies [1]. The first class includes simple statistics from the entire dataset (e.g., medians), linear models [130], support vector machines [223] or, more recently, deep neural architectures [187, 254, 153, 189].

Recently, it was noted that the unification of the two paradigms (i.e., inferring similar data points for each imputation and building global models from the overall dataset) can be beneficial for MDI. This can be done by exploiting GNNs [152, 45, 114, 172], a novel class of neural networks that can process graph-based data in a differentiable fashion. In particular, the Graph Imputer Neural Network (GINN) [189] explored the assumption of endowing tabular data with a graph topology based on a pre-computed similarity between points in the feature space, and then exploiting a GNNs to tackle the MDI problem. However, the proposed GINN model had two limitations. First, the method is tested only on the entire dataset (i.e., no mini-batching is performed, which is challenging with GNNs models), and scaling it up to large datasets is unfeasible due to the quadratic cost of computing the

similarity matrix. Second, graph connectivity must be computed beforehand in the feature space employing only those values observed by both data points, requiring the definition of a suitable distance metric and a customized procedure to sparsify the graph.

In general, building a connectivity beforehand as done in GINN was necessary, since the underlying assumption of most GNNs is that the graph topology is given and fixed; thus, convolution-like operations typically amount to modifying the node-wise features by averaging information from the neighbors. However, the hand-crafted connectivity might be sub-optimal and requires a number of hyper-parameters to be defined and manually optimized. Instead, automatically learning the latent graph structure can overcome the limitations of these methods by inferring the underlying graph relationships. Such latent graphs can capture the actual topology of structured data through the downstream tasks, which can be seen as a task-related topology, thus conveying model interpretability.

Based on these considerations, the main contributions of this chapter are as follows. (i) We introduce an end-to-end trainable network architecture to learn the optimal underlying latent graph for tabular data using a novel EdGe Generation Graph AutoEncoder (EGG-GAE) network module. The EGG-GAE module sampling scheme is optimized with respect to downstream task metrics utilizing a straight-through estimator [109] in the backward pass to ensure its differentiability. (ii) We demonstrate that employing the latent graph predictions for the MDI problem induces consistent improvement across a number of datasets in a large experimental evaluation, demonstrating significant improvements over baselines and reaching state-of-the-art performance. (iii) We propose the concept of tabular graph mini-batching along with an ensembling technique to resolve the MDI scalability issues [147]. (iv) We propose a novel concept of learnable *prototype nodes*, which encodes a learnable data representation in the form of an additional set of nodes added to the imputed graph to provide each data point in the mini-batch with a reliable neighborhood. Our code is available on GitHub [1]

3.2 Problem Formulation

Let the matrix $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{\mathrm{N}}$ denote a d-dimensional dataset and $\mathbf{Y}$ represents its corresponding target vector. Without loss of generality, we assume each data vector $\mathbf{x}_i$ contains numerical variables referred to $\mathrm{d} \in \{1, \ldots, \mathrm{d_n}\}$ and categorical variables are indexed by $\mathrm{d} \in \{\mathrm{d_{n+1}}, \ldots, \mathrm{d_n} + \mathrm{d_c}\}$ with $\mathrm{d_n} + \mathrm{d_c} = \mathrm{d}$. We assume that each $\mathrm{d_{th}}$ categorical variable takes values among $\mathrm{C_{d_{th}}}$ classes. The dataset $\mathbf{X}$ is referred as mixed if $\mathrm{d_n} > 0, \mathrm{d_c} > 0$, numerical if $\mathrm{d_c} = 0$ and categorical when $\mathrm{d_n} = 0$. By definition some percentage $\mathrm{I_{init}} \in [0, 1)$ of dataset entries $\mathrm{x}_{ij} \in \mathbf{X}$ are missing (corrupted). We associate a binary matrix $\mathbf{M} \in \{0, 1\}^{\mathrm{N} \times \mathrm{N}}$ to identify missing and observed variables, where 1 corresponds to the observed values, and 0 indicates missing ones. Note that the corruption process can be of many types: MCAR, MNAR, MAR, and it is generally unknown to the user. The imputation process aims to provide a plausible estimation $\hat{\mathrm{x}}_{ij}$ for unobserved values x_{ij}, such that (i) the imputed dataset $\widehat{\mathbf{X}}$ would be as close as possible to the real complete dataset

[1] https://github.com/levtelyatnikov/EGG_GAE

(if such exists), (ii) the imputed dataset $\widehat{\mathbf{X}}$ has to achieve strong downstream task performance if adopting $\widehat{\mathbf{X}}$ as input to predict the corresponding target vector $\mathbf{Y}$.

Dataset preprocessing We assume that the dataset $\mathbf{X}$, by definition, is properly normalized and corrupted in advance, hence the corresponding corruption matrix $\mathbf{M}$ is determined. Training different imputation approaches discussed in this work requires distinct data preprocessing strategies. A straightforward way is to employ statistics of observed values. In our work, numerical values are initially assessed with *mean* statistics, while categorical ones are approximated with the corresponding *most frequent class* unless otherwise stated. Some of the imputation baselines discussed in this chapter also require one-hot encoding of categorical values. The preprocessed dataset is denoted as $\overline{\mathbf{X}}$, which is the input to the subsequent EGG-GAE module.

3.3 Method

We propose a general solution for the tabular data MDI problem based on graph representation learning, whose overall pipeline is depicted in Fig. 3.1. At every iteration, a mini-batch of data is sampled and preprocessed, according to the procedure described in Section 3.3.1. This mini-batching is necessary, as it allows the algorithm to scale to large datasets. Next, we build a graph where each node is a row of the sampled mini-batch, and its corresponding node features are a non-linear mapping of the original inputs. The connectivity between nodes is learned through a differentiable sampling procedure, instead of being fixed as in previous works (e.g., [189]). We describe different variations of this last component in Section 3.3.2. We also propose two extensions to this basic architecture, i.e., ensembling and what we call *prototype nodes*, in Section 3.3.3. The entire system is trained end-to-end with a combination of imputation losses and a downstream classification loss, as described in Section 3.3.4.

3.3.1 Mini-batch preprocessing

Like in Spinelli et al. [189], we turn MDI into a predictive task by employing a surrogate task, we randomly remove elements from the mini-batch and impute them to enable our model to reconstruct missing values. In contrast to denoising auto-encoders [217], we only predict the removed elements rather than reconstructing the entire input. In order to obtain a surrogate batch-level corruption matrix M we use the MCAR mechanism to mask a certain percentage $\mathrm{I_b} \in [0, 1 - \mathrm{I_{init}})$ of the sampled batch $\mathrm{X} \sim \overline{\mathbf{X}}$. Precomputed statistics replace numerical masked values, while unobserved categorical entries are replaced with auxiliary tokens. We replace each categorical variable with a trainable dense embedding, whose size is a hyperparameter. Note that initial missing values, which have to be imputed during the inference, are represented as observed variables in the surrogate batch-level corruption matrix M. The concatenation of the preprocessed batch parts: categorical $\mathrm{X^c}$ and numerical $\mathrm{X^n}$, yields the final preprocessed batch. In order to avoid cumbersome notation, we refer to the final batch representation as X.

3.3.2 Architecture

The core feature of the proposed EGG-GAE model is the EGG block, which endows the tabular representation of the sampled mini-batch with a graph topology in order to predict the missing values with a GNN module. The general EGG module comprises three components, as depicted schematically in Fig. 3.2: (i) a *node projector* transforms input features $\mathrm{H_k}$ by projecting each row into a new space $\mathrm{H_k^g}$ that we call the graph embedding space; (ii) A *sampler*, which obtains the edge set $\mathcal{E}_\mathrm{k}$ by sparsifying all possible edges between nodes; (iii) a GNN head that operates on the obtained graph $\mathcal{G} = (\mathrm{H_k}, \mathcal{E}_\mathrm{k})$. The EGG blocks can be stacked subsequently while their outputs concatenated to obtain the final representation $\mathrm{H_{out}}$. In the following, we describe two variations of EGG blocks: the standard EGG blocks sample each edge independently, and a restricted k-EGG block that samples exactly k neighbors for each node.

Before the first EGG block, the preprocessed batch X is encoded via an initial mapping function:

$$\mathrm{H_0} = \mathrm{MLP_{FP}}(\mathrm{X}) \tag{3.1}$$

where $\mathrm{MLP_{FP}}$ is a two-layer MLP with a ReLU activation and batch normalization in between.

EGG Each EGG block first projects the input $\mathrm{H_k}$ with an additional row-wise operation, obtaining:

$$\mathrm{H_k^g} = \mathrm{MLP_k}(\mathrm{H_k}) \tag{3.2}$$

where $\mathrm{MLP_k}$ has the same architecture as $\mathrm{MLP_{FP}}$. The sampler block represents pairwise edge relations of the projected $\mathrm{H_k^g}$ by first forming a probability matrix P where the i-th, j-th element is computed as:

$$\mathrm{P}_{ij} = \exp^{-||\mathrm{h}_i - \mathrm{h}_j||^2} \tag{3.3}$$

where $\mathrm{h}_i, \mathrm{h}_j$ are i-th and j-th rows of matrix $\mathrm{H_k^g}$. Each element of P_{ij} represents the probability of sampling edge (i, j) in the output graph. In order to sample an undirected graph, corresponding to a strictly upper triangular adjacency matrix $\tilde{\mathrm{A}}_\mathrm{k}$, we combine a Gumbel-Softmax trick [109] for sampling from P with masking:

$$\tilde{\mathrm{A}}_{ij} = (1 + \exp(^{(\log(\mathrm{P}_{ij} \odot \mathrm{J}_{ij}) + \mathrm{G}_{ij} \odot \mathrm{J}_{ij})}/_{\tau}))^{-1} \tag{3.4}$$

where τ is a separate temperature hyperparameter, J is a strictly upper triangular matrix with ones above the main diagonal, $\odot$ is the Hadamard product, and $\mathrm{G}_{ij} \sim \mathrm{Gumbel}(0, 1)$. We obtain a sparse matrix by then thresholding $\tilde{\mathrm{A}}_\mathrm{k}$ at 0.5. The final sparse unweighted adjacency matrix $\mathrm{A_k}$ = is computed as:

$$\mathrm{A_k} = \tilde{\mathrm{A}}_\mathrm{k} + \tilde{\mathrm{A}}_\mathrm{k}^\mathrm{T} + \mathrm{I} \tag{3.5}$$

where I is the identity matrix. In order to have a valid gradient for back-propagation with respect to the thresholding operation, we use a straight-through estimator in the backward pass [109] to allow the gradient flow through the sampling scheme. The final input of the GCN head is then a sparse unweighted adjacency matrix $\mathrm{A_k}$ along

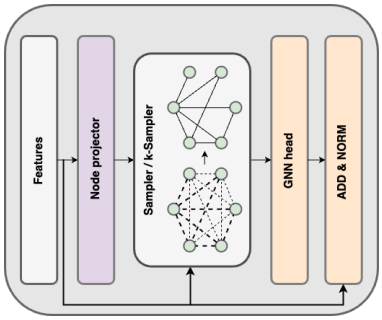


Figure 3.2. Schematic depiction of the EGG module.

with the corresponding feature representation $\mathrm{H_k}$. The updated node representation $\mathrm{H_{k+1}}$ is computed as:

$$\hat{\mathrm{H}}_{\mathrm{k+1}} = \mathrm{GCN}(\mathrm{H_k}, \mathrm{A_k}) \tag{3.6}$$

$$\mathrm{H_{k+1}} = \mathrm{LayerNorm}(\hat{\mathrm{H}}_{\mathrm{k+1}} + \mathrm{H_k}) \tag{3.7}$$

where GCN is a graph convolutional layer [122] or any other message-passing layer that operates on the graph connectivity.

***k*-EGG** We also explore a variant of EGG, called k-EGG, which is more directly inspired to the sampling procedure in Kool et al. [127]. It forces a sparsity of the adjacency matrix by limiting the number of neighbors sampled per node to a fixed constant k. For each row i of $\mathrm{A_k}$, instead of thresholding each entry, we extract the first k edges corresponding to the k highest values, obtaining $\{(i, j_1), \ldots, (i, j_k)\}_{i=1}^{\mathrm{n}}$ and filling with ones the corresponding positions of matrix $\tilde{\mathrm{A}}_{\mathrm{k}} \in \{0, 1\}^{\mathrm{n}\times\mathrm{n}}$. The unweighted adjacency matrix $\mathrm{A_k}$ is computed as in Eq. equation 3.5.

Comparisons with DGM Our graph sampling procedure is inspired to the recently proposed DGM block [118], with a number of important differences that we mention here. First, DGM was designed for a graph scenario where the full set of nodes is available beforehand and a *single* underlying graph connectivity is assumed to exist. Instead, we work on randomly sampled mini-batches of data coming from a generic tabular dataset. Second, since the size of mini-batch is a user-defined parameter, we use directly the straight-through estimator in the backward pass, avoiding an additional surrogate loss as in Kazi et al. [118]. The proposed module does not require to limit the output dimensions of the node projector to small values, hence obtaining greater model expressivity. Finally, the GNN head in this chapter follows a design with skip-connections and layer normalization.

3.3.3 Extensions to the basic formulation

This section describes two simple extensions of the model that we have found to work consistently better in our experimental evaluation.

Ensembling The proposed model has two sources of stochasticity: mini-batch sampling and continuous relaxation of discrete random variables (the edge sampling procedure). We propose to exploit this stochasticity during inference by sampling the batches from the test set until each datum (node) has the desired number of predictions, so that each node has multiple predictions relying on different neighborhoods. In the classification case, we select the maximum average soft prediction, whereas we use the mean prediction for the regression case.

Prototype nodes In this chapter, we also propose to use trainable prototype nodes which we refer to as H_{pr}. Applying mini-batching to tabular or graph-structured data may lead in exceptional cases to a lack of data expressiveness during the forward pass due to the same sources of stochasticity mentioned above. For instance, in the case of tabular data, there is no guarantee that the sampled mini-batch composes a reliable set of neighbors for the particular datum prediction. Therefore, prototype nodes are designed to encode common data patterns and allow each data point to have reliable neighbors regardless of the sampled mini-batch. The number of prototypes nodes is a hyperparameter. We initialize the prototypes nodes randomly. However, other strategies such as data mean can be applied. The prototype nodes are then added to H_k before every EGG block. H_{pr} does not participate explicitly in the objective function while contributing as sampled neighbors.

3.3.4 Objective

The objective function consists of two main terms: downstream loss and imputation loss. The downstream loss $\mathcal{L}_{\mathrm{task}}$ paired with the model construction allows to perform end-to-end training, including sampling scheme optimization (in contrast to the DGM paper [118]). In the experiments, we use classification task datasets. Therefore, the proposed model is optimized through a cross-entropy loss.

The imputed data $\widehat{\mathrm{X}}$ along with the network predictions $\widehat{\mathrm{Y}}$ are obtained through representation $\mathrm{H}_{\mathrm{out}}$ as:

$$\widehat{\mathrm{X}} = [\mathbf{H}^{\mathrm{n}}(\mathrm{H}_{\mathrm{out}}), \mathbf{H}^{\mathrm{c}}_i(\mathrm{H}_{\mathrm{out}})] \tag{3.8}$$

$$\widehat{\mathrm{Y}} = \mathbf{H}^{\mathrm{task}}(\mathrm{H}_{\mathrm{out}}) \tag{3.9}$$

where $\mathbf{H}^{\mathrm{n}}, \{\mathbf{H}^{\mathrm{c}}_i\}^{\mathrm{c}}_{i=1}, \mathbf{H}^{\mathrm{task}}$ are linear projectors, acting row-wise on the input matrix $\mathrm{H}_{\mathrm{out}}$. The downstream task loss $\mathcal{L}_{\mathrm{task}}$ depends on the dataset, in our case that is cross entropy. Note that downstream task loss implicitly contributes to finding the best solution for the imputation problem and latent graph representation learning. The prototype nodes do not participate in any losses explicitly. However, implicit contribution through neighbor batch nodes allows the gradient to flow backwards and learn their representation. The mini-batch masking procedure introduces a surrogate objective to simulate the presence of missing data for which the reconstruction loss is computed. We optimize the MDI solution with the sum of Eq. equation 3.10 and equation 3.11, $\mathrm{M}^{\mathrm{n}}, \mathrm{M}^{\mathrm{c}}$ are numerical and categorical parts of surrogate batch-level corruption matrix M corresponding to $\mathrm{X}^{\mathrm{n}}, \mathrm{X}^{\mathrm{c}}$ subsequently.

$$\mathcal{L}_{\text{imp}}^{\text{n}} = \frac{\text{MSE}(\text{X}^{\text{n}} \odot \text{M}^{\text{n}}, \widehat{\text{X}}^{\text{n}} \odot \text{M}^{\text{n}})}{\sum(1 - \text{M}^{\text{n}})} \tag{3.10}$$

$$\mathcal{L}_{\text{imp}}^{\text{c}} = \frac{\text{CrossEntropy}(\text{X}^{\text{c}} \odot \text{M}^{\text{c}}, \widehat{\text{X}}^{\text{c}} \odot \text{M}^{\text{c}})}{\sum(1 - \text{M}^{\text{c}})} \tag{3.11}$$

$$\mathcal{L}_{\text{h}} = \sum_{i=1}^{\text{N}} \sum_{j=1}^{\text{N}} \bar{\text{a}}_{ij} \text{a}_{ij} \tag{3.12}$$

Graph learning generally assumes a homophilic structure of the data, i.e., similar patterns are connected with a higher probability, which in our case means that we expect patterns of the same class to be linked. We can explicitly enforce homophily by penalising interclass entries within the sampled adjacency matrix A. The target vector $\mathbf{Y}$ provides an idealized adjacency matrix $\text{A}_{\mathbf{Y}}$, where $\text{A}_{\mathbf{Y}_{ij}} = 1$ if $\mathbf{Y}_i$ and $\mathbf{Y}_j$ belong to the same class, otherwise zero. The complement of the idealized adjacency is denoted as $\bar{\text{A}}_{\mathbf{Y}}$. Equation equation 3.12 represents the penalization term, where $\bar{\text{a}}_{ij} \in \bar{\text{A}}_{\mathbf{Y}}$ and $\text{a}_{ij} \in \text{A}$. The final objective $\mathcal{L}$ is computed as a weighted combination of losses (3.10-3.12):

$$\mathcal{L} = \alpha \mathcal{L}_{\text{task}} + \beta(\mathcal{L}_{\text{imp}}^{\text{n}} + \mathcal{L}_{\text{imp}}^{\text{c}}) + \gamma \mathcal{L}_{\text{h}} \tag{3.13}$$

where α, β, γ are loss weights.

3.4 Evaluation

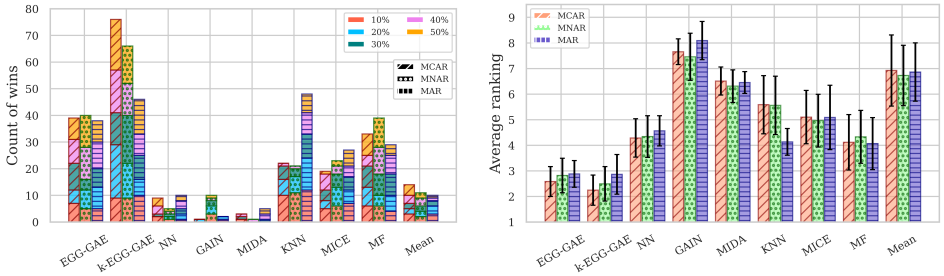
The primary experimental comparison is discussed in Section 3.4.1, whereas in Section 3.4.2, we concentrate on architectural ablations. All experiments are conducted at the following noise levels: $10\%, 20\%, 30\%, 40\%$, and 50%.

Datasets We validate our method on 15 datasets from the UCI repository by artificially introducing missing values using MCAR, MNAR, or MAR mechanisms (the setup for MAR and MNAR is replicated from [151]). Table 3.1 displays the aggregate statistics for the datasets. The training and test parts of the SUSY dataset were subsampled so that they could be compared to the majority of imputation baselines. The remaining datasets are divided into training sets (70%) and validation sets (30%). To satisfy a real-world scenario, we optimise the model with respect to additional noise introduced into the validation set (using the MCAR mechanism regardless of the source of initial missingness) and report the results concerning the initially missing values. Following the relevant literature, we evaluate the imputation and post-imputation prediction performance for each experiment.

Algorithms In the experiments the proposed architectures EGG-GAE and k-EGG-GAE, are compared with two statistical imputation methods: Mean [141], KNN [207], two machine learning imputation approaches: MICE [210], MissForest (MF) [192] and four deep learning approaches: MIDA [78], GINN [189], GAIN [254], and NN (the proposed architecture wherein EGG block is substituted with an MLP

Table 3.1. Dataset statistics. For each dataset, we provide the number of samples, the number of numerical features, and the number of categorical datasets.

Data type	Dataset	#Samples	#Num.Fs.	#Cat.Fs.
Numerical	Yeast	1484	8	0
	Wireless	2000	7	0
	Abalone	4177	8	0
	Wine-quality	4898	11	0
	Page blocks	5473	10	0
	Electrical grid stability	10000	14	0
	SUSY (small)	25000	18	0
Mixed	Anuran	7195	22	3
	Default credit card	30000	14	10
	Adult	32561	6	8
Categorical	Car	1728	0	6
	Phishing websites	2456	0	9
	Letter	20000	0	16
	Chess	28056	0	6
	Connect	67557	0	42



(a) Unified count of wins **(b)** Unified average ranking

Figure 3.3. Unified average ranking computed for the MCAR, MNAR, and MAR scenarios.

one that is identical to MLP_{FP}). To provide a fair comparison with the baselines, we apply the hyper-parameters and data preprocessing steps from the original papers for all datasets. The proposed models and NN baseline utilize the data pipeline described in this chapter.

Proposed architecture details The surrogate batch level corruption is introduced with the MCAR mechanism and $\text{I}_b = 0.2$. The number of EGG blocks is equal to 1. The hidden representations of all MLPs (feature propagation block and node mapper) are equal to 300. The batch size is equal to 300. The regularization homophily parameter is equal to 0.1. The temperature parameter τ linearly decreases from 0.5 to 0.01. During inference, the ensembling parameter is equal to 5. EGG-GAE and k-EGG-GAE share the majority of architectural hyperparameters, and for k-EGG-GAE we fix the number of sampled neighbors per node at $k = 5$. We utilize RMSprop for the optimization with the learning rate equal to 0.0001.

3.4.1 Imputation

The main set of experiments addresses the imputation reconstruction and predictive performance of the proposed networks in comparison to baseline algorithms

utilising the MCAR, MNAR, and MAR mechanisms. The predictive performance of an MDI solution for tabular data is typically measured by classical machine learning (ML) algorithms for a downstream task. To assess post-imputation downstream task performance, we employ random forest [38]. We show a schematic result with a unified count of wins and a unified average ranking that takes all levels of noise into account.

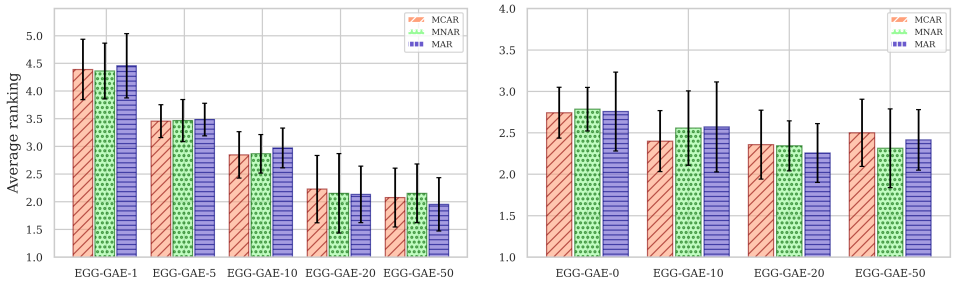
The unified count of wins shows the summary for each level of noise and missing mechanism (MCAR, MNAR and MAR). It represents the unified number of times that each method achieves the best performance with respect to the imputation or post-imputation task metrics, i.e., the lowest for RMSE, MAE and the highest in imputation and predictive accuracy. To compute the average ranking we first rank the model for each dataset according to the performance metric. The average ranking is then calculated by averaging the obtained rankings across all datasets. We obtain a separate average ranking for each level of noise, resulting in a matrix consisting of average rankings for every level of noise. The unified average ranking represents multiple average rankings based on a variety of performance metrics and is displayed as a bar graph with error bars. The bar height represents the mean of average rankings regarding the performance metrics and noise levels, while the error bars show corresponding variations. Note that GINN does not participate in the unified count of wins or unified average ranking, because execution time of GINN for big datasets is unfeasible within a reasonable amount of time.

In Fig. 3.3, it is evident that the suggested models outperform the baselines for every missing mechanism, especially for higher noise levels. Fig. 7.2a shows that k-EGG-GAE achieves the best score considerably more often than EGG-GAE, especially for MCAR and MNAR scenarios. Aggregating the results across every noise level, we observe that the EGG-GAE and k-EGG-GAE together accumulate $53\%, 49\%, 39\%$ of the best cases against the $15\%, 18\%$, and 22% of its best competitor for the MCAR, MNAR and MAR mechanisms, respectively. The machine learning baselines (KNN, MICE, and MF) are the strongest competitors, achieving the best performance in $33\%, 40\%$, and 48% of the cases (cumulatively) for the MCAR, MNAR, and MAR scenarios, accordingly. In Fig.7.2b, we can see that the proposed model framework using MLP instead of EGG (referred to as NN) performs just as well as MF on average, even though it rarely achieves the highest score. In addition, we can see that the proposed EGG-GAE and k-EGG-GAE models stay roughly on par.

3.4.2 Ablation experiments

We perform ablation studies over the numerical datasets (reported in Table 3.1). The ablation study examines model architecture alterations, evaluating ensembling and prototype nodes. The results are averaged across five runs and presented as unified average rankings based on end-to-end accuracy, RMSE and MAE. In Section 3.4.2 we analyze the proposed methods further by comparing training/inference timings for various neural baselines. The architectural experiments can be found in the supplementary materials, Appendix A.1.

Figure 3.4a shows that performing ensembling during inference enhances the performance of downstream and MDI problems regardless of missingness mechanisms.



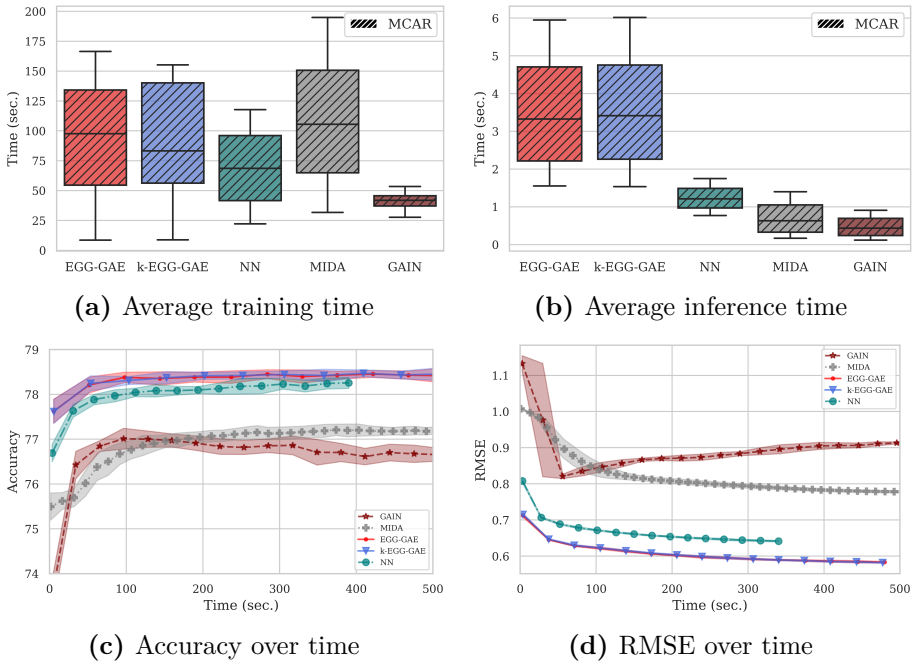
(a) Ensembling **(b)** Prototype nodes

Figure 3.4. Unified average ranking computed for the MCAR, MNAR, and MAR scenarios. A value following the model's name indicates the variable parameter: a number of prototype nodes or a number ensembling iterations.

We can see that increasing the number of ensembling iterations on average causes improvement for every type of missingness, as hypothesized. Both EGG-GAE-20 and EGG-GAE-50 have close mean values, while the corresponding variations share the same interval (around $[1.3, 2.8]$); this suggests that the plateau is reached when the number of iterations surpasses 20. We argue that performing ensembling during the inference leads to (i) reliable performance (ii) enhancing the performance of downstream and MDI problems. Figure 3.4b demonstrates that the models with learnable prototype nodes (EGG-GAE-10, EGG-GAE-20, and EGG-GAE-50) have a lower mean average ranking compared to the model without them (EGG-GAE-0), independent of the missingness mechanisms scenario. Increasing the number of prototype nodes helps to achieve better performance. However, we can see that introducing a significant number of prototype nodes might worsen the performance (EGG-GAE-50). We believe that the number of prototype nodes should be at least equal to the number of classes of downstream tasks. However, adding too many prototype nodes can impair the performance. We believe it affects the sampling scheme by increasing the likelihood that the graphs will be built primarily with prototype nodes, resulting in a poorer solution.

Time comparison

Figure 3.5a represents the average training time until convergence of the validation loss for the numerical datasets, while Figure 3.5b depicts the average models inference time. Note that the proposed architecture EGG-GAE does not vary a lot between average time training along with average inference time. This follows from the batch size (300) that was used to train EGG-GAE and k-EGG-GAE, which is fixed for all experiments. Increasing the batch size will result in quadratic time consumption growth due to the pairwise distance calculation. The average training time is approximately the same as MIDA and five times slower compared to GAIN. Average inference time is approximately 5-6 times greater compared to NN, MIDA and GAIN models, mostly due to the ensembling procedure. In Figures 3.5c and 3.5d we illustrate the evolution of accuracy and RMSE throughout training time in seconds for SUSY, averaged over 10 runs.



(a) Average training time **(b)** Average inference time

(c) Accuracy over time **(d)** RMSE over time

Figure 3.5. Average training/inference and performance time comparison for the neural approaches.

3.5 Related Work

Tabular missing data imputation

MDI algorithms can be categorized depending on whether they are discriminative or generative, univariate or multivariate, and on whether they provide one or multiple imputations for each missing data point. In this work we present a generative model which performs multivariate imputations and can provide multiple imputations for each missing datum.

The imputation strategies can be divided into three categories: 1) statistical; 2) machine learning (ML), and 3) deep learning (DL) based. Statistical methods exploit the observed data to obtain mean, median, or mode estimation of missing data points [70]. Among traditional machine learning imputation approaches, multiple imputation using chained equations (MICE) [210] is considered one of the most flexible and powerful. MICE iteratively imputes each dataset variable while keeping the others constant, selecting one or more observations from a predictive distribution on that variable. Although MICE has performed well in some cases, its underlying assumptions may result in biased forecasts and lower accuracy [7]. ML algorithms include k-nearest neighbors (KNN) [1], decision trees [130], support vector techniques [223] and several others. In practice, these approaches have mixed results compared to more straightforward techniques such as mean imputation [27]. KNN is generally limited to weighted averaging among similar feature vectors, whereas other algorithms are required to build a global dataset model for imputation. DL models include deep denoising autoencoders [78], recurrent neural networks [22], and generative models [254, 153]. Multilayer nonlinear computation allows these methods to capture more

complex correlations in data, however, they still require building a global model from the dataset while ignoring potentially significant contributions from similar points. GINN [189] addressed the problem of leveraging both the global aspect of the dataset and local dependencies between different data points utilizing GNNs. GINN requires the calculation of a pre-defined similarity matrix on the entire dataset in feature space, which is unfeasible for most real-world databases.

Latent graph learning

GNNs exploit the general idea of localized message-passing, e.g., using graph convolutional layers [122], GraphSAGE [95], edge convolutions [74], and graph attention [216]. In their basic formulation, GNNs layers require graph connectivity to be provided as input to the model. Mini-batches of data can be formed by sampling several graphs from a pool of graphs or sampling sub-graphs with a fixed number of nodes or connections from the original graph [266, 52]. The common disadvantages of these methods are that they require a given or fixed input graph and the requirement of a sparse graph to make the approach computationally feasible, especially for large graphs. Recently, methods which do not assume given or fixed graph connectivity were proposed. Such methods construct the graph dynamically during training. Wang et al. [225] proposed Dynamic Graph CNNs (DGCNN) using KNN to construct the graph on-the-fly in the feature space of the neural network. Later, a graph learning model was proposed in Cosmo et al. [54] that builds a probability graph as a weighted adjacency matrix for an optimal classification result. Thus, the graph is built in a fully connected manner, implying that the model cannot exploit the possible sparseness of the graph. A more general Differential Graph Module (DGM) [118] explicitly models sparse latent graphs employing the Gumbel-top-k trick [127], overcoming the dense graph limitations by fixing the number of neighbors for each node which allows working with bigger graphs. We take inspiration from DGM for our sampling scheme, and we detail the major differences in Section 3.3.

Node feature imputation in GNNs

GNNs models typically cannot deal with attribute-incomplete graph data directly, where rows represent nodes and columns feature channels. However, in real-world scenarios, features are often only observed for a subset of the nodes. Several works address missing node features in graph machine learning tasks. SAT [45] uses transformer-like models for feature completion followed by independent GNNs to solve the downstream task, which leads to a sub-optimal solution. GCNMF [198] overcomes this limitation by employing a Gaussian mixture model (GMM) to represent missing node features and jointly learns the GMM and GNNs parameters, however, this significantly increases the number of trainable parameters, implying high computational cost. PaGNNs introduced partial aggregation functions to propagate only the observed features [114]. However, these cannot scale to large graphs [172]. Recently a discrete diffusion-based feature reconstructions framework was proposed, which leads to a simple, fast and scalable iterative algorithm [172], though the method is designed to work for homophilic graphs.

3.6 Discussion

In this chapter, we introduce a generic framework for handling missing values in tabular data, employing graph deep learning. Particularly, we presented an end-to-end trainable graph autoencoder (EGG-GAE) model for learning the underlying graph representation of tabular data applied to the MDI problem. We performed extensive experiments with real-world datasets (from different fields) and determined that our model outperformed current state-of-the-art algorithms in terms of imputation and downstream task performance. We described several improvements to our model by demonstrating that ensembling improves MDI and dataset task performances; we further introduced novel learnable prototype nodes to encode common data patterns and serve as a generic, reliable subset of nodes for the predicted graphs. Finally, we introduced a regularization method that forces homophily in the learned latent graph representation.

As future works, the proposed EGG-GAE network can be applied to any type of data to introduce a graph topology for, e.g., imputing missing data over images, audio, or other types of high-dimensional data, exploiting the modularity of modern deep learning architectures. In addition, the Euclidean distance calculation can be substituted with a trainable network to construct probabilistic graphs based on a learned metric distance function. The assumptions and limitations of the proposed framework are described in supplementary materials, Appendix A.2.

Chapter 4

From Latent Graph to Latent Topology Inference

Contents

This chapter is organized as follows. In Section 4.1, we introduce the problem of inferring higher-order structures from data and discuss its motivation. Section 4.2 provides the necessary background for the following Section 4.3, where we present the methodology behind the proposed DCM module and its integration into a pipeline for inferring cell complexes from data. Section 4.4 presents comprehensive experimental results that support the proposed approach. Finally, the chapter concludes with a discussion in Section 4.5.

4.1 Introduction

As described in Section 1.1, GNNs have shown remarkable performance in learning tasks where data are represented over a graph domain due to their ability to combine the flexibility of neural networks with prior knowledge about data relationships, expressed in terms of the underlying graph topology.

The content of this chapter is adapted from the material published in [17]

The majority of GNNs assume the graph topology to be fixed (and optimal) for the task at hand. Therefore, the focus is usually on designing more sophisticated architectures with the aim of improving the message-passing process described in Section 2.2. As has been described in Chapter 3, very recently, a series of works [117, 58, 205] started to investigate LGI techniques for, where the intuition is that data can have some underlying but unknown (latent) graph structure, mainly in cases where only a point cloud of data is available but also when the given graph is suboptimal for the downstream task. At the same time, the field of TDL [12, 92] started to gain interest, motivated by the fact that many systems are characterized by higher-order interactions that cannot be captured by the intrinsically pairwise structure of graphs (see Chapter 2). However, TDL techniques usually transform graphs to higher-order complexes by means of deterministic lifting maps, assigning higher-order cells to cliques or induced cycles of the graph, thus implicitly assuming that these (task-agnostic, deterministic) complexes are optimal for the downstream task. In addition, whenever an input graph is not available, these strategies scale poorly in the size of the input set by requiring an exhaustive (combinatorial) search over all possible cells, making them unfeasible for even medium-sized sets of data points.

In this chapter, we introduce the concept of Latent Topology Inference (LTI) by generalizing Latent Graph Inference (LGI) to higher-order complexes. The goal of LTI is not (only) learning a graph structure describing pairwise interactions but rather learning a higher-order complex describing multi-way interactions among data points. As a first instance of LTI, we introduce the Differentiable Cell Complex Module (DCM), a novel deep learning architecture that dynamically learns a cell complex to improve the downstream task. The DCM implements a two-step inference procedure to alleviate the computational burden: first, learning the 1-skeleton of the complex (i.e., a graph) via a novel improved version of the Differentiable Graph Module (DGM) [117], and then learning which higher-order cells (polygons) should be included in the complex. Both steps leverage message-passing (at node and edge levels) and a sparse sampling technique based on the α-entmax class of functions, which allows overcoming the limitation of the original DGM, capable of learning only regular graph topologies. We generalize the training procedure of the Differentiable Graph Module (DGM) [117] to train the DCM in an end-to-end fashion. The DCM is tested on several homophilic and heterophilic datasets, and it is shown to outperform other state-of-the-art techniques, offering significant improvements *both when the input graph is provided or not.* In particular, accuracy gains on heterophilic benchmarks with provided input graphs indicate that the DCM leads to robust performance even when the input graph does not fit the data well.

4.2 Background

Most of the definitions required in this chapter are provided in Section 2. In particular, the definition of cell complex 2.1.4 is crucial. While Definition 2.2.2 outlines the concept of neighborhood functions for this chapter, it is essential to introduce boundary matrices that describe the structure of the cell complex. To do so, we must first establish an orientation of the cells. One approach to orienting cells, as proposed by Sardellitti et al. [179], involves a simplicial decomposition of the

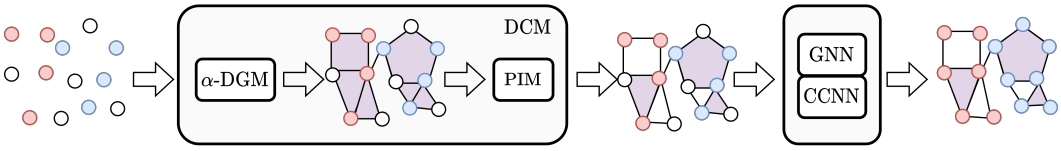


Figure 4.1. The proposed two-step procedure for Latent Topology Inference (LTI) via regular cell complexes. The Differentiable Cell Complex Module (DCM) is a function that first learns a graph describing the pairwise interactions among data points via the α-Differentiable Graph Module (α-DGM), and then it leverages the graph as the 1-skeleton of a regular cell complex whose 2-cells (polygons), describing multi-way interactions among data points, are learned via the PIM. The inferred topology is then used in two message-passing networks, at node (Graph Neural Network, GNN) and edge (Cell Complex Neural Network, CCNN) levels to solve the downstream task. The whole architecture is trained in an end-to-end fashion.

complex. This process subdivides the cell into a set of internal k-simplices [133, 12], such that: i) two simplices share exactly one $(k-1)$-simplicial boundary element, which is not the boundary of any other k-cell in the complex; and ii) two k-simplices induce an opposite orientation on the shared $(k-1)$-boundary. Consequently, by orienting a single internal simplex, the orientation propagates throughout the entire cell. Given an orientation, the structure of an oriented regular cell complex of order K is fully captured by the set of its incidence (or boundary) matrices $\mathbf{B}k \in \mathbb{R}^{Nk-1 \times N_k}$, $k = 1, \dots, K$. The entries of these matrices are defined as $\mathbf{B}_k(i,j) = 0$ if σ_i^{k-1} is not a face of σ_j^k, and $\mathbf{B}_k(i,j) = 1$ (or -1) if σ_i^{k-1} is a face of σ_j^k and its orientation is coherent (or not) with the orientation of σ_j^k. These boundary matrices describe the relations between $k-1$-cells and k-cells. The corresponding adjacency, termed lower/upper, can be derived for each level of the cell complex, describing the relations through the lower/upper boundary cells. Hence, the lower adjacent is defined as $\mathcal{N}d(\sigma i^K) = \mathbf{B}_k\mathbf{B}_k^T$ and the upper adjacent as $\mathcal{N}u(\sigma i^K) = \mathbf{B}_k^T\mathbf{B}_k$.

Definition 4.2.1 (k-skeleton) A *k-skeleton* of a cell complex $\mathcal{C}$ is the subcomplex of $\mathcal{C}$ consisting of cells of dimension at most k.

The 0-skeleton of a cell complex is a set of vertices, and the 1-skeleton is a set of vertices and edges, thus forming a graph. Consequently, given a graph $\mathcal{G} = \mathcal{V}, \mathcal{E}$, it is possible to construct a cell complex "on top of it", i.e., a cell complex $\mathcal{C}_\mathcal{G}$ whose 1-skeleton is isomorphic to $\mathcal{G}$. In this work, we attach order 2 cells to an inferred (learned) subset of induced cycles of the graph $\mathcal{G}$ up to length K_{max}. We refer to these as *polygons*, and we denote the resulting order-2 cell complex as $\mathcal{C}_\mathcal{G} = \mathcal{V}, \mathcal{E}, \mathcal{P}$, where $\mathcal{P}$ is the set of polygons, with $|\mathcal{V}| = N$, $|\mathcal{E}| = E$, and $|\mathcal{P}| = P$. This procedure is formally a skeleton-preserving lifting map; a detailed discussion about the lifting of a graph into a cell complex can be found in Bodnar et al. [31].

Signals over Cell Complexes A k-cell signal is defined as a mapping from the set $\mathcal{D}_k$ of all k-cells contained in the complex, with $|\mathcal{D}_k| = N_k$, to real numbers [179]:

$$x_k : \mathcal{D}_k \to \mathbb{R} \tag{4.1}$$

Therefore, for an order-2 complex $\mathcal{C}_\mathcal{G}$, the k-cell signals are defined as the following mappings:

$$x_0 : \mathcal{V} \to \mathbb{R}, \qquad x_1 : \mathcal{E} \to \mathbb{R}, \qquad x_2 : \mathcal{P} \to \mathbb{R}, \tag{4.2}$$

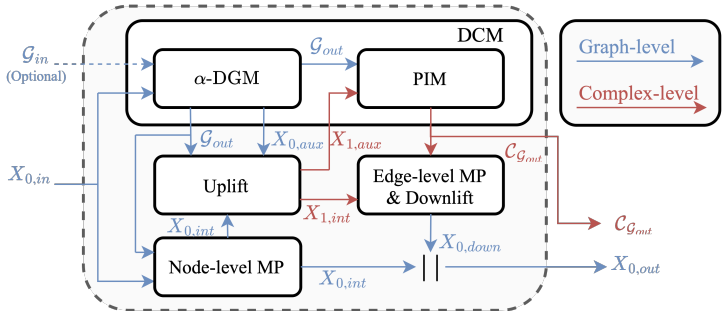


Figure 4.2. The DCM and the proposed architecture.

representing node, edge, and polygon signals, respectively, with $N_0 = N$, $N_1 = E$, and $N_2 = P$. If F $k-$cell signals are available, we collect them in a feature matrix $\mathbf{X}_{k,in} = \{\mathbf{x}_{k,in}(i)\}_{i=1}^{N_k} \in \mathbb{R}^{N_k \times F}$, where the i-th row $\mathbf{x}_{k,in}(i) = [x_{k,1}(\sigma_i^k), \dots, x_{k,F}(\sigma_i^k)] \in \mathbb{R}^F$ collects the features, i.e. the signals values, of the i-th $k-$cell σ_i^k. The definition of $k-$cell signal in equation 4.1 is sufficient and rigorous for the scope of this work, however more formal topological characterizations in terms of chain and cochain spaces can be given [179, 31, 92].

4.3 A Differentiable Layer for Latent Topology Inference

We propose a novel layer, depicted at high level in Figure 4.1 and in detail in Figure 4.2, comprising of a series of modules among which the most important one is the *Differentiable Cell Complex Module* (DCM), the first fully differentiable module that performs LTI, i.e. learns a cell complex describing the hidden higher-order interactions among input data points. The DCM is equipped with a novel sampling algorithm that, at the best of our knowledge, is the first graph/complex sampling technique that allows to generate topologies whose neighborhoods cardinality is not fixed a priori but can be freely learned in an end-to-end fashion.

The proposed layer takes as input the node feature matrix $\mathbf{X}_{0,in} \in \mathbb{R}^{N \times F_{in}}$, and gives as output the updated node feature matrix $\mathbf{X}_{0,out} \in \mathbb{R}^{N \times F_{out}}$ and the inferred latent cell complex $\mathcal{C}_{\mathcal{G}_{out}}$. Optionally, the layer can take as input also a graph $\mathcal{G}_{in}$. To make the layer self-consistent in a multi-layer setting, the output can be reduced to the updated node feature matrix $\mathbf{X}_{0,out}$ and the 1-skeleton (graph) $\mathcal{G}_{out}$ of $\mathcal{C}_{\mathcal{G}_{out}}$.

We employ a two-step inference procedure to keep the computational complexity tractable, i.e. the DCM module first samples the 1-skeleton of the latent cell complex (possibly in a sparse way, i.e. $E << N^2$), and then it samples the polygons among the induced cycles generated by the sampled edges. The first step is implemented via the novel α-Differentiable Graph Module ($\alpha-$DGM), while the second step is implemented via the novel Polygon Inference Module (PIM). Directly sampling among all the possible polygons, thus trivially generalizing the DGM framework [117], would have led to intractable complexity, e.g. even at triangles level the sampling would have had a complexity of the order of $\mathcal{O}(\sqrt{E^3}) = \mathcal{O}(N^3)$, being all edges candidate to be sampled. In the following, we describe in detail each module of the proposed layer.

4.3.1 The α-Differentiable Graph Module

The α-DGM is a novel variation of the DGM and it is responsible for inferring the 1-skeleton (graph) of the latent cell complex. One of the main limitations of the DGM [117] is the constraint of inferring only regular graphs; we solve this problem by proposing a novel sampling procedure based on the $\alpha-$entmax class of functions [164], recently introduced in the NLP community to obtain sparse alignments and zero output probabilities. With fixed $\alpha > 1$ and input $\mathbf{s} \in \mathbb{R}^d$, the $\alpha-$entmax function $\boldsymbol{\alpha}_{ENT} : \mathbb{R}^d \to \Delta^d$ is defined as:

$$\boldsymbol{\alpha}_{ENT}(\mathbf{s}) = \underset{\mathbf{u} \in \Delta^d}{\arg\max} \, \langle \mathbf{u}, \mathbf{s} \rangle_{\mathbb{R}^d} + H_\alpha(\mathbf{u}), \tag{4.3}$$

where $\Delta^d = \{\mathbf{u} \in \mathbb{R}^d : \mathbf{u} \geq 0, \|\mathbf{u}\|_1 = 1\}$ is the d-dimensional probability simplex and H_α is the family of Tsallis α-entropies parametrized by a parameter $\alpha \geq 1$ [164]. For $\alpha = 1$, equation equation 4.3 recovers the standard softmax, while for $\alpha > 1$ solutions can be sparse with a degree depending on α. In practice, α can be initialized at a reasonable value (e.g., 1.5) and adapted via gradient descent [53].

In α-DGM, we first compute auxiliary node features $\mathbf{X}_{0,aux} = \nu_0(\mathbf{X}_{0,in}) \in \mathbb{R}^{N \times d_0}$, where $\nu_0(\cdot)$ is a learnable function, e.g. a GNN if an initial graph $\mathcal{G}_{in}$ is provided or an MLP otherwise. At this point, a (pseudo-)similarity function $\text{sim}(\cdot)$ is chosen, the similarities among node embeddings are computed and collected in the vectors $\mathbf{z}_i = \{\text{sim}(\mathbf{x}_{0,aux}(i), \mathbf{x}_{0,aux}(j))\}_j \in \mathbb{R}^N$, $i, j = 1, \dots, N$. Valid examples of $\text{sim}(\cdot)$ are cosine similarity, inverse or minus Euclidean square distance.

The (sparse) edge probabilities are then obtained node-wise via $\alpha-$entmax, i.e. the vectors $\mathbf{p}_i = \boldsymbol{\alpha}_{ENT}(\mathcal{LN}(\mathbf{z}_i)) \in \mathbb{R}^N$, $i = 1, \dots, N$ are computed; layer normalization $\mathcal{LN}$ is employed to have better control on the similarities statistics and, consequently, more stability in the training procedure. The (i, j) edge is included in the inferred graph if $\mathbf{p}_i(j) > 0$. This procedure leads to a directed graph that would be incompatible with the cell complex structure, whose 1-skeleton is an undirected graph; the inferred directed graph is converted to the closest undirected graph $\mathcal{G}_{out} = \{\mathcal{N}, \mathcal{E}\}$, i.e. each inferred edge is considered as a bidirectional edge to obtain $\mathcal{E}$. We want further stress the fact that computing edge probabilities via the α-entmax leads to sparse and non-regular graphs whose sparsity level can be controlled (or learned) tuning the parameter α. A detailed pictorial description of the α-DGM is shown in Figure 3 (left).

Remark 2. The choice of applying the α-entmax in a node-wise fashion and not directly on a vector containing all the possible similarities (that would have naturally led to an undirected graph) is due to the fact that, in the latter case, we empirically observed that the α-entmax consistently leads to heavily disconnected graphs with few hub nodes (therefore, also leading to performance drops).

At this point, the intermediate node features $\mathbf{X}_{0,int} \in \mathbb{R}^{N \times F_{int}}$ are computed with one or more usual message-passing (MP) rounds over the inferred graph:

$$\mathbf{x}_{0,int}(i) = \gamma_0 \left(\mathbf{x}_{0,in}(i), \bigoplus_{j \in \mathcal{N}_u(\sigma_i^0)} \phi_0^{\mathcal{N}_u} \left(\mathbf{x}_{0,in}(i), \mathbf{x}_{0,in}(j) \right) \right), \tag{4.4}$$

where $\gamma_0(\cdot)$ and $\phi_0^{\mathcal{N}_u}(\cdot)$ are learnable functions and $\bigoplus$ is any aggregation operator.

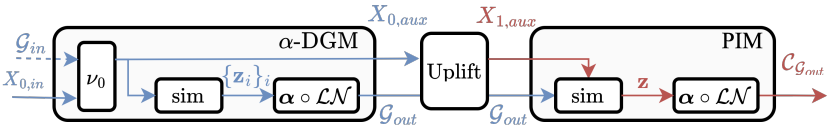


Figure 4.3. The α-Differentiable Graph Module (left) and the Polygon Inference Module (right).

4.3.2 Uplift Module

To fully exploit the potential of cell complexes, it is not sufficient to work at the node level. For this reason, intermediate edge features are learned (or computed) with a MP round of the form:

$$\mathbf{x}_{1,int}(i) = \psi_1\left(\bigoplus_{j \in \mathcal{B}(\sigma_i^1)} \phi_1^{\mathcal{B}}\left(\mathbf{x}_{0,int}(j)\right)\right), \tag{4.5}$$

where $\psi_1(\cdot)$ is a (possibly) learnable function. The boundary $\mathcal{B}(\sigma_i^1)$ of edge i is composed by its endpoints nodes σ_j^0 and σ_v^0. Therefore, the intermediate features $\mathbf{x}_{1,int}(i)$ of edge i are learned (or computed) as a function of the intermediate node features $\mathbf{x}_{0,int}(j)$ and $\mathbf{x}_{0,int}(v)$.

4.3.3 The Polygon Inference Module

The Polygon Inference Module (PIM) is a novel module responsible of inferring the polygons of the latent cell complex by sampling a subset of the induced cycles of the inferred 1-skeleton $\mathcal{G}_{out}$.

To this aim, auxiliary edge features are computed from the auxiliary node features with a MP round as in equation 4.5, which we denote in this context with $\mathbf{X}_{1,aux} = \nu_1(\mathbf{X}_{0,aux}) \in \mathbb{R}^{E \times d_1}$ for notation consistency.

At this point, we need a similarity function for the edge embeddings belonging to the same induced cycle. We decide to use the sum of the pairwise similarities, e.g. for a generic cycle of length k made by the edges whose indices are collected in a index set $\mathcal{I}$, the similarity is computed (with a slight abuse of notation) as:

$$\text{sim}(\{\mathbf{x}_{1,aux}(i)\}_{i \in \mathcal{I}}) = \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}, j \neq i} \text{sim}(\mathbf{x}_{1,aux}(i), \mathbf{x}_{1,aux}(j)). \tag{4.6}$$

The similarity in equation 4.6 for an induced cycle of length $k \leq K_{max}$ contains $\frac{k!}{(k-2)!2!}$ terms, however, K_{max} is usually a very small integer and the sum can be trivially distributed. In general, $\text{sim}(\{\mathbf{x}_{1,aux}(i)\}_{i \in \mathcal{I}})$ doesn't need to be the sum of pairwise similarities. It can be arbitrarily designed, as long as it is a similarity measure, i.e. it is higher if the involved embeddings are similar.

The similarities are collected in a vector $\mathbf{z} \in \mathbb{R}^{\widetilde{P}}$, where $\widetilde{P}$ is the number of induced cycles, and the polygons probabilities are computed as $\mathbf{p} = \boldsymbol{\alpha}_{ENT}(\mathcal{LN}(\mathbf{z})) \in \mathbb{R}^{\widetilde{P}}$. We set $\mathcal{C}_{\mathcal{G}_{out}} = \{\mathcal{N}, \mathcal{E}, \mathcal{P}\}$, where $\mathcal{P}$ are the induced cycles with positive probabilities and $|\mathcal{P}| = P \leq \widetilde{P}$ (possibly $P << \widetilde{P}$).

The updated edge features $\mathbf{X}_{1,out} \in \mathbb{R}^{E \times F_{out}}$ are computed with one or more MP rounds as:

$$\mathbf{x}_{1,out}(i) = \gamma_1\left(\mathbf{x}_{1,int}(i), \bigotimes_{\mathcal{N}_k \in \{\mathcal{N}_d, \mathcal{N}_u\}} \bigoplus_{j \in \mathcal{N}_k(\sigma_i^1)} \phi_1^{\mathcal{N}_k}\left(\mathbf{x}_{1,int}(i), \mathbf{x}_{1,int}(j)\right)\right), \tag{4.7}$$

where $\bigotimes$ is a neighborhoods aggregation operator [92], $\gamma_1(\cdot)$, $\phi_1^{\mathcal{N}_d}(\cdot)$, and $\phi_1^{\mathcal{N}_u}(\cdot)$ are learnable functions. A detailed pictorial description of the PIM is shown in Figure 3 (right).

Remark 3. The message-passing rounds in equation 4.7 and in equation 4.5 (as a special case) are instances of message-passing neural networks over cell complexes. Several MP schemes can be defined for cell (simplicial, combinatorial) complexes, please refer to [31, 92] for more details. We employ the schemes in equation 4.5-equation 4.7 for two reasons: the first one is that using more sophisticated MP rounds (e.g. moving to cells of higher order than polygons or designing more intensive messages exchange among different cell orders) would have lead to computational intractability; the second one is that moving to higher order cells also introduces a series of tricky theoretical issues (in terms of the structure of the complex) that are not trivial to tackle [96]. We plan to investigate these directions in future works.

4.3.4 Downlift Module and Output Computation

At this point, the output node features $\mathbf{X}_{0,out} \in \mathbb{R}^{N \times F_{out}}$ are learned (or computed) from the updated edge features $\mathbf{X}_{1,out} \in \mathbb{R}^{E \times F_{out}}$ and the intermediate node features $\mathbf{X}_{0,int} \in \mathbb{R}^{N \times F_{int}}$ as:

$$\mathbf{x}_{0,out}(i) = \Big[\mathbf{x}_{0,int}(i) \,\Big|\Big|\, \mathbf{x}_{0,down}(i)\Big], \tag{4.8}$$

where the $\mathbf{X}_{0,down} \in \mathbb{R}^{N \times F_{out}}$ are obtained with a MP round of the form:

$$\mathbf{x}_{0,down}(i) = \psi_0\left(\bigoplus_{j \in \mathcal{CB}(\sigma_i^0)} \phi_0^{\mathcal{CB}}\left(\mathbf{x}_{1,out}(j)\right)\right) \tag{4.9}$$

with $\psi_0(\cdot)$ being a (possibly) learnable function. The coboundary $\mathcal{CB}(\sigma_i^0)$ of node i is composed by all the edges for which node i is an endpoint. Therefore, the output features $\mathbf{x}_{0,out}(i)$ of node i are learned (or computed) as the concatenation of its intermediate features and features obtained downlifiting the updated features of the edges for which i is an endpoint.

We present a comparison in terms of computational complexity between the DCM and the DGM [117] in Appendix B.3 .

4.3.5 Training of the Differentiable Cell Complex Module

The proposed sampling scheme based on the α-entmax does not allow the gradient of the downstream task loss to flow both through the graph and polygons inference branches of the DCM, due to the fact that they involve only auxiliary features and the entmax outputs are substantially just a way of indexing the edges and the polygons

present in the inferred complex. To enable a task-oriented end-to-end training of the DCM, we generalize the approach of DGM [117, 58] and design an additional loss term that rewards edges and polygons involved in a correct classification (in classification tasks) or in "good" predictions (in regression tasks), and penalizes edges and polygons that lead to misclassification or high error predictions. Refer to Appendix B.2 for the details.

4.4 Evaluation

In this Section, we evaluate the effectiveness of the proposed framework on several heterophilic and homophilic graph benchmarks. Graph homophily is the principle that nodes with similar labels are more likely to be connected. Traditional Graph/Cell Complex Convolutional Neural Networks (GCNs and CCCNs) implicitly rely on the homophily assumption, and performance drops are typically observed in heterophilic settings [34, 190, 191].

Our main goal is to show that *latent topology inference* via the differentiable cell complex module (DCM) allows learning higher-order relationships among data points that lead to significant improvements w.r.t. *latent graph inference*. Since DCM is a generalization of the differentiable graph module (DGM), we use as a comparison the original (discrete) DGM [117] (denoted DGM-E) and its recently introduced non-Euclidean version [58] (denoted DGM-M). Moreover, we also report the results of a simplified variant of our model (denoted as DCM ($\alpha = 1$)), in which the graph is explicitly learned and all the polygons are taken, i.e. $\alpha = 1$ in the PIM; in this (lower complexity) case, the α-DGM guides both the edges (explicitly) and the polygons (implicitly) inference steps. We also test a variant, denoted with GCN-CCCN, where the complex is not imputed at all, i.e. the input graph is used in a GCN and all the polygons are used in a CCCN, and a variant, denoted with KCM, where the graph is imputed using a KNN after embedding the node features using an MLP/GCN, and all the polygons are used. Finally, we report the vanilla MLP and GCN [123], GCN2 [44], GAT [215], and (convolutional) CWN [31] as further baselines and comparisons. In all the experiments, we utilize GCNs at the graph level and CCCNs at the edge level; the details about the architectures employed to obtain the results are given in Appendix B.5. The **first** and the **second** best results are highlighted.

We follow the same core experimental setup of [58] on transductive classification tasks; in particular, we first focus on standard graph datasets such as *Cora*, *CiteSeer* [251], *PubMed*, *Physics* and *CS* [185], which have high homophily levels ranging from 0.74 to 0.93. We then test our method on several challenging heterophilic datasets, *Texas*, *Wisconsin*, *Squirrel*, and *Chameleon* [175], which have low homophily levels ranging between 0.11 and 0.23. The results for the homophilic and heterophilic datasets are presented in Tables 4.1 and 4.2, respectively. All the models were tested in two settings: assuming the original graph is available (marked *w graph* in Tables 4.1 and 4.2) and a more challenging case in which the input graph is assumed to be unavailable (*w/o graph*).

From Tables 4.1 and 4.2, it is evident that the DCM exhibits consistently superior performance compared to alternative methods across both homophilic and

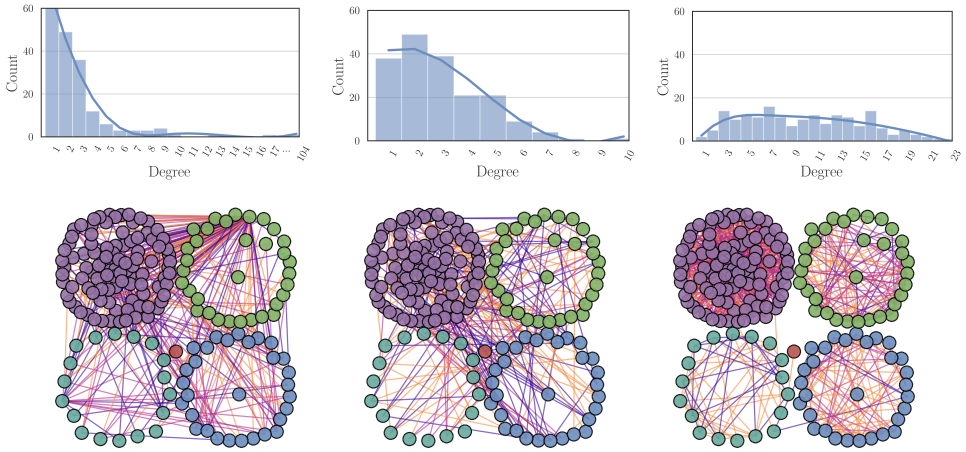
Table 4.1. Homophilic-graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

		Cora	CiteSeer	PubMed	Physics	CS
	Model/Hom. level	0.81	0.74	0.80	0.93	0.80
w/o graph	MLP	58.92 ± 3.28	59.48 ± 2.14	85.75 ± 1.02	94.91 ± 0.28	87.80 ± 1.54
	KCM	78.47 ± 2.09	75.20 ± 2.41	86.66 ± 0.91	95.61 ± 0.18	95.14 ± 0.32
	DGM-E	62.48 ± 3.24	62.47 ± 3.20	83.89 ± 0.70	94.03 ± 0.45	76.05 ± 6.89
	DGM-M	70.85 ± 4.30	68.86 ± 2.97	**87.43** ± 0.40	95.25 ± 0.36	92.22 ± 1.09
	DCM	**78.80** ± 1.84	**76.47** ± 2.45	87.38 ± 0.91	**96.45** ± 0.12	**95.40** ± 0.40
	DCM ($\alpha = 1$)	**78.73** ± 1.99	**76.32** ± 2.75	**87.47** ± 0.77	**96.22** ± 0.27	**95.35** ± 0.37
w graph	GCN	83.11 ± 2.29	69.97± 2.00	85.75 ± 1.01	95.51 ± 0.34	87.28 ± 1.54
	GCN2	**87.85** ± 1.41	78.53 ± 2.66	**89.60** ± 0.70	**97.41** ± 0.34	95.05 ± 0.38
	GAT	**89.81** ± 1.77	78.18 ± 2.31	88.53 ± 0.61	**98.87** ± 0.30	94.42 ± 0.70
	KCM	78.43 ± 2.11	75.23 ± 2.45	86.61 ± 0.95	96.16 ± 0.17	95.46 ± 0.36
	CWN	88.63 ± 1.91	75.53 ± 2.13	87.97 ± 0.77	96.23 ± 0.24	93.52 ± 0.59
	GCN+CCCN	86.09 ± 1.82	78.36 ± 3.33	88.59 ± 0.67	96.90 ± 0.30	95.31 ± 0.49
	DGM-E	82.11 ± 4.24	72.35 ± 1.92	87.69 ± 0.67	95.96 ± 0.40	87.17 ± 3.82
	DGM-M	86.63 ± 3.25	75.42 ± 2.39	87.82 ± 0.59	96.21 ± 0.44	92.86 ± 0.96
	DCM	85.78 ± 1.71	**78.72** ± 2.84	88.49 ± 0.62	96.99 ± 0.44	**95.79** ± 0.48
	DCM ($\alpha = 1$)	85.97 ± 1.86	**78.60** ± 3.16	**88.61** ± 0.69	96.69 ± 0.46	**95.78** ± 0.49

Table 4.2. Heterophilic-graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

		Texas	Wisconsin	Squirrel	Chameleon
	Model/Hom. level	0.11	0.21	0.22	0.23
w/o graph	MLP	77.78 ± 10.24	85.33 ± 4.99	30.44 ± 2.55	40.35 ± 3.37
	KCM	84.12 ± 11.37	87.10 ± 5.15	35.15 ± 1.38	52.12 ± 2.02
	DGM-E	80.00 ± 8.31	**88.00** ± 5.65	34.35 ± 2.34	48.90 ± 3.61
	DGM-M	81.67 ± 7.05	**89.33** ± 1.89	35.00 ± 2.35	48.90 ± 3.61
	DCM	**85.71** ± 7.87	87.49 ± 5.94	**35.55** ± 2.24	**53.63** ± 3.07
	DCM ($\alpha = 1$)	**84.96** ± 10.24	86.72 ± 6.02	**35.25** ± 2.22	**53.67** ± 3.19
w graph	GCN	41.66 ± 11.72	47.20 ± 9.76	24.19 ± 2.56	32.56 ± 3.53
	GCN2	75.50 ± 7.81	74.57 ± 5.38	33.09 ± 1.76	49.50 ± 3.02
	GAT	66.72 ± 11.22	60.52 ± 9.23	**35.07** ± 2.13	50.73 ± 3.12
	KCM	83.92 ± 11.15	84.92 ± 5.21	34.47 ± 1.49	**53.12** ± 2.02
	CWN	65.87 ± 6.33	64.57 ± 7.12	32.44 ± 2.75	43.86 ± 2.51
	GCN+CCCN	84.43 ± 9.11	84.03 ± 5.42	OOM	OOM
	DGM-E	60.56 ± 8.03	70.67 ± 10.49	29.87 ± 2.46	44.19 ± 3.85
	DGM-M	62.78 ± 9.31	76.00 ± 3.26	30.44 ± 2.38	45.68 ± 2.66
	DCM	**84.87** ± 10.04	**86.33** ± 5.14	34.95 ± 2.59	53.05 ± 3.00
	DCM ($\alpha = 1$)	**84.96** ± 5.60	**85.36** ± 5.05	**35.13** ± 2.27	**53.76** ± 3.72

heterophilic datasets, both with and without provided input graphs. As expected, our method achieves top performance without an input graph for the heterophilic datasets and with an input graph for the homophilic datasets. We achieve remarkable results considering the input graph for heterophilic datasets. Despite exposing the model to the "wrong" input graphs (in the sense that the topology does not match the structure of the node features), the overall performance of the DCM remains stable, yielding remarkable improvements exceeding 20% for Texas and approximately 10% for Wisconsin compared to DGM-M [58]. Therefore, observing the results across both homophilic and heterophilic datasets when the input graph is provided, we can appreciate how DCM exploits the available "good" graphs in the first case while being less sensitive to the "wrong" ones in the latter. Our performance on the heterophilic datasets suggests that the learned latent complex has a homophilic 1-skeleton (graph) that enables the involved GCNs and CCCNs to reach higher accuracies. To corroborate this hypothesis, in Figure 4.4, we show the evolution of the latent topology during the training for the Texas dataset along with the nodes



(a) Input, $\%_p = 100$, $h = 0.11$ **(b)** Epoch 40, $\%_p = 90$, $h = 0.40$ **(c)** Epoch 180, $\%_p = 20$, $h = 0.99$

Figure 4.4. Evolution of the latent complex for the Texas dataset, along with homophily level and nodes degree distribution. Edges in orange, triangles in lilac, squares in purple ($K_{max} = 4$)

degree distribution, the percentage of sampled polygons $\%_p$, and the homophily level h that we note reaching 0.99 on the final inferred graph from the initial 0.11. Moreover, the fact that most of the inferred polygons belong to the same class and the pretty high spread of the degree distribution further confirm the effectiveness of the proposed architecture. In Appendix B.4, we report ablation studies, plots, and interpretability hints on the inferred complexes.

4.5 Discussion

To our knowledge, DCM is the first differentiable approach for latent topology inference. The promising results open several avenues for future work.

Methodological. Although cell complexes are very flexible topological objects, other instances of LTIcould leverage hypergraphs or combinatorial complexes (CCs) [89], which are able to handle both hierarchical and set-type higher-order interactions. Second, remaining within cell complexes, different MP schemes [92], lifting maps [31], or model spaces [58] are of interest for future work. Third, extending our framework to weighted [16] or directed complexes [56] could give further insights. Finally, one potentially interesting direction is merging LTIand Sheaf Neural Networks in order to learn cellular sheaves defined on latent higher-order complexes.

Computational. While most of our experimental validation of the DCM focused on transductive tasks, we could also tackle inductive ones [58]. Future works could tailor the DCM to fit challenging specific applications as the ones presented in the introduction for LGI. Moreover, though computationally tractable thanks to the proposed two-step inference procedure, DCM may benefit from further improvements to effectively scale on very large datasets. The main bottlenecks are MP operations and the search for the induced cycles of the learned graph. A possible solution for

the former could be neighbor samplers [58], while the latter could be tackled by leveraging stochastic search methods or moving to more flexible topological spaces, e.g. CCs. Finally, our sampling strategy implicitly assumes that there are no specific edges and polygons distribution requirements, e.g. a sampling budget is given or a particular correlation structure needs to be imposed on the cells. In these cases, incorporating more sophisticated sampling methods like IMLE [136, 184] could be beneficial.

Chapter 5

Topological Network Traffic Compression

Contents

This chapter is organized as follows. In Section 5.1, we introduce the problem of network traffic compression and provide the motivation for the proposed TDL-based lossy traffic compression framework. In Section 5.2, we detail the methodology for detecting higher-order structures in traffic data, along with the application of TNN to achieve a compressed representation. Section 5.3 presents the evaluation of the proposed methodology, while Section 5.4 reviews related work. The chapter concludes with a discussion of the approach in Section 5.5.

5.1 Introduction

Computer Network's traffic has significantly increased in recent years [208], specially driven by the development of new applications –such as vehicular networks, Internet of Things, virtual reality, video streaming– and the advancement of network technology –e.g. the fast improvements in link speed. In fact, current Internet Service Providers (ISP) and datacenter networks can easily produce hundreds of

The content of this chapter is adapted from the material published in [25, 24]

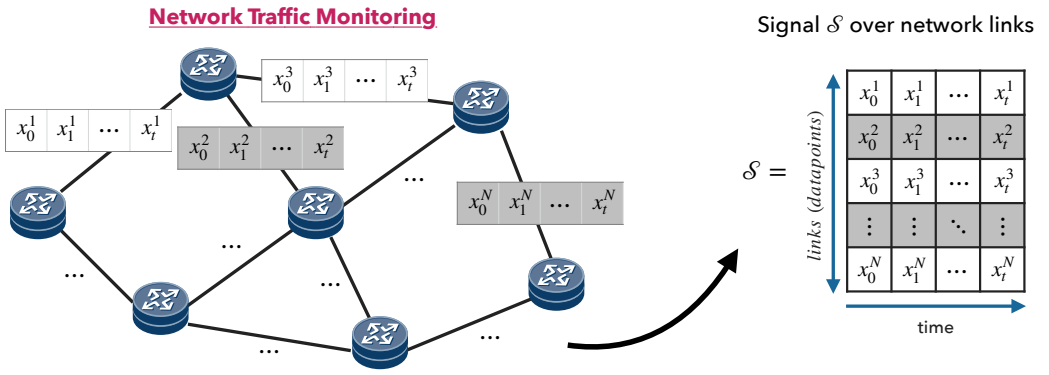


Figure 5.1. The goal is to compress a signal $\mathcal{S}$ over the network representing the temporal evolution of each link utilization.

terabytes of traffic traces per day [173], and this keeps growing. However, network operators continuously need to store and analyze network traffic data for various network management purposes, including network planning, traffic engineering, traffic classification, anomaly detection or network forensics. With those huge amounts of generated data, the efficient storage of all this information is then becoming a crucial aspect [237].

Given that most –if not all– of these network management tasks admit a reasonable loss tolerance, we argue that flexible *lossy* compression methods can be taken into consideration for the storage of traffic traces. Moreover, Network Traffic provides with complex relational data naturally represented in the graph domain. These two facts motivated us to explore the design of *lossy* compression methods based on Machine Learning (ML) –and specifically on GNNs– for the Network Traffic Compression task. To the best of our knowledge, there already exist ML and GNN-based models in the literature that target lossy compression tasks [98, 250], but they are mainly entangled to Information Theory concepts used in classical compression algorithms, and applied to Computer Vision domains.

GNNs have demonstrated remarkable performance in modelling and processing relational data by naturally encompassing its binary interactions [263]. However, the traffic patterns we aim to compress may go beyond the provided graph-based network structure (e.g. with distant elements possibly having strong correlations, such as generator and sink nodes), and current GNN models suffer from over-smoothing and over-squashing issues that might prevent them from exploiting these long-range interactions [75]. In this regard, TDL methods, as discussed in Chapter 2, take GNNs a step further by working on domains that can feature higher-order relations, beyond pairwise connections.

With the objective of understanding the benefits that TNNs can bring to the field of network traffic representation, we propose a novel TDL-based lossy traffic compression framework to *(a)* first detect higher-order correlated structures over a given traffic data, and *(b)* then apply TNNs to obtain compressed representations within those multi-element sets. In particular, this work aims to assess if the proposed framework naturally exploits multi-datapoint interactions –between possibly distant elements– in a way that makes it more suitable for compression tasks than GNN-

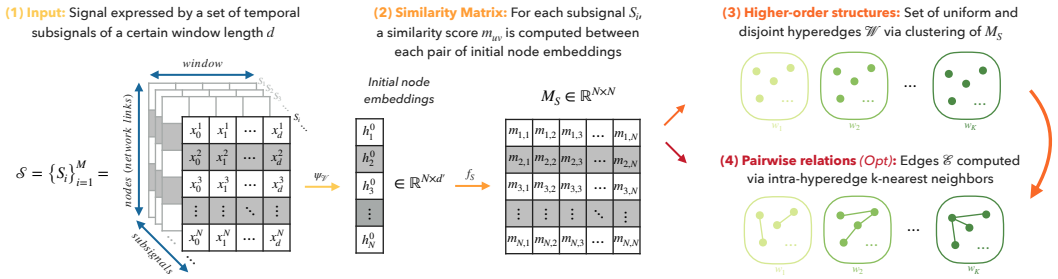


Figure 5.2. Topology Inference Module. For each subsignal $S_i \in \mathcal{S}$, it outputs a topological object $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ determined by K disjoint hyperedges.

based architectures (even if data comes from the graph domain). To do so, we target the problem of compressing the temporal per-link traffic evolution (Figure 5.1) of networks with multiple vantage points, and consider two real-world datasets –Abilene and Geant backbone networks– extracted from [154]. Once the original link-based signal is divided into processable temporal windows, we benchmark our method against a curated set of GNN-based architectures –and a Multi-Layer Perceptron (MLP)– properly designed for compression as well. Obtained results reveal that our topological framework defines the best baseline for *lossy neural compression*.

5.2 Methodology

This section describes the proposed Topological Network Traffic Compression framework, which is divided into the following three primary modules:

5.2.1 Topology Inference

Notably, our proposed methodology does not necessarily rely on the *physical network topology*; instead, it first infers from scratch a new *computational topology* –containing higher-order relationships– by leveraging the data that is aimed to be compressed. In particular, the framework is conceived to compress signals $\mathcal{S}$ over graphs; in many networking scenarios, such as sensor networks or network traffic monitoring, those signals represent temporal measurements of a specific magnitude (e.g. temperature, link traffic) in nodes/edges of the graph. In practice, the graph signal $\mathcal{S}$ can be divided into temporal windows of a certain length d to tackle this inherent continuum time component; this results in having a set of M subsignals $\mathcal{S} = \{S_i\}_{i=1}^{M}$, each of them corresponding to a set of d-dimensional measurements over the graph, i.e. $S_i = \{x_j\}_{j=1}^{N}$, $x_j \in \mathbb{R}^d$.[1] The following described pipeline (see Figure 5.2) is independently applied to every signal S_i.

Similarity Matrix: The initial signal $\{x_j\}_{j=1}^{N}$ is encoded with a MLP into an embedding space

$$h_j^0 = \psi_{\theta_\mathcal{V}}(x_j) \in \mathbb{R}^{d'}, \ \forall j \in \{1, \dots, N\}.$$

[1] As abuse of notation, we will avoid writing the superscript i when referring to the measurements $\{x_j\}_{j=1}^{N}$ of a generic signal $S_i \in \mathcal{S}$.

Next, we compute the pairwise similarity matrix $M_S = (m_{uv}) \in \mathbb{R}^{d' \times d'}$, where $m_{uv} := f_S\left(h_u^0, h_v^0\right)$ and $f_S : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}$ is a similarity function.

Higher-order Relationships: We use clustering techniques on the similarity matrix M_S to deduce K higher-order structures, over which a Topological Message Passing pipeline –see Section 5.2.2– performs the compression. In fact, the idea is to compress the signal within the inferred multi-element sets and encode compressed representations of the data into the final hidden states of these hyperedges. Therefore, the number of higher-order structures K is desired to be considerably lower than the number N of datapoints ($K \ll N$); we design the following *clustering* scheme for this purpose:

1. The number of hyperedges are defined as $K = \lfloor N/p \rceil$, where p is a hyperparameter that identifies the maximum hyperedge length.

2. For every row in the similarity matrix M_S, we get the top $p-1$ highest column entries and calculate their sum. We then select the row that corresponds to the highest summation value. This chosen row becomes the basis for forming a hyperedge as we gather the indices of the $p-1$ selected columns along with the index of the row itself. Then the gathered indices are removed from the rows and columns of the similarity matrix M_S, obtaining a reduced $\hat{M}_S \in \mathbb{R}^{(d'-p)\times(d'-p)}$.

3. Previous step 2 is repeated with subsequent $\hat{M}_S$ until K disjoint hyperedges are obtained.[2]

On the other hand, the choice of the similarity function becomes a crucial aspect for the compression task. Supported by our early experiments (see Section 5.3), our framework makes use of the **Signal to Noise Ratio (SNR)** distance metric presented in [257], proposed in the context of deep metric learning as it jointly preserves the semantic similarity and the correlations in learned features [257].

Pairwise relationships: Besides higher-order structures, our framework can optionally leverage graph-based relational interactions, either *(i)* by considering the original graph connectivity if it is known, or *(ii)* by inferring the edges via the similarity matrix as well –by connecting each element with a subset of top k row-based entries in M_S. In our experiments only intra-hyperedge edges have been considered to keep the inferred higher-order structures completely disjoint from each other.

5.2.2 Compression Module

We implemented two topological Message Passing (MP) compression pipelines, named **SetMP** and **CombMP**. **SetMP** is a purely set-based architecture that operates only over hyperedges and nodes; **CombMP**, our most general architecture, leverages the three different structures (nodes, edges, hyperedges) in a hierarchical

[2] When N/p is not an even division, at some point the ranking starts considering the row-wise $p-2$ higher entries to form $p-1$-length hyperedges, so that a total of $K = \lfloor N/p \rceil$ hyperedges of lengths p and $p-1$ are obtained.

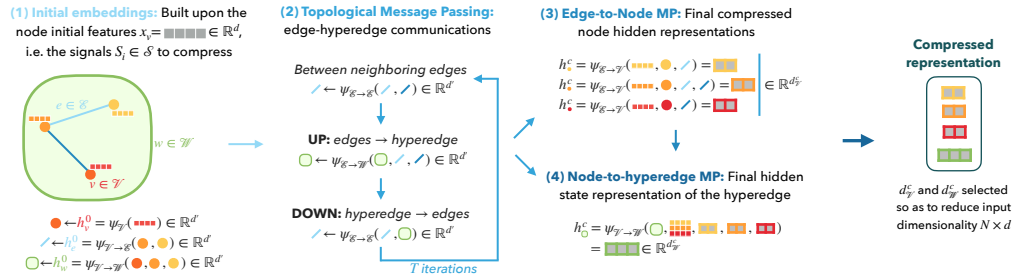


Figure 5.3. Compression Module workflow for the **CombMP** architecture. It is independently applied to each hyperedge $w \in \mathcal{W}$ of the inferred topological object $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$.

way,[3] and can be seen as a generalisation of **SetMP**. We describe both solutions as follows:

Remark: We follow the standard notation of Message Passing Neural Networks (MPNN): $\psi_{\theta_\cdot}$ and $\phi_{\theta_\cdot}$ are learnable functions, and $\oplus$ represent permutation-invariant aggregator functions.

CombMP Architecture For a given signal S_i and its corresponding initial embeddings $\{h_j^0\}_{j=1}^N$, **CombMP** operates over a topological object $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ where $\mathcal{V}$ denotes the set of elements or nodes, $|\mathcal{V}| = N$; $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ represent the set of edges; and $\mathcal{W} \in (\mathcal{V} \times \cdots \times \mathcal{V})$ the set of hyperedges. The compression pipeline (visualized in Figure 5.3) can be described as follows:

Initial embeddings: First, we generate initial embeddings for the three considered topological structures. For nodes, we use the previously computed embeddings $\{h_v^0\}_{v=1}^N$. For edges and hyperedges, (learnable) permutation invariant functions are applied over the initial embeddings of the nodes they contain; respectively, $h_e^0 = \phi_{\theta_\mathcal{E}}\left(\oplus_{v \in e} h_v^0\right)$ for each $e \in \mathcal{E}$, and $h_w^0 = \phi_{\theta_\mathcal{W}}\left(\oplus_{v \in w} h_v^0\right)$ for each $w \in \mathcal{W}$. The same dimension d' is used for all initial and intermediate hidden representations.

Edge-Hyperedge Message Passing: We define a hierarchical propagation of messages between edges and hyperedges. First, neighboring edges communicate to each other to update their representations; denoting the edge neighbors of an edge $e \in \mathcal{E}$ by

$$\mathcal{N}_e^\mathcal{E} := \{e' = (u, v) \in \mathcal{E} | e' \neq e, u \in e \vee v \in e\},$$

its new hidden state becomes

$$h_e^1 = \phi_{\theta_{\mathcal{E}\to\mathcal{E}}}\left(\oplus_{e' \in \mathcal{N}_e^\mathcal{E}} \psi_{\theta_{\mathcal{E}\to\mathcal{E}}}\left(h_e^0, h_{e'}^0\right)\right).$$

Next, hyperedges also update their hidden states based on the updated edge representations according to

$$h_w^1 = \phi_{\theta_{\mathcal{E}\to\mathcal{W}}}\left(\oplus_{e \in \mathcal{E}, e \subset w} \psi_{\theta_{\mathcal{E}\to\mathcal{W}}}\left(h_w^0, h_e^1\right)\right),$$

[3] Edges and hyperedges are distinguished because, analogously to recent Combinatorial Complexes (CCC) models[92], edges can hierarchically communicate with hyperedges if they are contained in them; in fact, the name **CombMP** relates to these general topological constructions.

for each $w \in \mathcal{W}$. Then the idea is to propagate downwards towards the edges, i.e. from hyperedges to edges,

$$h_e^2 = \phi_{\theta_{\mathcal{W}\to\mathcal{E}}}\left(\oplus_{w\in\mathcal{W},e\subset w}\psi_{\theta_{\mathcal{W}\to\mathcal{E}}}\left(h_e^1, h_w^1\right)\right);$$

and only between edges again. We note that this whole communication process can be iterated T times.

Edge-to-Node Compression: At this point, we perform a first compression step over the nodes by leveraging the updated edge hidden representations, the initial node embeddings, as well as the original node data as a residual connection. Formally, for each node $v \in \mathcal{V}$ we get a compressed hidden representation

$$h_v^c = \phi_{\theta_{\mathcal{E}\to\mathcal{V}}}\left(\oplus_{e\in\mathcal{E},v\in e}\psi_{\theta_{\mathcal{E}\to\mathcal{V}}}\left(x_v, h_v^0, h_e^t\right)\right) \in \mathbb{R}^{d_{\mathcal{V}}^c},$$

forcing it to have a much lower dimension than the original signal, i.e. $d_{\mathcal{V}}^c \ll d$.

Node-to-Hyperedge Compression: Finally, a second and last compression step is performed over the hypergraph representations, in this case leveraging a residual connection to the original measurements, the previously computed compressed representations of nodes, as well as the updated hidden representations of hyperedges. More in detail, each hyperedge $w \in \mathcal{W}$ obtains its final compressed hidden representation as

$$h_w^c = \phi_{\theta_{\mathcal{V}\to\mathcal{W}}}\left(\oplus_{v\in\mathcal{V},v\in w}\psi_{\theta_{\mathcal{V}\to\mathcal{W}}}\left(x_v, h_v^c, h_w^t\right)\right) \in \mathbb{R}^{d_{\mathcal{W}}^c}.$$

In this case, since the number K of hyperedges is set to be much lower than the number of nodes, the dimension $d_{\mathcal{W}}^c$ of these final hyperedge representations is not as critical as that of the nodes.

SetMP Architecture: In contrast to **CombMP**, this architecture disregards binary connections and consequently operates over a topological object $\mathcal{T} = (\mathcal{V}, \mathcal{W})$, where again $\mathcal{V}$ denotes the set of N nodes and $\mathcal{W} \in (\mathcal{V} \times \cdots \times \mathcal{V})$ the set of K inferred hyperedges. Initial embeddings for nodes and hyperedges are generated in the same way. Then, without edges as intermediaries, we directly perform the node compression –in this case based on both the initial node and hyperedge embeddings and the original signal S_i. Formally, for each node $v \in \mathcal{V}$ we get a compressed hidden representation

$$h_v^c = \phi_{\theta_{\mathcal{W}\to\mathcal{V}}}\left(\oplus_{w\in\mathcal{W},v\in e}\psi_{\theta_{\mathcal{W}\to\mathcal{V}}}\left(x_v, h_v^0, h_w^0\right)\right) \in \mathbb{R}^{d_{\mathcal{V}}^c}.$$

The second and last compression step over the hyperedge representations is exactly the same as in the **CombMP** –except for the fact that now the considered hyperedge hidden states have not been updated. Therefore, each hyperedge $w \in \mathcal{W}$ obtains its final compressed hidden representation as

$$h_w^c = \phi_{\theta_{\mathcal{V}\to\mathcal{W}}}\left(\oplus_{v\in\mathcal{V},v\in w}\psi_{\theta_{\mathcal{V}\to\mathcal{W}}}\left(x_v, h_v^c, h_w^0\right)\right) \in \mathbb{R}^{d_{\mathcal{W}}^c}.$$

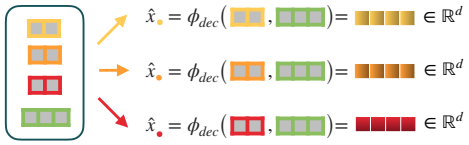


Figure 5.4. Decompression Module. It is applied over each hyperedge-dependent compressed representation set generated by the Compression Module.

Table 5.1. Reconstruction Mean Squared Error (MSE) and Mean Absolute Error (MAE) over the test set of the considered datasets for two different compression factors. **Top:** Considered ML-based baselines. **Middle:** Our proposed TDL-based architectures. **Bottom:** State-of-the-art *zfp* [62] method.

	Abilene				Geant			
	$r_c = 1/3$		$r_c = 2/3$		$r_c = 1/3$		$r_c = 2/3$	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
GNN	$1.95 \cdot 10^{-2}$	$1.08 \cdot 10^{-1}$	$1.95 \cdot 10^{-2}$	$1.08 \cdot 10^{-1}$	$2.33 \cdot 10^{-2}$	$1.21 \cdot 10^{-1}$	$2.32 \cdot 10^{-2}$	$1.20 \cdot 10^{-1}$
MPNN	$7.88 \cdot 10^{-4}$	$1.24 \cdot 10^{-2}$	$7.92 \cdot 10^{-4}$	$1.24 \cdot 10^{-2}$	$8.45 \cdot 10^{-3}$	$4.13 \cdot 10^{-2}$	$1.82 \cdot 10^{-3}$	$2.39 \cdot 10^{-2}$
MLP	$1.04 \cdot 10^{-3}$	$1.88 \cdot 10^{-2}$	$9.71 \cdot 10^{-4}$	$1.80 \cdot 10^{-2}$	$3.76 \cdot 10^{-3}$	$3.96 \cdot 10^{-2}$	$3.62 \cdot 10^{-3}$	$3.89 \cdot 10^{-2}$
SetMP	$\mathbf{3.22 \cdot 10^{-4}}$	$\mathbf{8.75 \cdot 10^{-3}}$	$\mathbf{2.03 \cdot 10^{-4}}$	$\mathbf{6.80 \cdot 10^{-3}}$	$\mathbf{6.93 \cdot 10^{-4}}$	$\mathbf{1.52 \cdot 10^{-2}}$	$\mathbf{2.90 \cdot 10^{-4}}$	$\mathbf{1.05 \cdot 10^{-2}}$
CombMP	$5.81 \cdot 10^{-4}$	$1.12 \cdot 10^{-2}$	$3.76 \cdot 10^{-4}$	$1.06 \cdot 10^{-2}$	$1.07 \cdot 10^{-3}$	$1.88 \cdot 10^{-2}$	$7.04 \cdot 10^{-4}$	$1.61 \cdot 10^{-2}$
SetMP (Gen.)	–	–	–	–	$8.50 \cdot 10^{-4}$	$1.63 \cdot 10^{-2}$	$5.26 \cdot 10^{-4}$	$1.26 \cdot 10^{-2}$
zfp	$9.19 \cdot 10^{-5}$	$7.34 \cdot 10^{-3}$	$4.02 \cdot 10^{-7}$	$4.84 \cdot 10^{-4}$	$1.04 \cdot 10^{-4}$	$7.83 \cdot 10^{-3}$	$4.18 \cdot 10^{-7}$	$4.95 \cdot 10^{-4}$

Compressed Representations: In both **CombMP** and **SetMP** architectures the final node and hyperedge states,

$$\{\{h_v^c\}_{v\in\mathcal{V}}, \{h_w^c\}_{w\in\mathcal{W}}\}\,,$$

encode the compressed representation of a signal $S_i = \{x_j\}_{j=1}^{N}$. Consequently, the number K of higher-order structures together with the dimensions of those compressed hidden states define the compression factor r_c:

$$r_c = \frac{N \cdot d_{\mathcal{V}}^c + K \cdot d_{\mathcal{W}}^c}{N \cdot d} \tag{5.1}$$

5.2.3 Decompression

This last module learns to reconstruct the original signal of every node through its compressed representation and the final hidden state of the hyperedge it belongs to. More formally, for each $v \in \mathcal{V}$ and its corresponding hyperedge $v \in w \in \mathcal{W}$, the reconstructed signal $\hat{x}_v$ is obtained as

$$\hat{x}_v = \phi_{dec}\left(h_v^c, h_w^c\right),$$

where ϕ_{dec} is implemented as a MLP in our framework. The whole model is trained end-to-end to minimize the (mean squared) reconstruction error.

5.3 Evaluation

The main goal of this section is to analyze the potential of current ML techniques –and particularly TNNs– to perform network traffic compression.

5.3.1 Experimental Setup

For the evaluation, we use two public real-world datasets –Abilene, Geant– from [154]. They are pre-processed to generate subsignals S_i of network link-level traffic measurements in temporal windows of length $d = 10$, to which then a random 60/20/20 split is performed for training, validation and test, respectively. In this context, our method is compared against three ML-based baselines.

- **GNN**: we implemented several standard GNN architectures (GCN [123], GAT [214], GATv2 [39], GraphSAGE [95]) to perform signal compression leveraging the network physical topology; the **GNN** baseline accounts for the best result among them in each evaluation scenario.
- **MPNN**: a custom graph-based MPNN scheme –over the original physical network topology as well– whose modules and pipeline are similar to our topological methodologies.
- **MLP**: a feed-forward auto-encoder architecture applied to flattened subsignals S_i.

GNN and **MPNN** baselines implement a decompression module similar to that of our TDL-based methods.

Analogously to what we do with our proposed architectures, we have fine-tuned the involved hyperparameters of all implemented baselines to perform the compression task. All models –including ours– are trained for a maximum of 200 epochs in Abilene dataset and 500 in Geant using Adam optimizer (with learning rate 0.003 and epsilon 0.001) and using batches of 25 samples if required. The model state corresponding to the best performing iteration over the validation set is selected for the test evaluation, whose results are represented in Table 5.1.

5.3.2 Experimental Results

Table 5.1 shows the reconstruction error –Mean Squared Error (MSE) and Mean Absolute Error (MAE)– obatained by our framework and the baselines in both datasets for two different compression ratios, 1/3 and 2/3. We can see that **SetMP** performs better than our most generic **CombMP** architecture, and observe that both TDL-based approaches consistently outperform GNN-based models and **MLP** in all scenarios. Among the ML baselines, the customised **MPNN** is the best performing solution, followed by the **MLP** and with **GNN** in the last place.

Table 5.1 also shows the performance of the state-of-the-art lossy compression method *zfp* [62] for the desired precision. Despite *zfp* gets better reconstruction errors than our model in all scenarios, we note that differences are considerably lower for smaller (i.e. more challenging) compression factors. Overall, we reckon that these are promising results; our TDL-based models are postulated as strong ML-based baselines for network traffic compression, and by addressing some of their current limitations (see next Section 5.5) there can be room for further improvements.

Finally, we also performed an exploratory evaluation of the generalization capabilities of our proposed topological inspired architectures. To do so, we took the best performing **SetMP** models trained on Abilene and evaluated them in the Geant

network; results are shown in Table 5.1, labelled by "SetMP (Gen.)". Despite the slight decrease of performance w.r.t. the **SetMP** models trained on Geant, these models obtain better reconstruction errors than all other ML architectures, including **CombMP**.

5.4 Related Work

The general pipeline of our introduced TDL-based models is aligned with *zfp* [62], the state-of-the-art lossy method for floating-point data compression. As detailed in [62], the first step of *zfp* consists in dividing floating matrices or tensors in disjoint blocks of a fixed dimension, which then are independently processed to extract compressed representations. This has similarities with our search of higher-order structures, with the difference that *zfp*'s divisions are totally determined by the input elements' order.

Additionally, the proposed topology inference module has some resemblances to that of [236], which is also based on grouping entries of an affinity matrix computed over a set of element's neural embeddings. Nevertheless, due to the constraints imposed by the compression task, there exist relevant differences between their inference procedure and ours: whereas we design a clustering methodology to get disjoint and uniform-length sets, [236] implements a multi-scale hyperedge forming pipeline where each node ends up belonging to an arbitrary number of higher-order structures.

Regarding our topological-inspired architectures, we highlight the paper [92] on Combinatorial Complexes (CCC). Our most general method –**CombMP**, which in fact stands for *Combinatorial Message Passing*– was conceptualized before the publication of these works, but we note that it can be formalized in terms of CCCs' notation by considering nodes as 0-cells, edges as 1-cells, and hyperedges as 2-cells. **SetMP**, on the other hand, belongs to the hypergraph family according to the classification of [159].

Finally, a special mention should be given to [4], which also leverages a ML model to specifically perform network traffic compression. However, authors of this work implement a spatio-temporal GNN that acts as a *predictor* of a traditional lossless compression method (Arithmetic Coding), whisch defines a totally different conceptual approach.

5.5 Discussion

Obtained results support our hypothesis that taking into account higher-order interactions could help in designing more expressive ML-based models for network traffic compression tasks, specially due to the fact that these higher-order structures can go beyond the graph-based local neighborhood and connect possibly distant datapoints whose signals may be strongly correlated (e.g. generator and sink nodes). In that regard, TDL can provide us with novel methodologies that naturally encompass and exploits those multi-element relations. Moreover, our preliminary analysis on generalization suggests that our trained framework could potentially compress traffic on networks not previously seen in training.

However, there are several aspects and limitations of our proposed methodology that are being investigated and will be addressed in future work. The following list summarizes some of the main lines of research:

Topology Inference Apart from exploring other metrics beyond SNR, it would be interesting to consider more flexible clustering approaches that could dynamically adapt the length/number of the inferred higher-order structures.

Implementation Issues Our current proposal does not scale well with the original (graph) signal dimension, as it mainly relies on an iterative pair-wise similarity computation between all (graph) elements. More research about how to relax this aspect is required in order to make its deployment feasible , and in this regard some ideas we want to explore are:

- To test whether static higher-order structures generated from training samples can perform well in testing time; not only this would reduce the execution time, but also the necessity of storing the cluster sequence at each iteration.
- To check the feasibility of storing simultaneously sparse matrices indicating when the compression loss exceeds a certain threshold; apart from becoming a potential indicator of anomalies in the signal, if the distribution of big reconstruction errors is really sparse, combining them with the compressed representations will provide with loss tolerance guarantees, and could be key for matching *zfp* performance.
- For large graphs, to divide the original graph into independent subgraphs, to which then our method is applied.

Topological MP According to current results, it seems that the lower complexity of **SetMP** is a clear advantage w.r.t. **CombMP**, and suggests that intermediate edge communications noisily interfere in the compression process. This should be further validated with other datasets, and possibly by testing some variations of the edge-based communication pipeline in the general **CombMP** architecture.

Chapter 6

Topology-Driven Token Compression for Goal-Oriented Communications

Contents

This chapter is organized as follows. In Section 6.1, we introduce deep Joint Source-Channel Coding (JSCC) and provide the motivation for the transformer-based deep JSCC enhanced with topology inference. Section 6.2 outlines the deep Joint Source-Channel Coding (JSCC) framework, followed by Section 6.3, which describes the proposed topology-driven token compression framework. In Section 6.4, we present the evaluation, and finally, Section 6.5 presents a discussion on the proposed approach and potential directions for future research.

6.1 Introduction

Recent advances in semantic and goal-oriented communication have greatly improved how distributed network nodes coordinate to accomplish tasks more efficiently [194, 195]. This new approach emphasizes the semantic understanding of data, allowing communication to be driven by task-specific objectives. Among the various approaches, Joint Source-Channel Coding (JSCC) has emerged as a promising technique, especially when enhanced by neural networks—an approach

Currently under double-blind review at an A1 conference.

commonly referred to as *deep* JSCC [239]. These deep JSCC schemes capitalize on the powerful approximation capabilities of modern neural networks, including Convolutional Neural Networks (CNNs) [37] and transformers [235], to jointly optimize data coding and wireless transmission by incorporating the communication channel as a non-trainable layer within the system. This approach allows for an end-to-end learning framework for task-specific optimization and has demonstrated performance advantages over traditional coding schemes, particularly in semantic communication, where it mitigates the steep performance degradation observed when the Signal-to-Noise Ratio (SNR) drops below information-theoretic limits [37].

A deep JSCC system consists of three primary components: an encoder, a communication channel, and a decoder. Both the encoder and decoder are modeled as neural networks, while the communication channel introduces non-trainable physical constraints. The main function of the encoder is to compress input data into a form that can be efficiently transmitted through the channel, while the decoder's role is to reconstruct the transmitted data and carry out the designated downstream task. The flexibility of deep JSCC lies in its ability to train *task-specific* encoders and decoders, enabling the transmission of only task-relevant information, thus achieving greater compression and efficiency in communication.

Transformer-based architectures have found applications across various fields, including natural language processing and computer vision [108]. However, each transformer block necessitates the computation of the self-attention mechanism, which exhibits quadratic complexity with respect to the size of the input. This computational demand poses significant challenges for deploying transformer-based models in resource-constrained environments. Consequently, transformer optimization methods that initially emerged outside of the communication domain, such as quantization [55, 230, 60], distillation [100, 3], sparsification [132, 101], and lately dynamic networks [226, 146, 234], have been successfully adopted in the deep JSCC scenario to enforce latency or communication bottleneck constraints for real-time communication scenarios [239, 248, 219, 61]. Several authors have proposed trainable techniques for designing adaptive deep JSCC schemes, such as partitioning the latent space of the model [239], masking non-relevant channels in a CNN [248], or projecting transformer outputs—which consist of a set of tokens that represent encoded image patches—into variable-length vectors [57]. In [219], the authors proposed an adaptive source allocation paradigm based on quantization, which improves transmission reliability, while in [264], a Universal Transformer is trained to achieve flexible communication, enabling better performance under various channel conditions. More recently, [61] proposed a token-dropping method to achieve adaptive compute allocation.

This chapter proposes a principled approach for transformer-based deep JSCC algorithms for image processing, designed to address both communication bottleneck and computational latency constraints within a unified framework. Our method is motivated by the observation that not all tokens carry equal importance for a given task. For example, in object classification, patches representing the object itself are essential, while background patches offer minimal contribution to the task. By identifying the most relevant tokens, we optimize computational efficiency and streamline the transmission process, focusing resources on transmitting the crucial information needed for accurate task execution. This *selective transmission* poses a

key challenge: determining which tokens should be transmitted based on the task and the data. In this work, we tackle the challenge by leveraging the transformer's capability to operate over a set structure, selectively merging patches that contain similar information. These merging operations form a topology over the tokens, denoted as $\mathcal{T}$, represented by a Directed Acyclic Graph (DAG). This approach reduces the number of tokens that need to be transmitted and enables the decoder to be conditioned on the new topology $\mathcal{T}$ implicitly or explicitly. We experimentally demonstrate the advantages of token-level information compression in scenarios when one or multiple downstream tasks are required on the receiver side. Specifically, our study focuses on classification and regression tasks. For regression, we explicitly condition the decoder on the topology $\mathcal{T}$ and use this information to progressively unmerge the tokens, following the DAG from the bottom up. Additionally, we show that the proposed approach is compatible with the integration of other compression techniques, such as variable-length token projection [57], to further optimize the transmission process. To the best of our knowledge, we are the first to tackle both communication and computational bottlenecks in JSCC by modifying the data topology.

6.2 Deep JSCC Framework

We consider a deep JSCC framework where the entire communication pipeline is described by a neural network. The pipeline consists of three main components: the encoder, the communication channel, and the decoder. We first describe the communication pipeline and then the specific parts of it.

The input data are images, denoted as $x \in \mathbb{R}^{C\times H\times W}$, where C, H, and W represent the number of channels, the height, and the width of the image, respectively. The encoder, denoted as f_e, is placed at the transmitter side and processes the image x into a sequence of tokens $t_e \in \mathbb{R}^{T\times D}$, where T is the number of tokens, and D is the hidden dimensionality. The token sequence t_e is then transmitted through the channel, denoted as $\mathcal{C}$. On the receiver side, the decoder, denoted as f_d, processes the received tokens t_e to perform the downstream task, such as classification. The overall process can be summarized as:

$$y = f_d(\mathcal{C}(f_e(x))) = f_d \circ \mathcal{C} \circ f_e(x)$$

where y is the output of the downstream task.

In this work, we utilize the Vision Transformer architecture (ViT [65]), which has recently emerged as a powerful model in computer vision, often outperforming traditional CNNs. A ViT takes an input image $x \in \mathbb{R}^{C\times H\times W}$ and processes it through a sequence of transformer layers $\mathcal{B}^l$, where $l = 1, \ldots, L$. A ViT model $f(x)$ is defined as $f(x) = \mathcal{P} \circ \mathcal{B}^L \circ \mathcal{B}^{L-1} \circ \cdots \circ \mathcal{B}^1 \circ \mathcal{F}(x)$, where $t^0 = \mathcal{F}(x)$ is a pre-processing layer that splits the image into equally sized patches and projects them into the initial sequence $t^0 \in \mathbb{R}^{T\times D}$ where T is the number of tokens and D is the hidden dimensionality, while $\mathcal{P}$ is a post-processing layer. Each transformer block $\mathcal{B}$ with multi-head attention transforms the input via self-attention and MLP layers so that $t^l = \mathcal{B}^l(t^{l-1})$, where $t^l \in \mathbb{R}^{T\times D}$ denotes the updated tokens. The internal computational flow keeps the dimensions of the tokens D and the sequence

length T fixed. Each transformer block $\mathcal{B}$ is defined as:

$$\mathcal{B}^l(t^{l-1}) = \text{MLP}(\text{LayerNorm}(t^{l-1} + \text{MHSA}(t^{l-1})))$$
$$\text{MHSA}(t^{l-1}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$
$$\text{head}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$
$$\text{MLP}(x) = \sigma(xW_1 + b_1)W_2 + b_2$$

where MHSA stands for the multi-head self-attention, head_i is an attention head, $Q_i = t^{l-1}W_Q^i$, $K_i = t^{l-1}W_K^i$, $V_i = t^{l-1}W_V^i$, and W_Q^i, W_K^i, W_V^i are the projection matrices, and the final output is concatenated and multiplied by the output projection matrix W_O, and the where W_1, W_2, b_1, b_2 are learnable parameters, and σ is the activation function.

6.2.1 Encoder-decoder architecture

We employ a transformer-based architecture for both the encoder and decoder. The encoder $f_e(\cdot)$ is composed of L_e layers and used to extract meaningful features from the input, which are then used by the decoder $f_d(\cdot)$, composed of $L_d = L - L_e$ layers, to perform a downstream task. So that:

$$f_e(x) = \mathcal{B}^{L_e} \circ \cdots \circ \mathcal{B}^1 \circ \mathcal{F}(x),\ f_d(x) = \mathcal{P} \circ \mathcal{B}^{L_d} \circ \cdots \circ \mathcal{B}^{L_{e+1}}(x).$$

The encoder extracts features from an input sample, and the decoder receives and uses them for the downstream task. More precisely, given a ViT, we select a splitting point L_e, and divide the network into encoder and decoder.

6.2.2 Communication channel

For the channel $\mathcal{C}$, we adopt a conventional configuration as described in [239], implementing it as a layer that introduces noise to the symbols with a given power level. Specifically, noise is added based on a predetermined SNR, where the received signal $\mathbf{h}'$ at the decoder can be represented as $\mathbf{h}' = \mathbf{h} + \mathbf{n}$, where $\mathbf{n}$ is drawn from a zero-mean Gaussian distribution with variance σ^2, such that the SNR is given by:

$$\text{SNR}_{\text{db}}(x) = 10 \log_{10}\left(\frac{\|\mathbf{h}\|^2}{\|\mathbf{n}\|^2}\right)$$

6.3 Proposed Method

Topological Encoder: We propose an encoder that is capable of compressing the number of tokens required to be sent over the communication channel. A generic transformer block operates on *sets* of tokens, allowing to adjust the number of tokens within each block without modifying the model architecture. We leverage this observation to design an encoder capable of outputting a desired number of tokens for a given image while tracking which tokens have been merged to construct the topology $\mathcal{T}$ in the form of a sequence of DAGs. This method reduces the number

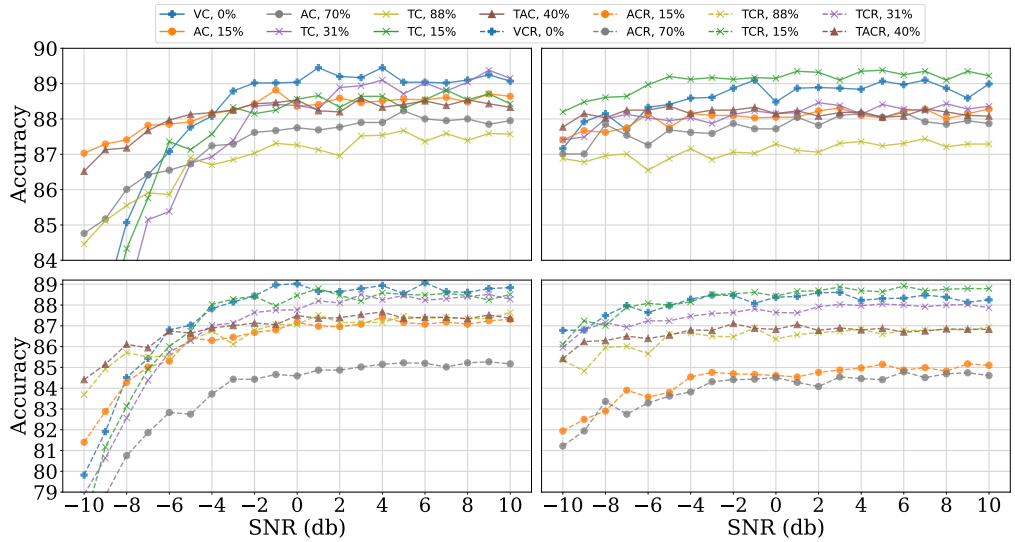


Figure 6.1. Performance comparison of models. The left column shows performance without noise during training and the right column with noise. The top row represents single downstream task and the bottom row double downstream task scenarios.

of tokens in a transformer network by merging a fixed number r of tokens at each layer.

For each block, a desired number of tokens r is merged using a bipartite graph matching algorithm. The keys $K \in \mathbb{R}^{T^l \times D}$ summarize each token's information; thus, following [35], we employ cosine similarity between the keys, $K = \frac{1}{h}\sum_{i=1}^{h} K_i$, where the mean is taken over the keys K_i of each attention head i. The token merging procedure for the l-th transformer layer proceeds as follows: (i) The tokens are partitioned into two sets, $\mathbb{A}$ and $\mathbb{B}$, such that $\mathbb{A} \cup \mathbb{B} = \mathbb{T}^l$ and $|\mathbb{A}| \approx |\mathbb{B}|$, where $|\mathbb{T}^l|$ is the set of all tokens at layer l. (ii) An edge is created from each token $a \in \mathbb{A}$ to its most similar token $b \in \mathbb{B}$ based on a similarity metric, such as cosine similarity: $\text{sim}(a, b) = \frac{a \cdot b}{\|a\|\|b\|}$(iii) The r most similar edges are retained, with the top-r edges selected based on their similarity scores $\text{sim}(a, b)$. (iv) Tokens connected via the selected edges are merged by averaging their features, following the direction of the edges. The resulting sets $\mathbb{A}'$ and $\mathbb{B}'$ have dimensionalities $|\mathbb{A}| - r$ and $|\mathbb{B}|$, respectively, with at most r tokens updated in $\mathbb{B}'$. (v) Finally, the two sets $\mathbb{A}'$ and $\mathbb{B}'$ are concatenated, producing the new set of tokens $\mathbb{T}^{l+1}$, where $T^{l+1} = |\mathbb{T}^{l+1}|$. The incidence matrix $H^l \in \{0, 1\}^{T^{l+1}|\times|T^l}$, where an entry of 1 indicates that tokens have been merged and 0 otherwise, represents the topology at layer l. The sequence of incidence matrices $\mathcal{T} = \{H_r^L, \ldots, H_r^1\}$, where L represents the number of transformer blocks enhanced with the token merging procedure, and r denotes the number of tokens merged, forms the final topology. This procedure achieves computational efficiency nearly comparable to random token dropping, significantly accelerating computations while having minimal impact on downstream tasks. The transformer block enhanced with token merging is referred to as *tome* block $\mathcal{B}_r$. By selecting r, a tome block output is defined as $t^l = \mathcal{B}_r^l(t^{l-1})$ where r is a parameter of bipartite graph matching applied executed within the tome transformer block $\mathcal{B}_r$. The input sequence length is $t^{l-1} \in R^{T^{l-1} \times D}$ and $t^l \in R^{T^l \times D}$, where $T^l = T^{l-1} - r$. The

encoder, denoted as $f_e^{\mathcal{B}_r}$, consists of a sequence of $\mathcal{B}_r^l$ blocks and is defined as $f_e^{\mathcal{B}_r}(x) = \mathcal{B}_r^{L_e} \circ \cdots \circ \mathcal{B}_r^1 \circ \mathcal{F}(x)$. Applying the encoder $f_e^{\mathcal{B}_r}(x)$ results in a total reduction of rL_e tokens after L_e tome blocks $\mathcal{B}_r$. For brevity, the encoder composed of standard transformer blocks $\mathcal{B}$ is referred to as $f_e(x)$.

Decoder: In Section 6.4, we explore scenarios where one or two downstream tasks need to be performed on the receiver side, specifically focusing on the tasks of image classification and reconstruction. In the case of classification, the decoder, denoted as f_d^{cl}, and in the case of topological encoder $f_e^{\mathcal{B}_r}$, is implicitly conditioned on the topology by learning to classify an image from tokens whose topology has been adjusted by the encoder. For transformer architecture, a common approach is to prepend a special trainable token, the class token CLS, to the received tokens sequence t_e, resulting in $t_d^{cl} = [\text{CLS}^0, t_e]$, to enable classification. This sequence is then passed through the L_d transformer blocks, and, after the last block, a post-processing module, usually a linear layer, $\mathcal{P}$ takes as input the CLS^{L_d} token to produce the prediction. Hence the decoder $f_d^{cl}(x)$ takes as input t_d^{cl}. In the case of the regression task and the encoder $f_e^{\mathcal{B}_r}$, the decoder, denoted as $f_{d,\mathcal{T}}^{reg}$ is explicitly conditioned on the topology $\mathcal{T}$. The topology facilitates a progressive deconvolution process to unmerge tokens. In this process, the output transformer block $\mathcal{B}^{l_d}$ is deconvolved in a bottom-up manner using the incidence matrix $H_r^{L_d - l_e}$, defined as $\mathcal{B}_{\mathcal{T}}^l = \mathbf{T}(H_r^{L_d - l_e}) \circ \mathcal{B}^{l_d}$, where $\mathbf{T}$ represents the transposition operation. It is important to note that in this scenario, the number of transformers blocks $\mathcal{B}_{\mathcal{T}}^l$ in the decoder $f_{d,\mathcal{T}}^{reg}$ must satisfy $L_d \geq L_e$ to ensure the complete deconvolution of all tokens and takes as input received token sequence t_e and a sequence of incidence matrices $\mathcal{T} = (H_r^{L_e}, \ldots, H_r^1)$, resulting in $f_{d,\mathcal{T}}^{reg}(t_e, \mathcal{T})$. In the case of encoder f_e, the decoder, denoted as f_d^{reg}, is employed, which takes as input only the received token sequence t_e.

6.4 Evaluation

Setup: we evaluate the proposed technique under various conditions, focusing on model performance with and without noise introduced by the communication channel $\mathcal{C}$. When noise is present during training, we assume that $\mathcal{C}$ adds noise to the transmitted tokens, with values randomly sampled between -10dB and 10dB, consistent with the noise levels tested during inference. This setup allows us to assess the model's robustness to channel-induced noise during transmission. We evaluate two scenarios on the receiver side: one involving a single task, image classification, and another where two tasks—image classification and reconstruction—are performed simultaneously, referred to as the double downstream task. All experiments are conducted on the Imagemette dataset [103]. These evaluations help us understand the technique's compression capabilities, particularly when multiple tasks are executed on the receiver side. For both scenarios, we initialize all models from a pre-trained *Vision Transformer* (ViT) architecture [206], with $L_e = L_d = 6$. In (i), the first setup uses the encoder f_e and decoder f_d, referred to as VC for the single downstream task and VCR for the double downstream task. These models do not perform compression and serve as baselines. (ii) The second baseline setup enhances the encoder f_e with an additional non-linear projection layer to compress the hidden dimensions [57], and

Table 6.1. The image reconstruction error (RMSE) is averaged over various noise, ranging from -10 dB to 10 dB. The results are shown for models trained with and without noise.

Training	VCR	ACR		TCR		TACR
Noise	0%	15%	30%	15%	31%	40%
No	3.68 ± 1.52	3.88 ± 1.15	3.98 ± 1.34	3.75 ± 1.47	3.85 ± 1.34	3.96 ± 1.25
Yes	3.72 ± 0.7	3.85 ± 0.73	4.02 ± 0.82	3.53 ± 0.79	3.64 ± 0.74	3.98 ± 0.73

the decoders f_d^{cl} and f_d^{reg} are similarly enhanced with non-linear projection layers to uncompress the hidden representation before passing it through transformer blocks. These models are referred to as AC and ACR for the single and double downstream tasks, respectively.(iii) The encoder $f_e^{\mathcal{B}_r}$ employed to compress the input. For the single downstream task, the decoder f_d^{cl} is used, and this model is referred to as TC. For the double downstream task, the decoders f_d^{cl} and $f_{d,\mathcal{T}}^{reg}$ are employed, and this model is referred to as TCR. (iv) Lastly, this setup combines both the non-linear projection and token merging approaches. The encoder $f_e^{\mathcal{B}r}$ is further enhanced with a non-linear projection layer to further compress the merged tokens, while the decoders f_d^{cl} and $f_{d,\mathcal{T}}^{reg}$ are enhanced with non-linear projection layers to uncompress the hidden representation before the transformer blocks. These models are referred to as TAC and TACR for the single and double downstream tasks.

Compression evaluation: the compression evaluation is calculated as $\frac{T_\mathcal{C} \cdot D_\mathcal{C}}{T \cdot D}$, where $T_\mathcal{C}$ is the number of tokens passed to the communication channel, and T is the initial number of tokens. Similarly, $D_\mathcal{C}$ and D represent the hidden dimensionalities of the tokens transmitted through the communication channel and the initial tokens, respectively. Given that the incidence matrices consist solely of $\{0, 1\}$ entries, we neglect the compression overhead to simplify the comparison.

Analysis: From Fig. 6.1, it is clear that all models trained without noise perform poorly under high noise from the communication channel $\mathcal{C}$, whereas models trained with noise demonstrate significantly improved performance. The performances of the models with a single downstream task are more tightly clustered, with accuracy ranging from 84% to 89.5%. With two downstream tasks, the performance gap widens, with accuracy ranging from 79% to 89%. Notable that in three out of four cases, the TC at compression level 15% model outperforms VC, indicating that the proposed techniques can facilitate deep JSCC. Clearly, in the scenarios when the model is trained with noise, the proposed topology-enhanced models (TC, TCR, TAC, and TACR) perform better compared to AC and ACR models. At the same time, we can see that TAC and TACR perform worse than TC and TCR in most of the scenarios. Additionally, it is evident that the compression of hidden dimensions saturates when the models are trained with noise, while topology models do not exhibit such a phenomenon. Additionally, Table 6.1 shows that TCR and TACR models outperform the baseline ACR models. Notably, TCR 15% outperforms the baseline VCR 0%. Fig. 6.2 demonstrates the top 5 largest merged tokens across four distinct images. The tokens can extend beyond the immediate local neighborhood, capturing non-contiguous areas of the image; this allows the model to represent complex structures.

Figure 6.2. Top 5 merged tokens produced by the TCR 88% model, with distinct colors representing individual token boundaries.

6.5 Discussion

In this chapter, we propose utilizing data topology to address both communication and computational bottlenecks in JSCC. We evaluate the proposed technique under various conditions, demonstrating its effectiveness compared to compression methods based on internal dimensionality reduction for both classification and regression tasks. For future work, we plan to enhance this approach by incorporating a selective choice of the r parameter conditioned on the input. Additionally, we aim to extend this work by exploring a wider range of downstream tasks and applying it to other data domains.

Part III

Higher-Order Networks: Formalization, Standardization, and Architectural Advances

Chapter 7

Hypergraph Neural Networks through the Lens of Message Passing: A Common Perspective to Homophily and Architecture Design

Contents

The content of this chapter is adapted from the material submitted to Transactions on Machine Learning Research [201]

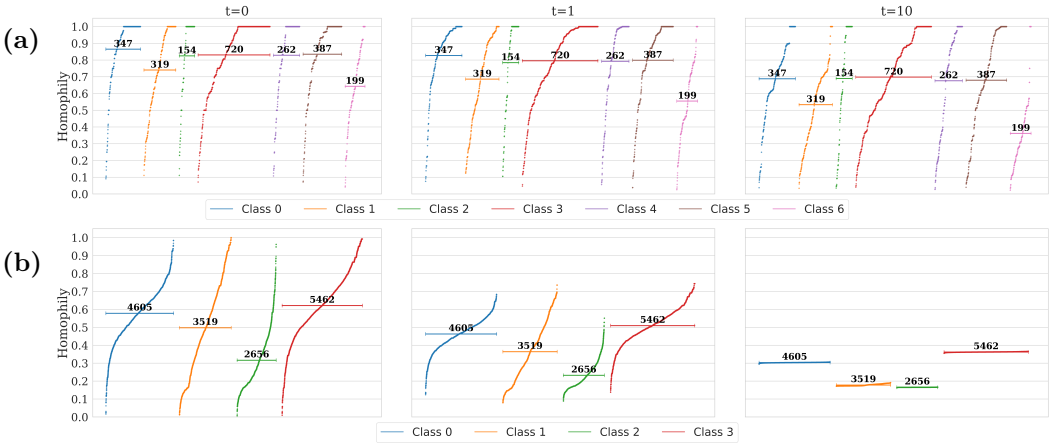


Figure 7.1. Node Homophily Distribution Scores for CORA-CA (a) and 20Newsgroups (b) using Equation 7.2 at $t = 0, 1$, and 10. Horizontal lines depict class mean homophily, with numbers above indicating the quantity of points visualized per class.

This chapter is organized as follows. In Section 7.1, we introduce the problem of overshadowing the hypergraph network foundations question due to the recent rapid development relied on primarily adapting GNN approaches. Section 7.2 provides the novel homophily measure for hypergraph networks, while in Section 7.3, we present the novel MultiSet framework capable of describing most of the current hypergraph architectures. Section 7.4 presents a comprehensive experimental evaluation of the raised questions. Section 7.5 reviews the related work, and the chapter concludes with a discussion in Section 7.6.

7.1 Introduction

Hypergraph learning techniques have multiplied in recent years, demonstrating their effectiveness in processing higher-order interactions in numerous fields, spanning from recommender systems [256, 129], to bioinformatics [261, 244] and computer vision [135, 236]. However, so far, the development of Hypergraph Neural Networks (HNNs) has been largely influenced by the well-established Graph Neural Network (GNN) field. In fact, most of the current methodologies and benchmarking datasets in the hypergraph realm are obtained by *lifting* procedures from their graph counterparts.

The advancement of hypergraph research has been significantly propelled by drawing inspiration from graph-based models [49, 71, 243], but it has simultaneously led to overshadowing hypergraph network foundations. In this chapter, we argue that it is time to address fundamental questions in order to pave the way for further innovative ideas in the field. In that regard, we explore some of these open questions to understand better current HNN architectures and benchmarking datasets along three axes. **Q1** Can the concept of homophily play a crucial role in HNNs, similar to its significance in graph-based research? **Q2** Given that current HNNs are predominantly extensions of GNN architectures adapted to the hypergraph domain, are these extended methodologies suitable, or should we explore new

strategies tailored specifically for handling hypergraph-based data? **Q3** Are existing hypergraph benchmarking datasets truly *meaningful* and representative to draw robust conclusions?

To begin with, we explore how the concept of homophily can be characterized in complex, higher-order networks. Notably, there are many ways of characterizing homophily in hypergraphs –such as the distribution of node features, the analogous distribution of the labels, or the group connectivity similarity (as already discussed in [213]). In particular, this work places the *node class distribution* at the core of the analysis, and introduces a novel definition of homophily that relies on a Message Passing (MP) scheme. This enables us to analyze both hypergraph datasets and architecture designs from the same perspective, as well as to successfully describe model performances (see Section 7.2 and 7.4.2).

Next, we shift our focus towards the design of hypergraph-specific methodologies that HNNs could benefit from, no longer relying on lifting strategies. To this end, after examining state-of-the-art HNN architectures, we first describe the most versatile MP framework up to date, called MultiSet (see Section 7.3.2). Our formulation, which enables hyperedge-dependent node representations and residual connections, inherently generalizes most existing HNN frameworks and models, including AllSet [49], UniGCNII [106], EDHNN [222] and a recently proposed model WHATsNet that allows for hyperedge-dependent node representations [51].

Subsequently, to facilitate a comparison between standard lifted hypergraph models and a new class of hypergraph-specific architectures, in Section 7.3.3, we present the realization of a hypergraph model—MultiSetMixer—that accommodates multiple hyperedge-based node hidden representations.[1] By allowing multiple node representations based on hyperedge memberships—meaning the same node is represented differently depending on the hyperedges it belongs to—this new class of architectures results in an exponential increase in potential node representations as the number of connections within the hypergraph network grows. To address this complexity, we propose and evaluate novel connectivity-based mini-batching strategies tailored to the specific characteristics of hypergraph networks, as detailed in Section 7.3.4. These sampling procedures not only facilitate processing large hyperedges but also give rise to an interesting behavior –which we term connectivity-based distribution shift– thoroughly discussed in the chapter (see Section 7.4.3).

Last but not least, we provide an extensive set of experiments that, driven by the general questions stated above, aim to gain a better understanding of the fundamental aspects of hypergraph representation learning. In fact, the obtained results not only help us contextualize the proposals introduced in this work but open new challenges in hypergraph modeling, such as signal oversquashing caused by large hyperedges, which impacts the performance of all HNNs (see Section 7.4.1) or the tendency of HNNs to ignore connectivity for common benchmark datasets (see Section 7.4.4).

Summary of contributions:

- We introduce a novel definition of homophily for hypergraphs, capable of

[1] Note that MultiSetMixer is a simple yet powerful model architecture that serves primarily as a baseline, providing an example of a model that incorporates and employs the unique characteristics of hypergraph networks.

effectively describing HNN model performances (**Q1** and **Q3**, Sections 7.2 and 7.4.2).

- We present the novel MultiSet framework, which incorporates hyperedge-dependent node representations and generalizes most of the existing hypergraph models in the literature (**Q2**, Section 7.3.2).
- We explore novel mini-batch sampling strategies for architectures with and without hyperedge-dependent node classification (**Q2**, Section 7.3.3).
- We perform a large set of experiments assessing benchmarking both datasets and HNN architectures, as well as connecting the proposed MP homophily with the models' performance (**Q1**, **Q2**, **Q3**, Sections 7.2 and 7.4).

Notation. A hypergraph is an ordered pair of sets $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of hyperedges. Each hyperedge $e \in \mathcal{E}$ is a subset of $\mathcal{V}$, i.e., $e \subseteq \mathcal{V}$. A hypergraph is a generalization of the concept of a graph where (hyper)edges can connect more than two nodes. A vertex v and a hyperedge e are said to be incident if $v \in e$. For each node v, we denote its class by y_v, and we denote by $\mathcal{E}_v = \{e \in \mathcal{E} : v \in e\}$ the subset of hyperedges in which it is contained. We represents by $d_v = |\mathcal{E}_v|$ the node degree. The set of classes is represented by $\mathcal{C} = \{c_i\}_{i=1}^{|\mathcal{C}|}$.

7.2 Homophily Metrics in Hypergraphs

In this section, we present a new definition of homophily that employs a two-step message passing scheme applicable to general, non-uniform hypergraphs, in contrast to the definition by Veldt et al. [213]. In essence, our definition focuses on capturing hyperedge interconnections by the exchange of information following the message passing scheme. Following that, we illustrate its applicability in examining higher-order networks through qualitative analysis. Finally, we demonstrate the applicability of the proposed concept by deriving a Δ homophily measure. In Section 7.4.2, we show its capability to describe HNNs' performance. These play a pivotal role in our attempt to answer the fundamental question **Q1** raised in the Introduction.

Message passing homophily. Given a a hyperedge $e \in \mathcal{E}$, we define the 0-level hyperedge homophily $h_e^0(c)$ as the fraction of nodes within e that belong to class c, i.e.

$$h_e^0(c) = \frac{1}{|e|} \sum_{v \in e} \mathbb{1}_{y_v = c}. \tag{7.1}$$

This score describes how homophilic the initial connectivity is with respect to class c. Computing the score for each class $c_i \in \mathcal{C}$ generates a categorical distribution for each $e \in \mathcal{E}$, i.e. $h_e^0 = [h_e^0(c_0), \ldots, h_e^0(c_{|C|})]$. Using this information as a starting point, we calculate higher-level homophily measurements for both nodes and hyperedges through the two-step message passing approach. Formally, we define the t-level

homophily score as

$$h_v^{t-1} = \texttt{AGG}_{\mathcal{E}}\left(\{h_e^{t-1}(y_v)\}_{e\in\mathcal{E}_v}\right), \tag{7.2}$$

$$h_e^t(c) = \texttt{AGG}_{\mathcal{V}}\left(\{h_v^{t-1}\}_{v\in e, y_v=c}\right), \tag{7.3}$$

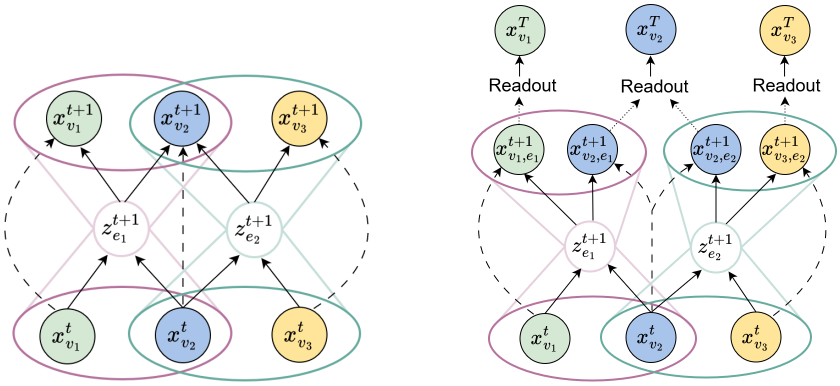
where $\texttt{AGG}_{\mathcal{E}}$ and $\texttt{AGG}_{\mathcal{V}}$ are functions aggregating edge and node homophily scores, respectively (we consider the mean operator in our implementation). We note that our homophily measure enables the definition of a score for each node and hyperedge for any neighborhood resolution.

Qualitative analysis. One straightforward way to make use of the message passing homophily measure is to visualize how the node homophily score dynamically changes, as described in Eq. 7.2. Figure 7.1 depicts this process for non-isolated nodes on CORA-CA and 20NewsGroup datasets. Looking at CORA-CA (Figure 7.1 (a)), we note that there are a significant number of nodes with high 0-level homophily at each class (except number 6), and this homophily distribution is kept mostly unchanged as we move to the 1-hop neighborhood ($t = 1$). Interestingly, the same trend holds even when shifting to the 10-level node homophily –only classes 1 and 6 show a relevant drop in highly homophilic nodes. This suggests the presence of isolated homophilic subnetworks within the hypergraph. In contrast, 20Newsgroups dataset (Figure 7.1 (b)) displays relatively low node homophily scores from the 0-level (specifically for class 2, with a mean value around 0.3). Moving to $t = 1$, there is a significant decrease in the homophily scores for every class. Finally, at time step $t = 10$, we can observe that all the classes converge to approximately the same homophily values within each class. This convergence and low homophily scores suggest that the network is highly interconnected.

Δ homophily. Rather than measuring homophily in individual discrete timestamps, we next derive Δ *homophily* measure, which offers a dynamic perspective to explore hypernetworks. This measure is based on the assumption that if the 1-hop neighborhood of a node $u \in \mathcal{V}$ is predominantly homophilic, then the change in homophily score between two consecutive timestamps will be small. Conversely, a substantial change in u's homophily implies that the node resides in a neighborhood characterized as heterophilic. Specifically, for each node, we quantify the homophily change after t-th step of message passing by computing the difference between the node homophily at that step from its value at previous one, $t-1$. Subsequently, we look at the proportion of nodes whose homophily difference is below a certain threshold $\mu \in \mathbb{R}^+$, i.e.

$$\Delta_\mu^t = \frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}} \mathbb{1}_{|h_v^t - h_v^{t-1}|<\mu}. \tag{7.4}$$

As we show in Section 7.4.2, this dynamic measure becomes a helpful tool to analyze the performance of HNNs.



(a) AllSet layout. **(b)** MultiSet layout.

Figure 7.2. Visual representation of the AllSet and MultiSet frameworks.

7.3 Methods

Current HNNs aim to generalize GNN concepts to the hypergraph domain, and are specifically focused on redefining graph-based propagation rules to accommodate higher-order structures. In this regard, the work of Chien et al. [49] introduced a general notation framework, called AllSet, that encompasses most of currently available HNN layers, including CEGCN/CEGAT, HNN [71], HNHN [64], HCHA [10], HyperGCN [243], as well as AllDeepSet and AllSetTransformer presented in the same work [49]. This section first revisits the original AllSet formulation, and then introduces a new framework –MultiSet– which extends AllSet by allowing multiple hyperedge-dependent representations of nodes. Finally, we present a particular realization of an architecture that takes into account hyperedge dependent-node representations. Then, we open a discussion on the scalability issues of hypergraph architectures and propose a novel mini-batching scheme. In contrast to previous formulations, our proposed framework and implementation are inspired by hypergraph needs and features, and motivated by the fundamental question **Q2**.

7.3.1 AllSet Propagation Setting

For a given node $v \in \mathcal{V}$ and hyperedge $e \in \mathcal{E}$ in a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, let $\boldsymbol{x}_v^{(t)} \in \mathbb{R}^f$ and $\boldsymbol{z}_e^{(t)} \in \mathbb{R}^d$ denote their vector representations at propagation step t. We say that a function f is a multiset function if it is permutation invariant w.r.t. each of its arguments in turn. Typically, $\boldsymbol{x}_v^{(0)}$ and $\boldsymbol{z}_e^{(0)}$ are initialized based on the corresponding node and hyperedge original features, if available. The vectors $\boldsymbol{x}_v^{(0)}$ and $\boldsymbol{z}_e^{(0)}$ represent the initial node and hyperedge features, respectively. In this context, the AllSet framework [49] consists in the following two-step update rule:

$$\boldsymbol{z}_e^{(t+1)} = f_{\mathcal{V}\to\mathcal{E}}(\{\boldsymbol{x}_u^{(t)}\}_{u:u\in e}; \boldsymbol{z}_e^{(t)}), \tag{7.5}$$

$$\boldsymbol{x}_v^{(t+1)} = f_{\mathcal{E}\to\mathcal{V}}(\{\boldsymbol{z}_e^{(t+1)}\}_{e\in\mathcal{E}_v}; \boldsymbol{x}_v^{(t)}), \tag{7.6}$$

where $f_{\mathcal{V}\to\mathcal{E}}$ and $f_{\mathcal{E}\to\mathcal{V}}$ are two permutation invariant functions with respect to their first input. Equations 7.5 and 7.6 describe the propagation from nodes to

hyperedges and vice versa, respectively. We extend the original AllSet formulation to accommodate UniGCNII [106] by modifying the node update rule (Eq. 7.6) as follows

$$\boldsymbol{x}_v^{(t+1)} = f_{\mathcal{E}\to\mathcal{V}}(\{\boldsymbol{z}_e^{(t+1)}\}_{e\in\mathcal{E}_v}; \{\boldsymbol{x}_v^{(k)}\}_{k=0}^{t}), \tag{7.7}$$

i.e. allowing residual connections. Again, the only requirement is to be invariant w.r.t. the first input.

Proposition 7.3.1 *UniGCNII [106] is a special case of AllSet considering 7.5 and 7.7.*

In the practical implementation of a model, $f_{\mathcal{V}\to\mathcal{E}}$ and $f_{\mathcal{E}\to\mathcal{V}}$ are parametrized and learnt for each dataset and task; particular choices of these functions give rise to most of the HNN architectures considered in this chapter (see Appendix C.2).

7.3.2 MultiSet Framework

In this Section, we introduce our proposed MultiSet framework, which can be seen as an extension of AllSet where nodes can have multiple co-existing hyperedge–based representations. For a given hyperedge $e \in \mathcal{E}$ in a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we denote by $\boldsymbol{z}_e^{(t)} \in \mathbb{R}^d$ its vector representation at step t. For a node $v \in \mathcal{V}$, MultiSet allows for as many representations of the node as the number of hyperedges it belongs to. We denote by $\boldsymbol{x}_{v,e}^{(t)} \in \mathbb{R}^f$ the vector representation of node v in a hyperedge $e \in \mathcal{E}_v$ at propagation time t, and by $\mathbb{X}_v^{(t)} = \{\boldsymbol{x}_{v,e}^{(t)}\}_{e\in\mathcal{E}_v}$ the set of all d_v hidden states of that node in the specified time-step. Accordingly, the hyperedge and node update rules of MultiSet are formulated to accommodate hyperedge–dependent node representations:

$$\boldsymbol{z}_e^{(t+1)} = f_{\mathcal{V}\to\mathcal{E}}(\{\mathbb{X}_u^{(t)}\}_{u:u\in e}; \boldsymbol{z}_e^{(t)}), \tag{7.8}$$

$$\boldsymbol{x}_{v,e}^{(t+1)} = f_{\mathcal{E}\to\mathcal{V}}(\{\boldsymbol{z}_e^{(t+1)}\}_{e\in\mathcal{E}_v}; \{\mathbb{X}_v^{(k)}\}_{k=0}^{t}), \tag{7.9}$$

where $f_{\mathcal{V}\to\mathcal{E}}$ and $f_{\mathcal{E}\to\mathcal{V}}$ are two multiset functions with respect to their first input. After T iterations of message passing, MultiSet also considers a last readout-based step to obtain a unique final representation $\boldsymbol{x}_v^T \in \mathbb{R}^{f'}$ for each node from the set of its hyperedge–based representations:

$$\boldsymbol{x}_v^{(T)} = f_{\mathcal{V}\to\mathcal{V}}(\{\mathbb{X}_v^{(k)}\}_{k=0}^{T}) \tag{7.10}$$

where $f_{\mathcal{V}\to\mathcal{V}}$ is also a multiset function.

As we show in the following propositions (proofs can be found in Appendices C.3.2 C.3.3 and C.3.4), both AllSet framework and the more recent EDHNN and WHATsNet architectures [222, 51] can be expressed in terms of the general MultiSet notation.

Proposition 7.3.2 *AllSet 7.5-7.6, as well as its extension 7.5-7.7, are special cases of MultiSet 7.8-7.9-7.10.*

Proposition 7.3.3 *EDHNN [222] is a special case of MultiSet 7.8-7.9-7.10.*

Proposition 7.3.4 *WHATsNet [51] is a special case of MultiSet 7.8-7.9-7.10.*

7.3.3 Training MultiSet Networks

This section describes the main characteristics of MultiSet layer implementation, along with a possible realization, which we refer to as MultiSetMixer.

Learning MultiSet layers. Following the mixer-style block designs [204] and standard practice, we use the following MultiSet layer implementation:

$$\begin{aligned} \boldsymbol{z}_e^{(t+1)} &= f_{\mathcal{V}\to\mathcal{E}}(\{\boldsymbol{x}_{u,e}^{(t)}\}_{u:u\in e}; \boldsymbol{z}_e^{(t)}) \qquad (7.11)\\ &:= \frac{1}{|e|}\sum_{v\in e}\boldsymbol{x}_{u,e}^{(t)} + \mathrm{MLP}\left(\mathrm{LN}\left(\frac{1}{|e|}\sum_{v\in e}\boldsymbol{x}_{u,e}^{(t)}\right)\right),\\ \boldsymbol{x}_{v,e}^{(t+1)} &= f_{\mathcal{E}\to\mathcal{V}}(\boldsymbol{z}_e^{(t+1)}; \boldsymbol{x}_{v,e}^{(t)}) \qquad (7.12)\\ &:= \boldsymbol{x}_{v,e}^{(t)} + \mathrm{MLP}\left(\mathrm{LN}(\boldsymbol{x}_{v,e}^{(t)})\right) + \boldsymbol{z}_e^{(t+1)},\\ \boldsymbol{x}_v^{(T)} &= f_{\mathcal{V}\to\mathcal{V}}(\mathbb{X}_v^{(T)}) := \frac{1}{d_v}\sum_{e\in\mathcal{E}_v}\boldsymbol{x}_{v,e}^{(t)} \qquad (7.13) \end{aligned}$$

where MLPs are composed of two fully-connected layers, and LN stands for layer normalization. This architecture, which we call MultiSetMixer, is based on a mixer-based pooling operation for *(i)* updating hyperedges from its node's representations, and *(ii)* generate and update hyperedge-dependent representations of the nodes.

Proposition 7.3.5 *The functions $f_{\mathcal{V}\to\mathcal{E}}$, $f_{\mathcal{E}\to\mathcal{V}}$ and $f_{\mathcal{V}\to\mathcal{V}}$ defined in MultiSetMixer are permutation invariant. Furthermore, these functions are universal approximators of multiset functions when the size of the input multiset is finite.*

7.3.4 Mini-batching

The motivation for introducing a new strategy to iterate over hypergraph datasets is twofold. First, current HNN pipelines face scalability issues when processing datasets with a large number of nodes and hyperedges. This problem is particularly pronounced in architectures that allow for hyperedge-dependent node representations, as each node must be represented multiple times. This scaling challenge becomes significant as the number of nodes, hyperedges, and hyperedge sizes grow (see Table C.2 in the Appendix for relevant statistics). Second, pooling operations over large sets can lead to signal oversquashing, negatively impacting the performance of HNNs that do not account for this issue (see Table 7.1), as demonstrated by the 20Newsgroups dataset, where hyperedges have a median of 537 nodes per hyperedge (see Table C.1 in the Appendix for detailed statistics).

To address these issues, we propose sampling X mini-batches of a certain size B at each iteration in two steps. At *step 1*, we sample B hyperedges from $\mathcal{E}$. The hyperedge sampling over $\mathcal{E}$ can be either uniform or weighted (e.g. by taking into account hyperedge cardinalities). Then in *step 2* L nodes are in turn sampled from each sampled hyperedge e, padding the hyperedge with $L-|e|$ special padding tokens if $|e| < L$ –consisting of $\mathbf{0}$ vectors that can be easily discarded in some computations. Overall, the shape of the obtained mini-batch X is $B \times L$.

Step 0 is particularly beneficial for hypergraph datasets with a large number of hyperedges, but it can be skipped when the entire network fits into memory. In

Sections 7.4.1 and 7.4.3, we demonstrate that *step 1* (node mini-batching within a hyperedge) is useful for two key reasons: (i) pooling operations over large sets may lead to signal oversquashing, while pooling over smaller sets helps mitigate this issue, and (ii) node batching introduces an intriguing effect—what we refer to as connectivity-based distribution shift—which offers a novel way to leverage connectivity to rebalance the training distribution. Therefore, it can still be beneficial to use node mini-batching even when the entire hyperedge fits into memory.

When both *step 1* and *step 2* are employed, the memory required during the forward pass is $B \times L \times d$, where d is the hidden dimension size. If only *step 2* is used, the batch size becomes $|\mathcal{E}| \times L \times d$, where $|\mathcal{E}|$ is the total number of hyperedges in the hypernetwork. Finally, if no mini-batching is applied, the batch size is $|\mathcal{E}| \times \max_{e \in \mathcal{E}} |e| \times d$, where $\max_{e \in \mathcal{E}} |e|$ denotes the size of the largest hyperedge. For a detailed empirical analysis of memory utilization across different datasets and sampling schemes, see Section 7.4.3.

7.4 Evaluation

The questions that we raised in Section 7.1 have shaped our research, leading to a new definition of higher-order homophily and unexplored architectural designs that can potentially fit better the properties of hypergraph networks. In subsequent subsections, we set four questions that follow up from these fundamental inquiries and can help contextualize the technical contributions of this chapter.

Datasets and models. We use the same datasets used in Chien et al. [49], which includes Cora, Citeseer, Pubmed, ModelNet40, NTU2012, 20Newsgroups, Mushroom, ZOO, CORA-CA, and DBLP-CA. More information about datasets and corresponding statistics are in Appendix C.4.2. We also utilize the benchmark implementation provided by Chien et al. [49] to conduct the experiments with several models, including AllDeepSets, AllSetTransformer, EDHNN, UniGCNII, CEGAT, CEGCN, HCHA, HNN, HNHN, HyperGCN, HAN, and HAN mini-batching. Since the design of these architectures is primarily inspired by the GNN literature, we refer to them as lifted architectures. Additionally, we consider vanilla MLP applied to node features and a transformer architecture, as well as MultiSetMixer representing a baseline that employs hyperedge-dependent node representations and a new MLP baseline leveraging Connectivity Batching (MLP CB). We refer to Section 7.3.3 for more details about all these architectures. All models are optimized using 15 splits with 2 model initializations, resulting in a total of 30 runs; see Appendix C.4.1 for further details.

7.4.1 How do the lifted models perform in comparison to hypergraph-specific models?

Our first experiment directly targets our fundamental **Q2** by assessing the performance of lifted architectures in comparison with those that leverage unique characteristics of hypergraph networks, such as hyperedge-dependent node representations and hyperedge-based mini-batching.

Table 7.1. Hypergraph model performance benchmarks. Test accuracy in % averaged over 15 splits.

Model	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	77.11 ± 1.00	70.67 ± 1.42	89.04 ± 0.45	82.23 ± 1.46	91.34 ± 0.27	99.96 ± 0.05	86.49 ± 1.86	96.70 ± 0.25	81.19 ± 0.49	89.10 ± 7.00	6.80
AllSetTransformer	79.54 ± 1.02	72.52 ± 0.88	88.74 ± 0.51	84.43 ± 1.14	91.61 ± 0.19	99.95 ± 0.05	88.22 ± 1.42	98.00 ± 0.12	81.59 ± 0.59	**91.03 ± 7.31**	3.25
UniGCNII	78.46 ± 1.14	73.05 ± 1.48	88.07 ± 0.47	83.92 ± 1.02	91.56 ± 0.18	99.89 ± 0.07	88.24 ± 1.56	97.84 ± 0.16	81.16 ± 0.49	89.61 ± 8.09	4.75
EDHNN	**80.74 ± 1.00**	73.22 ± 1.14	**89.12 ± 0.47**	**85.17 ± 1.02**	**91.94 ± 0.23**	99.94 ± 0.11	88.04 ± 1.65	97.70 ± 0.19	81.64 ± 0.49	89.49 ± 6.99	**2.90**
CEGAT	76.53 ± 1.58	71.58 ± 1.11	87.11 ± 0.49	77.50 ± 1.51	88.74 ± 0.31	96.81 ± 1.41	82.27 ± 1.60	92.79 ± 0.44	NA	44.62 ± 9.18	12.11
CEGCN	77.03 ± 1.31	70.87 ± 1.19	87.01 ± 0.62	77.55 ± 1.65	88.12 ± 0.25	94.91 ± 0.44	80.90 ± 1.74	90.04 ± 0.47	NA	49.23 ± 6.81	12.56
HCHA	79.53 ± 1.33	72.57 ± 1.06	86.97 ± 0.55	83.53 ± 1.12	91.21 ± 0.28	98.94 ± 0.54	86.60 ± 1.96	94.50 ± 0.33	80.75 ± 0.53	89.23 ± 6.81	7.85
HGNN	79.53 ± 1.33	72.24 ± 1.08	86.97 ± 0.55	83.45 ± 1.22	91.26 ± 0.26	98.94 ± 0.54	86.71 ± 1.48	94.50 ± 0.33	80.75 ± 0.52	89.23 ± 6.81	7.95
HNHN	77.68 ± 1.08	73.47 ± 1.36	87.88 ± 0.47	78.53 ± 1.15	86.73 ± 0.40	**99.97 ± 0.04**	88.28 ± 1.50	97.84 ± 0.15	81.53 ± 0.55	89.23 ± 7.85	5.55
HyperGCN	74.78 ± 1.11	66.06 ± 1.58	82.32 ± 0.62	77.48 ± 1.14	86.07 ± 3.32	69.51 ± 4.98	47.65 ± 5.01	46.10 ± 7.95	80.84 ± 0.49	51.54 ± 9.88	14.30
HAN	80.73 ± 1.37	**73.69 ± 0.95**	86.34 ± 0.61	84.19 ± 0.81	91.10 ± 0.20	91.33 ± 0.91	83.78 ± 1.75	93.85 ± 0.33	79.67 ± 0.55	80.26 ± 6.42	8.90
HAN minibatch	80.24 ± 2.17	73.55 ± 1.13	85.41 ± 2.32	82.04 ± 2.56	90.52 ± 0.50	93.87 ± 1.04	80.62 ± 2.00	92.06 ± 0.63	79.76 ± 0.56	70.39 ± 11.29	10.60
MultiSetMixer	78.06 ± 1.24	71.85 ± 1.50	87.19 ± 0.53	82.74 ± 1.23	90.68 ± 0.19	99.58 ± 0.16	**88.90 ± 1.30**	**98.38 ± 0.21**	**88.57 ± 1.96**	88.08 ± 8.04	6.20
MLP CB	74.06 ± 1.26	71.93 ± 1.53	85.83 ± 0.51	74.39 ± 1.40	84.91 ± 0.44	99.93 ± 0.08	85.43 ± 1.51	96.41 ± 0.32	86.13 ± 2.82	81.61 ± 10.98	10.30
MLP	73.27 ± 1.09	72.07 ± 1.65	87.13 ± 0.49	73.27 ± 1.09	84.77 ± 0.41	99.91 ± 0.08	79.70 ± 1.56	95.31 ± 0.28	80.93 ± 0.59	85.13 ± 6.90	11.50
Transformer	74.15 ± 1.17	71.82 ± 1.51	87.37 ± 0.49	73.61 ± 1.55	85.26 ± 0.38	99.95 ± 0.08	82.88 ± 1.93	96.29 ± 0.29	81.17 ± 0.54	88.72 ± 10.25	9.85

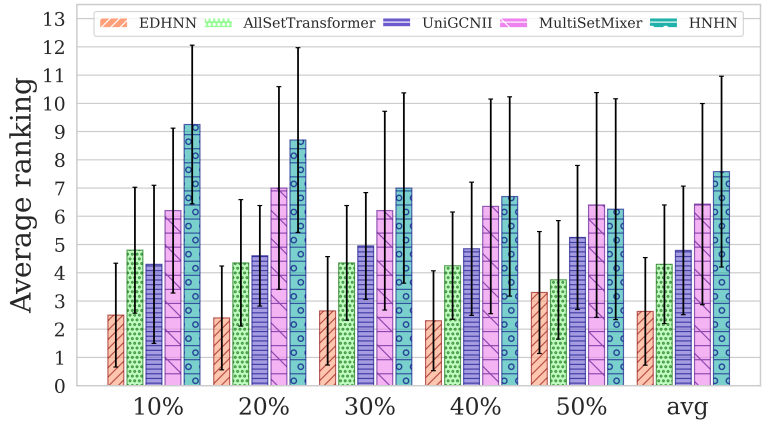


Figure 7.3. Average rankings at different training percentages, with the overall average performance shown on the right. The error bars indicate the standard deviation of the average rankings across multiple datasets.

Figure 7.3 shows the average rankings –across all models and datasets– of the top-5 best-performing models for different training splits, exhibiting that those splits can impact the relative performance among models. In the main body of this work, we focus our analysis on the 50% split results presented in Table 7.1,[2] while the corresponding tables for other scenarios are provided in Appendix C.5.1. Table 7.1 emphasizes MultiSetMixer solid performance, obtaining the highest test accuracy on NTU2012, ModelNet40, and 20Newsgroups datasets. Notably, MultiSetMixer and MLP CB share similar patterns, and both significantly outperform all the other architectures on 20Newsgroups, which we further discuss in Section 7.4.3.

In fact, the comparable performance among the rest of HNN models on this dataset suggests that existing architectures can not account for the dataset connectivity. According to what we observed in the qualitative homophily analysis performed in Section 7.2, 20Newsgroup is densely interconnected, making it highly heterophilic as the MP evolves; we argue this presents a challenge for most of current HNNs architectures. In contrast, CORA-CA exhibits a high degree of homophily within its hyperedges and shows the most significant performance gap between HNNs and

[2]Unless otherwise specified, all tables in the main body of the chapter use a 50%/25%/25% split between training and testing. The results are shown as Mean Accuracy Standard Deviation, with the best result highlighted in bold and shaded in grey, and results within one standard deviation are displayed in blue-shaded boxes.

the baselines. Please refer to Section 7.4.4 for more experiments on the impact of connectivity.

Finally, we highlight the strong overall performance of the non-inductive baselines (MLP, MLP CB) across most datasets. Notably, HNN architectures significantly outperform them in only 3 out of 10 cases (Cora, CORA-CA, DBLP-CA). This fact showcases that, so far, features are being more representative than connectivity in most considered hypergraph datasets –a relevant insight for **Q3**.

7.4.2 When are HNNs exploiting the connectivity?

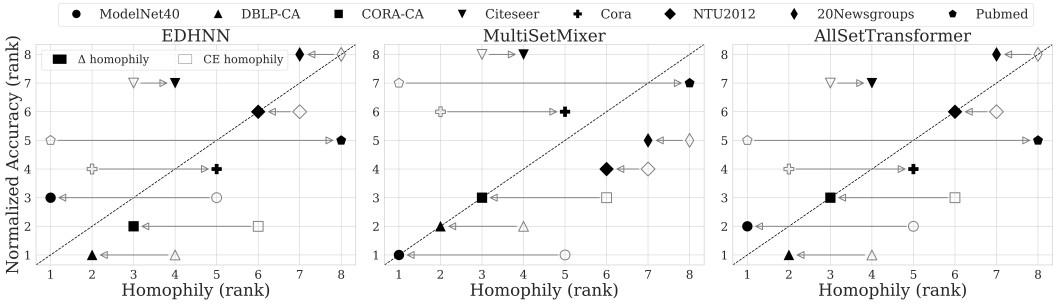


Figure 7.4. Joint visualization of rank dependencies, showing Norm. Acc. versus Δ Homophily (Eq. 7.4 at step $t = 1$ and $\mu = 0.1$) and CE Homophily [222]. Norm. Acc. (Eq. 7.14) is assessed for various instances of model A (specified in column titles), with model B being MLP CB. Both axes represent rank values, with lower values indicating better metrics. Arrows denote the rank shift in homophily between CE homophily and Δ homophily for each dataset.

Motivated by the previous observation of the general results, in this section, we investigate when HNN models actually take advantage of the inductive bias provided by the message-passing scheme. To do so, we first propose a way to capture the influence of the bias in the downstream task performance –in an attempt to decouple it from the impact of just the dataset features–, and then investigate if the datasets' homophily scores are able to account for the resulting observations. We argue this study provides a valuable contribution to questions **Q2** and **Q3**.

In order to quantitatively assess the impact of inductive bias, we compare the results of HNNs –i.e., model A–, with those of another architecture that does not leverage the connectivity –i.e. a non-inductive baseline, model B. Specifically, we measure the difference between the accuracy of model A (acc_A) and B (acc_B) (Table 7.2 shows the computed differences considering MLP and MLP CB as baselines). The real-world datasets employed in this study span diverse domains and, as depicted in Table 7.1, this implies considerable variations in performance values across datasets. In order to mitigate such variability, we introduce the following normalized accuracy relative to acc_B:

$$\text{Norm. Acc.} = (acc_A - acc_B)/(100 - acc_B). \tag{7.14}$$

Next, we are interested in assessing if dataset's homophily can shed some light on the resulting normalized accuracy measurements. To that end, we consider two

Table 7.2. Difference in Accuracy between Model A and Model B; they represent, respectively, models with and without inductive bias.

Model A	Model B	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO
AllDeepSets	MLP	3.84	-1.40	1.91	8.96	6.57	0.05	6.79	1.39	0.26	3.97
	MLP CB	3.05	-1.26	3.21	7.84	6.43	3.13	1.06	0.29	-4.94	7.49
AllSetTransformer	MLP	6.27	0.45	1.61	11.16	6.84	0.04	8.52	2.69	0.66	5.90
	MLP CB	5.48	0.59	2.91	10.04	6.70	3.12	2.79	1.59	-4.54	9.42
EDHNN	MLP	7.47	1.15	1.99	11.90	7.17	0.03	8.34	2.39	0.71	4.36
	MLP CB	6.68	1.29	3.29	10.78	7.03	0.01	2.61	1.29	-4.49	7.88
MultiSetMixer	MLP	4.79	-0.22	0.06	9.47	5.91	-0.33	9.20	3.07	7.64	2.95
	MLP CB	4.00	-0.08	1.36	8.35	5.77	-0.35	3.47	1.97	2.44	6.47

different homophily measures: on the one hand, our proposed Δ homophily between the two first steps of the MP (Eq 7.4 with $t = 1$ and $\mu = 0.1$). On the other hand, Clique Expansion (CE) homophily, calculated over the clique expansion of the hypergraph following the approach of Wang et al. [222]. Figure 7.4 illustrates the rank dependency of normalized accuracy against these two homophily measures, with MLP CB serving as the strongest non-inductive baseline (as indicated by Table 7.1). Additionally, we show the ideal correlation –performance directly proportional to homophily– with the dashed diagonal, as well as the rank shift in homophily between the two homophily measures through the arrows. The difference between EDHNN, MultiSetMixer, and AllSetTransformer lies in the way message passing propagates information (see Section 7.3.2). Mushroom and Zoo datasets were excluded due to Mushroom's discriminatory node features and Zoo's small hypernetwork size.

The comparison between CE homophily and Δ homophily reveals a notable trend, with Δ homophily consistently aligning closer, on average, to the middle dashed line –indicating a higher positive correlation between performance and Δ homophily level in comparison to CE homophily. Remarkably, across all architectures, the highest normalized accuracy is consistently distributed across ModelNet40, DBLP-CA, and CORA-CA, with Δ homophily ranking them as the top three accordingly. A striking shift in rankings is observed for the Pubmed dataset, transitioning from the most homophilic under the CE homophily measure to the least homophilic under Δ homophily. We associate this to the high percentage of isolated nodes (80.52%, see Table C.2): while CE homophily scores are largely influenced by them, our proposed measure ignores self-connections. Additionally, the 20Newsgroup dataset occupies the last positions in both homophily ranks, aligning with our quantitative analysis findings.

In summary, we show the applicability of Δ homophily by showing that it exhibits a positive correlation with respect to the ability of exploiting the connectivity by HNN architectures, significantly stronger than the CE homophily that is commonly used nowadays. Our findings underscore the crucial role of accurately expressing homophily in hypergraph networks, entangling with the complexity in capturing higher-order dependencies.

7.4.3 What is the impact of the mini-batch sampling?

Next, we examine the role of our proposed mini-batching sampling in explaining the general results shown in Table 7.1, and investigate how it influences other models' performance. These experiments provide valuable insights on **Q2**.

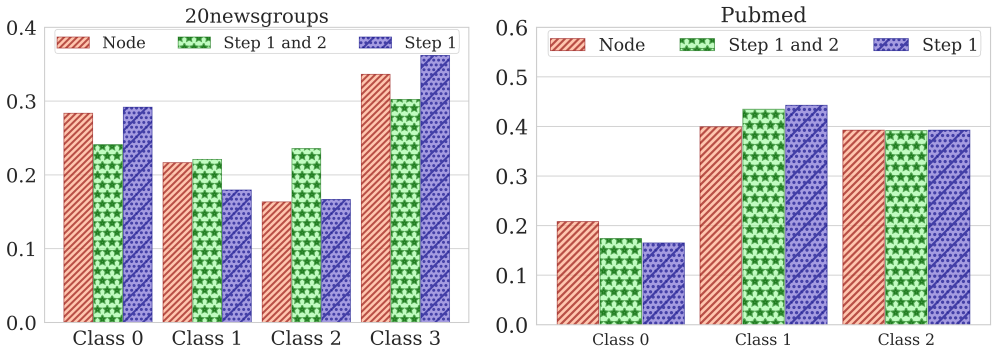


Figure 7.5. Class distribution shift induced by mini-batching: '*Node*' represents the original node class distribution, '*Step 1 and 2*' the resulting one after sampling both hyperedges and nodes, and '*Step 1*' when only sampling hyperedges.

Table 7.3. Mini-batching experiment. Test accuracy in % averaged over 15 splits.

Model	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllSetTransformer (batched)	74.34 ± 1.08	69.67 ± 1.46	**87.75 ± 0.30**	75.75 ± 1.46	86.06 ± 0.22	99.91 ± 0.05	87.55 ± 0.86	96.42 ± 0.17	81.37 ± 0.28	**93.20 ± 5.38**	2.70
EDHNN	77.88 ± 0.69	69.51 ± 0.87	86.82 ± 0.33	**83.12 ± 0.89**	90.45 ± 0.28	**99.95 ± 0.04**	87.64 ± 0.99	97.55 ± 0.17	81.23 ± 0.31	90.00 ± 4.43	2.40
MultiSetMixer	78.06 ± 1.24	71.85 ± 1.50	87.19 ± 0.53	82.74 ± 1.23	**90.68 ± 0.19**	99.58 ± 0.16	**88.90 ± 1.30**	**98.38 ± 0.21**	**88.57 ± 1.96**	88.08 ± 8.04	**1.80**
HAN minibatch	**80.24 ± 2.17**	**73.55 ± 1.13**	85.41 ± 2.32	82.04 ± 2.56	90.52 ± 0.50	93.87 ± 1.04	80.62 ± 2.00	92.06 ± 0.63	79.76 ± 0.56	70.39 ± 11.29	3.10

Class distribution analysis. To evaluate and motivate the potential of the proposed mini-batching sampling, we investigate the reason behind both *(i)* the superior performance of MultiSetMixer and MLP CB on 20NewsGroup, NTU2012, ModelNet40, and *(ii)* their poor performance on Pubmed. Framing mini-batching from the connectivity perspective presents a challenge that conceals significant potential for improvement [203]. It is important to note that connectivity, by definition, describes relationships among the nodes, implying that some parts of the dataset might interconnect more densely, creating some sort of hubs within the network. Thus, mini-batching might introduce unexpected skew in training distribution. In particular, Figure 7.5, depicts the original class distribution of the dataset, and compares it to the skewed distributions resulting from employing the corresponding steps of mini-batching (see Section 7.3.3). Note that, when sampling both hyperedges and nodes ('*Step 1 and 2*'), dominant classes 0 and 3 undergo undersampled, contributing to a more balanced distribution in the case of 20Newsgroup. Conversely, for Pubmed, class 2 is undersampled, while the predominant 1st class experiences oversampling, further skewing the distribution in this dataset. This observation leads to the hypothesis that, in some cases, the sampling procedure produces a natural shift that rebalances the class distributions, which in turn helps to improve the performance.

Application to other models. Furthermore, we explore the proposed mini-batch sampling procedure with the AllSetTransformer and EDHNN models by just sampling hyperedges without additional hyperparameter optimization. From Table 7.3, we can observe a drop in performance for most of the datasets both for AllSetTransformer and for EDHNN, but overall they in turn outperform the HAN (mini-batching) model. This suggests the substantial potential of the proposed sampling procedure.

Scalability. To evaluate the scalability of the proposed mini-batching approach, we compare memory utilization across the mid-sized datasets used in this study, as well as the large-scale Amazon dataset, which consists of 100k hyperedges and 547k nodes. This comparison highlights the efficiency of our method when applied to larger datasets. On Amazon, MultiSetMixer achieves a substantial improvement in memory efficiency, reducing utilization by an order of magnitude compared to AllSetTransformer (from 19020MiB to 1232MiB) while achieving comparable performance (92.22 ± 0.84 vs 93.22 ± 0.089). In contrast, the batched version of AllSetTransformer performs worse on the same dataset, achieving 82.634±0.029, while consuming around eight times more memory memory compared to MultiSetMixer.

Table 7.4. Memory usage comparison across datasets.

Dataset	Pubmed	DBLP CA	NTU2012	ModelNet40	20newsW100	Amazon
AllSetTransformer	2536MiB	2306MiB	1206MiB	2426MiB	2352MiB	19020MiB
AllSetTransformer (batched)	2652MiB	1940MiB	1138MiB	2038MiB	4224MiB	8022MiB
MultiSetMixer	1290MiB	2364MiB	1174MiB	1280MiB	1112MiB	1232MiB

7.4.4 How do connectivity changes affect performance?

In this section, we extensively explore the structure of datasets and assess model performance by manipulating the original connectivity of the datasets. The extent to which the performance of the models is affected by changes in connectivity provides valuable information both on the properties of the datasets and on the considered architectures.

We design two different experimental approaches, aiming to systematically modify the original connectivity of datasets. The first experiment tests the performance when some hyperedges are removed following different *drop connectivity* strategies. The second one examines the models performance by introducing two preprocessing strategies on the hypergraph connectivity. Our findings below shed some light on the fundamental questions **Q1**, **Q2** and **Q3**.

Reducing connectivity

This experiment aims to investigate the significance of connectivity information in datasets and the extent to which it influences the performance of the models. We divide this experiment into two parts: (i) drop connectivity and (ii) connectivity rewiring. In the first part of the experiment, we employ three strategies to introduce variations in the initial dataset's connectivity. The first two strategies involve ordering hyperedges based on their lengths in **ascending order**. In the first approach, referred to as *trimming*, we remove the initial $x\%$ of ordered hyperedges, for a certain fixed fraction x. The second approach, referred to as *retention*, involves keeping the first $x\%$ of hyperedges and discarding the remaining $100 - x\%$. The last strategy instead involves randomly dropping $x\%$ of hyperedges from the dataset, and it is referred to as *random drop*.

Results. The results are shown in Table 7.5, and they indicate that connectivity minimally impacts CEGCN and AllSetTransformer for the Citeseer and Pubmed

datasets. On the other hand, MultiSetMixer performs better at the *trimming 25%* setting, although the achieved performance is on par with MLP, as reported in Table 7.1. This indicates that the proposed model is negatively affected by the distribution shift. Conversely, for the Pubmed dataset, we observe that MultiSetMixer's performance improves, likely due to the reduced impact of the distribution shift in this case. Notably, CEGCN shows improvements on 7 out of 9 datasets, with the most significant gains seen in the ZOO dataset, where performance nearly doubles. Another interesting pattern is observed in the Cora, CORA-CA, and DBLP-CA datasets, where retaining only 25% of the highest relationships (*retention 25%*) consistently results in better performance compared to retaining 50% or 75%. This outcome is surprising, as at the 25% level, only a small fraction of the higher-order relationships is preserved. In contrast, the *trimming* strategy shows the opposite trend, with performance declining as more higher-order relationships are removed. This phenomenon remains consistent across all models. Finally, when hyperedges are removed randomly, the performance consistently degrades as the percentage of removed hyperedges increases.

Rewiring connectivity

In this experiment, we preserve the original connectivity while investigating the influence of homophilic hyperedges on performance. To do so, we adjust the given connectivity in two different ways. The first strategy aims to unveil the full potential of the homophily measure for each dataset, by splitting the given hyperedges into fully homophilic ones based on their *node labels*. In contrast, the second strategy explores the possibility of partitioning hyperedges based on their *initial node features*. The hyperedge splitting results from applying multiple times k-means algorithm for each hyperedge e, varying at each iteration the number of centroids m from 2 to $\min(C, |e|)$; the elbow method is then used to determine the optimal hyperedge partitioning.

Results. As shown in Table 7.6, the Label Based strategy enhances performance for all datasets and models. Notably, the graph-based method CEGCN achieves similar results to HNNs with this strategy. Additionally, on average, only CEGCN performs better with the k-means strategy, and this method also mitigates distribution shifts for MultiSetMixer. These findings collectively suggest the crucial role of connectivity preprocessing, especially for graph-based models.

7.4.5 Benefits and challenges of hyperedge-dependent node representations in hypergraphs

In this section, we provide a deeper comparison between lifted models, such as AllSetTransformer, AllDeepSets, and EDHNN, and the MultiSetMixer, which allows for hyperedge-dependent node representation and thus leverages specific characteristics of hypergraph networks.

Table 7.1 highlights MultiSetMixer's superior performance on three datasets: NTU2012, ModelNet40, and 20Newsgroups. The improved results on 20Newsgroups can be attributed to the distribution shift and the pooling operations over restricted

Table 7.5. Drop connectivity. Test accuracy in % averaged over 15 splits.

Model	Type	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	Original	**77.11 ± 1.00**	70.67 ± 1.42	**89.04 ± 0.45**	**82.23 ± 1.46**	**91.34 ± 0.27**	99.96 ± 0.05	**86.49 ± 1.86**	96.70 ± 0.25	81.19 ± 0.49	89.10 ± 7.00	**2.85**
	Random 25%	76.65 ± 1.03	70.83 ± 1.70	88.87 ± 0.47	80.39 ± 1.51	90.36 ± 0.28	99.91 ± 0.09	85.49 ± 1.74	96.85 ± 0.26	81.15 ± 0.52	88.72 ± 5.97	4.05
	Random 50%	75.66 ± 1.18	70.70 ± 1.77	88.86 ± 0.41	77.97 ± 1.18	89.36 ± 0.25	99.93 ± 0.05	84.42 ± 1.75	**96.88 ± 0.27**	81.21 ± 0.39	84.49 ± 6.66	4.90
	Random 75%	74.96 ± 1.08	70.46 ± 1.66	88.76 ± 0.45	76.09 ± 1.27	87.68 ± 0.30	99.53 ± 0.13	81.70 ± 1.57	96.84 ± 0.27	81.31 ± 0.50	81.15 ± 6.88	7.50
	Retention 25%	76.87 ± 0.98	**70.96 ± 1.82**	88.94 ± 0.48	81.63 ± 1.26	90.93 ± 0.18	99.83 ± 0.09	85.13 ± 2.05	96.85 ± 0.26	81.09 ± 0.46	86.67 ± 7.26	3.65
	Retention 50%	75.80 ± 1.06	70.44 ± 1.63	88.84 ± 0.40	80.50 ± 1.38	90.55 ± 0.21	99.91 ± 0.09	82.25 ± 2.21	96.42 ± 0.23	81.04 ± 0.45	69.61 ± 9.28	6.75
	Retention 75%	75.52 ± 1.49	70.36 ± 1.71	88.78 ± 0.47	79.50 ± 1.09	89.48 ± 0.21	**99.97 ± 0.04**	78.85 ± 1.93	96.44 ± 0.27	81.19 ± 0.45	74.49 ± 9.97	6.85
	Trimming 25%	74.12 ± 1.30	70.95 ± 1.92	88.77 ± 0.45	74.87 ± 1.32	86.39 ± 0.31	99.85 ± 0.15	77.47 ± 1.67	96.18 ± 0.28	**81.61 ± 0.47**	**89.23 ± 8.11**	7.10
	Trimming 50%	75.24 ± 0.99	70.42 ± 1.62	88.87 ± 0.46	75.89 ± 1.53	87.14 ± 0.31	99.93 ± 0.06	82.76 ± 1.61	96.75 ± 0.23	81.47 ± 0.48	86.28 ± 8.73	6.15
	Trimming 75%	76.03 ± 1.39	70.86 ± 1.48	88.83 ± 0.48	77.50 ± 1.52	88.64 ± 0.27	99.93 ± 0.09	84.74 ± 1.81	96.82 ± 0.20	81.20 ± 0.48	86.03 ± 8.48	5.20
AllSetTransformer	Original	**79.54 ± 1.02**	72.52 ± 0.88	**88.74 ± 0.51**	**84.43 ± 1.14**	**91.61 ± 0.19**	99.95 ± 0.05	**88.22 ± 1.42**	**98.00 ± 0.12**	81.59 ± 0.59	91.03 ± 7.31	**1.95**
	Random 25%	79.11 ± 0.99	**72.75 ± 1.14**	88.67 ± 0.47	82.36 ± 1.38	90.61 ± 0.29	99.94 ± 0.09	87.50 ± 1.36	97.98 ± 0.17	**81.70 ± 0.52**	89.87 ± 7.66	3.10
	Random 50%	77.77 ± 1.34	72.21 ± 1.25	88.50 ± 0.45	79.73 ± 1.58	89.46 ± 0.27	99.96 ± 0.04	87.34 ± 1.55	97.83 ± 0.17	81.55 ± 0.66	89.49 ± 6.30	5.85
	Random 75%	76.92 ± 1.20	72.40 ± 1.22	88.54 ± 0.47	77.88 ± 1.74	87.73 ± 0.32	99.76 ± 0.15	86.31 ± 1.34	97.52 ± 0.20	81.46 ± 0.62	87.69 ± 6.09	8.10
	Retention 25%	79.19 ± 1.11	72.49 ± 0.86	88.73 ± 0.40	83.58 ± 1.30	91.18 ± 0.17	99.93 ± 0.09	87.21 ± 1.58	97.82 ± 0.17	81.63 ± 0.48	86.92 ± 7.18	4.10
	Retention 50%	78.16 ± 0.98	72.55 ± 1.13	88.70 ± 0.37	82.90 ± 1.15	90.80 ± 0.22	99.89 ± 0.18	86.67 ± 1.64	97.36 ± 0.21	81.61 ± 0.49	88.08 ± 7.51	5.20
	Retention 75%	77.38 ± 1.35	72.43 ± 0.98	88.71 ± 0.39	81.07 ± 1.20	89.83 ± 0.25	**99.97 ± 0.04**	85.58 ± 1.70	97.27 ± 0.22	81.58 ± 0.48	88.97 ± 6.91	5.80
	Trimming 25%	75.83 ± 1.31	72.39 ± 1.50	88.40 ± 0.45	76.51 ± 1.35	86.38 ± 0.32	99.84 ± 0.13	86.88 ± 1.66	97.10 ± 0.24	81.55 ± 0.55	**93.08 ± 7.79**	8.05
	Trimming 50%	77.37 ± 1.17	72.32 ± 1.30	88.49 ± 0.40	77.41 ± 1.73	87.03 ± 0.27	99.91 ± 0.12	86.86 ± 1.53	97.86 ± 0.21	81.45 ± 0.50	89.74 ± 8.53	7.50
	Trimming 75%	78.15 ± 1.11	72.67 ± 1.00	88.48 ± 0.39	78.91 ± 1.54	88.55 ± 0.26	99.92 ± 0.09	87.68 ± 1.56	97.90 ± 0.23	81.41 ± 0.61	91.03 ± 7.17	5.35
UniGCNII	Original	78.46 ± 1.14	**73.05 ± 1.48**	88.07 ± 0.47	**83.92 ± 1.02**	**91.56 ± 0.18**	99.89 ± 0.07	**88.24 ± 1.56**	97.84 ± 0.16	81.16 ± 0.49	89.61 ± 8.09	**2.25**
	Random 25%	78.31 ± 1.29	72.91 ± 1.24	88.09 ± 0.47	82.42 ± 1.05	90.86 ± 0.22	99.85 ± 0.10	87.82 ± 1.46	**97.87 ± 0.19**	81.08 ± 0.52	89.10 ± 7.76	3.55
	Random 50%	77.36 ± 1.34	72.54 ± 1.40	87.94 ± 0.52	80.17 ± 1.16	89.96 ± 0.24	99.85 ± 0.12	87.42 ± 1.44	97.77 ± 0.15	81.06 ± 0.55	87.31 ± 8.21	6.10
	Random 75%	76.70 ± 1.35	72.23 ± 1.81	87.91 ± 0.50	77.75 ± 1.31	88.54 ± 0.25	99.87 ± 0.09	87.56 ± 1.56	97.49 ± 0.18	81.06 ± 0.54	87.05 ± 6.50	7.30
	Retention 25%	**78.80 ± 0.92**	72.67 ± 1.24	**88.10 ± 0.51**	83.68 ± 0.96	91.26 ± 0.20	99.87 ± 0.06	87.56 ± 1.43	97.76 ± 0.16	81.01 ± 0.49	87.18 ± 7.58	4.05
	Retention 50%	77.18 ± 1.32	72.51 ± 1.54	88.02 ± 0.47	82.81 ± 1.32	90.99 ± 0.17	99.83 ± 0.08	87.16 ± 1.33	97.29 ± 0.17	80.82 ± 0.49	86.15 ± 8.49	7.15
	Retention 75%	76.63 ± 1.23	72.64 ± 1.15	88.07 ± 0.52	81.33 ± 1.27	90.17 ± 0.20	99.83 ± 0.14	86.71 ± 1.33	97.04 ± 0.16	80.87 ± 0.45	87.44 ± 7.49	7.20
	Trimming 25%	75.34 ± 1.26	72.68 ± 1.57	87.81 ± 0.47	76.18 ± 1.19	87.42 ± 0.30	99.87 ± 0.10	87.43 ± 1.53	97.00 ± 0.17	**81.50 ± 0.47**	**92.95 ± 8.15**	6.70
	Trimming 50%	76.75 ± 1.10	72.30 ± 1.64	87.87 ± 0.48	77.19 ± 1.42	88.00 ± 0.27	**99.90 ± 0.10**	87.47 ± 1.56	97.71 ± 0.18	81.22 ± 0.54	90.26 ± 7.40	6.15
	Trimming 75%	77.27 ± 1.08	72.69 ± 1.31	87.93 ± 0.52	78.68 ± 0.96	89.26 ± 0.28	99.84 ± 0.10	87.93 ± 1.69	97.86 ± 0.16	81.22 ± 0.45	90.64 ± 6.90	4.55
EDHNN	Original	**80.74 ± 1.00**	**73.22 ± 1.14**	**89.12 ± 0.47**	**85.17 ± 1.02**	**91.94 ± 0.23**	99.94 ± 0.11	**88.04 ± 1.65**	**97.70 ± 0.19**	**81.64 ± 0.49**	89.49 ± 6.99	**1.30**
	Random 25%	79.88 ± 1.17	72.03 ± 1.59	89.02 ± 0.33	80.54 ± 1.37	89.97 ± 0.28	99.64 ± 0.16	85.37 ± 1.51	97.19 ± 0.27	78.70 ± 0.49	89.23 ± 7.03	3.75
	Random 50%	78.45 ± 1.39	72.21 ± 1.60	88.81 ± 0.37	78.35 ± 1.52	88.77 ± 0.25	99.82 ± 0.16	84.19 ± 1.67	97.02 ± 0.24	74.37 ± 0.59	87.69 ± 6.18	6.00
	Random 75%	77.55 ± 1.60	72.33 ± 1.61	88.94 ± 0.35	77.29 ± 1.29	87.41 ± 0.24	99.45 ± 0.20	82.72 ± 1.43	96.44 ± 0.22	69.37 ± 0.61	80.13 ± 6.78	7.85
	Retention 25%	79.79 ± 1.31	71.97 ± 1.39	89.09 ± 0.44	81.57 ± 1.34	90.37 ± 0.22	99.93 ± 0.09	84.49 ± 1.51	97.03 ± 0.22	78.43 ± 0.58	86.28 ± 5.95	4.00
	Retention 50%	78.71 ± 1.44	72.26 ± 1.44	88.97 ± 0.36	80.26 ± 1.28	89.91 ± 0.27	99.97 ± 0.06	82.12 ± 1.91	96.40 ± 0.28	73.97 ± 0.53	69.10 ± 7.86	5.90
	Retention 75%	78.37 ± 1.45	72.60 ± 1.38	88.94 ± 0.41	78.86 ± 1.14	88.82 ± 0.18	**99.99 ± 0.03**	81.01 ± 1.69	96.18 ± 0.24	67.38 ± 0.76	73.97 ± 6.44	6.55
	Trimming 25%	77.03 ± 1.37	72.71 ± 1.45	88.63 ± 0.52	77.34 ± 1.46	87.00 ± 0.25	98.75 ± 0.46	82.00 ± 1.66	95.87 ± 0.30	78.58 ± 0.54	86.15 ± 9.07	8.00
	Trimming 50%	78.42 ± 1.33	72.49 ± 1.30	88.80 ± 0.38	78.10 ± 1.48	87.48 ± 0.26	99.74 ± 0.14	82.92 ± 1.35	96.83 ± 0.25	77.78 ± 0.62	83.59 ± 10.00	6.80
	Trimming 75%	79.10 ± 1.30	72.18 ± 1.34	88.82 ± 0.48	78.68 ± 1.32	88.44 ± 0.28	99.86 ± 0.11	85.04 ± 1.50	96.96 ± 0.28	78.70 ± 0.64	**89.74 ± 7.78**	4.85
CEGAT	Original	**76.53 ± 1.58**	71.58 ± 1.11	87.11 ± 0.49	77.50 ± 1.51	**88.74 ± 0.31**	96.81 ± 1.41	82.27 ± 1.60	92.79 ± 0.44	NA	44.62 ± 9.18	5.17
	Random 25%	75.88 ± 1.53	71.81 ± 1.05	87.03 ± 0.47	**78.00 ± 1.68**	87.58 ± 0.35	94.56 ± 2.09	82.03 ± 1.47	93.14 ± 0.34	NA	46.03 ± 9.01	5.94
	Random 50%	75.34 ± 1.52	71.86 ± 1.22	86.91 ± 0.48	76.92 ± 1.00	86.81 ± 0.32	95.14 ± 2.00	81.73 ± 1.44	93.62 ± 0.35	NA	47.69 ± 8.78	6.89
	Random 75%	75.26 ± 1.45	72.17 ± 1.59	87.02 ± 0.47	76.37 ± 1.26	85.79 ± 0.35	96.90 ± 1.40	82.66 ± 1.39	94.54 ± 0.43	NA	59.87 ± 8.55	5.22
	Retention 25%	75.96 ± 1.16	71.39 ± 1.33	87.13 ± 0.50	77.35 ± 1.52	88.48 ± 0.30	96.65 ± 1.49	80.39 ± 1.53	93.20 ± 0.45	NA	45.26 ± 9.40	6.39
	Retention 50%	75.36 ± 1.30	71.56 ± 1.27	87.16 ± 0.53	77.35 ± 1.52	88.14 ± 0.31	96.73 ± 1.59	80.56 ± 2.16	93.86 ± 0.41	NA	45.38 ± 9.97	6.00
	Retention 75%	75.02 ± 1.64	72.06 ± 1.41	87.22 ± 0.48	77.20 ± 1.47	87.54 ± 0.28	97.49 ± 0.89	81.82 ± 1.23	**94.94 ± 0.29**	NA	45.38 ± 9.22	5.06
	Trimming 25%	75.40 ± 1.45	**72.67 ± 1.76**	**87.68 ± 0.52**	76.14 ± 1.10	85.32 ± 0.42	99.72 ± 0.10	**84.94 ± 1.57**	94.42 ± 0.33	NA	**89.23 ± 7.38**	**3.67**
	Trimming 50%	75.90 ± 1.48	72.15 ± 1.62	87.41 ± 0.51	76.09 ± 1.65	85.69 ± 0.40	**99.80 ± 0.12**	82.04 ± 1.32	93.22 ± 0.38	NA	67.05 ± 7.87	4.44
	Trimming 75%	76.19 ± 1.68	71.82 ± 1.32	87.03 ± 0.56	76.04 ± 0.95	86.18 ± 0.38	99.31 ± 0.24	81.74 ± 1.48	92.79 ± 0.33	NA	53.33 ± 6.60	6.22
CEGCN	Original	**77.03 ± 1.31**	70.87 ± 1.19	87.01 ± 0.62	**77.55 ± 1.65**	**88.12 ± 0.25**	94.91 ± 0.44	80.90 ± 1.74	90.04 ± 0.47	NA	49.23 ± 6.81	4.61
	Random 25%	76.08 ± 1.55	71.35 ± 1.44	86.89 ± 0.59	76.51 ± 1.53	87.01 ± 0.39	93.11 ± 0.46	80.68 ± 1.86	90.36 ± 0.46	NA	49.74 ± 6.22	6.22
	Random 50%	75.55 ± 1.63	71.42 ± 1.60	86.70 ± 0.48	75.27 ± 1.22	86.24 ± 0.35	93.28 ± 0.61	80.63 ± 1.78	90.69 ± 0.54	NA	56.92 ± 7.24	6.33
	Random 75%	75.34 ± 1.62	71.73 ± 1.90	86.97 ± 0.51	74.53 ± 1.56	85.36 ± 0.26	93.01 ± 0.45	80.56 ± 1.76	91.91 ± 0.54	NA	63.20 ± 5.59	6.33
	Retention 25%	76.12 ± 1.58	70.87 ± 1.42	86.94 ± 0.56	76.98 ± 1.53	87.90 ± 0.29	94.94 ± 0.48	79.20 ± 1.42	90.59 ± 0.59	NA	49.87 ± 7.59	5.50
	Retention 50%	75.43 ± 1.28	70.83 ± 1.52	86.95 ± 0.54	76.87 ± 1.49	87.58 ± 0.28	94.97 ± 0.40	78.53 ± 1.90	90.09 ± 0.56	NA	45.77 ± 6.88	6.89
	Retention 75%	75.53 ± 1.25	71.72 ± 1.42	87.11 ± 0.53	76.36 ± 1.42	87.03 ± 0.28	94.74 ± 0.39	79.82 ± 1.41	**92.29 ± 0.46**	NA	40.38 ± 5.42	5.44
	Trimming 25%	75.58 ± 1.56	**72.26 ± 1.52**	**87.36 ± 0.51**	74.84 ± 1.31	84.97 ± 0.31	**99.60 ± 0.11**	**83.10 ± 1.69**	91.85 ± 0.42	NA	**87.69 ± 7.31**	**3.44**
	Trimming 50%	76.57 ± 1.47	71.81 ± 1.44	87.07 ± 0.55	74.66 ± 1.68	85.24 ± 0.33	99.54 ± 0.18	80.72 ± 1.64	90.64 ± 0.54	NA	71.28 ± 6.60	4.00
	Trimming 75%	76.53 ± 1.50	71.45 ± 1.45	86.75 ± 0.54	74.56 ± 1.32	85.56 ± 0.33	99.14 ± 0.23	80.38 ± 1.91	90.06 ± 0.37	NA	58.46 ± 7.17	6.22
HCHA	Original	**79.53 ± 1.33**	**72.57 ± 1.06**	**86.97 ± 0.55**	**83.53 ± 1.12**	**91.21 ± 0.28**	98.94 ± 0.54	**86.60 ± 1.96**	94.50 ± 0.33	**80.75 ± 0.53**	89.23 ± 6.81	**2.40**
	Random 25%	78.74 ± 1.30	72.33 ± 1.28	86.84 ± 0.56	81.98 ± 1.34	90.09 ± 0.35	98.55 ± 0.55	85.94 ± 1.76	94.78 ± 0.28	80.16 ± 0.46	89.10 ± 6.71	4.30
	Random 50%	77.65 ± 1.46	72.11 ± 1.42	86.67 ± 0.48	79.23 ± 1.41	88.88 ± 0.35	98.61 ± 0.48	85.32 ± 1.75	95.17 ± 0.28	79.68 ± 0.50	87.56 ± 6.97	6.60
	Random 75%	76.56 ± 1.60	72.23 ± 1.33	86.72 ± 0.56	77.11 ± 1.28	87.07 ± 0.34	98.59 ± 0.76	84.88 ± 1.44	**95.57 ± 0.34**	79.49 ± 0.43	82.18 ± 6.58	7.30
	Retention 25%	79.09 ± 1.25	72.29 ± 1.17	86.95 ± 0.52	83.06 ± 1.09	90.63 ± 0.25	98.82 ± 0.50	85.56 ± 1.66	94.73 ± 0.34	80.27 ± 0.44	86.03 ± 5.20	3.85
	Retention 50%	77.77 ± 1.38	72.20 ± 1.20	86.82 ± 0.48	82.16 ± 1.27	90.17 ± 0.29	98.22 ± 0.28	84.80 ± 1.79	94.54 ± 0.22	79.96 ± 0.44	75.77 ± 6.86	6.70
	Retention 75%	77.05 ± 1.53	72.37 ± 1.20	86.79 ± 0.47	80.79 ± 0.95	88.97 ± 0.24	97.62 ± 0.30	84.35 ± 1.65	95.21 ± 0.30	79.40 ± 0.52	84.36 ± 6.77	6.70
	Trimming 25%	75.68 ± 1.21	72.15 ± 1.72	86.83 ± 0.47	75.94 ± 1.35	85.85 ± 0.40	**99.87 ± 0.11**	85.60 ± 1.92	95.02 ± 0.26	80.66 ± 0.56	**91.92 ± 7.10**	5.50
	Trimming 50%	77.76 ± 1.28	72.10 ± 1.49	86.96 ± 0.48	77.26 ± 1.17	86.57 ± 0.36	99.84 ± 0.11	84.58 ± 1.37	94.73 ± 0.28	80.18 ± 0.60	83.08 ± 8.72	6.45
	Trimming 75%	78.38 ± 1.30	72.22 ± 1.12	86.81 ± 0.53	78.82 ± 0.95	88.09 ± 0.27	99.73 ± 0.18	86.05 ± 1.60	94.61 ± 0.27	80.02 ± 0.57	89.36 ± 8.49	5.20
HGNN	Original	**79.53 ± 1.33**	72.24 ± 1.08	**86.97 ± 0.55**	**83.45 ± 1.22**	**91.26 ± 0.26**	98.94 ± 0.54	**86.71 ± 1.48**	94.50 ± 0.33	**80.75 ± 0.52**	89.23 ± 6.81	**2.50**
	Random 25%	78.74 ± 1.30	72.15 ± 1.36	86.84 ± 0.56	81.94 ± 1.31	90.11 ± 0.34	98.55 ± 0.55	85.82 ± 1.65	94.78 ± 0.28	80.16 ± 0.43	89.10 ± 6.71	4.70
	Random 50%	77.65 ± 1.46	72.20 ± 1.62	86.67 ± 0.48	79.20 ± 1.48	88.84 ± 0.42	98.61 ± 0.48	85.59 ± 1.49	95.17 ± 0.28	79.68 ± 0.50	87.56 ± 6.97	5.95
	Random 75%	76.56 ± 1.60	72.16 ± 1.56	86.72 ± 0.56	77.03 ± 1.37	86.95 ± 0.34	98.59 ± 0.76	85.12 ± 1.27	**95.57 ± 0.34**	79.50 ± 0.42	82.18 ± 6.58	7.30
	Retention 25%	79.09 ± 1.25	72.13 ± 1.17	86.95 ± 0.52	83.11 ± 1.09	90.66 ± 0.23	98.82 ± 0.50	85.33 ± 1.52	94.73 ± 0.34	80.23 ± 0.44	86.03 ± 5.20	4.25
	Retention 50%	77.77 ± 1.38	72.20 ± 1.32	86.82 ± 0.48	82.20 ± 1.29	90.21 ± 0.27	98.22 ± 0.28	84.51 ± 1.77	94.54 ± 0.22	79.93 ± 0.46	75.77 ± 6.86	6.45
	Retention 75%	77.05 ± 1.53	**72.35 ± 1.40**	86.79 ± 0.47	80.88 ± 0.93	89.02 ± 0.23	97.62 ± 0.30	84.19 ± 1.49	95.21 ± 0.30	79.36 ± 0.54	84.36 ± 6.77	6.60
	Trimming 25%	75.68 ± 1.21	71.91 ± 1.61	86.83 ± 0.47	75.73 ± 1.44	85.78 ± 0.41	**99.87 ± 0.11**	85.89 ± 1.67	95.02 ± 0.26	80.66 ± 0.56	**91.92 ± 7.10**	5.55
	Trimming 50%	77.76 ± 1.28	71.91 ± 1.50	86.96 ± 0.48	77.29 ± 1.18	86.48 ± 0.34	99.84 ± 0.11	84.67 ± 1.43	94.73 ± 0.28	80.18 ± 0.60	83.08 ± 8.72	6.30
	Trimming 75%	78.38 ± 1.30	72.07 ± 1.25	86.81 ± 0.53	78.78 ± 1.04	87.99 ± 0.34	99.73 ± 0.18	86.00 ± 1.55	94.61 ± 0.27	80.02 ± 0.57	89.36 ± 8.49	5.40
HyperGCN	Original	**74.78 ± 1.11**	**66.06 ± 1.58**	**82.32 ± 0.62**	**77.48 ± 1.14**	**86.07 ± 3.32**	69.51 ± 4.98	47.65 ± 5.01	46.10 ± 7.95	**80.84 ± 0.49**	51.54 ± 9.88	**3.40**
	Random 25%	35.60 ± 1.76	34.71 ± 1.62	68.80 ± 0.62	55.31 ± 1.83	81.18 ± 0.39	**69.61 ± 4.77**	57.42 ± 3.22	47.78 ± 7.33	77.50 ± 0.54	51.41 ± 9.82	6.05
	Random 50%	33.75 ± 2.58	39.94 ± 1.72	69.37 ± 0.59	40.11 ± 1.97	67.36 ± 2.94	67.59 ± 6.63	49.36 ± 3.42	48.12 ± 5.98	71.74 ± 0.58	51.67 ± 9.40	6.80
	Random 75%	42.42 ± 2.51	49.31 ± 1.85	70.99 ± 0.65	37.25 ± 1.94	50.33 ± 0.74	66.01 ± 8.15	45.31 ± 3.01	49.08 ± 2.52	62.76 ± 0.73	51.92 ± 9.02	6.60
	Retention 25%	37.56 ± 1.65	35.87 ± 1.80	68.73 ± 0.53	63.64 ± 1.22	84.26 ± 0.32	**69.61 ± 4.81**	61.33 ± 2.63	72.36 ± 3.39	79.24 ± 0.48	51.54 ± 8.84	4.55
	Retention 50%	34.87 ± 2.14	37.98 ± 1.70	69.04 ± 0.53	56.45 ± 1.70	77.98 ± 0.36	69.58 ± 4.75	76.59 ± 2.60	81.69 ± 1.75	75.60 ± 0.57	51.54 ± 9.45	4.70
	Retention 75%	36.71 ± 1.95	44.39 ± 1.69	69.98 ± 0.52	45.09 ± 2.09	63.78 ± 3.04	69.20 ± 5.16	77.44 ± 3.62	**84.44 ± 2.23**	67.99 ± 0.51	**52.18 ± 8.61**	4.60
	Trimming 25%	50.59 ± 1.72	55.15 ± 1.57	74.16 ± 0.66	52.78 ± 1.99	68.13 ± 0.79	52.37 ± 1.41	**79.05 ± 2.74**	81.31 ± 4.34	73.13 ± 0.92	46.67 ± 21.96	4.60
	Trimming 50%	36.20 ± 2.74	44.47 ± 1.38	71.35 ± 0.58	39.84 ± 2.35	55.53 ± 0.45	54.57 ± 7.40	59.52 ± 1.81	65.29 ± 3.52	68.07 ± 1.26	51.03 ± 10.20	6.70
	Trimming 75%	34.73 ± 1.52	36.60 ± 1.89	69.59 ± 0.54	37.81 ± 1.86	61.89 ± 1.82	61.73 ± 3.19	65.10 ± 2.77	73.05 ± 1.78	73.42 ± 0.63	50.90 ± 11.14	7.00
MultiSetMixer	Original	**78.06 ± 1.24**	71.85 ± 1.50	87.19 ± 0.53	**82.74 ± 1.23**	**90.68 ± 0.19**	99.58 ± 0.16	**88.90 ± 1.30**	**98.38 ± 0.21**	**88.57 ± 1.96**	88.08 ± 8.04	**2.65**
	Random 25%	77.73 ± 1.24	71.73 ± 1.73	86.98 ± 1.02	81.14 ± 1.19	89.74 ± 0.21	99.42 ± 0.30	88.00 ± 1.25	97.83 ± 0.20	80.84 ± 1.56	88.46 ± 7.00	4.75
	Random 50%	76.76 ± 1.28	71.69 ± 1.75	87.29 ± 0.63	78.32 ± 1.20	88.55 ± 0.31	99.43 ± 0.35	86.68 ± 1.55	97.46 ± 0.22	77.28 ± 1.16	86.33 ± 7.05	6.40
	Random 75%	76.06 ± 1.28	72.11 ± 1.81	87.31 ± 0.51	76.54 ± 1.41	86.32 ± 0.33	99.42 ± 0.48	85.50 ± 1.91	96.89 ± 0.26	77.35 ± 0.65	86.28 ± 8.14	7.25
	Retention 25%	77.88 ± 1.13	71.33 ± 1.43	86.86 ± 0.96	82.27 ± 1.40	90.30 ± 0.26	99.67 ± 0.18	87.24 ± 1.76	97.82 ± 0.27	87.12 ± 1.36	86.77 ± 8.32	4.40
	Retention 50%	76.86 ± 1.20	71.49 ± 1.76	87.02 ± 0.99	81.15 ± 1.06	89.95 ± 0.24	99.56 ± 0.30	86.11 ± 1.93	96.88 ± 0.27	85.15 ± 1.42	**89.12 ± 7.59**	5.10
	Retention 75%	76.29 ± 1.62	71.23 ± 2.00	87.16 ± 0.77	79.59 ± 1.05	88.75 ± 0.25	98.84 ± 1.00	85.16 ± 1.35	96.76 ± 0.32	83.87 ± 1.27	88.80 ± 5.97	7.00
	Trimming 25%	75.07 ± 1.44	**72.41 ± 1.61**	87.19 ± 0.56	75.84 ± 1.35	85.86 ± 0.31	**99.95 ± 0.07**	84.61 ± 1.47	96.68 ± 0.25	79.29 ± 0.55	88.85 ± 8.23	6.55
	Trimming 50%	76.34 ± 1.33	72.17 ± 1.46	**87.42 ± 0.50**	76.97 ± 1.50	86.46 ± 0.37	99.75 ± 0.13	85.29 ± 2.04	97.47 ± 0.26	74.42 ± 0.68	86.92 ± 8.24	5.70
	Trimming 75%	77.11 ± 1.37	71.76 ± 1.56	87.35 ± 0.56	77.80 ± 0.94	88.16 ± 0.25	99.59 ± 0.20	87.19 ± 1.58	97.73 ± 0.21	67.00 ± 1.22	88.43 ± 8.07	5.20

neighbors within hyperedges (see Section 7.4.3). However, this same distribution shift negatively impacts MultiSetMixer's performance on the Zoo, Mushroom, and Pubmed datasets. As shown in Section 7.4.4, this distribution shift can be mitigated by modifying the initial connectivity. Specifically, Table 7.6 demonstrates

Table 7.6. Rewiring connectivity. Test accuracy in % averaged over 15 splits.

Model	Type	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	Label Based	**82.24 ± 1.12**	**75.65 ± 1.57**	**90.49 ± 0.40**	**91.12 ± 0.92**	**96.59 ± 0.17**	**99.96 ± 0.04**	**93.13 ± 1.29**	**99.52 ± 0.11**	**99.79 ± 0.13**	**91.54 ± 7.24**	**1.05**
	k-means	75.20 ± 1.11	70.87 ± 1.54	88.96 ± 0.48	79.59 ± 1.42	89.75 ± 0.25	99.94 ± 0.09	84.23 ± 1.50	97.17 ± 0.13	81.18 ± 0.54	86.92 ± 7.73	2.80
	Original	77.11 ± 1.00	70.67 ± 1.42	89.04 ± 0.45	82.23 ± 1.46	91.34 ± 0.27	**99.96 ± 0.05**	86.49 ± 1.86	96.70 ± 0.25	81.19 ± 0.49	89.10 ± 7.00	2.15
AllSetTransformer	Label Based	**83.43 ± 1.36**	**76.45 ± 1.43**	**90.19 ± 0.42**	**91.71 ± 0.89**	**96.75 ± 0.16**	**99.96 ± 0.05**	**94.81 ± 1.04**	**99.68 ± 0.09**	**99.93 ± 0.03**	**94.10 ± 6.91**	**1.05**
	k-means	77.14 ± 1.46	72.83 ± 1.07	88.60 ± 0.41	81.92 ± 1.35	89.79 ± 0.30	**99.96 ± 0.06**	87.95 ± 1.28	97.29 ± 0.20	81.58 ± 0.55	88.72 ± 7.69	2.75
	Original	79.54 ± 1.02	72.52 ± 0.88	88.74 ± 0.51	84.43 ± 1.14	91.61 ± 0.19	99.95 ± 0.05	88.22 ± 1.42	98.00 ± 0.12	81.59 ± 0.59	91.03 ± 7.31	2.20
UniGCNII	Label Based	**82.12 ± 1.11**	**75.23 ± 1.64**	**89.18 ± 0.50**	**89.80 ± 0.95**	**94.78 ± 0.13**	**99.93 ± 0.07**	**92.87 ± 1.32**	**99.31 ± 0.10**	**99.70 ± 0.10**	**94.74 ± 6.35**	**1.00**
	k-means	76.49 ± 1.01	72.73 ± 1.50	88.02 ± 0.48	81.13 ± 1.41	90.13 ± 0.26	99.88 ± 0.07	88.05 ± 1.78	97.10 ± 0.21	81.06 ± 0.48	89.23 ± 7.52	3.00
	Original	78.46 ± 1.14	73.05 ± 1.48	88.07 ± 0.47	83.92 ± 1.02	91.56 ± 0.18	99.89 ± 0.07	88.24 ± 1.56	97.84 ± 0.16	81.16 ± 0.49	89.61 ± 8.09	2.00
EDHNN	Label Based	**84.51 ± 1.23**	**76.76 ± 1.51**	**90.63 ± 0.53**	**92.28 ± 0.86**	**97.35 ± 0.15**	**99.96 ± 0.09**	**93.64 ± 1.12**	**99.59 ± 0.09**	**99.88 ± 0.08**	**92.44 ± 8.72**	**1.00**
	k-means	78.43 ± 1.08	73.21 ± 1.25	88.98 ± 0.43	82.99 ± 1.33	90.45 ± 0.25	99.94 ± 0.08	86.91 ± 1.51	96.83 ± 0.16	81.34 ± 0.55	86.54 ± 7.68	2.95
	Original	80.74 ± 1.00	73.22 ± 1.14	89.12 ± 0.47	85.17 ± 1.02	91.94 ± 0.23	99.94 ± 0.11	88.04 ± 1.65	97.70 ± 0.19	81.64 ± 0.49	89.49 ± 6.99	2.05
CEGAT	Label Based	**83.05 ± 1.08**	**77.82 ± 1.59**	**90.25 ± 0.39**	**91.42 ± 0.88**	**96.25 ± 0.13**	**99.91 ± 0.07**	**94.23 ± 0.77**	**99.26 ± 0.14**	OOM	**93.85 ± 7.39**	**1.00**
	k-means	75.45 ± 1.54	72.57 ± 1.12	87.32 ± 0.47	77.11 ± 1.51	87.27 ± 0.29	97.66 ± 0.72	85.48 ± 1.66	96.39 ± 0.26	OOM	68.08 ± 8.28	2.33
	Original	76.53 ± 1.58	71.58 ± 1.11	87.11 ± 0.49	77.50 ± 1.51	88.74 ± 0.31	96.81 ± 1.41	82.27 ± 1.60	92.79 ± 0.44	OOM	44.62 ± 9.18	2.67
CEGCN	Label Based	**83.70 ± 1.02**	**77.50 ± 1.53**	**90.08 ± 0.42**	**91.28 ± 0.97**	**96.68 ± 0.14**	**99.95 ± 0.05**	**94.03 ± 1.24**	**99.30 ± 0.14**	OOM	**95.00 ± 7.08**	**1.00**
	k-means	75.89 ± 1.53	72.07 ± 1.18	87.13 ± 0.51	76.43 ± 1.41	86.76 ± 0.24	94.84 ± 0.47	85.34 ± 1.71	95.77 ± 0.31	OOM	73.72 ± 7.89	2.44
	Original	77.03 ± 1.31	70.87 ± 1.19	87.01 ± 0.62	77.55 ± 1.65	88.12 ± 0.25	94.91 ± 0.44	80.90 ± 1.74	90.04 ± 0.47	OOM	49.23 ± 6.81	2.56
HCHA	Label Based	**84.06 ± 1.08**	**77.12 ± 1.37**	**88.81 ± 0.43**	**92.77 ± 0.73**	**96.70 ± 0.12**	**99.96 ± 0.06**	**95.21 ± 1.27**	**99.67 ± 0.09**	**99.93 ± 0.04**	**94.61 ± 6.97**	**1.00**
	k-means	77.51 ± 1.41	72.62 ± 1.33	86.89 ± 0.48	81.19 ± 1.31	89.42 ± 0.29	99.56 ± 0.27	87.62 ± 1.33	96.98 ± 0.15	80.58 ± 0.57	84.10 ± 9.83	2.60
	Original	79.53 ± 1.33	72.57 ± 1.06	86.97 ± 0.55	83.53 ± 1.12	91.21 ± 0.28	98.94 ± 0.54	86.60 ± 1.96	94.50 ± 0.33	80.75 ± 0.53	89.23 ± 6.81	2.40
HGNN	Label Based	**84.06 ± 1.08**	**77.11 ± 1.47**	**88.81 ± 0.43**	**92.86 ± 0.65**	**96.70 ± 0.11**	**99.96 ± 0.06**	**95.34 ± 1.07**	**99.67 ± 0.09**	**99.93 ± 0.04**	**94.61 ± 6.97**	**1.00**
	k-means	77.51 ± 1.41	72.41 ± 1.55	86.89 ± 0.48	81.19 ± 1.38	89.42 ± 0.27	99.56 ± 0.27	87.52 ± 1.51	96.98 ± 0.15	80.58 ± 0.57	84.10 ± 9.83	2.60
	Original	79.53 ± 1.33	72.24 ± 1.08	86.97 ± 0.55	83.45 ± 1.22	91.26 ± 0.26	98.94 ± 0.54	86.71 ± 1.48	94.50 ± 0.33	80.75 ± 0.52	89.23 ± 6.81	2.40
HyperGCN	Label Based	72.88 ± 1.23	**66.10 ± 1.79**	82.18 ± 0.62	76.20 ± 1.50	84.86 ± 0.39	**69.68 ± 4.90**	43.37 ± 4.65	**47.19 ± 6.42**	**82.14 ± 0.43**	**53.97 ± 8.24**	**1.50**
	k-means	45.76 ± 1.97	49.96 ± 1.68	77.97 ± 0.75	47.63 ± 1.36	40.88 ± 4.04	69.53 ± 4.91	32.21 ± 2.57	41.96 ± 2.40	80.85 ± 0.46	53.46 ± 8.65	2.70
	Original	**74.78 ± 1.11**	66.06 ± 1.58	**82.32 ± 0.62**	**77.48 ± 1.14**	**86.07 ± 3.32**	69.51 ± 4.98	**47.65 ± 5.01**	46.10 ± 7.95	80.84 ± 0.49	51.54 ± 9.88	1.80
MultiSetMixer	Label Based	**83.31 ± 1.09**	**74.98 ± 1.45**	**89.42 ± 0.56**	**92.34 ± 1.09**	**97.37 ± 0.20**	**99.98 ± 0.03**	**93.89 ± 1.34**	**99.49 ± 0.10**	**99.87 ± 0.06**	**91.28 ± 7.35**	**1.00**
	kmeans based	76.02 ± 1.33	72.47 ± 1.50	87.43 ± 0.50	79.80 ± 1.50	88.38 ± 0.33	99.92 ± 0.08	87.55 ± 1.21	97.03 ± 0.19	80.54 ± 0.57	88.85 ± 8.65	2.60
	Original	78.06 ± 1.24	71.85 ± 1.50	87.19 ± 0.53	82.74 ± 1.23	90.68 ± 0.19	99.58 ± 0.16	88.90 ± 1.30	98.38 ± 0.21	88.57 ± 1.96	88.08 ± 8.04	2.40
MLP CB	Label Based	**74.54 ± 1.51**	**72.41 ± 1.47**	**86.02 ± 0.50**	**74.71 ± 1.16**	84.88 ± 0.38	**99.97 ± 0.06**	85.94 ± 1.59	96.38 ± 0.32	81.09 ± 0.52	**87.56 ± 7.33**	**1.45**
	k-means	74.53 ± 1.34	72.23 ± 1.55	85.99 ± 0.39	74.46 ± 1.32	84.78 ± 0.35	99.92 ± 0.07	**86.16 ± 1.54**	96.31 ± 0.27	81.09 ± 0.55	87.44 ± 7.75	2.25
	Original	74.06 ± 1.26	71.93 ± 1.53	85.83 ± 0.51	74.39 ± 1.40	**84.91 ± 0.44**	99.93 ± 0.08	85.43 ± 1.51	**96.41 ± 0.32**	**86.13 ± 2.82**	81.61 ± 10.98	2.30

that applying k-means clustering to adjust the initial connectivity allows MultiSetMixer to achieve better performance on Zoo, Mushroom, and Pubmed. In the case of Mushroom, this approach enables performance comparable to models like AllSetTransformer, AllDeepSets, and EDHNN. At the same time, modifying the connectivity on 20Newsgroups mitigates the distribution shift but results in a performance decline, bringing MultiSetMixer's results closer to those of AllSetTransformer, AllDeepSets, and EDHNN.

Additionally, we observed that MultiSetMixer excels in processing Computer Vision/Graphics datasets such as NTU2012 and ModelNet40. This success is due to the initial graph construction, which involves lifting the k-uniform graph by constructing hyperedges based on the one-hop neighborhood of each node. On the Cora and Citeseer datasets, MultiSetMixer outperforms AllDeepSets and performs comparably to AllSetTransformer, all without requiring an attention mechanism. However, on CORA-CA, AllSetTransformer outperforms MultiSetMixer, which instead produces results similar to AllDeepSets.

Finally, MultiSetMixer's lower performance on Pubmed and DBLP-CA can be attributed to the difficulty of processing the entire hypergraph in a single forward pass due to memory constraints when storing all hyperedge-dependent node representations. However, employing the proposed mini-batching scheme still allows MultiSetMixer to achieve strong performance on these datasets.

At the conclusion of this section, we summarize the main benefits and challenges of hyperedge-dependent node representations in hypergraphs. The key advantage lies in their ability to model and capture complex interactions within hypergraph structures, as demonstrated by their superior performance on certain datasets, especially those involving computer vision and large-scale hyperedges. However, challenges arise in managing the scalability of these models, particularly in scenarios with large hyperedges or datasets, where memory constraints and distribution shifts can negatively impact performance. Despite these issues, careful modifications to the connectivity and the use of mini-batching strategies can mitigate some of the

limitations, offering a promising path forward for further optimization.

7.5 Related Work

Homophily in hypergraphs. Homophily measures are typically defined only for pairwise relationships. In the context of Graph Neural Networks (GNNs), many of the current models implicitly use the homophily assumption, which is shown to be crucial for achieving a robust performance with relational data [263, 47, 94]. Nevertheless, despite the pivotal role that homophily plays in graph representation learning, its hypergraph counterpart mainly remains unexplored. In fact, to the best of our knowledge, Veldt et al. [213] is the only work that faces the challenge of defining homophily for higher-order networks. This work introduces a framework in which homophily is quantified through group interactions, measuring the distribution of classes among hyperedges. However, their definition of homophily is restricted to uniform hypergraphs –where all hyperedges have exactly the same size–, which hinders its practical application to a great extent. Clique-expanded (CE) homophily, as used in Wang et al. [222], is calculated by determining node homophily [163] on the graph derived from the clique-expanded hypergraph. Thus, CE homophily is not directly defined on the hypergraph but requires a clique expansion. It is important to note that this expansion is a non-invertible transformation, which leads to a loss of information.

Hypergraph Neural Networks. Numerous machine-learning techniques have been developed for hypergraph data processing. A common early approach is transforming hypergraphs into graphs via clique expansion (CE), where each hyperedge is replaced with edges between all pairs of vertices in the hyperedge, enabling graph-based algorithm analysis [2, 262, 261, 137].

Hypergraph Neural Networks (HNNs) have also been applied to semi-supervised learning. One early method extends graph convolution by incorporating the normalized hypergraph Laplacian [71], with weighted CE as spectral convolution [64]. HyperGCN [243] uses mediators for a reduced CE, lowering the number of edges required to represent a hyperedge, and applying spectral convolution for information diffusion. Hypergraph Convolution and Hypergraph Attention (HCHA) [10] introduces modified normalizations and attention weights dependent on node and hyperedge features.

However, CE can result in the loss of structural information, leading to suboptimal performance [99, 49]. These models often perform best with shallow architectures, while deeper layers can cause oversmoothing [106]. Recent work has tried to mitigate oversmoothing with residual connections but still relies on hypergraph Laplacians and clique expansion [43]. Another method, line expansion (LE), treats vertices and hyperedges equally and models vertex-hyperedge pairs to induce a homogeneous structure, though both CE and LE demand significant computational resources [245].

Another research line focuses on two-stage message passing: nodes communicate with hyperedges, which then relay information back to the nodes [227, 252, 64, 6, 106, 244]. HyperSAGE [6] exemplifies this approach, offering improvements over

spectral methods but limited by a single learnable transformation and inefficient computation due to nested loops [49].

The work of Chien et al. [49] introduced AllSet, a general framework that describes HNNs through the composition of two learnable permutation invariant functions, defining a two-step message passing based mechanism –from nodes to hyperedges, then back from hyperedges to nodes. In particular, AllSet is shown to generalize most commonly used HNNs, including all clique expansion based (CE) methods, HNN [71], HNHN [64], HCHA [10], HyperSAGE [6] and HyperGCN [243]. Chien et al. [49] also proposes two novel AllSet-like learnable layers: the first one –AllDeepSet– exploits Deep Set [259], and the second one –AllSetTransformer– Set Transformer [134], both of them achieving state-of-the-art results in the most common hypergraph benchmarking datasets. Concurrent to AllSet, Huang and Yang [106] also aimed at designing a common framework for graph and hypergraph NNs, and its more advanced UniGCNII method leverages initial residual connections and identity mappings in the hyperedge-to-node propagation to address over-smoothing issues; notably, UniGCNII does not fall under the AllSet framework due to these residual connections. Likewise, the more recent EDHNN model [222] also goes beyond this framework by incorporating hyperedge-dependent messages from hyperedges to nodes, a step closer to the hyperedge-dependent node representations that we propose in this work.

Recently, Choe et al. [51] introduced WHATsNet, the first model capable of producing hyperedge-dependent node representations. In particular, WHATsNet model is a particular instance of the MultiSet framework introduced in our work (see Section 7.3.2), but its architecture is specifically designed to perform hyperedge-dependent node classification tasks –where each node has as many labels as the number of hyperedges it belongs to. This prevents benchmarking this model in our evaluation, where we focus on standard node-level classification tasks. Please refer to Appendix C.1 for an extended literature review.

7.6 Discussion

This section summarizes key findings from our extensive evaluation and proposed frameworks. Here we recap the question and summarize the answers to these questions revealed by this work.

Q1: **Can the concept of homophily play a crucial role in HNNs, similar to its significance in graph-based research?** We show that the concept of homophily in higher-order networks is considerably more complicated compared to networks that exhibit only pairwise connections. To address the issue, we introduce a novel message passing homophily framework that is capable of characterizing homophily in hypergraphs through the distribution of node features as well as node class distribution. In Section 7.2, we present Δ homophily, based on the dynamic nature of the proposed message passing homophily, showing that it correlates better with HNN models' performance than classical homophily measures over the clique-expanded hypergraph. Our findings underscore the crucial role of accurately expressing homophily in HNNs, emphasizing its complexity and the potential in capturing higher-order dynamics. Moreover, in our experiments (see

Section 7.4.4), we demonstrate that rewiring hyperedges for perfect homophily leads to similar results for graph-based methods (CEGCN, CEGAT) and HNN models. In conclusion, the proposed framework offers a robust foundation for exploring homophily in higher-order networks and can be seamlessly adapted to incorporate other characteristics, such as the distribution of node features, which parallel the node label homophily examined in this study. While we presented this framework in the context of hypergraphs, its adaptability allows for straightforward extensions to other topological domains.

Q2: **Given that current HNNs are predominantly extensions of GNN architectures adapted to the hypergraph domain, are these extended methodologies suitable, or should we explore new strategies tailored specifically for handling hypergraph-based data?** The three main contributions presented in this work —Message Passing Homophily, the MultiSet framework that integrates most existing HNNs with a comparison of their strengths and weaknesses across different settings, and the formulation of a novel mini-batch sampling scheme to address scalability issues—are directly inspired by the inherent properties of hypernetworks and their higher-order dynamics. Based on our experimental results and analysis, the proposed methodologies open an interesting discussion about the impact of ways of processing hypergraph data and defining HNNs. For instance, our mini-batching sampling strategy –which helps address scalability issues of current solutions– allowed us to realize the implicit introduction of node class distribution shifts in the process. This study could potentially lead to the definition of meaningful connectivity rewiring techniques, as we already explore in Section 7.4.4. Furthermore, we show that the introduced message passing and Δ homophily measures allow for a deeper understanding of the hypernetwork topology and its correlation to the HNN models' performances. Overall, this study focuses on comparing graph-based extensions for modeling hypergraph networks and methodologies that incorporate hypergraph-specific characteristics. This approach allows for identifying common failure modes in current hypergraph modeling techniques (Section 7.4). We argue that these contributions offer a new *perspective* on processing higher-order networks that extend beyond the graph domain.

Q3: **Are the existing hypergraph benchmarking datasets meaningful and representative enough to draw robust and valid conclusions?** In Section 7.4.4, we demonstrate that the significant performance gap of HNNs models on Cora, CORA-CA, and DBLP-CA is primarily attributed to the hyperedges with the largest cardinalities. Further analysis using Δ homophily reveals that their notable improvement is strongly tied to the homophilic nature of the one-hop neighborhood. Additionally, the experimental results in Section 7.4.1 and 7.4.4 highlight challenges for current HNNs with certain benchmark hypergraph datasets. Specifically, we find that lifted HNN models ignore connectivity for Citeseer, Pubmed, and 20Newsgroups, as well as for the Mushroom dataset, due to highly discriminative features. Furthermore, we observe that models that do not rely on inductive bias (i.e. do not use connectivity in the architecture), consistently exhibit good performance across the majority of datasets. This suggests that the expressive power of node features alone is sufficient for efficient task execution. Addressing this gap presents an open challenge for future research endeavors, and we posit the necessity for additional benchmark datasets where connectivity plays a pivotal role. In addition to this,

we believe it would be also interesting to analyze datasets involving higher-order relationships where node classes explicitly depend on hyperedges, as introduced in Choe et al. [51]. This represents an insightful line of research to further exploit hyperedge-based node representations.

Chapter 8

TopoBenchmark: A Framework for Benchmarking Topological Deep Learning

Contents

This chapter is organized as follows. In Section 8.1, we discuss and motivate the need for the benchmarking tool in TDL. Section 8.2 outlines existing software for graph-based learning and benchmarking, as well as Geometric Deep Learning (GDL). Section 8.3 revises the notion of featured cell complex and formalizes the lifting of a graph to a cell complex. Section 8.4 introduces the main components

The content of this chapter is adapted from the material submitted to Data-centric Machine Learning Research [201] and the Section 8.6 is adapted from the material published in Proceedings of Machine Learning Research through the GRaM Workshop at ICML 2024 [26]

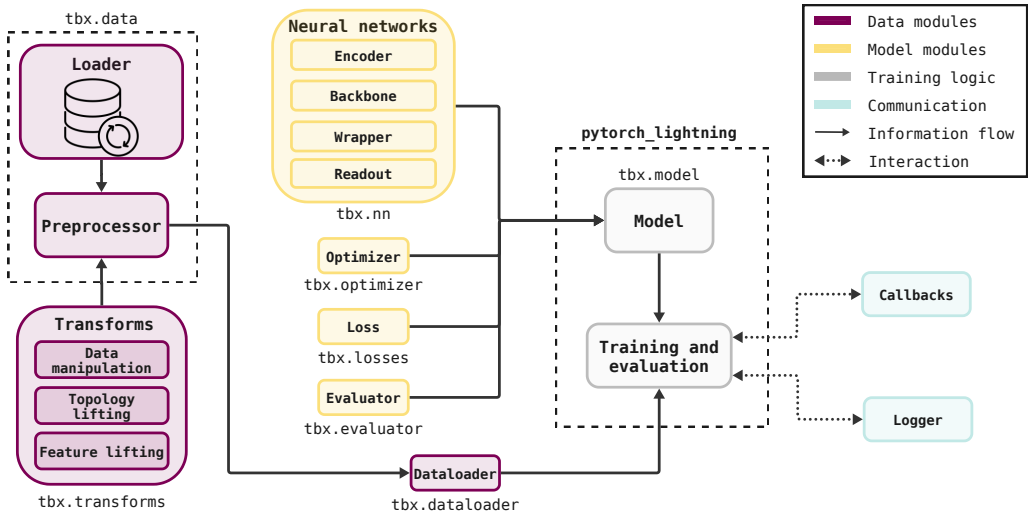


Figure 8.1. Workflow of `TopoBenchmark`, which consists of four main components, organized by their functionality: data modules, model modules, training modules, and communication modules.

of `TopoBenchmark`, and Section 4.4 demonstrates its usage via benchmarking experiments. Section 8.6 introduces the Topological Deep Learning (TDL) challenge that was based on the proposed `TopoBenchmark`library. The chapter concludes with some final remarks, current limitations, and suggestions for future work in Section 8.7.

8.1 Introduction

TDL has evolved rapidly recently, providing novel algorithms for learning higher-order relational data supported on various topological domains, such as simplicial, cell, and combinatorial complexes. The works of [90] and [159] have recently unified theory and notation of many TNNs. Moreover, the TAG ICML challenge 2023 [158] has contributed to software implementations of topological message passing for most TNNs under `TopoX` [93]. Despite these theoretical and practical advancements, the rapid acceleration of TDL research poses challenges for the reproducibility and systematic comparative evaluation of TNNs [157]. In particular, open software, standardization, and benchmarking were recognized as key open challenges for TDL in a recent ICML 2024 position paper [157]. `TopoBenchmark` provides a robust framework to address these issues and ensure consistent and reliable progress in the field.

`TopoBenchmark`[1] is an open-source and modular benchmarking framework for comparative evaluations of TNNs. Figure 8.1 visualizes the overall workflow and modularity of `TopoBenchmark`; see Section 8.4 for a more detailed outline. The proposed `TopoBenchmark` tool can generate topological datasets for benchmarking purposes, and carry out TNN comparisons based on various performance metrics. `TopoBenchmark` provides a high-level interface that reduces coding efforts, making it user-friendly. It can facilitate reproducibility of TNN research, and serves as a

[1] https://github.com/geometric-intelligence/TopoBenchmark

testbed to experiment with new ideas in TDL. Finally, `TopoBenchmark` is built with compatibility in mind, allowing integration with other tools and frameworks in the broader `PyTorch` ecosystem.

Three aspects of TDL make benchmarking less straightforward: the scarcity of topological data, the non-trivial standardization of input and output across topological domains, and the diversity of TNN architectures. `TopoBenchmark` addresses these three limitations.

Generation of topological data. Although higher-order data arise in various physical systems, the complexity of capturing these interactions and the limitations of current experimental designs constrain data collection. As a result, higher-order data is often generated by lifting (i.e., transforming) graph datasets and equipping them with higher-order features. `TopoBenchmark` implements graph lifting algorithms to automate this generation process, thus mitigating the scarcity of topological datasets.

Standardization of input and output across topological domains. Each topological domain encodes higher-order relations differently. Further, each TNN architecture provides distinct mappings between inputs and outputs, and therefore processes higher-order data differently. `TopoBenchmark` standardizes the input-output pipeline through an interface that automatically manages the transition between topological domains and their associated TNNs.

Diversity of TNN architectures. There is a broad spectrum of TNNs in the TDL literature. Each TNN typically employs its own techniques to preprocess and encode data. This diversity complicates performance comparisons between models on different datasets. To address this issue, `TopoBenchmark` implements a pipeline for data preprocessing and predictive performance evaluation metrics. The pipeline is easily accessible to newcomers through a configuration file.

8.2 Existing software

Graph-based learning and GDL are supported by several software packages, including `NetworkX` [84], `KarateClub` [174], `PyG` [72], and `DGL` [220]. `NetworkX` enables computations on graphs, while `KarateClub` implements unsupervised learning algorithms for graph-structured data. `PyG` and `DGL` provide functionality for GDL as well as graph-based learning.

Various software packages implement machine learning algorithms for higher-order domains such as `HyperNetX` [142], `XGI` [131], `DHG` [71], and `TopoX` [93]. `HyperNetX` facilitates computing with hypergraphs, while `XGI` enables computing with both hypergraphs and simplicial complexes. `DHG` implements deep learning algorithms for graphs and hypergraphs. `TopoX` is a suite of three packages: `TopoNetX`, `TopoEmbedX`, and `TopoModelX`, which together enable learning with TNNs. `TopoNetX` supports computations on graphs and higher-order domains, including simplicial complexes, cell complexes, path complexes, combinatorial complexes, and colored hypergraphs. `TopoEmbedX` implements algorithms to embed higher-order domains into Euclidean spaces. `TopoModelX` implements most of the TNNs surveyed in [159].

The Open Graph Benchmark (OGB) is a set of datasets and a benchmark framework to facilitate reproducible graph machine learning research [104, 105]. The OGB

is applicable to graph-based learning and GDL but not to TDL. `TopoBenchmark` aims to fill this gap by providing algorithms that generate higher-order datasets and benchmark TNNs.

8.3 Featured topological domains and lifting

The definitions of featured graphs and featured topological domains, along with the formalization of lifting a featured graph to a featured topological domain, are presented in this section (see Section 2.1).

8.3.1 Liftings

In this chapter, lifting refers to the process of mapping a featured graph to a featured topological domain through a well-defined procedure [90]. We provide a precise definition for the case of cell complexes in Section 2.1.5

Examples of liftings are given in Figure 2.2. Observe that a featured cell complex in Definition 2.1.5 acts as a template for a general topological domain to which the structural and feature information of the graph is lifted. In the general case, even if the graph features are empty (that is, if F_V and F_E are not defined), lifting still provides a way to map the graph structure to the cell complex structure.

Lifting maps can be fixed [32, 90] or learnable [17, 24]. Fixed lifting maps are predefined and follow a rule-based procedure to lift a featured graph to a featured topological domain. For example, one could define a fixed lifting that always maps a node in the graph to a node in the topological domain, and an edge to a corresponding cell. In contrast, learnable lifting maps are determined using machine learning models that are trained to optimize the lifting process based on specific criteria or objective functions. During training, the model learns to embed the nodes and edges of the input graph into the cells of a topological domain in a way that optimizes the representation for downstream tasks, such as classification or clustering. When lifting a graph to a topological domain, both the features on the higher-order cells and the structure of the topological domain itself can be computed deterministically [169] or learned [90].[2]

8.4 Outline of library

implements a unified and flexible workflow that facilitates the addition of new datasets, transforms, and deep learning models, thus ensuring applicability across a wide range of tasks and enabling a broad cross-domain comparison, which is lacking in the TDL literature. Each module within `TopoBenchmark` is assigned a distinct role, while maintaining a consistent input-output structure, which provides a modular interface across all topological domains. Figure 8.1 outlines the `TopoBenchmark` modules, which are organized by functionality to data, model, training, and communication modules. Algorithm 1 exemplifies the `TopoBenchmark` execution pipeline via pseudo-code.

[2] An unpooling operation from lower-rank cells, where features are given, to higher-rank cells, where features are to be lifted, is called a learnable feature lifter in [90].

8.4.1 Modules

Data modules. These modules include `Loader`, `Transforms`, `PreProcessor`, and `Dataloader`.
Loader. The `loader` module provides an interface for downloading and storing data; it includes the `GraphLoader`, `HypergraphLoader`, `SimplicialLoader`, and `CellLoader` classes. Their interfaces are built upon the widely-adopted and popular `InMemoryDataset`, provided by `PyG` to enhance interoperability. The project webpage provides multiple in-depth tutorials on the library, including a step-by-step overview of incorporating customized data with these interfaces.
Transforms. *Transforms* modules are developed as sub-classes of the `BaseTransform` class provided by `PyG`. These modules are `data manipulation`, `topology lifting`, and `feature lifting`. The `data manipulation` module allows for performing any data transformation; e.g., the transformations available in `PyG` or `TopoEmbedX` can be easily adapted in `TopoBenchmark`. The `topology lifting` and `feature lifting` modules are responsible for lifting data from one topological domain to another. In particular, the former module relies on `TopoNetX` to lift the topology, while the latter utilizes the lifted topology to infer (e.g., project, concatenate) the features for higher-order topological structures. Each transform takes a `Data` object from `PyG` as input, performs the necessary computations, and outputs the modified `Data` object. The *Transforms* modules are composable operators that can be easily customized to perform various tasks.
Pre-processor. The `PreProcessor` class applies a sequence of transforms to a dataset. It takes a dataset object and a sequence of transforms as input, and iterates over the dataset, applying a set of transforms sequentially to each instance in the dataset. To avoid re-computing the same transforms multiple times, the pre-processed dataset is automatically saved in a unique folder for each transform configuration. This ensures that each dataset needs to be processed only once per configuration, addressing the potential time-consuming nature of preprocessing large datasets. `PreProcessor` also generates and/or loads data splits according to the chosen strategy; for example, random splits with predefined proportions, k-fold cross-validation, or fixed splits.
Dataloader. The `dataloader` module provides a consistent interface for batch training across all topological domains. `TopoBenchmark` supports mini-batching for graphs, hypergraphs, simplicial complexes, and cell complexes.
Model modules. *Neural networks* modules form the core of the modeling pipeline. The `encoder` module maps initial data features into a latent space and applies a learnable transformation before passing the data to the TNN model. The `backbone` TNN can be imported from `TopoModelX`, `PyG`, or it can be built on a custom basis, allowing flexibility in model selection. The `wrapper` ensures that the correct input is provided to the forward pass of the model and collects the output in a dictionary. The `wrapper` plays a crucial role in the integration of any model in `TopoBenchmark`, making it possible to use existing and newly developed models. The `readout` module processes latent representations of the neural network into final output predictions. The `Loss` module is responsible for defining the loss function from the `PyTorch` library. The `optimizer` module defines the optimizer and scheduler. It allows for easy integration of any optimizer and scheduler from `torch.optim`, which provides

Algorithm 1 Execution pipeline for model training in `TopoBenchmark`

```
1: Input: General cfg configuration file
2: dataset ← Loader(cfg.dataset)                                  # Dataset loading
3: splits ← PreProcessor(dataset, cfg.transforms)                 # Transforms and
   splits
4: dataloader ← Dataloader(dataset)                               # Batch generator
5: model ← Model(                                                 # Model initialization
6:    nn.Encoder(cfg.model),
7:    nn.Backbone(cfg.model),
8:    nn.BackboneWrapper(cfg.model),
9:    nn.Readout(cfg.model),
10:              *[Evaluator(cfg.evaluator), Optimizer(cfg.optimizer),
   Loss(cfg.loss)]
   )
11: trainer ← lightning.Trainer(cfg.trainer, cfg.callbacks,
   cfg.logger)

12: Model training:
13: trainer.fit(model, dataloader)                                # Model training

14: Model step for each batch:
15: batch ← self.encoder(batch)                                   # Feature encoder
16: model_out ← self.forward(batch)                               # TNN
17: model_out ← self.readout(model_out, batch)                    # Readout
18: model_out ← self.loss(model_out, batch)                       # Loss computation
19: self.evaluator.update(model_out)                              # Evaluator update
```

flexibility in optimizing model training. The `evaluator` module is responsible for computing metrics during training and inference. It is built upon `torchmetrics` and supports standard metrics for classification and regression tasks.

Training and communication modules. The `Model` class defines a training pipeline for all domains (lines 14-19 of Algorithm 1). It inherits `LightningModule`, and requires the `Encoder`, `Wrapper`, `Backbone`, `Readout`, `Evaluator`, `Loss`, and `Optimizer` objects as inputs. The `lightning.Trainer` automates the training, evaluation, and testing processes. Extra functionalities can be added using *callbacks* modules, and users can monitor the training process using different *loggers* (e.g. `wandb`, `tensorboard`). Both are common practice in `Lightning`, and they are referred to as *communication* modules in `TopoBenchmark`.

8.4.2 Topological liftings

This section explores different (structural) liftings implemented in `TopoBenchmark`. In particular, liftings of graphs to cell complexes, simplicial complexes, and hypergraphs are discussed.[3]

From graphs to cell complexes: cycle-based liftings. A graph is lifted to a cell complex as follows. First, a finite set of cycles (closed loops) within the graph is identified. Second, each identified cycle in the graph is associated with a 2-cell,

[3]Note that the modular design of `TopoBenchmark` easily allows for the addition of new lifting transforms between any pair of topological domains.

such that the boundary of the 2-cell is the cycle. The nodes and edges of the cell complex are inherited from the graph.

From graphs to simplicial complexes: clique complexes. By lifting a graph to a simplicial complex, both pairwise and higher-order interactions are captured. For a graph, the corresponding clique complex is formed by considering every complete subgraph (clique) as a simplex. Specifically, each node is a 0-simplex, each edge (clique of size 2) is a 1-simplex, each triangle (clique of size 3) is a 2-simplex, and so forth. In general, a clique of size $k+1$ becomes a k-simplex.

From graphs to simplicial complexes: neighbor complexes. Neighbor complexes lift the neighborhoods of nodes to become simplices as follows. For each node in the graph, take the set of its neighbors and the node itself. These sets are then treated as simplices. The dimension of each simplex depends on the degree of the nodes. For example, a node with d neighbors forms a d-simplex.

From graphs to hypergraphs: k-hop liftings. Let $\mathcal{G} = (V, E)$ be a graph and $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. The k-neighborhood $N_k(v)$ of a node $v \in V$ in $\mathcal{G}$ consists of all nodes reachable within k steps from v. To lift $\mathcal{G}$ to $\mathcal{H}$, for each node $v \in V$, determine $N_k(v)$ and create a hyperedge e_v in $\mathcal{H}$ such that $e_v = N_k(v)$. Thus, the set of hyperedges in $\mathcal{H}$ is given by $\mathcal{E} = \{N_k(v) \mid v \in V\}$.

8.5 Evaluation

This section presents numerical experiments that exemplify the wide range of functionality of `TopoBenchmark`, and provide a first attempt to perform a broad cross-domain comparison. First, the general experimental setup is described. The experiments compare twelve graph, hypergraph and topological neural network models across four learning tasks related to twenty-two datasets. Subsequently, the main benchmark results are presented. Finally, it is shown how `TopoBenchmark` facilitates comparisons in TDL by conducting an ablation study for different types of signal propagation[4].

8.5.1 Setup

Learning tasks and datasets. Four learning tasks are carried out, namely node classification (seven datasets), node regression (seven datasets), graph classification (seven datasets), and graph regression (one dataset). For node classification, cocitation datasets (Cora, Citeseer, and Pubmed), as well as heterophilic datasets (Amazon rating, Minesweeper, Roman empire, and Tolokers) are used [166]. For node regression, the election, bachelor, birth, death, income, migration, and unemployment datasets from US election map networks are adapted [112]. In these datasets, nodes represent US states, edges connect neighboring states, and each state is characterized by demographic and election statistics. For each dataset, one statistic is set to be the output variable, while the remaining statistics serve as node features, referring to each such dataset with the output statistic's name. For graph classication, the TUDataset is used, consisting of the MUTAG, PROTEINS, NCI1,

[4]See Section 2.3 for an introduction to TNNs and higher-order message passing on topological domains.

NCI109, IMDB-BIN, IMDB-MUL, and REDDIT datasets; see [149]. For graph regression, the ZINC dataset is employed [107]. See Table D.5 in Appendix D.3 for descriptive summaries of the datasets.

Models. The twelve neural network models are supported on four domains: graphs, hypergraphs, simplicial complexes, and cell complexes. In particular, three GNNs (GCN, GIN, and GAT), three hypergraph neural networks (EDGNN, AllSet-Transformer, and UniGNN2), three simplicial neural networks (SCN, SCCN, and SCCNN), and three cell complex neural networks (CCXN, CWN, and CCCN) are used in the experiments.

Training and evaluation. Five splits are generated for each dataset. Each split comprises 50%/25%/25% of data points in the training, validation, and test set, respectively. As an exception, the pre-defined splits for the ZINC dataset are used [107]. The optimal hyperparameter set is selected based on the best average performance over five validation splits. See Appendix D.2.2 for more details on the hyperparameter search. One performance metric per dataset is reported. More specifically, the predictive accuuracy is reported for Cora, Citeseer, Pubmed, Amazon, Empire, MUTAG, PROTEINS, NCI1, NCI109, IMDB-BIN, IMDB-MUL, and REDDIT; the AUC-ROC metric is reported for Minesweeper and Tolokers; the mean-squared error is reported for election, bachelor, birth, death, income, migration, and unemployment; and the mean-absolute error is reported for ZINC. For each performance metric, its mean and standard deviation are computed over the five test sets per dataset and are reported in Table 8.1 (note that OOM stands for 'out of memory' in Table 8.1).

8.5.2 Main results

As seen from Table 8.1, higher-order neural networks (based on hypergraphs, simplicial and cell complexes) achieve the best performance on sixteen of twenty-two datasets, whereas GNNs achieve the best performance on seven datasets. GNNs perform best on node regression in the majority of cases (five out of seven). These best results obtained by GNNs are closely matched by TNNs, since the latter attain metrics within one standard deviation from the former. In contrast, in nine out of sixteen datasets TNNs outperform GNNs, and attain performance metrics that are higher by more than one standard deviation with respect to GNNs. In other words, in the cases where higher-order networks outperform GNNs, the performance gap is more pronounced. It is also noted that, for demonstration purposes, only one fixed lifting is considered to transform graph data to each of the considered topological domains (see Appendix D.2.3). These results suggest that TNNs hold an advantage over GNNs, even without lifting optimization. Most importantly for the context of this chapter, the benchmarks demonstrate the extent of comparisons that can be performed with `TopoBenchmark` across models and datasets.

8.5.3 Ablation study

Graph and hypergraph (neural network) models differ from simplicial and cell complex (neural network) models in terms of the domains and representations they support. Graph and hypergraph models can output two types of representations:

Table 8.1. Cross-domain comparison: results are shown as mean and standard deviation. The best result is bold and shaded in grey, while those within one standard deviation are in blue-shaded boxes.

	Dataset	GCN	GIN	GAT	AST	EDGNN	UniGNN2	CWN	CCCN	SCCNN	SCN
Node-level tasks	Cora	87.09 ± 0.20	87.21 ± 1.89	86.71 ± 0.95	**88.92 ± 0.44**	87.06 ± 1.09	86.97 ± 0.88	86.32 ± 1.38	87.44 ± 1.28	82.19 ± 1.07	82.27 ± 1.34
	Citeseer	75.53 ± 1.27	73.73 ± 1.23	74.41 ± 1.75	73.85 ± 2.21	74.93 ± 1.39	74.72 ± 1.08	75.20 ± 1.82	**75.63 ± 1.58**	70.23 ± 2.69	71.24 ± 1.68
	Pubmed	89.40 ± 0.30	89.29 ± 0.41	89.44 ± 0.24	**89.62 ± 0.25**	89.04 ± 0.51	89.34 ± 0.45	88.64 ± 0.36	88.52 ± 0.44	88.18 ± 0.32	88.72 ± 0.50
	Amazon	49.56 ± 0.55	49.16 ± 1.02	50.17 ± 0.59	50.50 ± 0.27	48.18 ± 0.09	49.06 ± 0.08	**51.90 ± 0.15**	50.26 ± 0.17	OOM	OOM
	Empire	78.16 ± 0.32	79.56 ± 0.20	84.02 ± 0.51	79.50 ± 0.13	81.01 ± 0.24	77.06 ± 0.20	81.81 ± 0.62	82.14 ± 0.00	**89.15 ± 0.32**	88.79 ± 0.46
	Minesweeper	87.52 ± 0.42	87.82 ± 0.34	89.64 ± 0.43	81.14 ± 0.05	84.52 ± 0.05	78.02 ± 0.00	88.62 ± 0.04	89.42 ± 0.00	89.0 ± 0.00	**90.32 ± 0.11**
	Tolokers	83.02 ± 0.71	80.72 ± 1.19	**84.43 ± 1.00**	83.26 ± 0.10	77.53 ± 0.01	77.35 ± 0.03	OOM	OOM	OOM	OOM
	Election	0.31 ± 0.02	**0.28 ± 0.02**	0.29 ± 0.02	0.29 ± 0.01	0.34 ± 0.02	0.37 ± 0.02	0.34 ± 0.02	0.31 ± 0.02	0.51 ± 0.03	0.46 ± 0.04
	Bachelor	0.29 ± 0.02	0.31 ± 0.03	**0.28 ± 0.02**	0.30 ± 0.03	0.29 ± 0.02	0.31 ± 0.02	0.33 ± 0.03	0.31 ± 0.02	0.34 ± 0.03	0.32 ± 0.02
	Birth	0.72 ± 0.09	0.72 ± 0.09	0.71 ± 0.09	0.71 ± 0.08	**0.70 ± 0.07**	0.73 ± 0.10	0.72 ± 0.09	0.71 ± 0.09	0.79 ± 0.12	0.71 ± 0.08
	Death	0.51 ± 0.04	0.52 ± 0.04	0.51 ± 0.04	**0.49 ± 0.05**	0.52 ± 0.05	0.51 ± 0.05	0.54 ± 0.06	0.54 ± 0.06	0.55 ± 0.05	0.52 ± 0.05
	Income	0.22 ± 0.03	0.21 ± 0.02	**0.20 ± 0.02**	0.21 ± 0.02	0.23 ± 0.02	0.23 ± 0.02	0.25 ± 0.03	0.23 ± 0.02	0.28 ± 0.03	0.25 ± 0.02
	Migration	0.80 ± 0.12	0.80 ± 0.10	**0.77 ± 0.13**	0.78 ± 0.12	0.80 ± 0.12	0.79 ± 0.12	0.84 ± 0.13	0.84 ± 0.12	0.90 ± 0.14	0.92 ± 0.20
	Unempl	0.25 ± 0.03	**0.22 ± 0.02**	0.23 ± 0.03	**0.22 ± 0.02**	0.26 ± 0.03	0.28 ± 0.02	0.25 ± 0.03	0.24 ± 0.03	0.43 ± 0.04	0.38 ± 0.04
Graph-level tasks	MUTAG	69.79 ± 6.80	79.57 ± 6.13	72.77 ± 2.77	71.06 ± 6.49	80.00 ± 4.90	**80.43 ± 4.09**	**80.43 ± 1.78**	77.02 ± 9.32	76.17 ± 6.63	73.62 ± 6.13
	PROTEINS	75.70 ± 2.14	75.20 ± 3.30	76.34 ± 1.66	**76.63 ± 1.74**	73.91 ± 4.39	75.20 ± 2.96	76.13 ± 2.70	73.33 ± 2.30	74.19 ± 2.86	75.27 ± 2.14
	NCI1	72.86 ± 0.69	74.26 ± 0.96	75.00 ± 0.99	75.18 ± 1.24	73.97 ± 0.82	73.02 ± 0.92	73.93 ± 1.87	**76.67 ± 1.48**	76.60 ± 1.75	74.49 ± 1.03
	NCI109	72.20 ± 1.22	74.42 ± 0.70	73.80 ± 0.73	73.75 ± 1.09	74.93 ± 2.50	70.76 ± 1.11	73.80 ± 2.06	75.35 ± 1.50	**77.12 ± 1.07**	75.70 ± 1.04
	IMDB-BIN	**72.00 ± 2.48**	70.96 ± 1.93	69.76 ± 2.65	70.32 ± 3.27	69.12 ± 2.92	71.04 ± 1.31	70.40 ± 2.02	69.12 ± 2.82	70.88 ± 2.25	70.80 ± 2.38
	IMDB-MUL	49.97 ± 2.16	47.68 ± 4.21	50.13 ± 3.87	**50.51 ± 2.92**	49.17 ± 4.35	49.76 ± 3.55	49.71 ± 2.83	47.79 ± 3.45	48.75 ± 3.98	49.49 ± 5.08
	REDDIT	76.24 ± 0.54	81.96 ± 1.36	75.68 ± 1.00	74.84 ± 2.68	83.24 ± 1.45	75.56 ± 3.19	**85.52 ± 1.38**	85.12 ± 1.29	77.24 ± 1.87	71.28 ± 2.06
	ZINC	0.62 ± 0.01	0.57 ± 0.04	0.61 ± 0.01	0.59 ± 0.02	0.51 ± 0.01	0.60 ± 0.01	**0.34 ± 0.01**	**0.34 ± 0.02**	0.36 ± 0.02	0.53 ± 0.04

node representations and edge or hyperedge representations. In contrast, the output of simplicial and cell models depends on the different types of cells present (0-cell up to n-cells) and the model itself. For example, a simplicial or cell complex model may process an n-cell input but may not produce an n-cell output. The `backbone_wrapper` in `TopoBenchmark` addresses these differences in the underlying domains of the models.

There is a second difference, which is inherent in the TNNs themselves. Consider a downstream classification task. For graphs, the standard practice is to perform classification over pooled node features. However, this aspect has not been extensively studied in the TDL literature. For instance, a simplicial or cell model may update 1-cell representations (edges) or 2-cell representations (cycles or triangles) while leaving 0-cell (nodes) representations unchanged, making direct pooling over nodes potentially meaningless. One can consider more complicated update processes in which different n-cell representations are combined, which makes the notion of pooling more intricate for higher-order domains, in general. These architectural implications are complex and are related to open research questions in TDL.

However, to compare different neural network architectures, the second aforementioned difference must be addressed. To this end, the ablation study of this section considers two types of readouts to perform a rigorous evaluation: direct readout (DR), in which the downstream task is performed directly over the 0-cell representation, and signal down-propagation (SDP), in which information from higher-order cell representations is iteratively fused down to 0-cell representations using appropriate incidence matrices, followed by a linear projection over the concatenated $n-1$-cell signal and the fused $n-1$-cell representation. For instance, if a simplicial or cell complex model outputs 0-cell, 1-cell, and 2-cell representations, the signal propagates from 2-cells to 1-cells, and from 1-cells to 0-cells during readout. The downstream task is then performed over the 0-cell representations.[5]

Table 8.2 shows that the best-performing readout type depends on the model's interior signal propagation. The CWN model does not update the 0-cell representa-

[5]See Section 2.3 for an introduction to TNNs and higher-order message passing on topological domains.

tion during propagation and the SDP readout strategy performs notably better, as expected. Conversely, CCCN, SCCNN, and SCN propagate information to 0-cells, so incorporating SDP readout leads to either small or minor variations in performance metrics. Please refer to Appendix D.2.3 for more detailed results.

These results highlight the importance of considering the structural properties of TNNs. The performance variations observed in this ablation study emphasize how critical architectural and lifting decisions are in higher-order learning models. By facilitating comparisons across a wide range of models and datasets, `TopoBenchmark` enables researchers to derive new insights and advance TDL.

Table 8.2. An ablation study compares the performance of CWN, CCCN, SCCNN, and SCN models on various datasets using two readout strategies, direct readout (DR) and signal down-propagation (SDP). SDP generally enhances CWN performance, whereas the effect of SDP on CCCN, SCCNN, and SCN varies based on their internal signal propagation mechanisms. Means and standard deviations of performance metrics are shown. The best results are shown in bold for each model and readout type.

	Dataset	CWN		CCCN		SCCNN		SCN	
		DR	SDP	DR	SDP	DR	SDP	DR	SDP
Node-level tasks	Cora	74.95 ± 0.98	**86.32 ± 1.38**	87.44 ± 1.28	**87.68 ± 1.17**	**82.19 ± 1.07**	80.65 ± 2.39	**82.27 ± 1.34**	79.91 ± 1.18
	Citeseer	70.49 ± 2.85	**75.20 ± 1.82**	**75.63 ± 1.58**	74.91 ± 1.25	**70.23 ± 2.69**	69.03 ± 2.01	**71.24 ± 1.68**	70.40 ± 1.53
	Pubmed	86.94 ± 0.68	**88.64 ± 0.36**	88.52 ± 0.44	**88.67 ± 0.39**	**88.18 ± 0.32**	87.78 ± 0.58	**88.72 ± 0.50**	88.62 ± 0.44
	Amazon	45.58 ± 0.25	**51.90 ± 0.15**	**50.55 ± 0.15**	50.26 ± 0.17	OOM	OOM	OOM	OOM
	Empire	66.13 ± 0.03	**81.81 ± 0.62**	82.14 ± 0.00	**82.51 ± 0.00**	**89.15 ± 0.32**	88.73 ± 0.12	85.89 ± 0.34	**88.79 ± 0.46**
	Minesweeper	48.82 ± 0.00	**88.62 ± 0.04**	89.42 ± 0.00	**89.85 ± 0.00**	87.40 ± 0.00	**89.00 ± 0.00**	**90.32 ± 0.11**	90.27 ± 0.36
	Election	0.60 ± 0.04	**0.34 ± 0.02**	**0.31 ± 0.02**	**0.31 ± 0.01**	**0.51 ± 0.03**	0.56 ± 0.04	**0.46 ± 0.04**	0.51 ± 0.03
	Bachelor	**0.33 ± 0.03**	**0.33 ± 0.03**	0.32 ± 0.02	**0.31 ± 0.02**	**0.34 ± 0.03**	**0.34 ± 0.03**	**0.32 ± 0.02**	**0.32 ± 0.03**
	Birth	0.81 ± 0.11	**0.72 ± 0.09**	**0.71 ± 0.09**	0.72 ± 0.05	**0.79 ± 0.12**	0.83 ± 0.12	**0.71 ± 0.08**	0.80 ± 0.11
	Death	0.55 ± 0.05	**0.54 ± 0.06**	**0.54 ± 0.06**	**0.54 ± 0.06**	**0.55 ± 0.05**	0.58 ± 0.05	**0.52 ± 0.05**	0.56 ± 0.05
	Income	0.36 ± 0.04	**0.25 ± 0.03**	**0.23 ± 0.02**	**0.23 ± 0.02**	**0.28 ± 0.03**	0.31 ± 0.03	**0.25 ± 0.02**	0.27 ± 0.02
	Migration	0.90 ± 0.16	**0.84 ± 0.13**	**0.84 ± 0.10**	**0.84 ± 0.12**	**0.90 ± 0.14**	0.93 ± 0.17	**0.92 ± 0.20**	0.96 ± 0.23
	Unempl	0.46 ± 0.04	**0.25 ± 0.03**	**0.24 ± 0.03**	0.25 ± 0.03	**0.43 ± 0.04**	0.45 ± 0.04	**0.38 ± 0.04**	0.41 ± 0.03
Graph-level tasks	MUTAG	69.68 ± 8.58	**80.43 ± 1.78**	**80.85 ± 5.42**	77.02 ± 9.32	**76.17 ± 6.63**	70.64 ± 3.16	71.49 ± 2.43	**73.62 ± 6.13**
	PROTEINS	**76.13 ± 1.80**	**76.13 ± 2.70**	**73.55 ± 3.43**	73.33 ± 2.30	74.19 ± 2.86	**74.98 ± 1.92**	**75.27 ± 2.14**	74.77 ± 1.69
	NCI1	68.52 ± 0.51	**73.93 ± 1.87**	76.67 ± 1.48	**77.65 ± 1.28**	**76.60 ± 1.75**	75.60 ± 2.45	**75.27 ± 1.57**	74.49 ± 1.03
	NCI109	68.19 ± 0.65	**73.80 ± 2.06**	**75.35 ± 1.50**	74.83 ± 1.18	**77.12 ± 1.07**	75.43 ± 1.94	74.58 ± 1.29	**75.70 ± 1.04**
	IMDB-BIN	**70.40 ± 2.02**	69.28 ± 2.57	69.12 ± 2.82	**69.44 ± 2.46**	**70.88 ± 2.25**	69.28 ± 5.69	**70.80 ± 2.38**	68.64 ± 3.90
	IMDB-MUL	49.71 ± 2.83	**49.87 ± 2.33**	**49.01 ± 2.63**	47.79 ± 3.45	**48.75 ± 3.98**	46.67 ± 3.13	48.16 ± 2.89	**49.49 ± 5.08**
	REDDIT	76.20 ± 0.86	**85.52 ± 1.38**	**85.12 ± 1.29**	83.32 ± 0.73	75.56 ± 3.46	**77.24 ± 1.87**	**71.28 ± 2.06**	69.68 ± 4.00
	ZINC	0.70 ± 0.00	**0.34 ± 0.01**	0.35 ± 0.02	**0.34 ± 0.02**	**0.36 ± 0.01**	**0.36 ± 0.02**	0.59 ± 0.01	**0.53 ± 0.04**

8.6 Topological Deep Learning Challenge: Beyond the Graph Domain

This section presents the Topological Deep Learning Challenge that was organized at the ELLIS Workshop on Geometry-grounded Representation Learning and Generative Modeling (GRaM) workshop. The main technical part of the challenge has been based on the TopoBenchmark library, and the output of the challenge, the new liftings, are in the process of being added to the main branch of the TopoBenchmark library.

Motivation

Despite the recent emergence of TDL it is already postulated to become a relevant tool in many research areas and applications, from complex physical systems [20] and signal processing [12] to molecular analysis [33] and social interactions [181], to name a few. However, a current limiting factor in the extensive use of

higher-order structures is that most datasets are usually stored as pointclouds or graphs. Although researchers have introduced various mechanisms for extracting higher-order elements (e.g. [236, 17, 24, 69, 87, 102]), it remains unclear how to optimize the process given a specific dataset and task.

In this context, developing tools and methods to move between different discrete (topological) domains represents one of the most pressing open challenges in TDL [157]. The process of mapping a data structure to a different topological domain is formalized using the concept of *topological lifting* (or, equivalently, *lifting*) [159]. Additionally, a *feature lifting* is a particular lifting that transfers data from an original domain where a signal (node/edge features) exists, to a new domain where new topological structures can emerge, such as simplicial/cell complexes. Fig. 2.2 depicts some visual examples of different liftings involving several topological domains.

In this framework, the main purpose of the *ICML 2024 Topological Deep Learning Challenge* is to foster new research and knowledge about effective liftings between different topological domains and data structures. The introduction of new topological liftings between different topological domains, as well as an efficient implementation of both new and existing mappings, might help to enforce the impact of TDL to a broader range of use cases and scenarios.

Besides reporting the main organizational aspects of the challenge, this section aims to give an overview of the main results achieved by participants.

8.6.1 Setup of the Challenge

The challenge[6] was hosted by the Geometry-grounded Representation Learning and Generative Modeling (GRaM) Workshop[7] at the International Conference on Machine Learning (ICML) 2024. Participants were asked to implement a topological lifting, apply it to a toy dataset, and test the results with an existing provided model for the considered target domain.

Guidelines

Participation was free and open to everyone –only principal PyT-Team developers were excluded. To enroll in the challenge, it was sufficient to:

- Send a valid Pull Request (PR) –i.e. passing all tests– before the deadline.
- Respect Submission Requirements (see below).

Teams were accepted, and there was no restriction on the number of team members. An acceptable PR automatically subscribed a participant/team to the challenge. A Pull Request could not contain more than one lifting. However, there was no restriction on the number of submissions (i.e. PRs) per participant/team.

Consistent with the aims of an open environment for sharing participation, in this activity was completely voluntary and no support or endorsement of any of the participating parties by any of the other participating parties was provided. All submissions are the views of the individual participants only and should be taken, as is with all faults and without any guarantee, promise or endorsement of any kind.

[6] Website: https://pyt-team.github.io/packs/challenge.html
[7] Workshop website: https://gram-workshop.github.io/

Submission Requirements

The submission had to have a valid lifting transformation between any pair of the following data structures: point-cloud/graph, hypergraph, simplicial complex, cell complex, and combinatorial complex. For a lifting to be valid, participants had to implement a mapping between the topological structures of two of the considered domains –*topological lifting*. Participants may optionally implement a procedure to define the features over the resulting topology –*feature lifting*.

Topology Lifting (Required)

Submissions could implement already proposed liftings from the literature, as well as propose novel approaches. In the case of original liftings, we note that neither the challenge nor its related publications would prevent participants from publishing their work: they keep all the credit for their implementations.

For a lifting from a certain source domain `src` (e.g. graph) to a topological destination `dst` (e.g. simplicial), a submission consisted of a PR to the ICML Challenge repository with the following files:

1. A Python script implementing the topological lifting in a single class using the provided `{src}2{dst}` transform primitives.
2. A configuration file defining the default parameters of the implemented transform.
3. A Jupyter notebook that loads a dataset from the `src` domain, applies the implemented lifting to transform the data into the `dst` domain, and runs a model from `TopoModelX` over the lifted dataset.
4. A Python script which contains the unit tests for all implemented methods and classes.

Feature Lifting (Optional)

Some TDL models require well-defined features on higher-order structures (e.g. 2-cells, hyperedges); therefore, in their more general formulation, liftings also need to produce initial features for every topological element of the target domain. Participants were more than welcome to implement new feature liftings mappings, although it was optional and only regarded as a bonus.

Award Categories

Given the lack of an exhaustive analysis of different types of procedures to infer the topological structure within TDL, there was no particular requirement for submitted liftings –apart from a high-quality code implementation. To promote and guide diversity in submissions, we proposed 4 general, non-mutually exclusive Award Categories (ACs) according to the following 2 taxonomies.

By target domain:

- **1st AC:** Best implementation of a lifting to Simplicial of Cell Domain.
- **2nd AC:** Best implementation of a lifting to Combinatorial Complex, Hypergraph, or Graph Domain.

By leveraged information:

- **3rd AC:** Best feature-based lifting, including liftings that leverage both the graph connectivity simultaneously.
- **4th AC:** Best implementation of a lifting using exclusively the graph connectivity.

In addition to the winners' award categories, the challenge also featured honorable mentions. These honorable mentions were determined based on aggregated reviewers' comments.

Evaluation Method

The Condorcet method was used to rank the submissions and decide on the winners in each category. The evaluation criteria were:

- Does the submission implement the lifting correctly? Is it reasonable and well-defined?
- How readable/clean is the implementation? How well does the submission respect the submission requirements?
- Is the submission well-written? Do the docstrings clearly explain the methods? Are the unit tests robust?

Note that these criteria did not reward the final model performance nor the complexity of the method. Rather, the goal was to implement well-written and accurate liftings that would unlock further experimental evidence and insights in this field.

Selected PyT-Team maintainers and collaborators, as well as each team whose submission(s) respect(s) the guidelines, voted once to express their preference for the best submission in each category. Note that each team was allowed only one vote per category, regardless of the number of team members.

Software Practices

All submitted code had to comply with the challenge's GitHub Action workflow, successfully passing all tests, linting, and formatting (i.e., ruff). Moreover, to ensure consistency, we asked participants to use TopoNetX's classes to manage simplicial/cell/combinatorial complexes whenever these topological domains are the target –i.e., destination– of the lifting. Moreover, we highly encouraged the use of TopoX [93] and NetworkX [83] libraries when possible.

8.6.2 Submissions and Winners

The challenge received 56 submissions in total, out of which 52 were valid according to the requirements (see Subsection 8.6.1). Regarding these valid submissions, they come from 31 different teams, with together sum up a total of 57 participants. Each award category accounted for 24, 28, 25, and 27 submissions correspondingly.

The winners were announced through multiple channels: at the ICML Workshop on Geometry-grounded Representation Learning and Generative Modeling, on social media platforms, and on the official challenge website. Regardless of the final rankings, we want to emphasize that all submissions demonstrated exceptional quality. For a complete list of winners, please refer to the publication [26].

8.7 Discussion

This chapter has introduced `TopoBenchmark`, an open-source benchmarking framework for TDL. `TopoBenchmark` simplifies the benchmarking process and accelerates research by organizing the TDL pipeline into a sequence of modular steps. A key feature of `TopoBenchmark` is its ability to lift between topological domains, allowing richer data representations and more detailed analyses. This capability supports the mapping of graph topology and features to higher-order topological domains, such as simplicial and cell complexes. The effectiveness and applicability of `TopoBenchmark` has been illustrated by benchmarking several TDL architectures on many learning tasks and datasets, gaining some preliminary insights about the relative advantages of different models.

The modular and scalable properties of `TopoBenchmark` open up several opportunities for future enhancements and simultaneously allow us to address its current limitations. One area for future work is the integration of learnable liftings, not yet implemented in the framework. This is a promising direction since learnable liftings can learn optimal representations of topological domains for specific downstream learning tasks. A second relevant limitation is the current lack of higher-order standard datasets. Thus, a second future work direction is to incorporate higher-order datasets into `TopoBenchmark`, giving users the option to choose between downloading built-in datasets (not yet available software feature) and generating synthetic higher-order data by lifting (already available software feature). Lastly, a third critical aspect pertains to the set of performance metrics. A wider range of alternative evaluation metrics specific to TDL needs to be developed and implemented to assess expressivity, explainability, and fairness [157]. More generally, users and researchers are encouraged to contribute to the development of `TopoBenchmark` in any of these three or other preferred directions.

Building upon the foundation laid by `TopoBenchmark`, we organized the 2nd edition of the Topological Deep Learning Challenge to further advance the field. Section 8.6 presented the motivation and outcomes of this challenge, hosted through the ICML 2024 Geometry-grounded Representation Learning and Generative Modeling (GRaM) Workshop. Challenge submissions implemented a wide variety of topological liftings between different pairs of discrete (topological) domains, providing a rich assortment of tools to infer and exploit higher-order structures. We hope that this community effort will help bridge the gap between TDL and most of the current

datasets, stored as point clouds or graphs. Therefore, the methods implemented in this challenge can potentially foster research and further methodological benchmarks in this growing TDL field.[8]

Last but not least, we remark that the participation statistics of this 2nd challenge edition almost doubled the numbers of the previous ICML 2023 TDL Challenge [158] –both in terms of participants and submissions–, indicating a notable increase in interest in the TDL field.

[8] In fact, all challenge submissions are by design compatible with the `TopoBenchmark` framework, which easily allows the use of a diverse set of datasets, models and topological liftings.

Chapter 9

Topological Deep Learning with State-Space Models: A Mamba Approach for Simplicial Complexes

Contents

This chapter is organized as follows. In Section 9.1, we introduce the challenge of the expanded notion of the neighborhood in higher-order topological domains and provide the motivation for the proposed solution that combines sequence modeling with TDL. In Section 9.2, we outline the notation used in this chapter, while Section 9.3 details the methodology of the proposed approach. Section 9.4 presents the evaluation of the proposed methodology, while Section 9.5 reviews related work. The chapter concludes with a discussion of the approach in Section 9.6.

The content of this chapter is adapted from the material submitted to Learning on Graphs Conference [148]

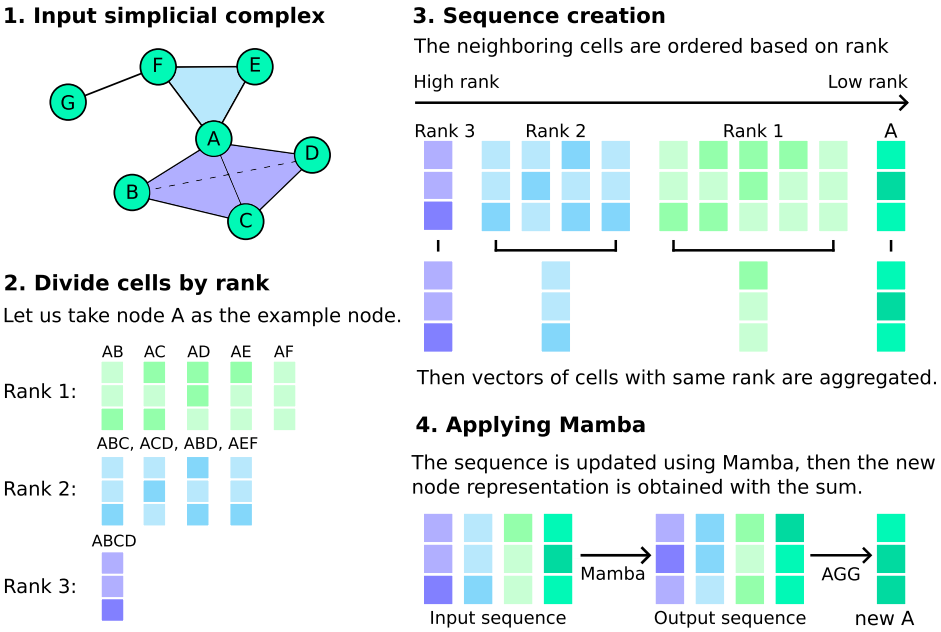


Figure 9.1. Scheme showing how our model updates the vector of node A. First, the neighboring cells of the target node are selected and then ordered by rank. A single vector for each rank is obtained by aggregating the cells with the same rank. Mamba is used on the resulting sequence which is then summed to obtain the new node representation.

9.1 Introduction

There are various challenges in GNN literature related to the Message Passing (MP) mechanism described in Section 2.2, including the challenge in modeling interactions between distant nodes, as multiple message-passing steps are required for information to propagate over long distances [138].
To try and overcome some of the GNNs limitations, several efforts have been made to adapt Transformers [212] for use with graph-structured data [258, 120]. These approaches treat the graph as a complete graph, computing attention between all pairs of nodes to facilitate communication, regardless of the distance between them. However, this method has the significant drawback of the attention matrix scaling quadratically with the number of nodes, making it impractical for large graphs. Recently State Space Models (SSMs) [81] have become a popular alternative to Transformer-based systems for sequence modeling. They can be seen as a combination of recurrent neural networks and convolutional neural networks [79], and their main advantage is that they have linear or non-linear scaling in the sequence length. By modeling long-range interactions with principled mechanisms [80], they have achieved state-of-the-art results in long-range datasets, like the Long Range Arena [199].

Mamba [79] has emerged as one of the most popular SSM. Its success can be attributed to two main factors: its ability to dynamically adapt parts of the parameters previously fixed in the SSMs equations based on input data, and its

hardware-aware implementation, which enables linear scaling with sequence length. By making its parameters input-dependent, Mamba effectively filters out irrelevant information while retaining relevant details indefinitely.

Still, determining how to leverage graph structures to generate sequences suitable for SSMs is an area of active research [21, 63].

TDL is a field of research also interested in addressing the limitations of graph neural networks. As discussed in depth in Section 2, TDL studies topological spaces such as simplicial, cellular, and combinatorial complexes. These domains are not limited to relations incorporating only two nodes, and they can instead work with structures containing multiple. Adapting sequence models to work with topological domains presents several challenges. In graphs, the concept of neighborhood is straightforward, as nodes are neighbors if they share an edge. However, in topological structures, the notion of neighborhood is more complex. As described in Section 9.2, simplicial complexes are composed of cells, which have different ranks based on the number of nodes they contain. Each group of cells at a given rank has both an upper and a lower neighborhood. The expanded definition of neighborhoods in topological domains complicates the development of message-passing strategies. Consequently, finding optimal methods for propagating information through these structures remains an ongoing area of research.

In this chapter, we present a novel approach that combines sequence modeling with topological deep learning. By creating a sequence of cells ordered by their rank, and then modeling it using the Mamba model, we can have information propagating from structures of one rank to any other in a single step. To validate our approach, we perform experiments on different graph datasets by converting them to simplicial complexes. We compare our results to state-of-the-art models developed to work on simplicial complexes. We also discuss how our approach allows for efficient batching during training. The effects of this batching technique on memory usage, training time, and overall performance are evaluated through a series of experiments. This approach is also compared to a standard batching technique for simplicial models.

9.2 Notation and Background

A graph $\mathcal{G} = (V, E)$ is a tuple composed of a finite set of nodes V and a finite set of edges E connecting pairs of nodes. Considering a graph with n_0 nodes, n_1 edges, node features of dimension d_0, and edge features with dimension d_1, we obtain a featured graph $\mathcal{G}_F = (V, E, F_V, F_E)$ where $F_V : V \to \mathbb{R}^{d_0}$ and $F_E : E \to \mathbb{R}^{d_1}$ are functions that map the nodes and the edges to their feature vectors respectively. One common choice to encode the structure of the graph is to use the adjacency matrix $A \in \mathbb{R}^{n_0 \times n_0}$ where

$$(A)_{i,j} = \begin{cases} 1 & e_{ij} \in E, \\ 0 & \text{otherwise}, \end{cases}$$

with e_{ij} representing an edge between nodes i and j.
Simplicial complexes expand on graphs by considering cells with different numbers of nodes. The precise definition of a simplicial complex is given in Section 2.1

We use n_i to indicate the number of cells of rank i in the simplicial complex, and $h_x^{(r)}$ to indicate the features of simplex x, which has rank r. $H^{(r)} \in \mathbb{R}^{n_r \times d_r}$ is the matrix where each row contains the features of a cell of rank r in the simplex.

Cell $x^{(r_1)}$ of rank r_1 is on the *boundary* of cell $y^{(r_2)}$ of rank r_2 if x is connected to y and $r_1 < r_2$, and we express this relation as $x^{(r_1)} \prec y^{(r_2)}$, as presented in [161]. We denote as $B_r \in \mathbb{R}^{n_{r-1} \times n_r}$ the unweighted incidence matrices defined as

$$(B_r)_{i,j} = \begin{cases} \pm 1 & x_i^{(r-1)} \prec x_j^{(r)}, \\ 0 & \text{otherwise.} \end{cases}$$

The sign of the entries of B_r encodes the orientation of the cells of the simplicial complex.

9.2.1 Clique lifting

While higher-order data naturally emerge in a variety of systems, from social networks [125] to proteins [111], the complexity of accurately capturing these interactions limits the collection of such data. Consequently, higher-order data are frequently derived by transforming graph datasets and augmenting them with higher-order features through the lifting procedure that is described in Section 2.1.5.

Connectivity lifting The lifting chosen is the *clique lifting*: given a graph $\mathcal{G}$ and a maximum rank R this lifting first finds the set of cliques $\mathcal{C}$ of the graph. Then for each clique $C \in \mathcal{C}$ the following sets of cells are added to the simplicial complex

$$\mathcal{X}_r = \bigcup_{C \in \mathcal{C}} \{X \subseteq C \mid |X| = r\}, \ \forall 1 \leq r \leq R.$$

This approach has the advantage of respecting the initial connectivity of the graph since the 1-rank cells of the obtained simplicial complex will be all the edges of the original graph but no pair of unconnected nodes will be included in the 1-cells. It is important to notice that given a clique of n elements, the number of rank r cells that the algorithm creates is the binomial coefficient $\binom{n}{r+1}$ which can lead to an exploding number of high-rank cells when the dataset presents large cliques. Despite this, the lifting works well in all the datasets considered in our experiments and we leave an exploration of other lifting techniques to future work.

Feature lifting Since the obtained higher-order cells do not have predefined features, it is necessary to *lift* the features as well. Many approaches are possible, using both fixed or learnable functions, but the most common approach is to obtain the features of a cell by simply summing the features of the lower rank cells that compose it. When using the sum, obtaining the features of the cells of any rank is straightforward when the features of the lower-rank cells are known, since we can use

$$H^{(r)} = B_r^\top H^{(r-1)}. \tag{9.1}$$

9.3 Method

This section presents in detail the proposed model architecture, which we refer to as *TopoMamba*, and then discusses batching for simplicial complexes, starting with presenting the general approach and then detailing how our model architecture allows for a more efficient batching implementation.

9.3.1 Model architecture

TopoMamba consists of three parts that will be explained in the following paragraphs: (i) the feature encoder, (ii) the Mamba block, and (iii) the task head.

Feature encoder The feature encoder performs the transformation of the input features via an initial mapping function

$$H^{(0)} = \text{ReLU}\left(H_{in}^{(0)} \cdot W_{in}^{\top} + b_{in}\right),$$

where $H_{in}^{(0)} \in \mathbb{R}^{n \times d_{in}}$ is the matrix of input features for the n nodes of the simplicial complex. The output of the feature encoder is $H^{(0)} \in \mathbb{R}^{n \times d_h}$, where d_h is the hidden dimension of the transformed features. $W_{in} \in \mathbb{R}^{d_h \times d_0}$ is a learnable weight matrix, and $b_{in} \in R^{d_h}$ is a learnable bias vector.

Mamba block The Mamba model takes as input a batch of sequences and outputs their updated representation. To be able to use Mamba we construct a sequence for each node from the simplicial complex structure. We use a parameter-free transformation (see Figure 9.1 steps 2, 3). In particular, for a given node A, we collect all the simplices it belongs to and divide them by rank: we obtain R sets of the form

$$Z_A^{(r)} = \{h^{(r)} \mid A \prec x^{(r)}\},$$

with R the maximum rank of the simplicial complex. The features for the higher-order cells are obtained from the node features using Equation 9.1. Next, the collected simplices of the same rank are aggregated using the AGG function, which pools a set of i-cells into a single representation. The AGG function can be any invariant aggregation operation, as there is no natural order among the cells of the same rank. The obtained sequence for node A is

$$\mathcal{S}_A = \left[\text{AGG}(Z_A^{(R)}),\ \text{AGG}(Z_A^{(R)}), ...,\ \text{AGG}(Z_A^{(1)}),\ h_A^{(0)}\right]. \tag{9.2}$$

The aggregation operation employed in our model is the sum, since it has better expressive power than the mean and the max aggregators [241]. The features of the node being considered are also added as the last element of the sequence. This ensures that the model can distinguish between different nodes with similar neighborhoods. The obtained sequences are then passed to the Mamba model $\mathcal{M}$ which outputs new sequences

$$\mathcal{S}_A^1 = \mathcal{M}(\mathcal{S}_A). \tag{9.3}$$

We also employ a separate Mamba model and apply it to the inverse of the sequence. This makes it possible to have the features of the higher-order cells be influenced by the node being studied, since as pointed out before the node vector is the last element. If we indicate with inv(·) the operator that inverts a sequence, then we can define

$$\mathcal{S}_A^2 = \mathcal{M}(\text{inv}(\mathcal{S}_A)). \tag{9.4}$$

The results of the two Mamba layers are then summed to obtain a single sequence

$$\mathcal{S}_A = \mathcal{S}_A^1 + \text{inv}(\mathcal{S}_A^2). \tag{9.5}$$

This equation demonstrates how cells of different ranks can directly influence one another. In sequential models, earlier elements in a sequence affect those that follow. Thus, in $\mathcal{S}_A^1$, higher-order cells influence lower-rank cells, while in $\mathcal{S}_A^2$, the influence is reversed, with lower-rank cells affecting higher-rank ones. This enables interactions across all ranks within a single Mamba block. The new features for node A are then obtained by summing the elements of the newly obtained sequence

$$\tilde{h}_A^{(0)} = \sum \mathcal{S}_A. \tag{9.6}$$

In parallel to this whole block, we also employ skip connections which were shown to improve the model performance (Section 9.4.2). The final node features can be written as

$$h_A^{(0)} \leftarrow h_A^{(0)} + \tilde{h}_A^{(0)}. \tag{9.7}$$

Equations 9.2 to 9.7 form the Mamba block. This module outputs the node features but does not update the higher-order cells, which can be obtained by applying Equation 9.1 again. This allows us to stack Mamba blocks one after the other, obtaining a final $h_A^{(0)}$. By exploiting the parallelization capabilities of the Mamba model we can stack the sequences for each node into a single matrix and obtain the desired output matrix $H^{(0)}$.

Task head Finally, the task head is another linear layer that transforms the hidden representations for each node into the shape needed for the desired task. In particular

$$H_{out}^{(0)} = \text{ReLU}\left(H^{(0)} \cdot W_{out}^{\top} + b_{out}\right),$$

where $H_{out} \in \mathbb{R}^{n \times d_{out}}$ is the final output of the network, $W_{out} \in \mathbb{R}^{d_{out} \times d_h}$ is the learnable weight matrix for the transformation, and $b_{out} \in \mathbb{R}^{d_{out}}$ is the bias vector. For classification d_{out} corresponds to the number of possible classes, while $d_{out} = 1$ is used for regression.

For clarity, we omitted from the previous discussions the inclusion of dropout layers and normalization layers. In particular, dropout is implemented after both the feature encoder and the task head, while layer normalization is used after creating the sequences.

9.3.2 Batching

Batching is essential for reducing memory requirements when working with large datasets. Unlike standard machine learning, batching for relational data requires careful handling to avoid loss of information. In message-passing networks, the limited spread of information, restricted by the number of message-passing steps, is leveraged by sampling algorithms. In fact, to avoid losing network information, they select only the batch nodes along with their corresponding n-hop neighborhoods. In TDL, the challenge of handling large higher-order networks becomes more pronounced due to the inclusion of higher-order structures. Generally, since the neighborhood structures of simplicial complexes are more complicated than those in graphs, it is required to handle each incidence matrix B_r independently. This involves first removing nodes that are too distant, followed by edges, triangles, and so on for each higher-order structure. While this approach can be effective, it requires a series of matrix operations, which slows down model training.

TopoMamba allows us to address the batching issue just mentioned by a sampling mechanism that is conceptually similar to, and just as straightforward as, graph neighborhood batching. For the creation of the sequences needed as input to the Mamba block, it is enough to rely solely on the boundary relations between higher-order structures and nodes, referred to as B^*, defined as:

Definition 9.3.1 Let $B^* \in \mathcal{R}^{n_0 \times n^*}$ be a matrix with $n^* = \sum_{i=1}^{R} n_i$ the sum of the number of all the higher order structures starting from rank 1 to the maximum rank. The *node incidence* is the matrix

$$(B^*)_{i,j} = \begin{cases} \pm 1 & x_i^{(0)} \prec x_j^{(r)}, \\ 0 & \text{otherwise.} \end{cases}$$

The rank r can be any $1 \leq r \leq R$.

By utilizing the node incidence matrix, we can directly apply existing neighborhood sampling algorithms developed for graphs to our model. In simplicial complexes, nodes that are part of a higher-order structure are all connected among themselves (see Definition 2.1.3). This ensures that when performing neighborhood sampling, all the cells belonging to the higher-order structures adjacent to a node are included in the selection. This method is efficient because it relies on a single incidence matrix and enables the use of well-optimized, graph-based libraries.

9.4 Evaluation

In this section, we present a comprehensive evaluation of our proposed model, comparing its performance against state-of-the-art models for simplicial complexes. To ensure generalizability, we evaluate the models on datasets from diverse domains. Section 9.4.1 outlines the experimental setup, while Section 9.4.2 discusses the main results, examines the impact of batching on the proposed and simplicial models, and includes an ablation study to analyze the effects of the model components.

9.4.1 Setup

We use several datasets in this study, including Cora [145], Citeseer [73], Pubmed [183], Minesweeper, Amazon Ratings, Roman Empire [165], and US-county-demos [113]. Table 9.1 provides key statistics for each dataset, where the numbers of triangles and tetrahedra correspond to the datasets after their transformation using clique lifting. Cora, Citeseer, and Pubmed are well-known cocitation datasets commonly used to benchmark graph neural networks for node classification tasks. Minesweeper, Amazon Ratings, and Roman Empire are recent heterophilious datasets, characterized by their non-homophilous structure, which poses a challenge for message-passing networks [165]. US-county-demos is a regression dataset where any node attribute can serve as the target label [113]. All datasets are randomly split into training, validation, and test sets, with 50% for training and 25% each for validation and testing. The results are averaged over ten randomly initialized runs. For classification tasks, accuracy is used as the evaluation metric, except for the Minesweeper dataset, where we follow the approach in [165] and use the ROC-AUC metric. For regression tasks, including the US-Birth and US-Migration datasets, we use the mean absolute error as the evaluation metric.

All models are optimized using the Adam optimizer [121]. A hyperparameter search is performed to identify the optimal settings for each dataset. Training is monitored using early stopping, with the best parameters selected based on validation performance. The hyperparameter search is executed on a Linux machine with 32 cores, 256GB of system RAM, and one NVIDIA A6000 GPU with 48GB of GPU memory.

	# of nodes	# of node features	# of edges	# of triangles	# of tetrahedra
Cora	2708	1433	5278	1630	220
Citeseer	3327	3703	4552	1167	255
Pubmed	19717	500	44324	12520	3275
Minesweeper	10000	7	39402	39402	9801
Roman	22662	300	7168	0	0
Amazon	24492	300	93050	110765	64195
US-county-demos	3224	6	9483	6490	225

Table 9.1. Statistics for the different datasets. The number of higher-order simplices is reported for the clique expansion algorithm. It is worth noticing that because of the peculiar structure of the Roman dataset, there are no cliques of three or more nodes, so the simplex obtained is just a graph.

9.4.2 Experiments

Models comparison To assess our model's performance, we compare it with two state-of-the-art models for simplicial complexes: SCN [231] and SCCNN [247]. We also test a variant of our model that substitutes the Mamba backbone with a recurrent neural network based on gated recurrent units [50].

Table 9.3 presents the performance results for the models evaluated in our experiments. TopoMamba generally performs better or is comparable to state-of-the-art simplicial models across all datasets, except for the Roman Empire dataset. The

fact that our model relies solely on graph structures to generate sequences for the Mamba model highlights its ability to effectively capture and leverage topological information. However, the Roman Empire dataset poses a challenge due to its chain-like structure, characterized by an average degree of 2.9 and a diameter of 6824, which makes it difficult for our model to learn effective node representations.

Batching The three models TopoMamba, SCN, and SCCNN are compared both when batching and when using the full-batch approach, where the entire graph is used for training. For our model the batching technique introduced in Section 9.3.2 is used. Since it cannot be adapted for the other models, with them we employ the standard batching approach also described in the same section. In addition to performance, we measure the maximum memory usage during training and the time required for each epoch.

Training times per epoch are reported in Table 9.5. As expected, the batching approach tends to result in longer training times compared to full-batch training. However, it is important to note that the model typically requires fewer training epochs when using batching. Our model also consistently demonstrates faster training times compared to other models while achieving comparable or superior performance.

Additional metrics are detailed in Table 9.2. The results suggest that there is no optimal batching method; for certain datasets, batching offers performance advantages, while for others, full-batch training is more effective. In terms of memory usage, the results confirm that our batching method is an effective technique for enabling large-scale dataset processing with limited memory resources.

Ablation Study: Effect of Model Components An ablation study is also performed to understand the impact of the various components of TopoMamba, including the presence or absence of skip connections and the backward Mamba model, as well as variations in hidden size and the number of blocks. The findings from the ablation study are summarized in Table 9.4. The results indicate that using two layers and a larger hidden dimension generally improves performance. As anticipated, the inclusion of skip connections is particularly beneficial when increasing the number of layers. Furthermore, considering the standard deviation in the results, our model appears robust to minor changes in architecture and hyperparameters.

9.5 Related Work

Recurrent Neural Networks (RNNs) are capable of capturing dependencies across sequences, which makes them suitable for problems involving temporal correlations [177, 186]. However, they face challenges such as vanishing gradients [162] and high computational costs due to their inherently sequential nature, which limits parallelization. State Space Models [81] provide a computationally efficient alternative to RNNs. The Mamba model [79] has recently gained traction for its ability to offer a more flexible state representation while preserving computational efficiency.
This concept is leveraged in [63] to develop a graph recurrent encoding-by-distance

Dataset	Model	Batch Size	Results	GPU Memory [Mb]	Epoch Time [s]
Cora	TopoMamba	128	84.93 ± 1.22	400	11
		full	85.08 ± 1.53	639	10
	SCN	128	84.62 ± 1.46	242	29
		full	84.71 ± 0.85	404	28
	SCCNN	128	85.06 ± 1.51	238	25
		full	85.58 ± 1.53	404	32
Pubmed	TopoMamba	128	88.28 ± 0.59	1546	116
		full	87.61 ± 0.39	4507	22
	SCN	128	87.20 ± 0.46	1520	414
		full	86.59 ± 0.47	2185	286
	SCCNN	128	87.37 ± 0.59	1464	508
		full	86.16 ± 0.63	2194	367
Minesweeper	TopoMamba	128	89.19 ± 0.61	665	48
		full	85.51 ± 0.89	2296	31
	SCN	128	85.77 ± 0.83	691	226
		full	83.02 ± 0.86	2402	409
	SCCNN	128	86.21 ± 0.80	688	184
		full	88.01 ± 0.92	2414	771
Roman	TopoMamba	128	82.55 ± 0.74	488	48
		full	80.59 ± 0.93	1098	109
	SCN	128	67.65 ± 1.31	113	90
		full	70.25 ± 1.22	1151	100
	SCCNN	128	76.47 ± 0.84	113	242
		full	76.83 ± 1.12	1161	332
Amazon	TopoMamba	128	47.88 ± 1.16	2322	633
		full	49.60 ± 0.47	24568	53
	SCN	128	43.52 ± 1.40	2669	2477
		full	43.42 ± 1.32	24887	217
	SCCNN	128	45.10 ± 0.74	2597	3186
		full	$44.96 \pm .1.12$	24807	241

Table 9.2. Table showing the results from the experiments comparing using full batches or batch size of 128. For the US Birth and US Migration datasets lower is better since the metric reported is the mean absolute error.

block, which creates sequences for each node by aggregating features from equidistant nodes, starting from the furthest nodes and ending with direct neighbors. This sequence can then be input into a recurrent neural network, providing an alternative to traditional message-passing architectures. Similarly, the authors in [21] construct sequences by performing random walks of varying lengths from each node and ordering these walks by their lengths, allowing for the generation of arbitrarily long sequences.

Building on these ideas, we explore the creation of sequences when working with simplicial complexes. Topological Deep Learning (TDL) [13, 91, 156] is increasingly recognized for its capacity to model complex relationships among graph nodes, leveraging various topological domains, including simplicial complexes, cellular complexes, hypergraphs, and combinatorial complexes [161].

Many models have been developed to work directly with simplicial complexes. In the SCN model presented in [231] the 0-cell representations are calculated by aggregating the 1-cell and the 2-cell representations by using the adjacency between nodes and edges and nodes and triangles, and by adding a learnable weight matrix. The same is

Datasets	TopoMamba	RNN	SCCNN	SCN
Cora	**86.50 ± 1.03**	84.87 ± 0.89	85.58 ± 1.53	84.71 ± 0.85
Citeseer	**73.96 ± 1.62**	72.53 ± 1.47	73.12 ± 1.62	73.36 ± 1.91
Pubmed	88.44 ± 0.48	88.20 ± 0.50	88.18 ± 0.32	**88.72 ± 0.50**
Minesweeper	89.19 ± 0.61	88.87 ± 0.59	89.02 ± 0.20	**90.32 ± 0.11**
Roman	82.74 ± 0.43	80.79 ± 0.60	**89.15 ± 0.32**	88.79 ± 0.46
Amazon	**48.22 ± 1.01**	47.21 ± 0.86	45.10 ± 0.74	43.52 ± 1.40
US Birth	**0.63 ± 0.02**	0.77 ± 0.04	0.64 ± 0.04	0.71 ± 0.08
US Migration	**0.58 ± 0.03**	0.77 ± 0.05	0.60 ± 0.06	0.75 ± 0.04

Table 9.3. Comparison between the different models. The results are shown as mean and standard deviation over 10 runs. The best results are in bold, while the second best are highlighted in blue. For *US Birth* and *US Migration* lower is better since the metric reported is the mean absolute error.

		Skip connection		No skip connection	
# layers	d_h	With $\mathcal{S}_A^2$	Without $\mathcal{S}_A^2$	With $\mathcal{S}_A^2$	Without $\mathcal{S}_A^2$
1	128	84.55 ± 1.53	84.33 ± 1.74	84.71 ± 1.30	84.36 ± 1.30
	256	85.36 ± 1.32	85.81 ± 1.23	85.35 ± 1.37	85.54 ± 1.00
2	128	85.32 ± 1.36	86.20 ± 1.10	85.95 ± 1.09	85.16 ± 1.16
	256	86.50 ± 1.03	86.15 ± 1.64	85.57 ± 1.11	85.45 ± 1.14

Table 9.4. Ablation study on the Cora dataset. *# layers* refers to the number of Mamba blocks used, while S_A^2 refers to using or not Mamba on the inverse of the sequence as presented in Equation 9.5.

done for the edge and the triangle representations, effectively having the cells of each rank influence every other rank directly. The main limitation of this approach is that it considers only 0, 1, and 2-cells, while simplicial complexes can in theory consider cells of any arbitrary rank. In [247] the authors propose the SCCNN model that learns the representations of the simplices by performing convolutions considering lower and upper adjacencies independently. While this approach can be applied to cells of any rank, when cells differ in rank by more that one, they no longer directly influence each other. As it will be presented shortly our work directly tackles both limitations presented by introducing a model in which cells of any rank directly influence the cells of any other rank.

9.6 Discussion

This chapter introduces a novel architecture for processing simplicial complexes. Our approach first constructs a sequence for each node by ordering and aggregating the representations of neighboring cells according to their rank. Then the Mamba model is used to iteratively update the representations of the nodes in the simplicial complex. The proposed approach eliminates the need to devise a specific higher-

Dataset	Batch Size	TopoMamba [s]	SCN [s]	SCCNN [s]
Cora	128	11	29	25
	full	10	28	32
Citeseer	128	12	18	21
	full	10	21	32
Pubmed	128	116	414	508
	full	22	286	367
Minesweeper	128	48	226	184
	full	31	409	771
Roman	128	48	90	242
	full	109	100	332
Amazon	128	633	2477	3186
	full	53	217	241
US Birth	128	14	79	84
	full	19	212	115
US Migration	128	16	48	47
	full	15	114	94

Table 9.5. Table showing the training times in seconds for an epoch. The results are reported for the studied models both when batching is used and when the whole graph is passed to the model.

order message-passing schema, as it allows cells of different ranks to communicate directly (see Section 9.3.1). The effectiveness of our model is evaluated against several state-of-the-art networks across various datasets, demonstrating that it either outperforms or matches these models while TopoMamba is considerably faster (see Section 9.4.2). Furthermore, the proposed architecture allows for a novel batching strategy relying on the introduced concept of node incidence (see Section 9.3.2) that is easy to implement, with minimal computational overhead during training, thereby enhancing the scalability of the approach.

For future work, TopoMamba can be extended to other topological domains, offering a novel paradigm for information propagation across different structures and potentially addressing the challenge of the increased time complexity in topological neural networks compared to GNNs. For example, this approach could be adapted to cellular and combinatorial complexes, which also exhibit hierarchical relationships among their constituent cells. Additionally, the node incidence matrix introduced in this work is analogous to the incidence matrix used for hypergraphs, suggesting that TopoMamba could be adapted to operate effectively in this domain as well. Additionally, Mamba can potentially allow the development of a single architecture for multiple topological domains (see open problem 6 [156]), by combining in a single sequence multiple topological domains.

Part IV

Conclusions and Future Works

Chapter 10

Conclusions and Future Work

This dissertation addresses fundamental challenges in relational data modeling. Our work tackles two critical areas: *topology learning* and *higher-order network modeling*. Through the development of novel methodologies and frameworks, we not only overcome existing limitations but also extend the scope of TDL –pushing the frontiers of this rapidly advancing field. The research contributions of this thesis are divided into two major parts, each advancing our understanding and capabilities in modeling complex relational structures.

Part II focuses on *latent topology inference*, where we develop techniques to uncover both graph-based and higher-order relational structures. These methods not only advance the methodological part of topology learning but also present practical applicability across various tasks. In particular, **Chapter 3** introduces a novel approach for Missing Data Imputation (MDI) in tabular data, leveraging latent graph structures to improve imputation accuracy and downstream task performance. **Chapter 5** applies TDL to network traffic compression, revealing how higher-order topological structures can capture intricate patterns in network data more effectively than traditional methods. **Chapter 6** introduces topology into goal-oriented semantic communications, showing how topological methods can enhance communication efficiency in Joint Source-Channel Coding (JSCC) tasks. These applications underscore the versatility and potential of *topology learning* in solving real-world challenges across multiple domains.

Part III of the dissertation addresses the challenges of *higher-order network modeling* and the broader issue of standardization within TDL. In **Chapter 7**, we explore fundamental questions in Hypergraph Neural Networks (HNNs) modeling. We introduce novel homophily measures applicable to any higher-order domain, presenting a new tool for a deeper understanding of complex networks. Moreover, we propose solutions to the issue of scalability in HNNs, enabling their application to larger, real-world datasets. This chapter also evaluates the reliability of established hypergraph benchmarking datasets, uncovering potential pitfalls in their evaluation. In **Chapter 8**, we address the urgent need for standardization and accessibility within TDL. We propose *TopoBenchmark*, a framework designed to accelerate progress and facilitate reproducible research. The framework's success was demonstrated through the TDL Challenge (see Section 8.6), which highlighted community engagement and the practical utility of the framework. Finally, **Chapter 9** tackles the challenge of

expanding the notion of "neighborhood" in higher-order topological domains. By combining sequence modeling with TDL, we offer a fresh perspective on information propagation within complex topological structures.

In conclusion, the contributions presented in this thesis provide a deeper understanding of relational data modeling by enabling more effective approaches to topology learning and higher-order network modeling. These advancements not only address existing limitations but also open up new directions for both theoretical development and practical applications within the field. The implications of this work, particularly in the context of TDL, set the stage for further exploration, as discussed in the following section.

10.1 Future Work

This thesis lays the groundwork for several promising research directions in relational data modeling. We outline key areas for further exploration.

The 2nd Topological Deep Learning Challenge at ICML 2024 [26] introduced numerous topological liftings, highlighting the need for systematic evaluation. Future work should focus on comprehensive benchmarking of these topological representations across diverse tasks. This involves assessing various topological domains, identifying their strengths and limitations, and evaluating their performance under different conditions. Such systematic evaluation will provide valuable insights into the most effective topological representations for specific applications.

As discussed in Chapters 7 and 8, there is a growing demand for benchmarking datasets that extend beyond standard node and graph classification tasks. The availability of the TopoBenchmark framework, combined with the liftings from the TDL challenge, presents a unique opportunity to introduce the first natural higher-order datasets and tasks. This development will enable fair evaluation of existing TNNs, facilitating the identification of optimal topological domains and corresponding TNN architectures for specific data and problem.

Building on the topology learning approaches presented in Chapters 4 and 5, a key research direction involves addressing the challenge of selecting the optimal topological domain for a given dataset and task. A more ambitious future goal is to develop a unified architecture capable of modeling cross-domain relational data representations. Such an architecture would potentially remove the need for manual domain selection, simplifying the modeling of complex systems. The transformer architecture, with its capacity to incorporate structural and positional encodings from multiple domains, seems particularly well-suited to this task. Enhancing this architecture with cross-domain topological positional and structural encoders, similar to the approach in [41], would offer a versatile solution that can seamlessly support a wide range of downstream applications. This direction could be further supported by the TopoBenchmark framework, providing a platform.

By pursuing these research avenues, we can advance our understanding of topological representations and their applications in machine learning, paving the way for more sophisticated and versatile models for complex systems analysis. The integration of comprehensive benchmarking, novel datasets, and unified architectures could drive significant progress in the field of relational data modeling.

Appendix A

EGG-GAE: scalable graph neural networks for tabular data imputation

Contents

A.1 Architectural experiments

We perform architectural experiments over the numerical datasets (reported in Table 3.1). The results are averaged across five runs and presented as unified average rankings based on end-to-end accuracy, RMSE and MAE. In Section A.1.1 we analyse the proposed homophily penalization term, evaluate different GNN heads in Sec. A.1.2. The effect of extra manipulation of the embedding space obtained by the node projector is investigated in Sec. A.1.3. Examine the impact of varying the number of neighbors sampled per node in the restricted sampling scheme of the k-EGG-GAE model in Section A.1.4.

A.1.1 Homophily experiment

Graph homophily is the tendency for nodes to connect with other nodes in the same class. We argue that boosting γ in Eq.3.13 enhances the sampling scheme of the EGG-GAE model by restricting the sampling of non-homophilic neighbors. We further investigate the influence of the proposed homophily loss adapted to EGG-GAE model. Fig. A.1 demonstrates that, on average, using the homophily regularisation term is beneficial. Increasing the regularisation hyperparameter γ

results in an improved unified solution on average. Although high penalization improves the performance, the variation of EGG-GAE-5 indicates that the performance enhancement has plateaued.

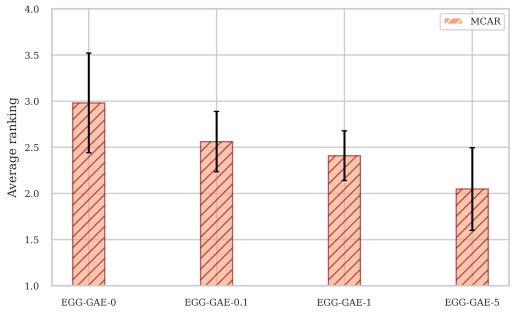


Figure A.1. Unified average ranking computed for the MCAR scenario. A value following the model name indicates the regularization hyperparameter γ.

A.1.2 Heads experiment

Here we inspect the performance change under different GNNs heads. We explore four heads: GCN [122], EdgeConv [225], ARMAConv [28] and SGConv [229]. As can be seen in Fig. A.2, ARMAConv and EdgeConv on average perform better than GCNConv and SGConv, which achieve roughly the same results, further improving the results from Section 3.4.1. We hypothesize a potential explanation of ArmaConv and EdgeConv superior performance compared to GCN and SGConv as follows. ArmaConv is more resistant to noise, which increases its resilience to incorrectly sampled connectivity, while EdgeConv intrinsically weights the contribution of each neighbor, providing additional noise resistance and reducing the contribution of not similar examples (which were sampled due to stochasticity) for concrete datum prediction. As a result, an additional filter is applied to the sampled nodes.

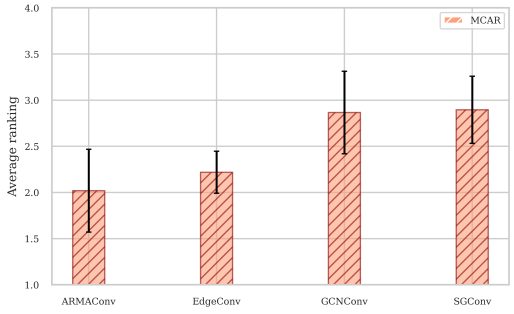


Figure A.2. Unified average ranking computed for the MCAR scenario. The model name indicates the head of EGG-GAE model.

A.1.3 Metric learning experiment

In this part, we investigate the possibility of influencing the embedding space acquired by the node projector. We add additional regularization on the embeddings $\mathrm{H}_{\mathrm{k}}^{g}$ obtained by Eq. 3.2 using triplet loss [182] which is calculated as:

$$\mathcal{L}_t = \eta \sum_{i}^{T} \left[\|\mathrm{h}_i^a - \mathrm{h}_i^+\|^2 + \|\mathrm{h}_i^a - \mathrm{h}_i^-\|^2 + \mathrm{m} \right] \tag{A.1}$$

where m is a margin and equal to 0.05, η is a regularization hyperparameter and $\{\mathrm{h}_i^{\mathrm{a}}, \mathrm{h}_i^+, \mathrm{h}_i^-\}_i^{\mathrm{T}}$ are the triplets formed from embeddings $\mathrm{H}_{\mathrm{k}}^{\mathrm{g}}$ forcing the homophily by selecting $\mathrm{h}_i^{\mathrm{a}}$ and h_i^+ from the same class and h_i^- from the other. We mine the triplets with distance weighted margin-based approach [228]. Fig. A.3 demonstrates applying further regularization on node embedding space can lead to a better solution; nevertheless, the scale parameter η has to be carefully chosen, since high values of η result in suboptimal solutions.

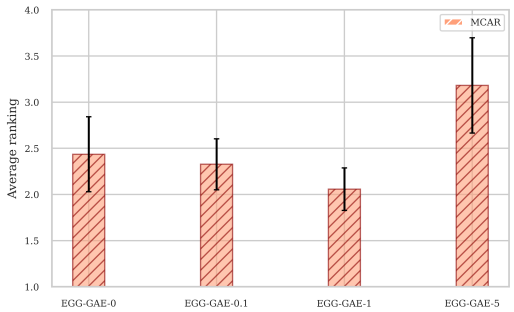


Figure A.3. Unified average ranking computed for the MCAR scenario. A value following the model name indicates the regularization hyperparameter η.

A.1.4 Restricted sampling

In this section, we investigate the restrictive sampling procedure by varying the number of sampled neighbors k per node. Figure A.4 demonstrates the corresponding experiment, where the model k-EGG-GAE-0 is a model which has only self-nodes. Models that rely on the sampled neighborhood consistently outperform models with only self-nodes in terms of MDI solution and predictive accuracy. Next, we observe that increasing the number k of sampled neighbors improves the performance on average, and that the optimal number of neighbors is 3. In addition, as the number of sampled neighbors increases, both the average ranking and the variation increase. We hypothesise that this indicates that as the number of sampled neighbors k increases, so does the proportion of noisy neighbors, which degrades performance.

A.2 Assumptions and limitations

1) The pairwise calculation in the sampling procedure of the EGG block requires n^2 operations, so increasing the batch size results in a quadratic increase in training/inference time. In addition, we believe that the procedure could generally be

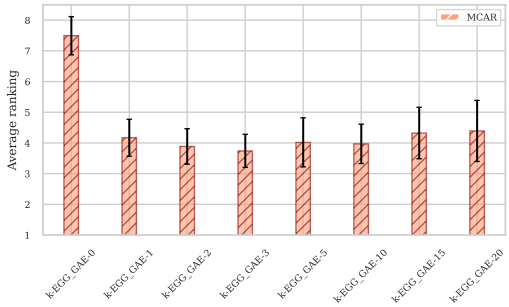


Figure A.4. Unified average ranking computed for the MCAR scenario. A value following the model name indicates the number of neighbors sampled per node.

replaced with a learnable block. In fact, we believe that the sampling process should be iterable, such that the first iteration provides an initial approximation of the neighborhood and subsequent iterations eliminate noisy neighbors.

2) In the chapter, we rely on a pretty simple graph construction approach: from the sampled batch X, we construct for each node its neighborhood and pass the obtained graph through a GNN head. There are a number of modifications that will allow for better gradients, resulting in a better solution.

Appendix B

From Latent Graph to Latent Topology Inference

Contents

B.1 Reproducibility

We include all the details about our experimental setting, including the choice of hyperparameters and the specifications of our machine, in Appendix B.5. We provide all the code, data splits, and virtual environment needed to replicate the experiments at the following anonymized repository: https://github.com/spindro/differentiable_cell-complex_module

B.2 Training Procedure

Here we discuss the key concepts about the training of the DCM. We detail how to back-propagate through the entmax-based discrete sampling at the 1-cell (edge) level (in the $\alpha-$DGM) and at the 2-cell (polygon) level (in the PIM). We follow the approach proposed by[117] and introduce a supplementary loss that rewards edges and polygons involved in a correct classification and penalizes the ones which result in misclassification. We define the reward function:

$$\delta\left(y_i, \hat{y}_i\right) = \mathbb{E}\left(a_i\right) - a_i\,, \tag{B.1}$$

as the difference between the average accuracy of the i-th sample and the current prediction accuracy, where y_i and $\hat{y}_i$ are the predicted and true labels, and $a_i = 1$ if $y_i = \hat{y}_i$ or 0 otherwise. We then define the loss associated to the edge sampling as follows:

$$L_{GL} = \sum_{i=1}^{N} \left(\delta\left(y_i, \hat{y}_i\right) \sum_{j=1}^{N} \mathbf{p}_i(j) \right) \tag{B.2}$$

We estimate $\mathbb{E}\left(a_i\right)$ with an exponential moving average on the accuracy during the training process

$$\mathbb{E}\left(a_i\right)^{(t+1)} = \mu \mathbb{E}\left(a_i\right)^{(t)} + (1-\mu) a_i \,, \tag{B.3}$$

with $\mu = 0.9$ in all our experiments and $\mathbb{E}\left(a_i\right)^{(0)} = \frac{1}{\#\text{ classes}}$.

Similarly, the loss associated with the polygon sampling is defined as follows:

$$L_{PL} = \sum_{i=1}^{N} \left(\delta\left(y_i, \hat{y}_i\right) \sum_{j \in \mathcal{P}_i} \mathbf{p}(j) \right) , \tag{B.4}$$

where $\mathcal{P}_i$ is the set of indices of the polygons for which node i is a vertex. Following [58], the accuracy can be replaced with the R2 score in the case of regression tasks. In every experiment, we set the initial value of α to 1.5.

B.3 Computational Complexity

We carry out a complexity analysis of the proposed architecture with a particular focus on the differences with respect to the DGM [117], which are:

(a) the introduction of α-entmax to sample cells compared to the Gumbel-Softmax sampler;

(b) the need to search for all cycles up to length K_{max} in the skeleton $\mathcal{G}_{out}$;

(c) the sampling operation over the cycles;

(d) the cell complex neural network;

For (a), the solution to equation 4.3 cannot be expressed in closed form except for specific values of α [53], but an ε-approximate solution can be obtained in $\mathcal{O}(1/\log \varepsilon)$ time with a bisection algorithm [29].

For (b), we leverage the algorithm in [31], which has complexity $\mathcal{O}((E + N\widetilde{P})\text{polylog}(N))$, where N is the number of vertices, E the number of sampled edges, and $\widetilde{P}$ the number of cycles induced by the skeleton up to length K_{max}. Note that in our implementation the cell complex is recomputed for each iteration, but this computation can be amortized across epochs, approximated with stochastic search algorithms, or leveraging more flexible topological spaces, e.g. Combinatorial Complexes.

For (c), the complexity is the same as (a), which is approximately linear in the number $\widetilde{P}$ of cycles.

For (d), the complexity of message-passing is also approximately linear in the size of the cell complex, due to the fact that we consider cells of a constant maximum dimension and boundary size [31].

Table B.1. Execution times expressed in Seconds.

		Graph MP	Graph Sampling	Lifting	Cell Sampling	Cell MP
Cora	GCN	$5e^{-5}$	-	-	-	-
	GCN+CCCN	-	-	$2e^{-1}$	-	$2e^{-4}$
	DGM	$6e^{-4}$	$4e^{-3}$	-	-	-
	DCM	$1e^{-5}$	$5e^{-3}$	$1e^{-1}$	$2e^{-2}$	$5e^{-4}$
Texas	GCN	$2e^{-5}$	-	-	-	-
	GCN+CCCN	-	-	$1e^{-2}$	-	$2e^{-4}$
	DGM	$3e^{-5}$	$9e^{-4}$	-	-	-
	DCM	$1e^{-5}$	$9e^{-4}$	$3e^{-2}$	$7e^{-2}$	$4e^{-5}$

As a further empirical analysis, in Table B.1 we break down the actual inference execution times on two reference datasets, Cora and Texas, for GCN, GCN+CCCN, DGM, and DCM, based on the (macro-)operations they require. As we can see from this empirical analysis and as we could expect from the above complexity analysis, the lifting operation (computing the cycles and lifting the node embeddings) and cell sampling (computing the similarities among edge embeddings and applying the α-entmax) are the main computational bottlenecks. The lifting operation needs to be performed in every complex-based architecture (we show DCM and GCN+CCCN, but for CWN would be the same). Architectures that do not perform LTI (CWN or GCN+CCCN) can mitigate this problem (only) on transductive tasks by computing the cycles offline. In our setting, additional solutions w.r.t. the ones presented in (b) could be accumulating the gradients and inferring the graph/cell-complexes once every t iteration rather than after every optimization step. The cell sampling bottleneck, unlike the lifting, is not given by any technical requirement, but it is just related to our actual implementation. To keep the code readable and reproducible for (mainly) academic purposes, we use basic data structures and functions, e.g. we store the cells in lists that we parse. However, the bottleneck could be mitigated by optimizing the code. We will soon update our repo.

B.4 Additional Results

We present a series of additional experiments and ablation studies to further validate the effectiveness of the DCM. In particular, in Section B.4.1 we investigate the advantages of employing the α-entmax as sampling strategy w.r.t. the Gumbel Top-k trick used in the DGM [58, 117]. Additionally, in Section B.4.2, we study the impact of the number of MP layers in equation 4.7-equation 4.4 (implemented, as explained in Section 4.4 of the body and Appendix B.5, with GCNs and CCCNs). Finally, in Section B.4.3, we investigate the usage of different values for the maximum cycle size K_{max}.

B.4.1 Sampling strategies

In this section, we assess the impact of employing the α-entmax as sampling strategy, compared to the Top-k sampler utilized in [117, 58], for both homophilic (Table B.2) and heterophilic datasets (Table B.3), with and without input graph.

Table B.2. Comparison among sampling strategies on homophilic graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

		Cora	CiteSeer	PubMed	Physics	CS
w/o graph	DCM	**78.80** ± 1.84	**76.47** ± 2.45	87.38 ± 0.91	96.45 ± 0.12	**95.40** ± 0.40
	DCM ($\alpha = 1$)	78.73 ± 1.99	76.32 ± 2.75	**87.47** ± 0.77	96.2 ± 0.27	95.35 ± 0.37
	Top-k DCM	76.16 ± 3.71	73.21 ± 2.73	87.13 ± 1.22	**96.51** ± 0.49	95.25 ± 0.08
	Top-k DCM All	76.24 ± 2.84	73.45 ± 3.12	87.38 ± 1.04	96.49 ± 0.64	95.17 ± 0.07
w graph	DCM	**85.78** ± 1.71	**78.72** ± 2.84	88.49 ± 0.62	**96.99** ± 0.44	**95.79** ± 0.48
	DCM ($\alpha = 1$)	85.97 ± 1.86	78.60 ± 3.16	**88.61** ± 0.69	96.69 ± 0.46	94.81 ± 0.49
	Top-k DCM	76.96 ± 3.46	75.50 ± 2.15	86.76 ± 0.89	96.19 ± 0.33	94.74 ± 0.28
	Top-k DCM All	77.55 ± 3.18	75.66 ± 2.67	86.64 ± 0.74	96.21 ± 0.34	94.79 ± 0.37

Table B.3. Comparison among sampling strategies on heterophilic graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

		Texas	Wisconsin	Squirrel	Chameleon
w/o graph	DCM	**85.71** ± 7.87	**87.49** ± 5.94	**35.55** ± 2.24	53.63 ± 3.07
	DCM ($\alpha = 1$)	84.96 ± 10.24	86.72 ± 6.02	35.25 ± 2.22	**53.67** ± 3.19
	Top-k DCM	84.21 ± 8.21	84.10 ± 6.60	33.90 ± 1.38	52.88 ± 4.24
	Top-k DCM All	82.71 ± 11.25	85.18 ± 6.89	33.80 ± 1.47	53.23 ± 3.86
w graph	DCM	84.87 ± 10.04	**86.33** ± 5.14	34.95 ± 2.59	53.05 ± 3.00
	DCM ($\alpha = 1$)	**84.96** ± 5.60	85.36 ± 5.05	**35.13** ± 2.27	**53.76** ± 3.72
	Top-k DCM	84.61 ± 8.47	82.87 ± 7.59	33.77 ± 0.91	52.84 ± 4.24
	Top-k DCM All	84.66 ± 9.33	82.10 ± 6.64	33.52 ± 1.08	51.61 ± 4.07

We compare the two variants of the DCM (again denoted with DCM and DCM ($\alpha = 1$)) against our same architecture but with Gumbel Top-k sampler in place of the $\alpha-$entmax, both when explicit sampling is performed at edge and polygons level (denoted with Top-k DCM) and when only the edge are sampled and all the polygons are taken (the counterpart of DCM ($\alpha = 1$), denoted with Top-K DCM All). Our investigation reveals that the capability of α-entmax of generating non-regular topologies leads to significant performance gains on almost all the tested datasets.

B.4.2 Number of message-passing Layers

In this section, we assess the impact of the number of MP layers on the performance of our model. In Table B.4 (for homophilic datasets) and Table B.5 (for heterophilic datasets), we show the accuracy as a function of the number of MP layers, i.e. GCNs and CCCNs layers at node and edge levels, respectively. We notice that the determination of an optimal number of message-passing layers is not governed by a universal rule but rather depends on the characteristics of the dataset under consideration. However, we observe that employing a single message-passing layer consistently yields favorable performance across most of the datasets, further confirming that integrating higher-order information is beneficial. Moreover, the 1-layer configuration maintains a comparable number of trainable parameters w.r.t. the DGM-M [58] settings, whose results are reported as a comparison. In particular, DGM-M employs 3-layer GCNs while we employ 1-layer GCNs and 1-layer CCCNs

Table B.4. Results varying the number of MP layers on homophilic graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

	# MP layers	Cora	CiteSeer	PubMed	Physics	CS
w/o graph	1	**78.80** ± 1.84	**76.47** ± 2.45	87.38 ± 0.91	96.45 ± 0.12	**95.40** ± 0.40
	2	76.90 ± 2.81	75.86 ± 2.35	87.77 ± 0.53	**97.10** ± 0.15	94.81 ± 0.40
	3	74.98 ± 4.20	74.59 ± 2.35	**88.10** ± 0.45	97.09 ± 0.11	94.15 ± 0.63
	4	69.74 ± 3.95	71.41 ± 2.47	87.79 ± 0.36	96.80 ± 0.08	93.54 ± 0.72
w graph	1	85.78 ± 1.71	**78.72** ± 2.84	88.49 ± 0.62	96.99 ± 0.44	**95.79** ± 0.48
	2	**88.89** ± 1.63	78.42 ± 2.63	88.27 ± 0.54	**97.03** ± 0.23	94.50 ± 0.55
	3	88.07 ± 2.26	76.95 ± 2.84	**88.97** ± 0.61	97.01 ± 0.12	94.47 ± 0.54
	4	86.00 ± 2.21	76.62 ± 2.86	88.62 ± 0.76	96.86 ± 0.19	93.66 ± 0.51

(having 3 times the number of parameters of a single GCN layer), see Appendix B.5 for details. For this reason, despite the fact that some results in Table B.4 and in Table B.5 are better than the ones reported in the body of the paper, we decided to show the ones that correspond to architectures whose number of trainable parameters are as similar as possible to the reported competitors.

B.4.3 Maximum Cycle Size

In this section, we conduct an ablation study on the maximum length K_{max} of induced cycles taken in consideration to sample the polygons of the latent cell complex. In Table B.6 and in Table B.7, we show the accuracy as a function of K_{max}, respectively. As for the choice of the number of MP layers, even the optimal K_{max} varies based on the specific dataset, however with top performance obtained in most cases when $K_{max} = 4$.

B.5 Model Architecture

In this appendix, we present a detailed description of the architectures employed to obtain the results in Table 4.1 and Table 4.2. To ensure uniformity and show the performance gain without intensive ad-hoc hyperparameters tuning (as the ones performed for the DGM [117] and the DGM-M [58]), we maintained a constant configuration for the number of layers, hidden dimensions, activation functions, K_{max}

Table B.5. Results varying the number of MP layers on heterophilic graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

	# MP layers	Texas	Wisconsin	Squirrel	Chameleon
w/o graph	1	85.71 ± 7.87	**87.49** ± 5.94	**35.55** ± 2.24	**53.63** ± 3.07
	2	**87.89** ± 10.24	84.80 ± 4.81	33.30 ± 1.67	52.22 ± 4.22
	3	82.02 ± 8.97	83.64 ± 4.87	33.61 ± 2.55	51.34 ± 3.25
	4	80.75 ± 10.80	79.49 ± 7.45	31.96 ± 2.56	47.82 ± 3.11
w graph	1	**84.87** ± 10.04	**86.33** ± 5.14	**34.95** ± 2.59	**53.05** ± 3.00
	2	77.34 ± 8.64	76.31 ± 4.92	33.53 ± 2.24	52.61 ± 2.73
	3	77.34 ± 8.09	80.95 ± 6.52	33.18 ± 1.88	51.47 ± 2.80
	4	74.76 ± 12.01	76.13 ± 7.43	33.45 ± 2.59	43.65 ± 3.07

Table B.6. Results varying K_{max} on homophilic graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

	K_{max}	Cora	CiteSeer	PubMed	Physics	CS
w/o graph	3	78.38 ± 2.50	75.56 ± 1.45	87.44 ± 0.91	96.32 ± 0.28	**95.41** ± 0.60
	4	**78.80** ± 1.84	**76.47** ± 2.45	87.38 ± 0.91	96.45 ± 0.12	95.40 ± 0.40
	5	78.60 ± 2.60	75.62 ± 1.26	**87.54** ± 0.98	96.20 ± 0.28	95.39 ± 0.60
w graph	3	85.54 ± 2.40	78.38 ± 1.56	**88.50** ± 0.78	97.01 ± 0.28	95.68 ± 0.63
	4	**85.78** ± 1.71	**78.72** ± 2.84	88.49 ± 0.62	96.99 ± 0.44	**95.79** ± 0.48
	5	85.54 ± 2.43	78.32 ± 1.47	85.75 ± 1.02	**97.07** ± 0.40	95.68 ± 0.56

Table B.7. Results varying K_{max} on heterophilic graph node classification benchmarks. Test accuracy in % averaged over 10 splits.

	K_{max}	Texas	Wisconsin	Squirrel	Chameleon
w/o graph	3	85.53 ± 9.85	84.80 ± 6.29	35.15 ± 1.76	53.41 ± 3.30
	4	**85.71** ± 7.87	**87.49** ± 5.94	**35.55** ± 2.24	**53.63** ± 3.07
	5	80.22 ± 11.54	85.57 ± 7.19	34.98 ± 2.15	53.01 ± 3.73
w graph	3	**85.49** ± 7.89	80.95 ± 7.36	33.11 ± 2.58	52.70 ± 3.21
	4	84.87 ± 10.04	**86.33** ± 5.14	**34.95** ± 2.59	**53.05** ± 3.00
	5	84.21 ± 9.15	81.71 ± 6.33	33.88 ± 1.49	52.92 ± 3.30

(4), (pseudo-)similarity functions (minus the euclidean distances among embeddings), dropout rates (0.5), and learning rates (0.01) across all datasets. The architecture details are shown in Table B.8. We conducted training for a total of 200 epochs for the homophilic datasets, with the exception of the physics dataset, which underwent 100 epochs like the heterophilic datasets. Our experiments were performed using a single NVIDIA RTX A6000 with 48 GB of GDDR6 memory. As mentioned in Section 4.4, in every experiment we employ Graph Convolutional Neural Networks (GCNs) and Cell Complex Convolutional Neural Network (CCCNs) as specific MP architectures at node and edge levels, respectively. We give a brief description of GCNs and CCCNs in the following.

Table B.8. Model and DCM Architecture.

Model Architecture			DCM Architecture		
No. Layer param.	Activation	Layer Type	No. Layer param.	Activation	Layer Type
(no. input features, 32)	ReLU	Linear	(32, 32)	ReLU	Linear
		DCM	(32, 32)	ReLU	Graph Conv
(32, 32)	ReLU	Graph Conv	(32, 32)	ReLU	Graph Conv
(32, 32)	ReLU	Cell Conv	(32, 32)	None	Graph Conv
(64, no. classes)	Softmax	Linear			

Remark 5. The DCM has not to be considered a novel GNN architecture that aims to achieve SOTA performance on graph benchmarks. To the best of our knowledge, DCM is the first model that permits learning of higher-order latent interactions while overcoming many of the LGI's (and TDL's) limitations and achieving and matching SOTA results across multiple datasets. In addition, please note that the majority of GNN and TNN architectures could be integrated into DCM as backbones at the node and edge levels as instead of the employed GCN and CCCN.

B.5.1 Graph and Cell Complex Convolutional Neural Networks

Graph Convolutional Neural Networks (GCNs) are one of the most famous and simple GNNs architectures. In GCNs, the output features of a node are computed as a weighted sum of linearly transformed features of its neighboring nodes. Therefore, the MP round in equation 4.4 is implemented as:

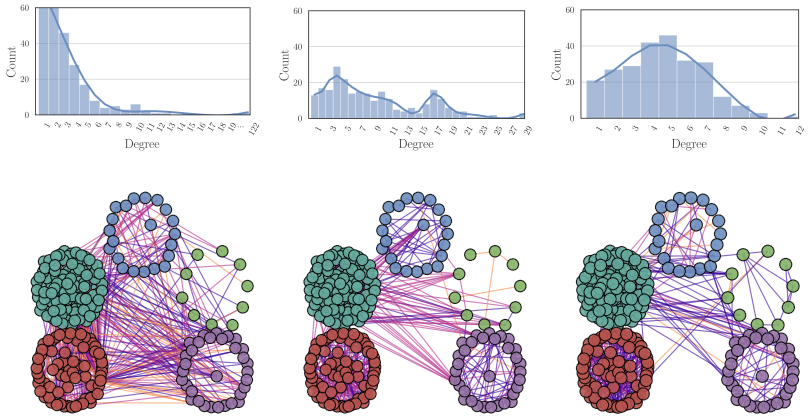
$$\mathbf{x}_{0,int}(i) = \gamma_0 \left(\sum_{j \in \mathcal{N}_u(\sigma_i^0)} a_{i,j} \mathbf{x}_{0,in}(j) \mathbf{W} \right), \tag{B.5}$$

where $\mathbf{W}$ are learnable parameters and the (normalized) weights $a_{i,j}$ are set as $a_{i,j} = \frac{1}{\sqrt{d_j d_i}}$, with d_i and d_j being the degrees of node i and node j, respectively [122].

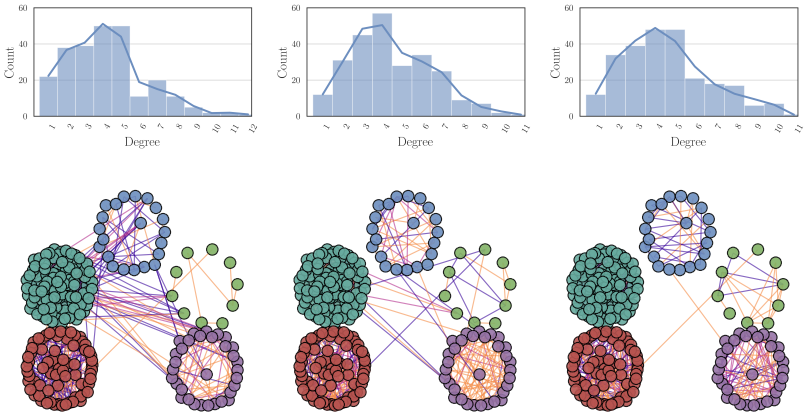
Cell Complex Convolutional Neural Networks (CCCNs) generalize GCNs to cell complexes using the adjacencies introduced in Definition 1. In our case, the output features of an edge are computed as a weighted sum of linearly transformed features of its neighboring edges, over the upper and lower adjacencies. Therefore, in this paper, we implement the MP round in equation 4.7 as:

$$\mathbf{x}_{1,out}(i) = \gamma_1 \left(\sum_{j \in \mathcal{N}_u(\sigma_i^1)} a_{u,i,j} \mathbf{x}_{1,int}(j) \mathbf{W}_u + \sum_{j \in \mathcal{N}_d(\sigma_i^1)} a_{d,i,j} \mathbf{x}_{1,int}(j) \mathbf{W}_d + \mathbf{x}_{1,int}(j) \mathbf{W} \right), \tag{B.6}$$

where $\mathbf{W}_u$, $\mathbf{W}_d$, and $\mathbf{W}$ are learnable parameters. The weights are normalized with the same approach of GCNs, with upper (for the $a_{u,i,j}$s) and lower (for the $a_{d,i,j}$s) degrees. The skip connection $\mathbf{x}_{1,int}(j)\mathbf{W}$ is as usual beneficial, and in the case of CCCNs it has a further theoretical justification in terms of Hodge Decomposition and signal filtering, see [171, 179] for further details.



(a) Input, $\%_p = 100$, $h = 0.19$ **(b)** Epoch 10, $\%_p = 75$, $h = 0.70$ **(c)** Epoch 50, $\%_p = 51$, $h = 0.89$



(d) Epoch 100, $\%_p = 55$ $h = 0.96$ **(e)** Epoch 150, $\%_p = 36$, $h = 0.98$ **(f)** Epoch 200, $\%_p = 18$, $h = 0.99$

Figure B.1. Evolution of the latent complex for the Wisconsin dataset, along with homophily level and nodes degree distribution. Edges in orange, triangles in lilac, squares in purple ($K_{max} = 4$)

Appendix C

Hypergraph Neural Networks through the Lens of Message Passing: A Common Perspective to Homophily and Architecture Design

Contents

C.1 Extended Related Works on Hypergraph Neural Networks

Numerous machine-learning techniques have been developed for processing hypergraph data. One commonly used approach in early literature is to transform the hypergraph into a graph through clique expansion (CE). This technique involves substituting each hyperedge with an edge for every pair of vertices within the hyperedge, creating a graph that can be analyzed using graph-based algorithms [2, 262, 261, 137].

Several techniques have been proposed that use Hypergraph Neural Networks (HNNs) for semi-supervised learning. One of the earliest methods extends the graph convolution operator by incorporating the normalized hypergraph Laplacian [71]. As pointed out in Dong et al. [64], spectral convolution with the normalized Laplacian corresponds to performing a weighted CE of the hypergraph. HyperGCN [243] employs mediators for incomplete CE on the hypergraph, which reduces the number of edges required to represent a hyperedge from a quadratic to a linear number of edges. The information diffusion is then carried out using spectral convolution for hypergraph-based semi-supervised learning. Hypergraph Convolution and Hypergraph Attention (HCHA) [10] employs modified degree normalizations and attention weights, with the attention weights depending on node and hyperedge features.

CE may cause the loss of important structural information and result in suboptimal learning performance [99, 49]. Furthermore, these models typically obtain the best performance with shallow 2-layer architectures. Adding more layers can lead to reduced performance due to oversmoothing [106]. In the recent study Chen and Zhang [43], an attempt was made to address oversmoothing in this type of network by incorporating residual connections; however, the method still relies on using hypergraph Laplacians to build a weighted graph through clique expansion. Another method presented in Yang et al. [245] introduces a new hypergraph expansion called line expansion (LE) that treats vertices and hyperedges equally. The LE bijectively induces a homogeneous structure from the hypergraph by modeling vertex-hyperedge pairs. In addition, the LE and CE techniques require significant computational resources to transform the original hypergraph into a graph and perform subsequent computations, hence making the methods unpractical for large hypergraphs.

Another line of research explores hypergraph modeling involving a two-stage procedure: information is transmitted from nodes to hyperedges and then back from hyperedges to nodes [227, 252, 64, 6, 106, 244]. This procedure can be viewed as a two-step message passing mechanism. HyperSAGE [6] is a prominent early example of this line of research allowing transductive and inductive learning over hypergraphs. Although HyperSAGE has shown improvement in capturing information from hypergraph structures compared to spectral-based methods, it involves only one learnable linear transformation and cannot model arbitrary multiset function [49]. Moreover, the algorithm utilizes nested loops resulting in inefficient computation and poor parallelism.

UniGNN [106] addresses some of these limitations by using a permutation-invariant function to aggregate vertex features within each hyperedge in the first

stage and using learnable weights only during the second stage to update each vertex with its incident hyperedges. One of the variations of UniGNN, called UniGCNII addresses the oversmoothing problem, which is common for most of the methods described above. It accomplishes this by adapting GCNII [44] to hypergraphs. The AllSet method, proposed in Chien et al. [49], employs a composition of two learnable multiset functions to model hypergraphs. It presents two model variations: the first one exploits Deep Set [259] and the second one Set Transformer [134]. The AllSet method can be seen as a generalization of the most commonly used hypergraph HNNs [243, 71, 10, 64, 6]. More implementation details and detailed drawbacks discussion can be found in Section 7.3.1. Although AllSet achieves state-of-the-art results, it suffers from the drawbacks of the message passing mechanism, including the local receptive field, resulting in a limited ability to model long-range interactions [82, 11]. Two additional issues are poor scalability to large hypergraph structures and oversmoothing that occurs when multiple layers are stacked.

Finally, we would like to mention two related papers that put the focus on hyperedge-dependent computations. On the one hand, EDHNN [222] incorporates the option of hyperedge-dependent messages from hyperedges to nodes; however, at each iteration of the message passing it aggregates all these messages to generate a unique node hidden representation, and thus it doesn't enable to keep different hyperedge-based node representations across the whole procedure –as our MultiSetMixer does. On the other hand, the work Aponte et al. [5] does allow multiple hyperedge-based representations across the message passing, but the theoretical formulation of this unpublished paper is not clear and rigorous, and the evaluation is neither reproducible nor comparable to other hypergraph models. Hence, we argue that our MultiSet framework represents a step forward by rigorously formulating a simple but general MP framework for hypergraph modeling that is flexible enough to deal with hyperedge-based node representations and residual connections, demonstrating as well that it generalizes previous hypergraph and graph models.

C.2 Details of the Implemented Methods

We provide a detailed overview of the models analyzed and tested in this work. In order to make their similarities and differences more evident, we express their update steps through a standard and unified notation.

Notation. A hypergraph with n nodes and m hyperedges can be represented by an incidence matrix $\boldsymbol{H} \in \mathbb{R}^{n \times m}$. If the hyperedge e_j is incident to a node v_i (meaning $v_i \in e_j$), the entry in the incidence matrix $H_{i,j}$ is set to 1. Instead, if $v_i \notin e_j$, then $H_{i,j} = 0$.

We denote with $\boldsymbol{W}$ and $\boldsymbol{b}$ a learnable weight matrix and bias of a neural network, respectively. Generally, $\boldsymbol{x}_v$ and $\boldsymbol{z}_e$ are used to denote features for a node v and a hyperedge e respectively. Stacking all node features together we obtain the node feature matrix $\boldsymbol{X}$, while $\boldsymbol{Z}$ is instead the hyperedge feature matrix. $\sigma(\cdot)$ indicates a nonlinear activation function (such as ReLU, ELU or LeakyReLU) that depends on the model used. Finally, we use $\|$ to denote concatenation.

C.2.1 AllSet-like models

This Section addresses the models that are covered in the AllSet unified framework introduced in 7.3.1, and that can potentially be expressed as particular instances of equations 7.5 and 7.7. For a detailed proof of the claim for most of the following models, refer to Theorem 3.4 in Chien et al. [49].

CEGCN / CEGAT. As introduced in the previous Sections, the CE of a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a weighted graph obtained from $\mathcal{G}$ with the same set of nodes. In terms of incidence matrix, it can be described as $\boldsymbol{H}^{(CE)} = \boldsymbol{H}\boldsymbol{H}^T$ [49]. A one-step update of the node feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times f}$ can be expressed both in a compact way as $\boldsymbol{H}^{(CE)}\boldsymbol{X}$ or directly as a node-level update rule, as

$$\boldsymbol{x}_v^{(t+1)} = \sum_{e \in \mathcal{E}_v} \sum_{u : u \in e} \boldsymbol{x}_v^{(t)}. \tag{C.1}$$

Some types of hypergraph convolutional layers in the literature adopt a CE-based propagation, for example generalizing popular graph-targeting models such as Graph Convolutional Networks [124] and Graph Attention Networks [214].

HNN. Before describing how HNN [71] works, it is necessary to define some notation. Let $\boldsymbol{H}$ be the hypergraph's incidence matrix. Suppose that each hyperedge $e \in \mathcal{E}$ is assigned a fixed positive weight z_e, and let $\boldsymbol{Z} \in \mathbb{R}^{m \times m}$ now denote the matrix stacking all these weights in the diagonal entries. Additionally, the vertex degree is defined as

$$d_v = \sum_{e \in \mathcal{E}_v} z_e, \tag{C.2}$$

while the hyperedge degree, instead, is

$$b_e = \sum_{v : v \in e} 1. \tag{C.3}$$

The degree values can be used to define two diagonal matrices, $\boldsymbol{D} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m \times m}$.

The core of the hypergraph convolution introduced in Feng et al. [71] can be expressed as

$$\boldsymbol{x}_v^{(t+1)} = \sigma\left(\sum_{e \in \mathcal{E}_v} \sum_{u : u \in e} z_e \boldsymbol{x}_v^{(t)} \boldsymbol{W}^{(t)} \right), \tag{C.4}$$

where σ is a non-linear activation function like LeakyReLU and ELU, and $\boldsymbol{W}^{(t)} \in \mathbb{R}^{f^{(t)} \times f^{(t+1)}}$ is a weight matrix between the (t)-th and $(t+1)$-th layer, to be learnt during training. Note that in this case the dimensionality of the node feature vectors $f^{(t)}$ can be layer-dependent.

The update step can be rewritten also in matrix form as

$$\boldsymbol{X}^{(t+1)} = \sigma(\boldsymbol{H}\boldsymbol{Z}\boldsymbol{H}^T\boldsymbol{X}^{(t)}\boldsymbol{W}^{(t)}), \tag{C.5}$$

where $\boldsymbol{X}^{(t+1)} \in \mathbb{R}^{n \times f^{(t+1)}}$ and $\boldsymbol{X}^{(t)} \in \mathbb{R}^{n \times f^{(t)}}$.

In practice, a normalized version of this update procedure is proposed. The matrix-based formulation allows to clearly express the symmetric normalization that is actually put in place through the vertex and hyperedge degree matrices $\boldsymbol{D}$ and $\boldsymbol{B}$ defined above:

$$\boldsymbol{X}^{(t+1)} = \sigma(\boldsymbol{D}^{-1/2}\boldsymbol{H}\boldsymbol{Z}\boldsymbol{B}^{-1}\boldsymbol{H}^T\boldsymbol{D}^{-1/2}\boldsymbol{X}^{(t)}\boldsymbol{W}^{(t)}). \tag{C.6}$$

HCHA. With respect to the previously described models, HCHA [10] uses a different kind of weights that depend on the node and hyperedge features. Specifically, starting from the same convolutional model proposed by Feng et al. [71] and described in Equation C.6, they explore the idea of introducing an attention learning model on $\boldsymbol{H}$.

Their starting point is the intuition that hypergraph convolution as implemented in Equation C.6 implicitly puts in place some attention mechanism, which derives from the fact that the afferent and efferent information flow to vertexes may be assigned different importance levels, which are statically encoded in the incidence matrix $\boldsymbol{H}$, hence depend only on the graph structure. In order to allow for such information on magnitude of importance to be determined dynamically and possibly vary from layer to layer, they introduce an attention learning module on the incidence matrix $\boldsymbol{H}$: instead of maintaining $\boldsymbol{H}$ as a binary matrix with predefined and fixed entries depending on the hypergraph connectivity, they suggest that its entries could be learnt during the training process. The entries of the matrix should express a probability distribution describing the degree of node-hyperedge connectivity, through non-binary and real values.

Nevertheless, the proposed hypergraph attention is only feasible when the hyperedge and vertex sets share the same homogeneous domain, otherwise, their similarities would not be compatible. In case the comparison is feasible, the computation of attention scores is inspired by [214]: for a given vertex v and a hyperedge e, the score is computed as

$$H_{v,e} = \frac{\exp(\sigma(\text{sim}(\boldsymbol{x}_v\boldsymbol{W}, \boldsymbol{z}_e\boldsymbol{W})))}{\sum_{g \in \mathcal{E}_v} \exp(\sigma(\text{sim}(\boldsymbol{x}_v\boldsymbol{W}, \boldsymbol{z}_g\boldsymbol{W})))}, \tag{C.7}$$

where σ is a non-linear activation function, and sim is a similarity function defined as

$$\text{sim}(\boldsymbol{x}_v, \boldsymbol{z}_e) = \boldsymbol{a}^T[\boldsymbol{x}_v \parallel \boldsymbol{z}_e], \tag{C.8}$$

in which $\boldsymbol{a}$ is a weight vector, and the resulting similarity value is a scalar.

HyperGCN. The method proposed by Yadati et al. [243] can be described as performing two steps sequentially: first, a graph structure is defined starting from the input hypergraph, through a particular procedure, and then the well known CGN model [124] for standard graph structures is executed on it. Depending on the approach followed in the first step, three slight variations of the same model can be identified: 1-HyperGCN, HyperGCN (enhancing 1-HyperGCN with so-called *mediators*) and FastHyperGCN.

Before analyzing the differences among the three techniques, we introduce some notation and express how the GCN-update step is performed. Suppose that the input hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is equipped with initial edge weights $\{z_e\}_{e\in\mathcal{E}}$ and node features $\{\boldsymbol{x}_v\}_{v\in\mathcal{V}}$ (if missing, suppose to initialize them randomly or with constant values). Let $\bar{\boldsymbol{A}}^{(t)}$ denote the normalized adjacency matrix associated to the graph structure at time-step (t). The node-level one-step update for a specific node v can be formalized as:

$$\boldsymbol{x}_v^{(t+1)} = \sigma\left(\left(\boldsymbol{W}^{(t)}\right)^T \sum_{u\in\mathcal{E}_v} \bar{A}_{u,v}^{(t)} \cdot \boldsymbol{x}_u^{(t)}\right), \tag{C.9}$$

in which $\boldsymbol{x}_v^{(t+1)}$ is the $(t+1)$-th step hidden representation of node v and $\mathcal{E}_v$ is the set of neighbors of v. For what concerns $\bar{A}_{u,v}^{(t)}$, it refers to the element at index u, v of $\bar{\boldsymbol{A}}^{(t)}$, which can be defined in the following ways according to the method:
1-HyperGCN: starting from the hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a simple graph is defined by considering exactly one representative simple edge for each hyperedge $e \in \mathcal{E}$, and it is defined as (v_e, u_e) such that $(v_e, u_e) = \text{argmax}_{v,u\in e}\|\left(\boldsymbol{W}^{(t)}\right)^T(\boldsymbol{x}_v^{(t)} - \boldsymbol{x}_u^{(t)})\|_2$. This implies that each hyperedge e is represented by just one pairwise edge (v_e, u_e), and this may also change from one step to the other, which leads to the graph adjacency matrix $\bar{\boldsymbol{A}}^{(t)}$ being layer-dependent, too.
HyperGCN: the model extends the graph construction procedure of 1-HyperGCN by also considering *mediator* nodes, that for each hyperedge e consist in $K_e := \{k \in e : k \neq v_e, k \neq u_e\}$. Once the representative edge (v_e, u_e) is determined and added to the newly defined graph, two edges for each mediator are also introduced, connecting the mediator to both v_e and u_e. Because there are $2|e| - 3$ edges for each hyperedge e, each weight is chosen to be $\frac{1}{2|e|-3}$ in order for the weights in each hyperedge to sum to 1. The generalized Laplacian obtained this way satisfies all the properties of the HyperGCN's Laplacian [243].
FastHyperGCN: in order to save training time, in this case the adjacency matrix $\bar{\boldsymbol{A}}^{(t)}$ is computed only once before training, by using only the initial node features of the input hypergraph.

UniGCNII. This model aims to extend to hypergraph structures the GCNII model proposed by Chen et al. [44] for simple graph structures, that is a deep graph convolutional network that puts in place an initial residual connection and identity mapping as a way to reduce the oversmoothing problem [106].

Let d_v denote the degree of vertex v, while $d_e = \frac{1}{|e|}\sum_{i\in e} d_i$ for each hyperedge $e \in \mathcal{E}$. A single node-level update step performed by UniGCNII can be expressed as:

$$\hat{\boldsymbol{x}}_v^{(t)} = \frac{1}{\sqrt{d_v}} \sum_{e\in\mathcal{E}_v} \frac{\boldsymbol{z}_e^{(t)}}{\sqrt{d_e}}, \tag{C.10}$$

$$\boldsymbol{x}_v^{(t+1)} = ((1-\beta)\boldsymbol{I} + \beta\boldsymbol{W}^{(t)})((1-\alpha)\hat{\boldsymbol{x}}_v^{(t)} + \alpha\boldsymbol{x}_v^{(0)}). \tag{C.11}$$

in which α and β are hyperparameters, $\boldsymbol{I}$ is identity matrix and $\boldsymbol{x}_v^{(0)}$ is the initial feature of vertex i.

HNHN. For the HNHN model by Dong et al. [64], hypernode and hyperedge features are supposed to share the same dimensionality d, hence in this case $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{Z} \in \mathbb{R}^{m\times d}$. The update rule in this case can be easily expressed using the incidence matrix as

$$\boldsymbol{Z}^{(t+1)} = \sigma(\boldsymbol{H}^T \boldsymbol{X}^{(t)} \boldsymbol{W}_Z + \boldsymbol{b_Z}), \tag{C.12}$$

$$\boldsymbol{X}^{(t+1)} = \sigma(\boldsymbol{H}\boldsymbol{Z}^{(t+1)} \boldsymbol{W}_X + \boldsymbol{b_X}). \tag{C.13}$$

in which σ is a nonlinear activation function, $\boldsymbol{W}_X, \boldsymbol{W}_Z \in \mathbb{R}^{d\times d}$ are weight matrices, and $\boldsymbol{b_X}, \boldsymbol{b_Z} \in \mathbb{R}^d$ are bias terms.

AllSet. The general formulation for the propagation setting of AllSet [49] is introduced in Subsection 7.3.1 and, starting from that, we now analyze the different instances of the model obtained by imposing specific design choices in the general framework.

In the practical implementation of the model, the update functions $f_{\mathcal{V}\to\mathcal{E}}$ and $f_{\mathcal{E}\to\mathcal{V}}$, that are required to be permutation invariant with respect to their first input, are parametrized and *learnt* for each dataset and task. Furthermore, the information of their second argument is not utilized in practice, hence their input can be more generally denoted as a set $\mathbb{S}$.

The two architectures AllDeepSets and AllSetTransformer are obtained in the following way, depending on whether the update functions are defined either as MLPs or Transformers:

1. AllDeepSets [49]: $f_{\mathcal{V}\to\mathcal{E}}(\mathbb{S}) = f_{\mathcal{E}\to\mathcal{V}}(\mathbb{S}) = \mathrm{MLP}(\sum_{s\in\mathbb{S}} \mathrm{MLP}(s))$;

2. AllSetTransformer [49], in which the update functions are defined iteratively through multiple steps as they were first designed by Vaswani et al. [211].

   The first set of operations corresponds to the self-attention module. Suppose that h attention heads are considered: first of all, h pairs of matrices $\boldsymbol{K}_i$ (keys) and $\boldsymbol{V}_i$ (values) with $i \in \{1, ..., h\}$ are computed from the input set through different MLPs. Additionally, h weights $\theta_i, i \in \{1, ..., h\}$ are also learned and together with the keys and values they allow for the computation of each head-specific attention value $\boldsymbol{O}_i$ using an activation function ω [211]. The h attention heads are processed in parallel and they are then concatenated, leading to a unique vector being the result of the multi-head attention module $\mathrm{MH}_{h,\omega}$. After that, a sum operation and a Layer Normalization (LN) [8] are applied:

$$\boldsymbol{K}_i = \mathrm{MLP}_i^K(\mathbb{S}), \boldsymbol{V}_i = \mathrm{MLP}_i^V(\mathbb{S}), \quad \text{where} \quad i \in \{1, ..., h\}, \tag{C.14}$$

$$\theta \triangleq \|_{i=1}^h \theta_i, \tag{C.15}$$

$$\boldsymbol{O}_i = \omega(\theta_i (\boldsymbol{K}_i)^T) \boldsymbol{V}_i, \quad \text{where} \quad i \in \{1, ..., h\}, \tag{C.16}$$

$$\mathrm{MH}_{h,\omega}(\theta, \mathbb{S}, \mathbb{S}) = \|_{i=1}^h \boldsymbol{O}^{(i)}, \tag{C.17}$$

$$\boldsymbol{Y} = \mathrm{LN}(\theta + \mathrm{MH}_{h,\omega}(\theta, \mathbb{S}, \mathbb{S})). \tag{C.18}$$

A feed-forward module follows the self-attention computations, in which a MLP is applied to the feature matrix and then sum and LN are performed again, corresponding to the last operations to be performed:

$$f_{\mathcal{V}\to\mathcal{E}}(\mathbb{S}) = f_{\mathcal{E}\to\mathcal{V}}(\mathbb{S}) = \mathrm{LN}(\boldsymbol{Y} + \mathrm{MLP}(\boldsymbol{Y})). \tag{C.19}$$

C.2.2 Other models

This Section describes the models that are considered for the experiments but that don't fall directly under the AllSet unified framework defined in Section 7.3.1.

EDHNN. The Equivariant Diffusion-based HNN model, shortened as EDHNN [222] represents the first attempt to draw a connection between the class of hypergraph diffusion algorithms and the design of Hypergraph Neural Networks. The underlying motivation is that, by enabling the model to approximate any continuous equivariant hypergraph diffusion operator, a broad spectrum of higher-order relations can be encoded.

In EDHNN, the hypergraph diffusion operators are learned directly from data, harnessing the expressive power of Neural Networks. This leads to the development of a novel Hypergraph Neural Network inspired by hypergraph diffusion solvers, with the subsequent operations executed at each layer described in the following.

The hyperedge-level feature update is performed as

$$\boldsymbol{z}_e^{(t+1)} = \sum_{u\in e} \hat{\phi}(\boldsymbol{x}_u^{(t)}), \tag{C.20}$$

and starting from that, the node-level update is defined as

$$\boldsymbol{x}_v^{(t+1)} = \hat{\psi}\left(\boldsymbol{x}_v^{(t)}, \sum_{e\in\mathcal{E}_v} \hat{\rho}(\boldsymbol{x}_v^{(t)}, \boldsymbol{z}_e^{(t+1)}), \boldsymbol{x}_v^0, d_v\right). \tag{C.21}$$

In the equations above, $\hat{\psi}$, $\hat{\phi}$ and $\hat{\rho}$ are three MLPs shared across layers.

HAN. The Heterogeneous Graph Attention Network model [224] is specifically designed for processing and performing inference on heterogeneous graphs. Heterogeneous graphs have various types of nodes and/or edges, and standard GNN models that treat all of them equally are not able to properly handle such complex information.

In order to apply this model on hypergraphs, [49] define a preprocessing step to derive a heterogeneous graph from a hypergraph. Specifically, a bipartite graph is defined such that there is a bijection between its nodes and the set of nodes and hyperedges in the original hypergraph. The nodes obtained in this way belong to one of two distinct types, that are the sets $\mathbb{V}$ and $\mathbb{E}$ (if they correspond to either a node or a hyperedge in the original hypergraph, respectively). Edges only connect nodes of two different types, and one edge exists between a node $u_v \in \mathbb{V}$ and a node $u_e \in \mathbb{E}$ if and only if $v \in e$ in the input hypergraph. We consider two types of so-called meta-paths (in this case, paths of length 2) in the heterogeneous graph, that are $\mathbb{V}\to\mathbb{E}\to\mathbb{V}$ and $\mathbb{E}\to\mathbb{V}\to\mathbb{E}$. We denote the sets of such meta-paths as

$\Phi_{\mathcal{V}}$ and $\Phi_{\mathcal{E}}$ respectively. Furthermore, let $\mathcal{N}_{u_v}^{\Phi_{\mathcal{V}}}$ denote the neighbors of node $u_v \in \mathbb{V}$ through paths $\gamma_v \in \Phi_{\mathcal{V}}$, and vice-versa let $\mathcal{N}_{u_e}^{\Phi_{\mathcal{E}}}$ denote the neighbors of node $u_e \in \mathbb{E}$ through paths $\gamma_e \in \Phi_{\mathcal{V}}$.

At each step, the model updates separately and sequentially the node features of nodes in $\mathbb{V}$ and $\mathbb{E}$. Consider for example the case of nodes in $\mathbb{V}$ (for nodes in $\mathbb{E}$ the process is the same, except that $\Phi_{\mathcal{E}}$ is considered instead of $\Phi_{\mathcal{V}}$). The node-level update is performed as follows, for a certain $u \in \mathbb{V}$:

$$\hat{\boldsymbol{x}}_u^{(t)} = \boldsymbol{W}_{\Phi_{\mathcal{V}}}^{(t)} \boldsymbol{x}_u^{(t)}, \tag{C.22}$$

$$\boldsymbol{x}_u^{(t+1)} = \sigma\left(\sum_{w \in \mathcal{N}_u^{\Phi_{\mathcal{V}}}} \alpha_{u,w}^{\Phi_{\mathcal{V}}} \hat{\boldsymbol{x}}_u^{(t)} \right). \tag{C.23}$$

In the equations above, $\boldsymbol{W}_{\Phi_{\mathcal{V}}}^{(t)}$ is a meta-path dependent weight matrix while $\alpha_{u,w}^{\Phi_{\mathcal{V}}}$ is an attention score computed between neighboring nodes in the same way as proposed in Veličković et al. [214], through similar equations as C.7 and C.8. More generally, h attention heads may be considered, that give rise to different attention scores for each head and consequently multiple results for the node feature update, that are then concatenated to obtain a unique feature vector $\boldsymbol{x}_u^{(t+1)}$.

MLP. We also add the MLP model as a baseline; this model doesn't use connectivity at all and only relies on the initial node features to predict their class. The node feature matrix $\boldsymbol{X}$ is obtained as

$$\boldsymbol{X}^{(t)} = \mathrm{MLP}(\boldsymbol{X}^{(t-1)}). \tag{C.24}$$

MLP CB. This model employs a sampling procedure as outlined in Section 7.3.2, in which we straightforwardly apply a Multilayer Perceptron to the initial features of nodes. During the training phase, we incorporate dropout by applying an MLP with distinct weights dropped out for each hyperedge, resulting in slightly different representations for nodes for each hyperedge they belong to. Furthermore, we execute the mini-batching procedure in accordance with the guidelines presented in Section 7.3.2. Importantly, that these two choices affect the training approach significantly so that the results of this model are very different from MLP's performances: see, for example, Table 7.1.

During the validation phase dropout is not utilized, ensuring that the representations used for each hyperedge remain exactly the same. Consequently, there is no need for the readout operation in this context. The node-level update is described by:

$$\boldsymbol{x}_{v,e}^{(t+1)} = \boldsymbol{x}_{v,e}^{(t)} + \mathrm{MLP}(\mathrm{LN}(\boldsymbol{x}_{v,e}^{(t)})), \tag{C.25}$$

$$\boldsymbol{x}_v^{(T)} = \frac{1}{d_v} \sum_{e \in \mathcal{E}_v} \boldsymbol{x}_{v,e}^{(T)}. \tag{C.26}$$

Transformer. Along with MLP, we consider another simple baseline that is the basic Transformer model [211].

Also in this case, let $\mathbb{S}$ denote the set of input vectors, and define as $\boldsymbol{S} = \text{MLP}(\mathbb{S})$ the matrix of input embeddings, obtained from the input set through a MLP. The operations performed on $\boldsymbol{S}$ generalize the ones described for the Transformer module adopted in AllSetTransformer, and they can be split in two main modules, that are the self-attention module and the feed-forward module:

1. Suppose that h attention heads are considered in the self attention module. First of all, h triples of matrices $\boldsymbol{K}_i$ (keys), $\boldsymbol{V}_i$ (values) and $\boldsymbol{Q}_i$ (queries) with $i \in \{1, ..., h\}$ are obtained from $\boldsymbol{S}$ through linear matrix multiplications with weight matrices $\boldsymbol{W}_i^K, \boldsymbol{W}_i^V$ and $\boldsymbol{W}_i^Q$ that are learned during training. The result for each attention module is computed through the key, query and key matrices using an activation function ω [211] and a normalization factor d_k, that corresponds to the dimension of the key and query vectors associated to each input element. The h outputs of the different attention heads are then concatenated, leading to a unique result matrix. After that, a sum operation and a Layer Normalization (LN) [8] are applied:

$$\boldsymbol{K}_i = \boldsymbol{S}\boldsymbol{W}_i^K, \boldsymbol{V}_i = \boldsymbol{S}\boldsymbol{W}_i^V, \boldsymbol{Q}_i = \boldsymbol{S}\boldsymbol{W}_i^Q, \quad \text{where} \quad i \in \{1, ..., h\}, \tag{C.27}$$

$$\boldsymbol{O}_i = \omega\left(\frac{\boldsymbol{Q}_i(\boldsymbol{K}_i)^T}{\sqrt{d_k}}\right)\boldsymbol{V}_i, \quad \text{where} \quad i \in \{1, ..., h\}, \tag{C.28}$$

$$\text{MH}_{h,\omega}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \|_{i=1}^{h}\boldsymbol{O}^{(i)}, \tag{C.29}$$

$$\boldsymbol{Y} = \text{LN}(\boldsymbol{S} + \text{MH}_{h,\omega}(\boldsymbol{S}, \boldsymbol{S})). \tag{C.30}$$

Since it will be useful in the following we introduce, we denote by MAB the output of the multihead self-attention block, namely

$$\text{MAB}(\boldsymbol{S}) = \text{LN}(\boldsymbol{S} + \text{MH}_{h,\omega}(\boldsymbol{S}, \boldsymbol{S})). \tag{C.31}$$

2. As described for AllSetTransformer [49], in the feed-forward module a MLP is applied to the feature matrix, followed by a sum operation and Layer Normalization. After that, the output of the overall Transformer architecture is obtained:

$$\boldsymbol{Y}_{out} = \text{LN}(\boldsymbol{Y} + \text{MLP}(\boldsymbol{Y})). \tag{C.32}$$

WHATsNet In [51], authors introduce a classification of edge-dependent node labels. In the following section, we will describe their method and define all the components. The author introduced WithinATT, inspired by the Transformer model [211], which adjusts a node embedding by focusing on interactions with other nodes within the same hyperedge. Specifically, it applies a set of node embeddings as queries, keys, and values in the attention mechanism. This approach effectively captures relationships between nodes by computing the dot-product for each node pair within the hyperedge. However, calculating the dot-product for all pairs

results in a computational complexity that is quadratic relative to the hyperedge size, posing challenges for large-scale, real-world hypergraphs. To address this, they utilize the inducing point method from SetTransformer [134], which achieves performance comparable to all-pair attention but with significantly greater efficiency. More precisely, Lee et al. [134] introduce a matrix of trainable inducing points $\boldsymbol{I}$ of shape way smaller than the shape of $\boldsymbol{Q}$ and $\boldsymbol{K}$ the $\mathrm{MAB}(\boldsymbol{Q}, \boldsymbol{K})$ is approximated by $\mathrm{MAB}(\boldsymbol{Q}, \mathrm{MAB}(\boldsymbol{I}, \boldsymbol{K}))$.

Let us now describe the WHATsNet model described in [51], first we need to define the WithinATT module. Let $\mathbf{X}_e^{(t)}$ be the set of embeddings of nodes in a hyperedge e in the t-th layer. Along with $\mathbf{X}_e^{(t)} \in \mathbb{R}^{|e| \times d}$, WithinATT uses number of trainable inducing points $\mathbf{I}_w \in \mathbb{R}^{k \times d}$, where k is typically much smaller than $\max_{e \in \mathcal{E}} |e|$, in particula in their experiments they fix $k = 4$. Then, WithinATT is formally expressed as follows:

$$\mathrm{WithinATT}(\boldsymbol{X}_e^{(t)}, \boldsymbol{I}_w) = \mathrm{MAB}(\boldsymbol{X}_e^{(t)}, \mathrm{MAB}(\boldsymbol{I}_w, \boldsymbol{X}_e^{(t)})). \tag{C.33}$$

where MAB is defined in C.31). They also define WithinOrderPE, a positional encodings used in edge-dependent attention between nodes within each hyperedge. To introduce WithinOrderPE, the authors first define the order of each element a in a set A as follows:

$$Order(a, A) = \sum_{a' \in A} \mathbb{1}_{a' \leq a}. \tag{C.34}$$

Then, given node centralities $F \in \mathbb{R}^{N \times d_f}$, where d_f represents the number of centrality measures, which corresponds to the dimensionality of the positional encodings, they define the WithinOrderPE of a node v within a hyperedge e as follows:

$$\mathrm{WithinOrderPE}(v, e; F) = ||_{i=1}^{d_f} \frac{1}{|e|} Order(F_{v,i}, \{F_{u,i}\}_{u \in e}). \tag{C.35}$$

This WithinOrderPE is then added to the node embeddings, which are subsequently fed into WithinATT. Four centrality measures are used: degree, eigenvector centrality, PageRank, and coreness. The notation $\boldsymbol{X}_{e,\boxplus}^{(t)}$ refers to the node embedding that incorporates the positional encodings from WithinOrderPE. To integrate these positional encodings, they first align the dimensionality of the positional encodings with that of the node features at layer t by using a learnable weight matrix. We are now prepared to outline the hyperedge and node update embeddings as presented in [51]. Specifically, the a hyperedge e's embedding $\boldsymbol{z}_e^{(t)}$ are obtained as

$$\boldsymbol{X}_{e,\boxplus}^{(t)} = \{\boldsymbol{x}_v^{(t)} \boxplus \mathrm{WithinOrderPE}(v, e) : v \in e\}, \tag{C.36}$$

$$\tilde{\boldsymbol{X}}_{e,\boxplus}^{(t)} = \mathrm{WithinATT}(\boldsymbol{X}_{e,\boxplus}^{(t)}) \tag{C.37}$$

$$\boldsymbol{z}_e^{(t)} = \mathrm{MAB}(\boldsymbol{z}_e^{(t-1)}, \tilde{\boldsymbol{X}}_e^{(t)}) \tag{C.38}$$

Similarly, node embeddings are updated by aggregating node-dependent hyperedge embeddings from WithinATT and WithinOrderPE. Specifically, a node v embedding $\boldsymbol{x}_v^{(t)}$ is updated as follows,

$$\boldsymbol{Z}_{v,\boxplus}^{(t)} = \{\boldsymbol{z}_e^{(t)} \boxplus \mathrm{WithinOrderPE}(v, e) : e \in \mathcal{E}_v\}, \tag{C.39}$$

$$\tilde{\boldsymbol{Z}}_{v,\boxplus}^{(t)} = \text{WithinATT}(\boldsymbol{Z}_{v,\boxplus}^{(t)}) \tag{C.40}$$

$$\boldsymbol{x}_v^{(t)} = \text{MAB}(\boldsymbol{x}_v^{(t-1)}, \tilde{\boldsymbol{Z}}_v^{(t)}) \tag{C.41}$$

C.3 Proof of Propositions

C.3.1 Proof of Proposition 7.3.1

UniGCNII inherits the same hyperedge update rule of other hypergraph models (e.g. HNN [71], HyperGCN [243]), so it directly follows from Theorem 3.4 of [49] that it can be expressed through 7.5. By looking at the definition of the node update rule of UniGCNII (Eq. C.10 and C.11), we can re-express it as

$$\begin{aligned}\boldsymbol{x}_v^{(t+1)} = ((1-\beta)\boldsymbol{I} + \beta\boldsymbol{W}^{(t)})\left((1-\alpha)\frac{1}{\sqrt{d_v}}\sum_{e\in\mathcal{E}_v}\frac{\boldsymbol{z}_e^{(t+1)}}{\sqrt{d_e}} + \alpha\boldsymbol{x}_v^{(0)}\right) \\ = f_{\mathcal{E}\to\mathcal{V}}(\{\boldsymbol{z}_e^{(t+1)}\}_{e\in\mathcal{E}_v}; \boldsymbol{x}_v^{(0)}).\end{aligned} \tag{C.42}$$

Note that this is a particular instance of the extended AllSet node update rule 7.7 where only a residual connection to the initial node features is considered. Lastly, it is straightforward to see that $f_{\mathcal{E}\to\mathcal{V}}$ is permutation invariant w.r.t the set $\{\boldsymbol{z}_e^{(t+1)}\}_{e\in\mathcal{E}_v}$, as it processes the set through a weighted mean.

C.3.2 Proof of Proposition 7.3.2

We prove this proposition by showing that we can obtain AllSet update rules 7.5-7.6 and 7.7 from our proposed MultiSet framework 7.8-7.9-7.10. This can easily follow by not distinguishing node representations among hyperedges, so $\mathbb{X}_v^{(t)} = \{\boldsymbol{x}_{v,e}^{(t)}\}_{e\in\mathcal{E}_v} = x_v^{(t)}$. With this particular choice, we directly get 7.5 from 7.8, and 7.7 can be obtained from 7.9 by further disregarding the hyperedge subscript –as there is only a single node representation to update. Analogously, we can get 7.6 from 7.9 if we additionally do not consider node residual connections, so $\{\mathbb{X}_v^{(k)}\}_{k=0}^{t}$ simply becomes $x_v^{(t)}$. Finally, the readout 7.10 can be defined as the identity function applied to the node representations at the last message passing step T.

C.3.3 Proof of Proposition 7.3.3

We also prove this proposition by showing that we can obtain EDHNN [222] update rules from our proposed MultiSet framework 7.8-7.9-7.10. EDHNN hyperedge update rule can be expressed as

$$z_e^{(t+1)} = \sum_{u\in e}\hat{\phi}(x_u^{(t)}) = f_{\mathcal{V}\to\mathcal{E}}(\{x_u^{(k)}\}_{u\in\mathcal{V}}),$$

which is a particular instance of 7.8 given that $\{x_u^{(k)}\}_{u\in\mathcal{V}} \subset \mathbb{X}_v^{(t)}$. As for the node update rule, we have

$$\begin{aligned}x_v^{(t+1)} = \hat{\psi}\left(x_v^{(t)}, \sum_{e\in\mathcal{E}_v}\hat{\rho}(x_v^{(t)}, z_e^{(t+1)}), x_v^0, d_v\right) \\ = f_{\mathcal{E}\to\mathcal{V}}\left(\{\boldsymbol{z}_e^{(t+1)}\}_{e\in\mathcal{E}_v}; \{\boldsymbol{x}_v^{(k)}\}_{k\in\{0,t\}}\right),\end{aligned} \tag{C.43}$$

where we recall that $d_v := |\mathcal{E}_v|$. By disregarding the hyperedge superscript in Eq. 7.9 –essentially making all hyperedge-based node representations the same one–, it is straightforward to see that the previous expression C.43 is a particular case of the MultiSet node update rule. Lastly, the readout step 7.10 can be just defined as the identity function, given that all involved hyperedge-based messages to nodes (i.e. $\hat{\rho}(x_v^{(t)}, z_e^{(t+1)})$) are already being aggregated at each iteration while updating the node state.

C.3.4 Proof of Proposition 7.3.4

The node updates in WHATsNet output a single node feature, while in MultiSet, the output of node updates results in multiple features for each node, i.e., one for each hyperedge to which it belongs. MultiSet framework preserves a higher level of generality by not enforcing pooling for edge-wise node representations at each layer, but a pooling operation could be intrinsically defined as part of the function in our Equation 7.9, leading to a single feature for each node as output.

We denote by $\boldsymbol{X}_e^{(t)}$ the set of the embeddings of nodes belonging to edge e. The functions are :

$$\begin{aligned} \boldsymbol{z}_e^{(t)} = \text{MAB}(\boldsymbol{z}_e^{t-1}, \text{WithinATT}(\{\boldsymbol{x}_v^{(t)} \boxplus \text{WithinOrderPE}(v,e) : v \in e\})) \\ = f_{\mathcal{V}\to\mathcal{E}}(\{\boldsymbol{x}_v^t\}_{v\in e}, z_e^{(t-1)}). \end{aligned} \tag{C.44}$$

with WithinOrderPE defined as in equation C.35. Similarly,

$$\begin{aligned} \boldsymbol{x}_v^{(t)} = \text{MAB}(\boldsymbol{x}_v^{t-1}, \text{WithinATT}(\{\boldsymbol{z}_e^{(t)} \boxplus \text{WithinOrderPE}(v,e) : e \in \mathcal{E}_v\})) \\ = f_{\mathcal{E}\to\mathcal{V}}(\{\boldsymbol{z}_e^t\}_{e\in\mathcal{E}_v}, x_v^{(t-1)}). \end{aligned} \tag{C.45}$$

where MAB is defined according to Equation C.31. In the work of [51], the centrality measures used to compute the vectors in $\text{WithinOrderPE}(v,e)$ rely solely on the topology of the graph. However, centrality measures based on node and edge features could also be considered, and our framework is capable of capturing these as well.

C.3.5 Proof of Proposition 7.3.5

It is straightforward that functions $f_{\mathcal{V}\to\mathcal{E}}$, $f_{\mathcal{E}\to\mathcal{V}}$ and $f_{\mathcal{V}\to\mathcal{V}}$ defined in MultiSet-Mixer (Equations 7.11-7.13) are permutation invariant w.r.t their first argument: hyperedge update rule 7.11 and readout 7.13 process it through a mean operation, whereas the node update rule only receives a single-element set. The rest of the proof follows from the proof of Proposition 4.1 of Chien et al. [49].

C.4 Experiments

C.4.1 Hyperparameters optimization

In order to implement the benchmark models, we followed the procedure described in Chien et al. [49]; in particular, the maximum epochs were set to 200 for all the models. The models were trained with categorical cross-entropy loss, and the best architecture was selected at the end of training depending on validation accuracy. For

the AllDeepSets [49], AllSetTransformer [49], UniGCNII [106], CEGAT, CEGCN, HCHA [10], HNN [71], HNHN [64], HyperGCN [243], HAN[224], and HAN (mini-batching) [224] and MLP, we performed the same hyperparameter optimization proposed in Chien et al. [49]. For both the proposed model and the introduced baseline, we conducted a thorough hyperparameter search across the following values:

- learning rate within the range of $0.001, 0.01$;
- weight decay values from the set $0.0, 1e-5, 1$;
- MLP hidden layer sizes of $64, 128, 256, 512$;
- mini-batch sizes set to $256, 512$, with full-batch utilization when memory resources allow;
- the number of sampled neighbors per hyperedge ranged from $2, 3, 5, 10$.

It's important to note that the limitation of the number of sampled neighbors per hyperedge to this small range was intentional. This limitation showcases that even for datasets with large hyperedges, effective processing can be achieved by considering only a subset of neighbors.

The models' hyperparameters were optimized for a 50% split and subsequently applied to all the other splits.

Reproducibility. We are committed to providing a comprehensive overview of our experimental setup, encompassing machine specifications, environmental details, and the optimal hyperparameters selected for each model. Additionally, the source code, including training/validation/test splits, will be supplied with both the initial release and the camera-ready version, ensuring the reproducibility of our results.

C.4.2 Further information about the datasets

For our experiments we utilized various benchmark datasets from existing literature on hypergraph neural networks, the statistical properties of which are in Table C.1. For what concerns co-authorship networks (Cora-CA and DPBL-CA) and co-citation networks (Cora, Citeseer, and Pubmed), we relied on the datasets provided in Yadati et al. [243]. Additionally, we employed the Princeton ModelNet40 [233] and the National Taiwan University [42] dataset introduced for 3D object classification. For these two datasets, we complied with what Feng et al. [71] and Yang et al. [245] proposed for the construction of the hypergraphs, using both MVCNN [196] and GVCNN [71] features. Additionally, we tested our model on three datasets with categorical attributes, namely 20Newsgroups, Mushroom, and ZOO, obtained from the UCI Categorical Machine Learning Repository [66]. In order to construct hypergraphs for these datasets, we followed the approach described in Yadati et al. [243], where a hyperedge is defined for all data points sharing the same categorical feature value.

We downloaded co-citation and co-authoring networks from Yadati et al. [243]. Below are the details on how the hypergraph was constructed. **Cora-CA**, **DBLP**: all documents co-authored by an author are in one hyperedge, following what was done

Table C.1. Statistics of hypergraph datasets: $|e|$ denotes the size of hyperedges while d_v denotes the node degree.

	Cora	Citeseer	Pubmed	CORACA	DBLP-CA	ZOO	20Newsgroups	Mushroom	NTU2012	ModelNet40
$\lvert\mathcal{E}\rvert$	1579	1079	7963	1072	22363	43	100	298	2012	12311
# classes	7	6	3	7	6	7	4	2	67	40
min $\lvert e\rvert$	2	2	2	2	2	1	29	1	5	5
med $\lvert e\rvert$	3	2	3	3	3	40	537	72	5	5
max d_v	145	88	99	23	18	17	44	5	19	30
min d_v	0	0	0	0	1	17	1	5	1	1
avg d_v	1.77	1.04	1.76	1.69	2.41	17	4.03	5	5	5
med d_v	1	0	0	2	2	17	3	5	5	4
CE Homophily	89.74	89.32	95.24	80.26	86.88	82.88	75.25	85.33	46.07	24.07
$\frac{1}{\lvert\mathcal{V}\rvert}\sum_{v\in\mathcal{V}} h_v^0$	84.10	78.25	82.05	80.81	88.86	91.13	81.26	88.05	53.24	42.16
$\frac{1}{\lvert\mathcal{V}\rvert}\sum_{v\in\mathcal{V}} h_v^1$	78.08	74.18	75.73	76.51	86.01	85.79	74.78	84.41	41.95	29.42

in Yadati et al. [243]. Co-citation data **Citeseer**, **Pubmed**, **Cora**: all documents cited by a document are connected by a hyperedge. Each hypernode (document abstract) is represented by bag-of-words features (feature matrix $\boldsymbol{X}$).

Citation and co-authorship datasets. In the co-citation and co-authorship networks datasets, the node features are the bag-of-words representations of the corresponding documents. In co-citation datasets (Cora, Citeseer, PubMed) all documents cited by a document are connected by a hyperedge. In co-authored datasets (CORA-CA, DBLP-CA), all documents co-authored by an author belong to the same hyperedge.

Computer vision/graphics. The hyperedges are constructed using the k-nearest neighbor algorithm in which $k = 5$.

Categorical datasets. There are instances with categorical attributes within the datasets. To construct the hypergraph, each attribute value is treated as a hyperedge, meaning that all instances (hypernodes) with the same attribute value are contained in a hyperedge. The node features of 20Newsgroups are the TF-IDF representations of news messages. The node features of mushrooms (in Mushroom dataset) represent categorical descriptions of 23 species. The node features of a zoo (in ZOO dataset) are a combination of categorical and numeric measurements describing various animals.

Table C.2. Node Connectivity Statistics. For brevity we use the following notation in this table: under the columns labeled $|\mathcal{E}_v| = k$, we report the amount of nodes that belong to k hyperedges. This value can be expressed in a more formal way as $|v \in \mathcal{V} : |\mathcal{E}_v| = k|$. Moreover, $|\mathcal{E}_v| = 0$, denotes the number of isolated nodes. In addition, the columns labeled "% $|\mathcal{E}_v| = k$" indicate the percentage of nodes belonging to k hyperedges relatively to the total number of nodes. Finally, $\sum_{e\in\mathcal{E}} |e|$ corresponds to the number of hyperedge-dependent node representations.

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}_v\rvert=0$	$\lvert\mathcal{E}_v\rvert=1$	$\lvert\{v:\lvert\mathcal{E}_v\rvert=2\}\rvert$	$\lvert\mathcal{E}_v\rvert=3$	$\lvert\mathcal{E}_v\rvert>3$	% $\lvert\mathcal{E}_v\rvert=0$	% $\lvert\mathcal{E}_v\rvert=1$	% $\lvert\mathcal{E}_v\rvert=2$	% $\lvert\mathcal{E}_v\rvert=31$	% $\lvert\mathcal{E}_v\rvert>3$	$\sum_{e\in\mathcal{E}}\lvert e\rvert$
Cora	2708	1274	575	327	156	376	47.05	21.23	12.08	5.76	13.88	6060
Citeseer	3312	1854	798	307	144	209	55.98	24.09	9.27	4.35	6.31	5307
Pubmed	19717	15877	339	313	292	2896	80.52	1.72	1.59	1.48	14.69	50506
CORA-CA	2708	320	995	951	287	155	11.82	36.74	35.12	10.60	5.72	4905
DBLP-CA	41302	0	8998	16724	9249	6331	0.00	21.79	40.49	22.39	15.33	99561
Mushroom	8124	0	0	0	0	8124	0.00	0.00	0.00	0.00	100.00	40620
NTU2012	2012	0	173	256	296	1287	0.00	8.60	12.72	14.71	63.97	10060
ModelNet40	12311	0	1491	1755	1594	7471	0.00	12.11	14.26	12.95	60.69	61555
20newsW100	16242	0	3053	3149	2720	7320	0.00	18.80	19.39	16.75	45.07	65451
ZOO	101	0	0	0	0	101	0.00	0.00	0.00	0.00	100.00	1717

Table C.3. Hypergraph model performance benchmarks. Test accuracy in % averaged over 15 splits. Training split: 50%.

Model	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	77.11 ± 1.00	70.67 ± 1.42	89.04 ± 0.45	82.23 ± 1.46	91.34 ± 0.27	99.96 ± 0.05	86.49 ± 1.86	96.70 ± 0.25	81.19 ± 0.49	89.10 ± 7.00	6.80
AllSetTransformer	79.54 ± 1.02	72.52 ± 0.88	88.74 ± 0.51	84.43 ± 1.14	91.61 ± 0.19	99.95 ± 0.05	88.22 ± 1.42	98.00 ± 0.12	81.59 ± 0.59	**91.03 ± 7.31**	3.25
UniGCNII	78.46 ± 1.14	73.05 ± 1.48	88.07 ± 0.47	83.92 ± 1.02	91.56 ± 0.18	99.89 ± 0.07	88.24 ± 1.56	97.84 ± 0.16	81.16 ± 0.49	89.61 ± 8.09	4.75
EDHNN	**80.74 ± 1.00**	73.22 ± 1.14	**89.12 ± 0.47**	**85.17 ± 1.02**	**91.94 ± 0.23**	99.94 ± 0.11	88.04 ± 1.65	97.70 ± 0.19	81.64 ± 0.49	89.49 ± 6.99	**2.90**
CEGAT	76.53 ± 1.58	71.58 ± 1.11	87.11 ± 0.49	77.50 ± 1.51	88.74 ± 0.31	96.81 ± 1.41	82.27 ± 1.60	92.79 ± 0.44	NA	44.62 ± 9.18	12.11
CEGCN	77.03 ± 1.31	70.87 ± 1.19	87.01 ± 0.62	77.55 ± 1.65	88.12 ± 0.25	94.91 ± 0.44	80.90 ± 1.74	90.04 ± 0.47	NA	49.23 ± 6.81	12.56
HCHA	79.53 ± 1.33	72.57 ± 1.06	86.97 ± 0.55	83.53 ± 1.12	91.21 ± 0.28	98.94 ± 0.54	86.60 ± 1.96	94.50 ± 0.33	80.75 ± 0.53	89.23 ± 6.81	7.85
HGNN	79.53 ± 1.33	72.24 ± 1.08	86.97 ± 0.55	83.45 ± 1.22	91.26 ± 0.26	98.94 ± 0.54	86.71 ± 1.48	94.50 ± 0.33	80.75 ± 0.52	89.23 ± 6.81	7.95
HNHN	77.68 ± 1.08	73.47 ± 1.36	87.88 ± 0.47	78.53 ± 1.15	86.73 ± 0.40	**99.97 ± 0.04**	88.28 ± 1.50	97.84 ± 0.15	81.53 ± 0.55	89.23 ± 7.85	5.55
HyperGCN	74.78 ± 1.11	66.06 ± 1.58	82.32 ± 0.62	77.48 ± 1.14	86.07 ± 3.32	69.51 ± 4.98	47.65 ± 5.01	46.10 ± 7.95	80.84 ± 0.49	51.54 ± 9.88	14.30
HAN	80.73 ± 1.37	**73.69 ± 0.95**	86.34 ± 0.61	84.19 ± 0.81	91.10 ± 0.20	91.33 ± 0.91	83.78 ± 1.75	93.85 ± 0.33	79.67 ± 0.55	80.26 ± 6.42	8.90
HAN minibatch	80.24 ± 2.17	73.55 ± 1.13	85.41 ± 2.32	82.04 ± 2.56	90.52 ± 0.50	93.87 ± 1.04	80.62 ± 2.00	92.06 ± 0.63	79.76 ± 0.56	70.39 ± 11.29	10.60
MultiSetMixer	78.06 ± 1.24	71.85 ± 1.50	87.19 ± 0.53	82.74 ± 1.23	90.68 ± 0.19	99.58 ± 0.16	**88.90 ± 1.30**	**98.38 ± 0.21**	**88.57 ± 1.96**	88.08 ± 8.04	6.20
MLP CB	74.06 ± 1.26	71.93 ± 1.53	85.83 ± 0.51	74.39 ± 1.40	84.91 ± 0.44	99.93 ± 0.08	85.43 ± 1.51	96.41 ± 0.32	86.13 ± 2.82	81.61 ± 10.98	10.30
MLP	73.27 ± 1.09	72.07 ± 1.65	87.13 ± 0.49	73.27 ± 1.09	84.77 ± 0.41	99.91 ± 0.08	79.70 ± 1.56	95.31 ± 0.28	80.93 ± 0.59	85.13 ± 6.90	11.50
Transformer	74.15 ± 1.17	71.82 ± 1.51	87.37 ± 0.49	73.61 ± 1.55	85.26 ± 0.38	99.95 ± 0.08	82.88 ± 1.93	96.29 ± 0.29	81.17 ± 0.54	88.72 ± 10.25	9.85

Table C.4. Hypergraph model performance benchmarks. Test accuracy in % averaged over 15 splits. Training split: 40%.

Model	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	76.09 ± 1.22	70.32 ± 1.39	88.58 ± 0.46	81.32 ± 1.27	90.96 ± 0.24	99.94 ± 0.08	85.60 ± 1.46	96.71 ± 0.21	81.11 ± 0.43	89.57 ± 5.91	7.20
AllSetTransformer	78.81 ± 0.99	71.65 ± 1.05	88.17 ± 0.45	83.26 ± 1.12	91.26 ± 0.24	99.94 ± 0.09	87.04 ± 1.07	97.92 ± 0.14	81.30 ± 0.41	**91.72 ± 6.38**	3.75
UniGCNII	77.78 ± 1.15	72.30 ± 1.45	87.86 ± 0.37	83.39 ± 0.95	91.32 ± 0.19	99.88 ± 0.09	87.30 ± 1.34	97.86 ± 0.16	81.14 ± 0.45	89.68 ± 6.42	4.45
EDHNN	**80.46 ± 0.91**	72.62 ± 0.98	**88.70 ± 0.34**	**84.41 ± 0.87**	**91.60 ± 0.20**	99.95 ± 0.09	87.33 ± 1.20	97.65 ± 0.20	81.41 ± 0.37	90.75 ± 5.54	**2.00**
CEGAT	75.68 ± 1.09	70.59 ± 0.89	86.39 ± 0.47	76.91 ± 1.22	88.18 ± 0.31	96.72 ± 1.50	80.97 ± 1.30	92.46 ± 0.29	NA	45.27 ± 9.41	11.67
CEGCN	76.19 ± 1.06	70.08 ± 1.26	86.22 ± 0.50	76.17 ± 1.44	87.61 ± 0.26	95.00 ± 0.38	79.41 ± 1.26	89.79 ± 0.39	NA	51.40 ± 7.24	12.72
HCHA	78.87 ± 1.04	71.73 ± 0.91	86.28 ± 0.43	83.05 ± 0.99	91.04 ± 0.23	99.00 ± 0.48	85.53 ± 1.43	94.53 ± 0.28	80.77 ± 0.31	90.54 ± 5.29	7.40
HGNN	78.87 ± 1.04	71.44 ± 1.00	86.28 ± 0.43	82.95 ± 1.06	91.06 ± 0.24	99.00 ± 0.48	85.71 ± 1.37	94.53 ± 0.28	80.77 ± 0.31	90.54 ± 5.29	7.40
HNHN	76.47 ± 0.90	72.25 ± 1.10	87.17 ± 0.45	77.27 ± 1.11	86.61 ± 0.31	**99.96 ± 0.08**	87.14 ± 1.23	97.82 ± 0.17	81.28 ± 0.49	89.46 ± 6.50	6.10
HyperGCN	73.56 ± 0.91	64.65 ± 1.28	82.09 ± 0.67	76.44 ± 1.06	86.22 ± 2.93	69.49 ± 5.02	46.78 ± 4.61	45.34 ± 7.34	80.82 ± 0.59	52.80 ± 8.90	14.00
HAN	79.89 ± 0.78	**73.16 ± 1.04**	86.11 ± 0.56	83.84 ± 0.91	90.96 ± 0.19	91.39 ± 0.93	82.79 ± 1.16	93.83 ± 0.27	79.52 ± 0.47	80.11 ± 6.46	8.75
HAN minibatch	80.07 ± 1.68	69.44 ± 6.58	86.08 ± 0.84	82.33 ± 1.91	NA	93.60 ± 0.91	79.41 ± 1.62	NA	79.43 ± 0.91	64.84 ± 12.59	11.56
MultiSetMixer	77.14 ± 1.25	70.96 ± 1.66	86.67 ± 0.42	82.09 ± 0.79	90.45 ± 0.25	99.61 ± 0.23	**87.41 ± 1.10**	**98.31 ± 0.17**	**88.82 ± 1.56**	88.71 ± 6.33	6.15
MLP CB	72.60 ± 1.44	71.08 ± 1.68	85.14 ± 0.47	72.63 ± 1.31	84.63 ± 0.31	99.91 ± 1.03	83.72 ± 1.40	96.25 ± 0.25	87.28 ± 3.50	81.42 ± 11.55	10.15
MLP	70.61 ± 7.44	70.96 ± 1.65	86.60 ± 0.40	70.70 ± 7.33	84.42 ± 0.28	99.91 ± 0.09	77.83 ± 1.63	95.24 ± 0.23	80.95 ± 0.54	85.38 ± 8.02	11.35
Transformer	72.65 ± 1.15	70.70 ± 1.50	86.79 ± 0.34	71.96 ± 1.03	84.97 ± 0.27	99.92 ± 0.09	80.69 ± 1.55	96.18 ± 0.24	80.95 ± 0.46	89.68 ± 7.31	9.80

Table C.5. Hypergraph model performance benchmarks. Test accuracy in % averaged over 15 splits. Training split: 30%.

Model	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	74.78 ± 1.02	69.10 ± 1.34	88.01 ± 0.39	79.69 ± 1.44	90.57 ± 0.20	**99.95 ± 0.06**	84.04 ± 1.39	96.49 ± 0.26	80.99 ± 0.41	87.04 ± 6.74	7.25
AllSetTransformer	77.67 ± 0.92	71.06 ± 1.00	87.62 ± 0.34	82.14 ± 0.96	91.10 ± 0.18	99.94 ± 0.09	85.73 ± 1.38	97.73 ± 0.21	81.05 ± 0.49	90.19 ± 6.18	4.05
UniGCNII	76.29 ± 1.05	71.38 ± 1.32	87.48 ± 0.35	81.93 ± 1.09	90.97 ± 0.20	99.88 ± 0.08	85.59 ± 1.60	97.89 ± 0.16	81.10 ± 0.44	88.33 ± 6.29	4.55
EDHNN	**79.30 ± 1.03**	71.85 ± 0.84	**88.25 ± 0.38**	**83.13 ± 1.03**	**91.37 ± 0.18**	99.94 ± 0.11	**86.09 ± 1.44**	97.55 ± 0.23	81.26 ± 0.34	89.44 ± 6.38	**2.35**
CEGAT	74.25 ± 1.24	69.75 ± 0.90	85.41 ± 0.44	75.24 ± 1.05	87.54 ± 0.23	96.86 ± 1.27	79.12 ± 1.60	91.89 ± 0.30	NA	42.87 ± 8.96	11.94
CEGCN	74.84 ± 1.35	69.17 ± 0.93	85.17 ± 0.41	74.83 ± 1.72	87.10 ± 0.25	95.08 ± 0.40	78.13 ± 1.35	89.34 ± 0.40	NA	48.70 ± 5.96	12.44
HCHA	77.81 ± 1.07	71.10 ± 1.11	84.97 ± 0.41	81.81 ± 1.08	90.85 ± 0.18	99.00 ± 0.47	84.34 ± 1.61	94.38 ± 0.28	80.78 ± 0.43	**90.65 ± 5.58**	7.40
HGNN	77.81 ± 1.07	70.88 ± 1.05	84.97 ± 0.41	81.78 ± 1.13	90.85 ± 0.16	99.00 ± 0.47	84.40 ± 1.38	94.38 ± 0.28	80.78 ± 0.43	**90.65 ± 5.58**	7.60
HNHN	74.85 ± 1.14	71.34 ± 1.03	86.34 ± 0.39	75.46 ± 1.02	86.33 ± 0.25	**99.95 ± 0.07**	84.93 ± 1.49	97.75 ± 0.20	81.10 ± 0.48	85.37 ± 7.96	6.30
HyperGCN	71.54 ± 1.26	63.82 ± 1.34	81.87 ± 0.56	74.44 ± 1.25	85.63 ± 2.89	69.44 ± 5.02	46.70 ± 4.01	45.28 ± 8.18	80.64 ± 0.48	55.46 ± 6.78	14.20
HAN	78.60 ± 1.28	**72.44 ± 1.05**	85.89 ± 0.44	82.69 ± 0.77	90.85 ± 0.19	91.47 ± 0.79	81.54 ± 1.44	93.79 ± 0.20	79.51 ± 0.58	79.81 ± 6.61	7.90
HAN minibatch	78.84 ± 1.19	72.26 ± 0.93	85.70 ± 0.81	79.81 ± 1.53	NA	93.59 ± 0.84	77.97 ± 1.63	NA	79.46 ± 1.10	45.74 ± 13.83	9.88
MultiSetMixer	76.03 ± 1.37	70.60 ± 1.15	86.12 ± 0.36	80.58 ± 1.11	90.19 ± 0.20	99.56 ± 0.16	85.95 ± 1.43	**98.20 ± 0.17**	**88.20 ± 1.24**	86.11 ± 6.96	5.90
MLP CB	71.14 ± 1.09	70.21 ± 1.14	84.24 ± 0.63	71.14 ± 1.61	84.17 ± 0.24	99.92 ± 0.08	81.18 ± 1.79	96.10 ± 0.22	85.94 ± 5.83	79.58 ± 8.22	10.40
MLP	66.14 ± 11.37	69.92 ± 1.32	85.86 ± 0.29	66.14 ± 11.37	83.96 ± 0.25	99.89 ± 0.11	75.08 ± 1.69	95.05 ± 0.31	80.82 ± 0.45	82.87 ± 7.37	11.70
Transformer	70.41 ± 1.13	69.75 ± 1.33	86.05 ± 0.28	69.93 ± 0.92	84.57 ± 0.25	99.93 ± 0.10	78.20 ± 1.73	95.92 ± 0.20	80.87 ± 0.37	86.85 ± 9.97	10.25

Table C.6. Hypergraph model performance benchmarks. Test accuracy in % averaged over 15 splits. Training split: 20%.

Model	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	72.56 ± 1.60	67.49 ± 1.57	87.24 ± 0.36	77.24 ± 1.52	89.91 ± 0.26	**99.92 ± 0.07**	80.71 ± 1.32	96.30 ± 0.24	80.59 ± 0.33	84.72 ± 7.95	7.05
AllSetTransformer	75.69 ± 1.09	69.39 ± 1.30	86.63 ± 0.40	80.54 ± 0.94	90.72 ± 0.17	**99.92 ± 0.07**	82.58 ± 1.31	97.48 ± 0.24	80.82 ± 0.28	85.61 ± 7.29	3.95
UniGCNII	74.11 ± 1.28	70.51 ± 1.48	86.97 ± 0.41	79.41 ± 1.23	90.47 ± 0.19	99.90 ± 0.06	82.54 ± 1.60	97.83 ± 0.15	80.88 ± 0.32	88.21 ± 5.55	4.20
EDHNN	**77.09 ± 1.30**	70.76 ± 1.39	**87.52 ± 0.40**	**81.06 ± 1.27**	**90.91 ± 0.20**	99.91 ± 0.10	**83.45 ± 1.34**	97.37 ± 0.17	80.92 ± 0.35	88.54 ± 6.38	**2.10**
CEGAT	71.86 ± 1.42	68.11 ± 1.34	84.03 ± 0.51	73.18 ± 1.32	86.98 ± 0.27	96.19 ± 1.38	76.14 ± 1.20	91.34 ± 0.31	NA	41.22 ± 6.16	11.67
CEGCN	73.25 ± 1.35	67.23 ± 1.39	83.47 ± 0.52	72.50 ± 1.44	86.29 ± 0.21	94.98 ± 0.31	75.55 ± 1.37	88.60 ± 0.41	NA	49.51 ± 6.31	12.44
HCHA	76.04 ± 1.30	69.90 ± 1.25	83.65 ± 0.37	80.03 ± 0.87	90.53 ± 0.17	99.03 ± 0.43	81.48 ± 1.29	94.31 ± 0.20	80.60 ± 0.29	**89.35 ± 5.89**	6.50
HGNN	76.04 ± 1.30	69.59 ± 1.22	83.65 ± 0.37	80.02 ± 0.88	90.51 ± 0.19	99.03 ± 0.43	81.60 ± 1.24	94.31 ± 0.20	80.60 ± 0.29	**89.35 ± 5.89**	6.60
HNHN	72.47 ± 1.06	69.44 ± 1.21	85.10 ± 0.47	73.16 ± 0.99	85.82 ± 0.21	99.88 ± 0.10	81.56 ± 1.54	97.61 ± 0.24	80.48 ± 0.32	82.60 ± 7.71	7.90
HyperGCN	68.59 ± 1.79	62.08 ± 1.28	81.57 ± 0.43	71.42 ± 1.27	85.45 ± 2.31	69.36 ± 5.10	44.01 ± 3.47	46.40 ± 8.41	80.38 ± 0.37	53.25 ± 8.74	14.10
HAN	76.73 ± 1.18	**71.21 ± 1.13**	85.72 ± 0.44	80.83 ± 0.89	90.56 ± 0.15	91.50 ± 0.98	79.46 ± 1.30	93.77 ± 0.23	79.33 ± 0.45	78.54 ± 6.50	7.45
HAN minibatch	76.89 ± 1.51	NA	85.59 ± 0.72	78.55 ± 1.43	NA	93.22 ± 1.34	73.79 ± 1.54	NA	79.50 ± 0.42	47.72 ± 14.96	10.14
MultiSetMixer	73.93 ± 1.15	68.95 ± 1.37	85.00 ± 0.62	78.32 ± 0.94	89.80 ± 0.19	99.42 ± 0.20	82.40 ± 1.41	**98.01 ± 0.19**	**87.85 ± 1.53**	81.06 ± 7.04	6.60
MLP CB	68.07 ± 1.50	68.64 ± 1.34	83.06 ± 0.58	68.37 ± 1.23	83.43 ± 0.25	99.84 ± 1.38	77.01 ± 1.69	95.85 ± 0.27	85.66 ± 5.70	75.61 ± 9.73	10.45
MLP	51.73 ± 17.51	68.20 ± 1.21	85.09 ± 0.34	51.73 ± 17.51	83.22 ± 0.21	99.84 ± 0.12	69.35 ± 9.49	94.69 ± 0.34	80.58 ± 0.31	78.54 ± 8.98	11.70
Transformer	67.34 ± 1.26	68.06 ± 1.39	85.31 ± 0.40	66.61 ± 1.50	83.93 ± 0.27	99.86 ± 0.13	74.17 ± 1.65	95.65 ± 0.29	80.51 ± 0.38	81.45 ± 6.81	10.70

Table C.7. Hypergraph model performance benchmarks. Test accuracy in % averaged over 15 splits. Training split: 10%.

Model	Cora	Citeseer	Pubmed	CORA-CA	DBLP-CA	Mushroom	NTU2012	ModelNet40	20Newsgroups	ZOO	avg. ranking
AllDeepSets	68.51 ± 1.64	64.50 ± 1.43	85.55 ± 0.38	73.67 ± 1.79	88.82 ± 0.25	**99.88 ± 0.08**	73.44 ± 1.91	95.96 ± 0.21	79.61 ± 0.36	76.81 ± 7.05	7.05
AllSetTransformer	71.82 ± 1.18	65.96 ± 1.48	84.71 ± 0.55	76.16 ± 1.36	90.09 ± 0.18	99.86 ± 0.09	75.78 ± 1.96	96.93 ± 0.21	80.18 ± 0.31	75.22 ± 10.78	4.70
UniGCNII	69.36 ± 1.63	66.41 ± 1.59	85.51 ± 0.50	75.84 ± 1.13	89.70 ± 0.25	99.86 ± 0.09	74.86 ± 2.20	**97.58 ± 0.18**	80.44 ± 0.26	**79.86 ± 7.97**	4.10
EDHNN	**72.78 ± 1.54**	67.90 ± 1.51	**86.00 ± 0.51**	77.20 ± 1.31	**90.30 ± 0.17**	99.86 ± 0.10	**76.60 ± 1.79**	96.91 ± 0.27	80.47 ± 0.29	79.06 ± 7.88	**2.30**
CEGAT	68.08 ± 1.65	64.15 ± 1.60	81.83 ± 0.43	69.04 ± 1.60	85.92 ± 0.23	96.01 ± 1.31	69.26 ± 2.27	90.17 ± 0.37	NA	39.20 ± 6.19	12.00
CEGCN	70.22 ± 1.62	62.68 ± 1.49	82.13 ± 0.44	67.45 ± 1.54	85.41 ± 0.26	94.85 ± 0.36	68.31 ± 2.08	87.28 ± 0.39	NA	49.20 ± 5.69	12.11
HCHA	72.76 ± 1.82	66.15 ± 1.27	82.41 ± 0.36	76.97 ± 0.95	90.00 ± 0.19	98.93 ± 0.41	74.44 ± 2.31	94.04 ± 0.21	80.23 ± 0.32	79.78 ± 7.89	5.95
HGNN	72.76 ± 1.82	65.69 ± 1.57	82.41 ± 0.36	76.96 ± 1.10	90.00 ± 0.18	98.93 ± 0.41	74.53 ± 2.44	94.04 ± 0.21	80.23 ± 0.32	79.78 ± 7.89	6.25
HNHN	67.43 ± 1.62	65.02 ± 1.40	82.33 ± 0.76	68.10 ± 1.67	84.74 ± 0.31	99.69 ± 0.18	73.82 ± 2.11	97.34 ± 0.25	80.00 ± 0.26	73.12 ± 6.57	8.80
HyperGCN	63.21 ± 1.95	57.81 ± 1.91	80.83 ± 0.46	65.58 ± 2.02	84.37 ± 1.73	67.56 ± 8.16	40.30 ± 3.67	45.92 ± 7.60	79.57 ± 0.38	51.96 ± 6.32	14.10
HAN	72.08 ± 1.47	67.86 ± 1.46	85.10 ± 0.43	**77.48 ± 1.22**	90.02 ± 0.17	91.67 ± 0.86	72.91 ± 1.88	93.52 ± 0.32	78.77 ± 0.49	70.94 ± 14.54	7.30
HAN minibatch	69.61 ± 6.86	**68.25 ± 1.15**	84.93 ± 0.65	76.27 ± 1.54	NA	NA	63.36 ± 2.66	NA	NA	43.62 ± 9.44	8.17
MultiSetMixer	69.69 ± 1.36	65.71 ± 1.46	83.01 ± 0.65	74.88 ± 1.01	89.11 ± 0.21	99.08 ± 0.33	74.82 ± 2.10	97.52 ± 0.20	**86.92 ± 1.66**	73.53 ± 7.58	6.10
MLP CB	62.42 ± 1.37	64.85 ± 1.30	81.43 ± 0.86	62.82 ± 1.80	82.02 ± 0.36	99.61 ± 0.18	68.80 ± 1.51	95.16 ± 0.26	85.21 ± 3.81	67.46 ± 9.14	10.70
MLP	38.64 ± 12.37	64.21 ± 1.53	83.56 ± 0.49	37.85 ± 11.79	81.88 ± 0.21	99.72 ± 0.15	63.39 ± 2.24	93.71 ± 0.38	79.63 ± 0.42	72.17 ± 9.21	11.60
Transformer	61.45 ± 1.66	63.75 ± 1.39	83.86 ± 0.50	60.65 ± 1.87	82.40 ± 0.47	99.74 ± 0.20	65.14 ± 1.61	94.66 ± 0.42	79.61 ± 0.39	68.82 ± 8.32	11.15

C.5 Experiment results

C.5.1 Benchmarking models across multiple training proportions splits

C.6 Comparisons with others Homophily measure in literature

K-uniform homophily measure

Hyperedge homophily. Veldt et al. [213] defines the group homophily measure for k-uniform hypergraphs as $G_k = (\mathcal{V}, \mathcal{E})$. The type t-affinity score for each $t \in \{1, \ldots, k\}$, indicates the likelihood of a node belonging to class c participating in groups in which exactly t group members also belong to class c and defined as in equation C.46. $d_t(v)$ is the number of hyperedges that v belongs to with exactly t members from class c. The authors also consider a standard baseline score $b_t(c)$, equation C.47, that measures the probability that a class-c node is in the group where t members are from class c, given that the other $k-1$ nodes were chosen uniformly at random.

$$\eta_t(c) = \frac{\sum_{v:y_v=c} d_t(v)}{\sum_{v:y_v=c} d_v}, \qquad \text{(C.46)} \qquad b_t(c) = \frac{\binom{n_c-1}{t-1}\binom{n-n_c}{k-t}}{\binom{n-1}{k-1}} \qquad \text{(C.47)}$$

n_c is the number of nodes in class c and n is the total number of nodes in the hypergraph. The k-uniform hypergraph homophily measure can be expressed as

a ratio of affinity and baseline scores, with a ratio value of 1 indicating that the group is formed uniformly at random, while any other number indicates that group interactions are either overexpressed or underexpressed for class c.

They suggest three possible ways for extending the concept of homophily to the hypergraph context. The first one is the *simple* homophily and it means that $\eta_t(c) > b_t(c)$ for $t = k$ and check whether a class has a higher-than-baseline affinity for group interactions that only involve members of their class. The second one is *order-j majority* homophily that is obtained when the top j affinity scores for one class are higher than the baseline, i.e. $\eta_{k-j+1}(c) > b_{k-j+1}(c), \ldots, \eta_k(c) > b_k(c)$. The last one they consider is *order-j monotonic* homophily, which corresponds to the case when top j ratio scores are increasing monotonic, i.e. $\eta_k(c)/b_k(c) > \eta_{k-1}(c)/b_{k-1}(c) > \cdots > \eta_{k-j+1}(c) > b_{k-j+1}(c)$.

Finally, considering that the value $\eta_t(c) - b_t(c)$ is the *bias* of class c for type t, they introduce a type-t *normalized bias* score that normalizes the maximum possible bias, hence the obtained metric is bounded in $[0, 1]$ and it is computed as:

$$f_t(c) = \begin{cases} \frac{\eta_t(c) - b_t(c)}{1 - b_t(c)} & \text{if } \eta_t(c) \geq b_t(c) \\ \frac{\eta_t(c) - b_t(c)}{b_t(c)} & \text{if } \eta_t(c) < b_t(c) \end{cases} \tag{C.48}$$

Comparisons to our measure Unlike [213] our measure of homophily does not assume a k-uniform hypergraph structure and can be defined for any hypergraph. Furthermore, the proposed measure enables the definition of a score for each node and hyperedge for any neighborhood resolution, i.e., the connectivity of the hypergraph can be explicitly investigated. It gives a definition of homophily that puts more emphasis on the connections following the two-step message passing mechanism starting from the hyperedges of the hypergraph.

Social equivalence

The social equivalence [197] is calculated through an expectation taken over pairs of users sampled from probability distributions. Specifically, $E(u, v \sim P(E_p))$ represents the expectation over positive user pairs sampled from the distribution of the hyperedges of the hypergraph $P(E_p)$. In contrast, $E(u_0, v_0 \sim P(V \times V \setminus E_p))$ represents the expectation over negative user pairs sampled from $P(V \times V \setminus E_p)$. The measure is a fraction between the numerator that involves the expected Jaccard index of environments for positive user pairs and the denominator, which comprises a similar calculation for negative user pairs. The Jaccard index is used to measure the similarity between two sets, and in this context, it assesses the similarity of hyperedges associated with positive user pairs. If the expected Jaccard index for positive user pairs is higher than that for negative pairs, the measure exceeds 1, indicating a significant level of observed social equivalence among users.

Comparisons to our measure The concept of Social Equivalence, as introduced by Sun et al. [197], differs significantly from our definition of homophily. In our approach, the measure is initially defined at both the node and hyperedge levels. Our primary objective is to address the question: 'How similar a node is to its neighbors?' Following a message passing scheme, our definition allows us to examine different time points, attempting to answer how similar a node is to the nodes it can reach

with t steps of message passing. This consideration also extends to edges, where we seek to understand the coherence or uniformity of an edge within itself. The guiding notion of similarity in our work is belonging to the same class. Specifically, in level-0 homophily, we evaluate whether a node is more connected with nodes of the same classes. We can then aggregate the node-level/hyperedge-level homophily to provide a definition for hypergraph homophily. In contrast, in the work of Sun et al. [197], the concept of social equivalence yields a single result for the entire hypergraph. The approach involves comparing the set of similar nodes that are connected with those that are not connected. The key question is whether the set of non-connected nodes is, on average, more similar or if the set of connected pairs exhibits greater similarity. It's important to note that this definition makes sense only for the entire hypergraph and captures a different notion of similarity.

Social conformity

Social conformity, as described in [197], involves leveraging learned representations of users and hyperedges within a model to understand and quantify the level of conformity among users in a social network.

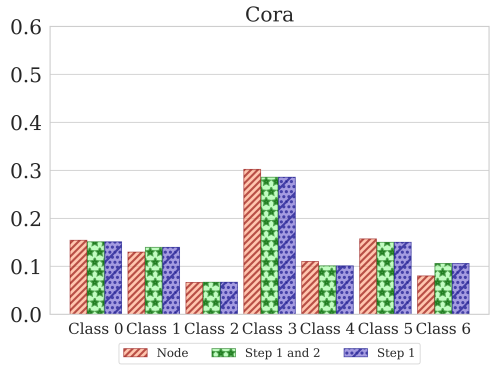
Node homophily computed on the clique expansion of the hypergraph

Clique-expanded (CE) homophily, employed in Wang et al. [222], is determined by calculating node homophily [163] on the graph derived from the clique-expanded hypergraphs.

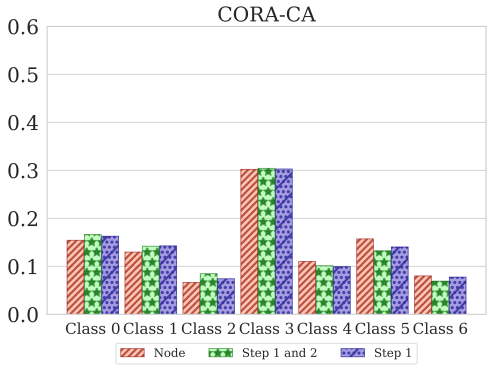
Comparisons to our measure The node homophily for a graph computes the fraction of neighbors with the same class for all nodes and then averages these values across the nodes. In contrast to our metric, CE homophily is not defined directly on the hypergraph but necessitates an expanded clique representation. While sharing some similarities with our node-wise measure h_0, it lacks the dynamic aspect inherent in the MP homophily measure, consequently failing to capture the dynamic information within connections. Our analysis in Section 7.2 underscores the significance of this dynamic element in understanding the correlation between homophily measures and the observed patterns in Hypergraph Neural Networks (HNNs).

C.7 Class Distribution Shift

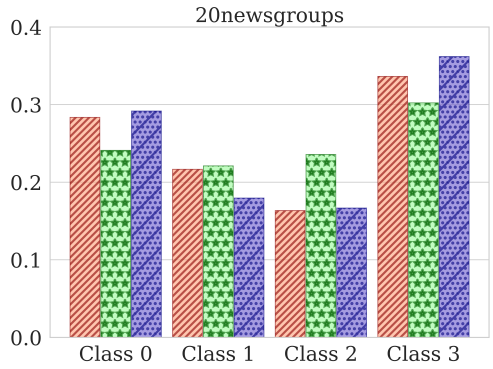
We now report the results for the class distribution shift obtained by applying the mini-batch sampling procedure described in Section C.2. For each dataset, we choose to illustrate 3 different distributions: the one corresponding to the original labels ("Node"), the one obtained by applying both *Step 1* and *Step 2* described in the mini-batch paragraph of Section 7.3.2 and the one obtained by only applying *Step 2*.



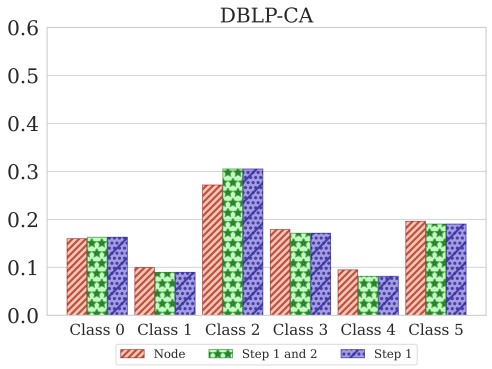
(a) Class distribution for Cora.



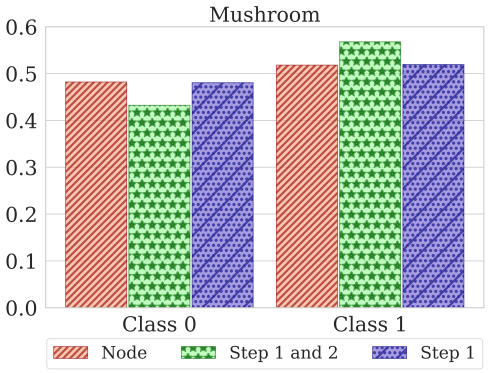
(b) Class distribution for CORA-CA.



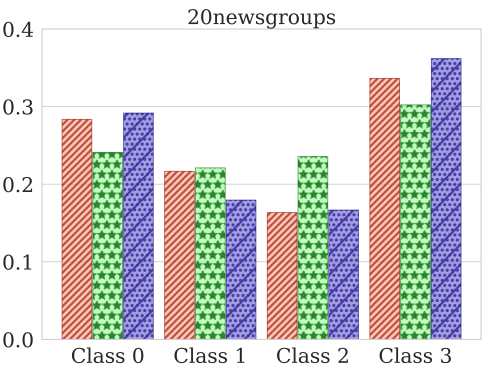
(c) Class distribution for 20Newsgroup.



(d) Class distribution for DBLP-CA.



(e) Class distribution for Mushroom.



(f) Class distribution for 20NewsW100.

Figure C.1. Class distribution shifts.

Appendix D

TopoBenchmark: A Framework for Benchmarking Topological Deep Learning

Contents

D.1 Code availability and reproducibility

The code for `TopoBenchmark` is available on GitHub under MIT license: https://github.com/pyt-team/TopoBenchmarkX. The codebase adheres to best software practices, employs continuous integration, is fully documented, and provides API documentation on its website https://pyt-team.github.io/topobenchmarkx/index.html.

The README.md comprehensively covers all aspects of library installation and development. To replicate the experiments outlined in this work, refer to the 'Experiments Reproducibility' section in the README.md. Additionally, the 'tutorials' folder contains various tutorials showcasing the integration of new models, datasets, and transforms.

D.2 Training details and additional results

In this section, we delve into the details of our hyperparameter search methodology, optimization strategy, computational resources utilized for conducting the experiments, and present additional results and analyses.

D.2.1 Experiments' configuration and model execution

To automate the configuration of the `TopoBenchmark` modules we utilize `hydra` package [242]. In particular, we rely on hierarchical configuration groups and registers to facilitate the easy use of the library. Specifically, there is no need to meticulously select each module to execute the pipeline in any of the domains. Simply choosing a dataset and a model automatically configures a full default pipeline, eliminating the need for manual intervention.

`TopoBenchmark` also automates the model execution and learning through the `lightning` library [140], orchestrating the training, validation and testing processes –as well as handling any logging and callbacks.

D.2.2 Hyperaparameter search

Five splits are generated for each dataset to ensure a fair evaluation of the models across domains. Each split comprises 50% training data, 25% validation data, and 25% test data. An exception is made for the ZINC dataset, where pre-defined splits are used [107].

Each model in every domain has numerous specific hyperparameters that can be tuned to enhance performance. TNNs, in particular, have multiple additional parameters that can potentially improve results. To avoid the combinatorial explosion of possible hyperparameter sets, the search space is limited to the hyperparameters common across all models. The grid-search strategy is employed to find the optimal hyperparameters for each model and dataset. Specifically, the encoder hidden dimension is varied over [32, 64, 128], the encoder dropout is varied over [0.25, 0.5], the number of backbone layers is varied over [1, 2, 3, 4], the learning rate is varied over [0.01, 0.001], and the batch size is varied over [128, 256]. Additionally, for the cellular and simplicial domains, the readout type is varied over [direct readout (DR), signal down-propagation (SDP)]. If the model cannot fit into memory, the batch size, encoder hidden dimension, and number of backbone layers are decreased until the model training fits into the GPU memory.

For node-level task datasets, validation is conducted after each training epoch, continuing until either the maximum number of epochs is reached or the optimization metric fails to improve for 50 consecutive validation epochs. The minimum number of epochs is set to 50. Conversely, for graph-level tasks, validation is performed every 5 training epochs, with training halting if the performance metric does not improve on the validation set for the last 10 validation epochs. To optimize the models, `torch.optim.Adam` is combined with `torch.optim.lr_scheduler.StepLR` wherein the step size was set to 50 and the gamma value to 0.5. In total more than 100,000 runs have been performed to obtain the results. The optimal hyperparameter set is generally selected based on the best average performance over five validation splits. For the ZINC dataset, five different initialization seeds are used to obtain the average performance.

The hyperparameter search is executed on a Linux machine with 256 cores, 1TB of system memory, and 8 NVIDIA A30 GPUs, each with 24GB of GPU memory.

D.2.3 Results

Table D.1 additionally presents the results for the CCXN and SCCN networks, which on average, perform slightly worse than the other models. As shown in Table D.2, the CCXN network performs better when using the SDP readout strategy, although the performance improvements are not as significant as those achieved by the CWN network with the same strategy. The SCCN model benefits more from utilizing the SDP readout compared to other simplicial domain models (SCN and SCCNN), showing performance improvements in 9 out of 21 cases, while SCCNN and SCN show improvements in only 3 and 5 cases, respectively. Overall, cellular models demonstrate performance enhancements on 15, 19, and 8 datasets for CCXN, CWN, and CCCN, respectively, with the SDP readout strategy. Conversely, simplicial domain models achieve 9, 3, and 5 performance improvements for SCCN, SCCNN, and SCN, respectively, with the same SDP strategy. Based on this analysis and the findings in Section 8.5.3, it is recommended for the new people in the TDL to opt for the SDP readout strategy if they prefer not to delve into the intricate details of interior propagation in cellular and simplicial models.

Note that for demonstration purposes, only one fixed lifting is considered for lifting graphs to each of the considered topological domains, leaving out of the scope of this paper the analysis of the optimal lifting in each particular scenario.[1] In particular, the clique complex lifting is applied to lift the graph to a simplicial domain. Cycle-based lifting is employed to lift the graph into the cellular domain, and k-hop lifting is utilized to lift the graph into a hypergraph, where $k = 1$. Projection lifting is applied to lift the features, specifically, the features of the $n-1$-cell projected to the n-cell by multiplying $n-1$-cell representation to corresponding incidence matrices.

Table D.1. Cross-domain comparison: results are shown as mean and standard deviation. The best result is bold and shaded in grey, while those within one standard deviation are in blue-shaded boxes.

	Dataset	GCN	GIN	GAT	AST	EDGNN	UniGNN2	CCXN	CWN	CCCN	SCCN	SCCNN	SCN
Node-level tasks	Cora	87.09 ± 0.2	87.21 ± 1.89	86.71 ± 0.95	**88.92 ± 0.44**	87.06 ± 1.09	86.97 ± 0.88	86.79 ± 1.81	86.32 ± 1.38	87.44 ± 1.28	80.86 ± 2.16	82.19 ± 1.07	82.27 ± 1.34
	Citeseer	75.53 ± 1.27	73.73 ± 1.23	74.41 ± 1.75	73.85 ± 2.21	74.93 ± 1.39	74.72 ± 1.08	74.67 ± 2.24	75.2 ± 1.82	**75.63 ± 1.58**	69.6 ± 1.83	70.23 ± 2.09	71.24 ± 1.68
	Pubmed	89.4 ± 0.3	89.29 ± 0.41	89.44 ± 0.24	**89.62 ± 0.25**	89.04 ± 0.51	89.34 ± 0.45	88.91 ± 0.47	88.64 ± 0.36	88.52 ± 0.44	88.37 ± 0.48	88.18 ± 0.32	88.72 ± 0.5
	Amazon	49.56 ± 0.55	49.16 ± 1.02	50.17 ± 0.59	50.5 ± 0.27	48.18 ± 0.09	49.06 ± 0.08	48.93 ± 0.14	**51.9 ± 0.15**	50.26 ± 0.17	OOM	OOM	OOM
	Empire	78.16 ± 0.32	79.56 ± 0.2	84.02 ± 0.51	79.5 ± 0.13	81.01 ± 0.24	77.06 ± 0.2	81.44 ± 0.31	81.81 ± 0.62	82.14 ± 0.0	88.27 ± 0.14	**89.15 ± 0.32**	88.79 ± 0.46
	Minesweeper	87.52 ± 0.42	87.82 ± 0.34	89.64 ± 0.43	81.14 ± 0.05	84.52 ± 0.05	78.02 ± 0.0	88.88 ± 0.36	88.62 ± 0.04	89.42 ± 0.0	89.07 ± 0.25	89.0 ± 0.0	**90.32 ± 0.11**
	Tolokers	83.02 ± 0.71	80.72 ± 1.19	**84.43 ± 1.0**	83.26 ± 0.1	77.53 ± 0.01	77.35 ± 0.03	OOM	OOM	OOM	OOM	OOM	OOM
	Election	0.31 ± 0.02	**0.28 ± 0.02**	0.29 ± 0.02	0.29 ± 0.01	0.34 ± 0.02	0.37 ± 0.02	0.35 ± 0.02	0.34 ± 0.02	0.31 ± 0.02	0.53 ± 0.03	0.51 ± 0.03	0.46 ± 0.04
	Bachelor	0.29 ± 0.02	0.31 ± 0.03	**0.28 ± 0.02**	0.3 ± 0.03	0.29 ± 0.02	0.31 ± 0.02	0.32 ± 0.03	0.33 ± 0.03	0.31 ± 0.02	0.36 ± 0.02	0.34 ± 0.03	0.32 ± 0.02
	Birth	0.72 ± 0.09	0.72 ± 0.09	0.71 ± 0.09	0.71 ± 0.08	**0.7 ± 0.07**	0.73 ± 0.1	0.74 ± 0.11	0.72 ± 0.09	0.71 ± 0.09	0.82 ± 0.09	0.79 ± 0.12	0.71 ± 0.08
	Death	0.51 ± 0.04	0.52 ± 0.04	0.51 ± 0.04	**0.49 ± 0.05**	0.52 ± 0.05	0.51 ± 0.05	0.54 ± 0.06	0.54 ± 0.06	0.54 ± 0.06	0.58 ± 0.06	0.55 ± 0.05	0.52 ± 0.05
	Income	0.22 ± 0.03	0.21 ± 0.02	**0.2 ± 0.02**	0.21 ± 0.02	0.23 ± 0.02	0.23 ± 0.02	0.25 ± 0.03	0.25 ± 0.03	0.23 ± 0.02	0.29 ± 0.03	0.28 ± 0.03	0.25 ± 0.02
	Migration	0.8 ± 0.12	0.8 ± 0.1	**0.77 ± 0.13**	0.78 ± 0.12	0.8 ± 0.12	0.79 ± 0.12	0.85 ± 0.18	0.84 ± 0.13	0.84 ± 0.12	0.91 ± 0.18	0.9 ± 0.14	0.92 ± 0.2
	Unempl	0.25 ± 0.03	**0.22 ± 0.02**	0.23 ± 0.03	**0.22 ± 0.02**	0.26 ± 0.03	0.28 ± 0.02	0.27 ± 0.03	0.25 ± 0.03	0.24 ± 0.03	0.43 ± 0.04	0.43 ± 0.04	0.38 ± 0.04
Graph-level tasks	MUTAG	69.79 ± 6.8	79.57 ± 6.13	72.77 ± 2.77	71.06 ± 6.49	80.0 ± 4.9	**80.43 ± 4.09**	74.89 ± 5.51	**80.43 ± 1.78**	77.02 ± 9.32	70.64 ± 5.9	76.17 ± 6.63	73.62 ± 6.13
	PROTEINS	75.7 ± 2.14	75.2 ± 3.3	76.34 ± 1.66	**76.63 ± 1.74**	73.91 ± 4.39	75.2 ± 2.96	75.63 ± 2.57	76.13 ± 2.7	73.33 ± 2.3	75.05 ± 2.76	74.19 ± 2.86	75.27 ± 2.14
	NCI1	72.86 ± 0.69	74.26 ± 0.96	75.0 ± 0.99	75.18 ± 1.24	73.97 ± 0.82	73.02 ± 0.92	74.86 ± 0.82	73.93 ± 1.87	**76.67 ± 1.48**	76.17 ± 1.39	76.6 ± 1.75	74.49 ± 1.03
	NCI109	72.2 ± 1.22	74.42 ± 0.7	73.8 ± 0.73	73.75 ± 1.09	74.93 ± 2.5	70.76 ± 1.11	75.66 ± 1.3	73.8 ± 2.06	75.35 ± 1.5	75.49 ± 1.39	**77.12 ± 1.07**	75.7 ± 1.04
	IMDB-BIN	**72.0 ± 2.48**	70.96 ± 1.93	69.76 ± 2.65	70.32 ± 3.27	69.12 ± 2.92	71.04 ± 1.31	70.08 ± 1.21	70.4 ± 2.02	69.12 ± 2.82	70.88 ± 3.98	70.88 ± 2.25	70.8 ± 2.38
	IMDB-MUL	49.97 ± 2.16	47.68 ± 4.21	50.13 ± 3.87	**50.51 ± 2.92**	49.17 ± 4.35	49.76 ± 3.55	47.63 ± 3.45	49.71 ± 2.83	47.79 ± 3.45	49.71 ± 3.7	48.75 ± 3.98	49.49 ± 5.08
	REDDIT	76.24 ± 0.54	81.96 ± 1.36	75.68 ± 1.0	74.84 ± 2.68	83.24 ± 1.45	75.56 ± 3.19	82.84 ± 2.54	**85.52 ± 1.38**	85.12 ± 1.29	74.44 ± 1.74	77.24 ± 1.87	71.28 ± 2.06
	ZINC	0.62 ± 0.01	0.57 ± 0.04	0.61 ± 0.01	0.59 ± 0.02	0.51 ± 0.01	0.6 ± 0.01	0.4 ± 0.04	**0.34 ± 0.01**	**0.34 ± 0.02**	0.46 ± 0.08	0.36 ± 0.02	0.53 ± 0.04

[1] Note that learnable liftings can further optimize the predictive capacity of higher-order networks.

Table D.2. An ablation study compares the performance of CCXN, CWN, CCCN, SCCN, SCCNN, and SCN models on various datasets using two readout strategies, direct readout (DR) and signal down-propagation (SDP). SDP generally enhances CWN performance, whereas the effect of SDP on CCCN, SCCNN, and SCN varies based on their internal signal propagation mechanisms. Means and standard deviations of performance metris are shown. The best results are shown in bold for each model and readout type.

	Dataset	CCXN		CWN		CCCN		SCCN		SCCNN		SCN	
		DR	SDP	DR	SDP	DR	SDP	DR	SDP	DR	SDP	DR	SDP
Node-level tasks	Cora	86.32 ± 1.22	**86.79 ± 1.81**	74.95 ± 0.98	**86.32 ± 1.38**	87.44 ± 1.28	**87.68 ± 1.17**	**80.86 ± 2.16**	80.06 ± 1.66	**82.19 ± 1.07**	80.65 ± 2.39	**82.27 ± 1.34**	79.91 ± 1.18
	Citeseer	72.87 ± 1.13	**74.67 ± 2.24**	70.49 ± 2.85	**75.2 ± 1.82**	**75.63 ± 1.58**	74.91 ± 1.25	**69.6 ± 1.83**	68.86 ± 2.40	**70.23 ± 2.69**	69.03 ± 2.01	**71.24 ± 1.68**	70.4 ± 1.53
	Pubmed	**88.91 ± 0.47**	88.38 ± 0.38	86.94 ± 0.68	**88.64 ± 0.36**	88.52 ± 0.44	**88.67 ± 0.39**	88.04 ± 0.51	**88.37 ± 0.48**	**88.18 ± 0.32**	87.78 ± 0.58	**88.72 ± 0.5**	88.62 ± 0.44
	Amazon	**48.93 ± 0.14**	48.34 ± 0.12	45.58 ± 0.25	**51.9 ± 0.15**	**50.55 ± 0.15**	50.26 ± 0.17	OOM	OOM	OOM	OOM	OOM	OOM
	Empire	80.46 ± 0.23	**81.44 ± 0.31**	66.13 ± 0.03	**81.81 ± 0.62**	82.14 ± 0.00	**82.51 ± 0.0**	88.2 ± 0.22	**88.27 ± 0.14**	**89.15 ± 0.32**	88.73 ± 0.12	85.89 ± 0.34	**88.79 ± 0.46**
	Minesweeper	88.88 ± 0.36	**89.76 ± 0.32**	48.82 ± 0.0	**88.62 ± 0.04**	89.42 ± 0.00	**89.85 ± 0.00**	88.85 ± 0.00	**89.07 ± 0.25**	87.4 ± 0.0	**89.0 ± 0.00**	**90.32 ± 0.11**	90.27 ± 0.36
	Election	0.39 ± 0.05	**0.35 ± 0.02**	0.6 ± 0.04	**0.34 ± 0.02**	**0.31 ± 0.02**	**0.31 ± 0.01**	**0.53 ± 0.03**	0.57 ± 0.02	**0.51 ± 0.03**	0.56 ± 0.04	**0.46 ± 0.04**	0.51 ± 0.03
	Bachelor	0.33 ± 0.03	**0.32 ± 0.03**	**0.33 ± 0.03**	**0.33 ± 0.03**	0.32 ± 0.02	**0.31 ± 0.02**	0.36 ± 0.02	**0.34 ± 0.02**	**0.34 ± 0.03**	**0.34 ± 0.03**	**0.32 ± 0.02**	**0.32 ± 0.03**
	Birth	0.80 ± 0.12	**0.74 ± 0.11**	0.81 ± 0.11	**0.72 ± 0.09**	**0.71 ± 0.09**	0.72 ± 0.05	**0.82 ± 0.09**	0.83 ± 0.10	**0.79 ± 0.12**	0.83 ± 0.12	**0.71 ± 0.08**	0.8 ± 0.11
	Death	0.57 ± 0.06	**0.54 ± 0.06**	0.55 ± 0.05	**0.54 ± 0.06**	**0.54 ± 0.06**	**0.54 ± 0.06**	0.58 ± 0.06	**0.56 ± 0.04**	**0.55 ± 0.05**	0.58 ± 0.05	**0.52 ± 0.05**	0.56 ± 0.05
	Income	**0.25 ± 0.03**	**0.25 ± 0.03**	0.36 ± 0.04	**0.25 ± 0.03**	**0.23 ± 0.02**	**0.23 ± 0.02**	**0.29 ± 0.03**	**0.29 ± 0.03**	**0.28 ± 0.03**	0.31 ± 0.03	**0.25 ± 0.02**	0.27 ± 0.02
	Migration	**0.80 ± 0.11**	0.85 ± 0.18	0.9 ± 0.16	**0.84 ± 0.13**	**0.84 ± 0.10**	**0.84 ± 0.12**	**0.91 ± 0.18**	0.93 ± 0.17	**0.90 ± 0.14**	0.93 ± 0.17	**0.92 ± 0.20**	0.96 ± 0.23
	Unempl	0.28 ± 0.05	**0.27 ± 0.03**	0.46 ± 0.04	**0.25 ± 0.03**	**0.24 ± 0.03**	0.25 ± 0.03	**0.43 ± 0.04**	0.47 ± 0.04	**0.43 ± 0.04**	0.45 ± 0.04	**0.38 ± 0.04**	0.41 ± 0.03
Graph-level tasks	MUTAG	69.79 ± 4.61	**74.89 ± 5.51**	69.68 ± 8.58	**80.43 ± 1.78**	**80.85 ± 5.42**	77.02 ± 9.32	70.64 ± 5.90	**73.62 ± 4.41**	**76.17 ± 6.63**	70.64 ± 3.16	71.49 ± 2.43	**73.62 ± 6.13**
	PROTEINS	**75.63 ± 2.57**	74.91 ± 1.85	**76.13 ± 1.80**	**76.13 ± 2.70**	**73.55 ± 3.43**	73.33 ± 2.30	**75.05 ± 2.76**	74.34 ± 3.17	74.19 ± 2.86	**74.98 ± 1.92**	**75.27 ± 2.14**	74.77 ± 1.69
	NCI1	72.43 ± 1.72	**74.86 ± 0.82**	68.52 ± 0.51	**73.93 ± 1.87**	76.67 ± 1.48	**77.65 ± 1.28**	**76.42 ± 0.88**	76.17 ± 1.39	**76.6 ± 1.75**	75.6 ± 2.45	**75.27 ± 1.57**	74.49 ± 1.03
	NCI109	73.22 ± 0.48	**75.66 ± 1.30**	68.19 ± 0.65	**73.8 ± 2.06**	**75.35 ± 1.50**	74.83 ± 1.18	**75.49 ± 1.39**	75.31 ± 1.36	**77.12 ± 1.07**	75.43 ± 1.94	74.58 ± 1.29	**75.7 ± 1.04**
	IMDB-BIN	**70.08 ± 1.21**	68.96 ± 2.03	**70.4 ± 2.02**	69.28 ± 2.57	69.12 ± 2.82	**69.44 ± 2.46**	**70.88 ± 3.98**	69.76 ± 3.16	**70.88 ± 2.25**	69.28 ± 5.69	**70.8 ± 2.38**	68.64 ± 3.90
	IMDB-MUL	47.63 ± 3.45	**48.75 ± 3.56**	49.71 ± 2.83	**49.87 ± 2.33**	**49.01 ± 2.63**	47.79 ± 3.45	**49.71 ± 3.70**	47.31 ± 3.12	**48.75 ± 3.98**	46.67 ± 3.13	48.16 ± 2.89	**49.49 ± 5.08**
	REDDIT	74.40 ± 1.50	**82.84 ± 2.54**	76.20 ± 0.86	**85.52 ± 1.38**	**85.12 ± 1.29**	83.32 ± 0.73	74.16 ± 1.54	**74.44 ± 1.74**	75.56 ± 3.46	**77.24 ± 1.87**	**71.28 ± 2.06**	69.68 ± 4.0
	ZINC	0.63 ± 0.02	**0.40 ± 0.04**	0.70 ± 0.0	**0.34 ± 0.01**	0.35 ± 0.02	**0.34 ± 0.02**	0.55 ± 0.01	**0.46 ± 0.08**	**0.36 ± 0.01**	**0.36 ± 0.02**	0.59 ± 0.01	**0.53 ± 0.04**

Table D.3. Model sizes corresponding to the best set of hyperparameters

Model	GCN	GAT	GIN	AST	EDGNN	UniGNN2	CWN	CCCN	CCXN	SCN	SCCN	SCCNN
Cora	234.63K	113.61K	105.03K	60.26K	113.29K	109.06K	343.11K	451.85K	735.37K	144.62K	155.88K	164.17K
Citeseer	525.06K	558.60K	122.05K	132.87K	258.50K	541.32K	1754.50K	1032.84K	758.41K	737.29K	782.34K	893.13K
Pubmed	114.69K	148.23K	53.38K	280.83K	147.59K	114.56K	163.72K	85.76K	277.51K	134.40K	457.99K	605.06K
MUTAG	67.97K	22.02K	38.40K	80.77K	5.73K	84.10K	334.72K	284.29K	73.86K	20.03K	398.85K	27.11K
PROTEINS	13.19K	10.11K	13.19K	14.34K	5.60K	21.31K	101.12K	34.56K	86.53K	10.24K	397.31K	26.72K
NCI1	6.72K	11.20K	154.37K	57.47K	88.19K	104.32K	124.10K	63.87K	15.87K	94.98K	131.84K	188.99K
NCI109	23.75K	11.23K	154.50K	221.57K	88.32K	4.61K	412.29K	17.67K	71.36K	26.08K	135.75K	49.54K
IMDB-BIN	21.70K	21.83K	9.89K	114.24K	9.86K	100.61K	68.80K	218.24K	19.04K	202.63K	563.07K	285.83K
IMDB-MUL	62.08K	6.37K	18.76K	111.30K	27.01K	8.32K	19.68K	45.64K	56.26K	27.94K	545.16K	121.22K
REDDIT	13.63K	30.66K	10.08K	106.18K	5.83K	68.10K	26.66K	47.94K	57.54K	7.84K	69.31K	286.59K
Amazon	122.37K	156.16K	89.35K	155.91K	122.24K	89.22K	578.95K	310.15K	200.96K	OOM	OOM	OOM
Minesweeper	9.28K	5.89K	51.46K	118.02K	21.70K	51.33K	22.24K	35.07K	8.83K	51.97K	25.15K	33.44K
Empire	37.39K	41.81K	33.23K	257.17K	41.49K	90.90K	142.74K	86.10K	43.83K	415.89K	612.50K	240.53K
Tolokers	84.87K	151.94K	3.75K	217.99K	21.89K	13.57K	12.07K	OOM	OOM	OOM	OOM	OOM
Election	84.22K	151.30K	150.27K	217.34K	21.57K	4.58K	118.08K	234.37K	253.18K	16.64K	11.52K	415.10K
Bachelor	17.47K	151.30K	7.81K	316.93K	84.10K	3.55K	43.78K	34.88K	12.86K	27.14K	43.52K	415.10K
Birth	4.64K	30.34K	5.70K	30.21K	84.10K	13.25K	26.24K	22.40K	253.18K	103.42K	11.52K	26.98K
Death	21.63K	10.18K	7.81K	316.93K	84.10K	51.07K	26.24K	15.58K	12.86K	27.14K	11.52K	415.10K
Income	17.47K	151.30K	9.92K	217.34K	21.57K	4.58K	85.18K	12.42K	12.86K	27.14K	268.03K	105.15K
Migration	67.71K	10.18K	38.27K	80.64K	21.57K	51.07K	26.24K	15.58K	65.15K	16.64K	11.52K	415.10K
Unempl	84.22K	151.30K	117.25K	105.86K	21.57K	5.60K	101.63K	234.37K	286.46K	27.14K	11.52K	105.15K
ZINC	22.59K	22.85K	10.40K	106.82K	22.53K	102.14K	88.06K	287.74K	16.48K	24.42K	617.86K	1453.82K
Average size	75K ± 111K	89K ± 119K	54K ± 53K	150K ± 88K	59K ± 60K	74K ± 109K	210K ± 367K	170K ± 229K	158K ± 212K	107K ± 172K	263K ± 251K	313K ± 340K

Table D.4. TNNs utilized in the experiments and their references

Acronym	Neural network name	Reference
	Graph neural networks	
GAT	Graph attention network	[214]
GIN	Graph isomorphism network	[240]
GCN	Semi-Supervised Classification with Graph Convolutional Networks	[122]
	Simplicial complexes	
SAN	Simplicial Attention Neural Networks	[76]
SCCN	Efficient Representation Learning for Higher-Order Data with Simplicial Complexes	[249]
SCCNN	Convolutional Learning on Simplicial Complexes	[246]
SCN	Simplicial Complex Neural Networks	[67]
	Cellular complexes	
CAN	Cell Attention Network	[77]
CCCN	Generalized simplicial attention neural networks [2]	[18]
CXN	Cell Complex Neural Networks	[86]
CWN	Weisfeiler and Lehman Go Cellular: CW Networks	[32]
	Hypergraphs	
AllSetTransformer	You are AllSet: A Multiset Function Framework for Hypergraph Neural Networks	[48]
EDGNN	Equivariant Hypergraph Diffusion Neural Operators	[221]
UniGNN	UniGNN: a Unified Framework for Graph and Hypergraph Neural Networks	[106]

D.3 Descriptive summaries of datasets

Table D.5 provides the statistics for each dataset lifted to three topological domains: simplicial complex, cellular complex, and hypergraph. The 0-cell, 1-cell, 2-cell, and 3-cell represents the count of n-cell each dataset have after the topology lifting procedure. The clique complex lifting is applied to lift the graph to a simplicial domain with a maximum n equal to three. Cycle-based lifting is employed to lift the graph into the cellular domain (maximum n is equal to two), and k-hop lifting is utilized to lift the graph into a hypergraph, where $k = 1$.

Table D.5. Descriptive summaries of the datasets used in the experiments.

	Cora	Citeseer	PubMed	MUTAG	NCI1	NCI109	PROTEINS	REDDIT-BINARY
Cellular								
0-cell	2708	3327	19717	3371	122747	122494	43471	859254
1-cell	5278	4552	44324	3721	132753	132604	81044	995508
2-cell	2648	1663	23605	538	14885	15042	38773	141218
3-cell	0	0	0	0	0	0	0	0
Simplicial								
0-cell	2708	3327	19717	3371	122747	122494	43471	859254
1-cell	5278	4552	44324	3721	132753	132604	81044	995508
2-cell	1630	1167	12520	0	186	183	30501	49670
3-cell	220	255	3275	0	0	0	3502	1303
Hypergraph								
0-cell	2708	3327	19717	3371	122747	122494	43471	859254
Num. Hyperedges	2708	3327	19717	3371	122747	122494	43471	859254

	IMDB-BINARY	IMDB-MULTI	ZINC	Amazon Ratings	Minesweeper	Roman Empire	Tolokers	US-county-demos
Cellular								
0-cell	19773	19502	277864	24492	10000	22662	OOM	3224
1-cell	96531	98903	298985	93050	39402	32927	OOM	9483
2-cell	77758	80901	33121	68553	28955	10266	OOM	6266
3-cell	0	0	0	0	0	0	OOM	0
Simplicial								
0-cell	19773	19502	277864	24492	10000	22662	OOM	3224
1-cell	96531	98903	298985	93050	39402	32927	OOM	9483
2-cell	391991	458850	769	110765	39204	7168	OOM	6490
3-cell	1694513	2343676	0	64195	9801	0	OOM	225
Hypergraph								
0-cell	19773	19502	277864	24492	10000	22662	11758	3224
Num. Hyperedges	19773	19502	277864	24492	10000	22662	11758	3224

Bibliography

[1] E. Acuna and C. Rodriguez. The treatment of missing values and its effect on classifier accuracy. In *Classification, clustering, and data mining applications*, pages 639–647. Springer, 2004. (see pages: 19, 29)

[2] S. Agarwal, K. Branson, and S. Belongie. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*, pages 17–24, 2006. (see pages: 79, 126)

[3] G. Aguilar, Y. Ling, Y. Zhang, B. Yao, X. Fan, and C. Guo. Knowledge distillation from internal representations. In *Proc. of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7350–7357, 2020. (see page: 54)

[4] P. Almasan, K. Rusek, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, and P. Barlet-Ros. Leveraging spatial and temporal correlations for network traffic compression. *arXiv preprint arXiv:2301.08962*, 2023. (see page: 51)

[5] R. Aponte, R. A. Rossi, S. Guo, J. Hoffswell, N. Lipka, C. Xiao, G. Chan, E. Koh, and N. Ahmed. A hypergraph neural network framework for learning hyperedge-dependent node embeddings. *arXiv preprint arXiv:2212.14077*, 2022. (see page: 127)

[6] D. Arya, D. K. Gupta, S. Rudinac, and M. Worring. Hypersage: Generalizing inductive representation learning on hypergraphs. *arXiv preprint arXiv:2010.04558*, 2020. (see pages: 79, 80, 126, and 127)

[7] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research*, 20(1):40–49, 2011. (see page: 29)

[8] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. (see pages: 131, 134)

[9] G. Bahl, M. Bahri, and F. Lafarge. Single-shot end-to-end road graph extraction. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1402–1411, 2022. doi: 10.1109/CVPRW56347.2022.00146. (see page: 3)

[10] S. Bai, F. Zhang, and P. H. Torr. Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637, 2021. (see pages: 67, 79, 80, 126, 127, 129, and 138)

[11] M. Balcilar, P. Héroux, B. Gauzere, P. Vasseur, S. Adam, and P. Honeine. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pages 599–608. PMLR, 2021. (see page: 127)

[12] S. Barbarossa and S. Sardellitti. Topological signal processing over simplicial complexes. *IEEE Transactions on Signal Processing*, 68:2992–3007, 2020. (see pages: 33, 34, and 92)

[13] S. Barbarossa and S. Sardellitti. Topological Signal Processing over Simplicial Complexes. *IEEE Transactions on Signal Processing*, 68:2992–3007, 2020. ISSN 1053-587X, 1941-0476. doi: 10.1109/TSP.2020.2981920. (see page: 107)

[14] S. Barbarossa and S. Sardellitti. Topological signal processing over simplicial complexes. *IEEE Transactions on Signal Processing*, 68:2992–3007, 2020. (see page: 2)

[15] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng. Spectral sparsification of graphs: theory and algorithms. *Communications of the ACM*, 56(8): 87–94, 2013. (see page: 3)

[16] C. Battiloro, S. Sardellitti, S. Barbarossa, and P. D. Lorenzo. Topological signal processing over weighted simplicial complexes. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023. (see page: 41)

[17] C. Battiloro, I. Spinelli, L. Telyatnikov, M. Bronstein, S. Scardapane, and P. D. Lorenzo. From latent graph to latent topology inference: Differentiable cell complex module. *arXiv preprint arXiv:2305.16174*, 2023. (see pages: 7, 32, 86, and 93)

[18] C. Battiloro, L. Testa, L. Giusti, S. Sardellitti, P. Di Lorenzo, and S. Barbarossa. Generalized simplicial attention neural networks. *arXiv preprint arXiv:2309.02138*, 2023. (see page: 148)

[19] C. Battiloro, E. Karaismailoğlu, M. Tec, G. Dasoulas, M. Audirac, and F. Dominici. E (n) equivariant topological neural networks. *arXiv preprint arXiv:2405.15429*, 2024. (see page: 4)

[20] F. Battiston, E. Amico, A. Barrat, G. Bianconi, G. Ferraz de Arruda, B. Franceschiello, I. Iacopini, S. Kéfi, V. Latora, Y. Moreno, et al. The physics of higher-order interactions in complex systems. *Nature Physics*, 17 (10):1093–1098, 2021. (see page: 92)

[21] A. Behrouz and F. Hashemi. Graph Mamba: Towards Learning on Graphs with State Space Models, Feb. 2024. (see pages: 100, 107)

[22] Y. Bengio and F. Gingras. Recurrent neural networks for missing or asynchronous data. *Advances in neural information processing systems*, 8, 1995. (see page: 29)

[23] A. R. Benson, R. Abebe, M. T. Schaub, A. Jadbabaie, and J. Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48):E11221–E11230, 2018. (see page: 1)

[24] G. Bernárdez, L. Telyatnikov, E. Alarcón, A. Cabellos-Aparicio, P. Barlet-Ros, and P. Liò. Topological graph signal compression. *arXiv preprint arXiv:2308.11068*, 2023. (see pages: 7, 43, 86, and 93)

[25] G. Bernárdez, L. Telyatnikov, E. Alarcón, A. Cabellos-Aparicio, P. Barlet-Ros, and P. Liò. Topological network traffic compression. In *Proceedings of the 2nd on Graph Neural Networking Workshop 2023*, GNNet '23, page 7–12, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400704482. doi: 10.1145/3630049.3630172. URL https://doi.org/10.1145/3630049.3630172. (see pages: 7, 43)

[26] G. Bernárdez, L. Telyatnikov, M. Montagna, F. Baccini, M. Papillon, M. Ferriol-Galmés, M. Hajij, T. Papamarkou, M. S. Bucarelli, O. Zaghen, et al. Icml topological deep learning challenge 2024: Beyond the graph domain. *arXiv preprint arXiv:2409.05211*, 2024. (see pages: 7, 83, 96, and 112)

[27] D. Bertsimas, C. Pawlowski, and Y. D. Zhuo. From predictive methods to missing data imputation: an optimization approach. *J. Mach. Learn. Res.*, 18 (1):7133–7171, 2017. (see pages: 19, 29)

[28] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. (see page: 114)

[29] M. Blondel, A. Martins, and V. Niculae. Learning classifiers with fenchel-young losses: Generalized entropies, margins, and algorithms. In *AISTATS*, pages 606–615. PMLR, 2019. (see page: 118)

[30] C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Lio, G. F. Montufar, and M. Bronstein. Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021. (see pages: 2, 12)

[31] C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Liò, G. F. Montufar, and M. Bronstein. Weisfeiler and lehman go cellular: Cw networks. In *Advances in Neural Information Processing Systems*, 2021. (see pages: 4, 34, 35, 38, 39, 41, and 118)

[32] C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. Montufar, P. Liò, and M. Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, 2021. (see pages: 86, 148)

[33] C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. F. Montufar, P. Lio, and M. Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pages 1026–1037. PMLR, 2021. (see pages: 2, 92)

[34] C. Bodnar, F. D. Giovanni, B. P. Chamberlain, P. Lio, and M. M. Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in GNNs. In *Advances in Neural Information Processing Systems*, 2022. (see page: 39)

[35] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman. Token merging: Your vit but faster. In *The Eleventh International Conference on Learning Representations*, 2022. (see page: 57)

[36] A. Bonifati, I. Holubová, A. Prat-Pérez, and S. Sakr. Graph generators: State of the art and open challenges. *ACM computing surveys (CSUR)*, 53(2):1–30, 2020. (see page: 3)

[37] E. Bourtsoulatze, D. B. Kurka, and D. Gündüz. Deep joint source-channel coding for wireless image transmission. *IEEE Trans. on Cognitive Communications and Netw.*, 5(3):567–579, 2019. (see page: 54)

[38] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (see page: 27)

[39] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *ICLR*, 2022. (see pages: 1, 14, and 50)

[40] M. Burgess, E. Adar, and M. Cafarella. Link-prediction enhanced consensus clustering for complex networks. *PloS one*, 11(5):e0153384, 2016. (see page: 3)

[41] S. Cantürk, R. Liu, O. Lapointe-Gagné, V. Létourneau, G. Wolf, D. Beaini, and L. Rampášek. Graph positional and structural encoder. *arXiv preprint arXiv:2307.07107*, 2023. (see page: 112)

[42] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003. (see page: 138)

[43] G. Chen and J. Zhang. Preventing over-smoothing for hypergraph neural networks. *arXiv preprint arXiv:2203.17159*, 2022. (see pages: 79, 126)

[44] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020. (see pages: 39, 127, and 130)

[45] X. Chen, S. Chen, J. Yao, H. Zheng, Y. Zhang, and I. W. Tsang. Learning on attribute-missing graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2020. (see pages: 19, 30)

[46] Y. Chen, Y. R. Gel, and H. V. Poor. BScNets: block simplicial complex neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022. (see page: 2)

[47] E. Chien, J. Peng, P. Li, and O. Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020. (see page: 79)

[48] E. Chien, C. Pan, J. Peng, and O. Milenkovic. You are allset: a multiset function framework for hypergraph neural networks. *arXiv preprint arXiv:2106.13264*, 2021. (see page: 148)

[49] E. Chien, C. Pan, J. Peng, and O. Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations*, 2022. (see pages: 63, 64, 67, 70, 79, 80, 126, 127, 128, 131, 132, 134, 136, 137, and 138)

[50] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. (see page: 105)

[51] M. Choe, S. Kim, J. Yoo, and K. Shin. Classification of edge-dependent labels of nodes in hypergraphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 298–309, 2023. (see pages: 64, 68, 80, 82, 134, 135, and 137)

[52] W. Cong, R. Forsati, M. Kandemir, and M. Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1393–1403, 2020. (see page: 30)

[53] G. M. Correia, V. Niculae, and A. F. T. Martins. Adaptively sparse transformers. In *EMNLP-IJCNLP*, pages 2174–2184. Association for Computational Linguistics, Nov. 2019. (see pages: 36, 118)

[54] L. Cosmo, A. Kazi, S.-A. Ahmadi, N. Navab, and M. Bronstein. Latent-graph learning for disease prediction. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 643–653. Springer, 2020. (see pages: 3, 30)

[55] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014. (see page: 54)

[56] O. T. Courtney and G. Bianconi. Dense power-law networks and simplicial complexes. *Phys. Rev. E*, 97:052303, May 2018. doi: 10.1103/PhysRevE.97.052303. (see page: 41)

[57] J. Dai, S. Wang, K. Tan, Z. Si, X. Qin, K. Niu, and P. Zhang. Nonlinear transform source-channel coding for semantic communications. *IEEE Journal on Selected Areas in Communications*, 40(8):2300–2316, 2022. (see pages: 54, 55, and 58)

[58] H. S. de Ocáriz Borde, A. Kazi, F. Barbero, and P. Lio. Latent graph inference using product manifolds. In *The Eleventh International Conference on Learning Representations*, 2023. (see pages: 3, 33, 39, 40, 41, 42, 118, 119, 120, and 121)

[59] H. S. de Ocáriz Borde, A. Kazi, F. Barbero, and P. Lio. Latent graph inference using product manifolds. In *The Eleventh International Conference on Learning Representations*, 2023. (see page: 3)

[60] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022. (see page: 54)

[61] A. Devoto, S. Petruzzi, J. Pomponi, P. D. Lorenzo, and S. Scardapane. Adaptive semantic token selection for ai-native goal-oriented communications. *ArXiv*, abs/2405.02330, 2024. URL https://api.semanticscholar.org/CorpusID:269606011. (see page: 54)

[62] J. Diffenderfer, A. L. Fox, J. A. Hittinger, G. Sanders, and P. G. Lindstrom. Error analysis of zfp compression for floating-point data. *SIAM Journal on Scientific Computing*, 41(3):A1867–A1898, 2019. (see pages: xiii, 49, 50, and 51)

[63] Y. Ding, A. Orvieto, B. He, and T. Hofmann. Recurrent Distance-Encoding Neural Networks for Graph Representation Learning. https://arxiv.org/abs/2312.01538v1, Dec. 2023. (see pages: 100, 106)

[64] Y. Dong, W. Sawin, and Y. Bengio. HNHN: hypergraph networks with hyperedge neurons. In *ICML Graph Representation Learning and Beyond Workshop*, 2020. (see pages: 67, 79, 80, 126, 127, 131, and 138)

[65] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. (see page: 55)

[66] D. Dua, C. Graff, et al. Uci machine learning repository, 2017. *URL http://archive. ics. uci. edu/ml*, 7(1), 2017. (see page: 138)

[67] S. Ebli, M. Defferrard, and G. Spreemann. Simplicial neural networks. In *NeurIPS Workshop on Topological Data Analysis and Beyond*, 2020. (see pages: 4, 148)

[68] Y. Eitan, Y. Gelberg, G. Bar-Shalom, F. Frasca, M. Bronstein, and H. Maron. Topological blind spots: Understanding and extending topological deep learning through the lens of expressivity. *arXiv preprint arXiv:2408.05486*, 2024. (see page: 4)

[69] Y. S. Elshakhs, K. M. Deliparaschos, T. Charalambous, G. Oliva, and A. Zolotas. A comprehensive survey on delaunay triangulation: Applications, algorithms, and implementations over cpus, gpus, and fpgas. *IEEE Access*, 2024. (see page: 93)

[70] A. Farhangfar, L. A. Kurgan, and W. Pedrycz. A novel framework for imputation of missing values in databases. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(5):692–709, 2007. (see page: 29)

[71] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565, 2019. (see pages: 4, 63, 67, 79, 80, 85, 126, 127, 128, 129, 136, and 138)

[72] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. (see page: 85)

[73] C. L. Giles, K. D. Bollacker, and S. Lawrence. CiteSeer: An automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, DL '98, pages 89–98, New York, NY, USA, May 1998. Association for Computing Machinery. ISBN 978-0-89791-965-4. doi: 10.1145/276675.276685. (see page: 105)

[74] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272. PMLR, 2017. (see page: 30)

[75] J. H. Giraldo, F. D. Malliaros, and T. Bouwmans. Understanding the relationship between over-smoothing and over-squashing in graph neural networks. *arXiv preprint arXiv:2212.02374*, 2022. (see page: 44)

[76] L. Giusti, C. Battiloro, P. Di Lorenzo, S. Sardellitti, and S. Barbarossa. Simplicial attention neural networks. *arXiv preprint arXiv:2203.07485*, 2022. (see pages: 4, 148)

[77] L. Giusti, C. Battiloro, L. Testa, P. Di Lorenzo, S. Sardellitti, and S. Barbarossa. Cell attention networks. In *International Joint Conference on Neural Networks*, 2023. (see pages: 4, 148)

[78] L. Gondara and K. Wang. Mida: Multiple imputation using denoising autoencoders. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 260–272. Springer, 2018. (see pages: 25, 29)

[79] A. Gu and T. Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, Dec. 2023. (see pages: 99, 106)

[80] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Re. HiPPO: Recurrent Memory with Optimal Polynomial Projections, Oct. 2020. (see page: 99)

[81] A. Gu, K. Goel, and C. Ré. Efficiently Modeling Long Sequences with Structured State Spaces, Aug. 2022. (see pages: 99, 106)

[82] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui. Implicit graph neural networks. *Advances in Neural Information Processing Systems*, 33: 11984–11995, 2020. (see page: 127)

[83] A. Hagberg and D. Conway. Networkx: Network analysis with python. *URL: https://networkx. github. io*, 2020. (see page: 95)

[84] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Lab (LANL), Los Alamos, NM, United States, 2008. (see page: 85)

[85] S. Haj Ali and M.-T. Hütt. Inferring missing edges in a graph from observed collective patterns. *Physical Review E*, 105(6):064610, 2022. (see page: 3)

[86] M. Hajij, K. Istvan, and G. Zamzmi. Cell complex neural networks. In *NeurIPS Workshop on Topological Data Analysis and Beyond*, 2020. (see pages: 4, 148)

[87] M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, and K. N. Ramamurthy. Higher-order attention networks. *arXiv preprint arXiv:2206.00606*, 2(3):4, 2022. (see page: 93)

[88] M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, K. N. Ramamurthy, T. Birdal, T. K. Dey, S. Mukherjee, S. N. Samaga, et al. Topological deep learning: Going beyond graph data. *arXiv preprint arXiv:2206.00606*, 2022. (see page: 12)

[89] M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, and K. N. Ramamurthy. Higher-order attention networks, 2022. (see page: 41)

[90] M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, K. N. Ramamurthy, T. Birdal, T. Dey, S. Mukherjee, S. Samaga, N. Livesay, R. Walters, P. Rosen, and M. Schaub. Topological deep learning: going beyond graph data. *arXiv preprint arXiv:1906.09068*, 2023. (see pages: 4, 12, 84, and 86)

[91] M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, K. N. Ramamurthy, T. Birdal, T. K. Dey, S. Mukherjee, S. N. Samaga, N. Livesay, R. Walters, P. Rosen, and M. T. Schaub. Topological Deep Learning: Going Beyond Graph Data, May 2023. (see page: 107)

[92] M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, K. N. Ramamurthy, T. Birdal, T. K. Dey, S. Mukherjee, S. N. Samaga, N. Livesay, R. Walters, P. Rosen, and M. T. Schaub. Topological deep learning: Going beyond graph data, 2023. (see pages: 2, 33, 35, 38, 41, 47, and 51)

[93] M. Hajij, M. Papillon, F. Frantzen, J. Agerberg, I. AlJabea, R. Ballester, C. Battiloro, G. Bernárdez, T. Birdal, A. Brent, P. Chin, S. Escalera, S. Fiorellino, O. H. Gardaa, G. Gopalakrishnan, D. Govil, J. Hoppe, M. R. Karri, J. Khouja, M. Lecha, N. Livesay, J. Meißner, S. Mukherjee, A. Nikitin, T. Papamarkou, J. Prílepok, K. N. Ramamurthy, P. Rosen, A. Guzmán-Sáenz, A. Salatiello, S. N. Samaga, S. Scardapane, M. T. Schaub, L. Scofano, I. Spinelli, L. Telyatnikov, Q. Truong, R. Walters, M. Yang, O. Zaghen, G. Zamzmi, A. Zia, and N. Miolane. TopoX: a suite of Python packages for machine learning on topological domains. *arXiv preprint arXiv:2402.02441*, 2024. (see pages: 84, 85, and 95)

[94] J. Halcrow, A. Mosoi, S. Ruth, and B. Perozzi. Grale: Designing networks for graph learning. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2523–2532, 2020. (see page: 79)

[95] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. (see pages: 1, 14, 30, and 50)

[96] J. Hansen and R. Ghrist. Toward a spectral theory of cellular sheaves. *Journal of Applied and Computational Topology*, 2019. (see page: 38)

[97] A. Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002. ISBN 0-521-79160-X; 0-521-79540-0. (see page: 10)

[98] M. Havasi. Advances in compression using probabilistic models. 2021. doi: 10.17863/CAM.79008. URL https://www.repository.cam.ac.uk/handle/1810/331555. (see page: 44)

[99] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram. The total variation on hypergraphs-learning on hypergraphs revisited. *Advances in Neural Information Processing Systems*, 26, 2013. (see pages: 79, 126)

[100] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. (see page: 54)

[101] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021. (see page: 54)

[102] J. Hoppe and M. T. Schaub. Representing edge flows on graphs via sparse cell complexes. In *Learning on Graphs Conference*, pages 1–1. PMLR, 2024. (see page: 93)

[103] J. Howard and S. Gugger. Imagenette: A subset of imagenet for benchmarking machine learning, 2020. URL https://github.com/fastai/imagenette. (see page: 58)

[104] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, pages 22118–22133, 2020. (see page: 85)

[105] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec. OGB-LSC: a large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021. (see page: 85)

[106] J. Huang and J. Yang. Unignn: a unified framework for graph and hypergraph neural networks. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth*

International Joint Conference on Artificial Intelligence, IJCAI-21, pages 2563–2569. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/353. URL https://doi.org/10.24963/ijcai.2021/353. Main Track. (see pages: 64, 68, 79, 80, 126, 130, 138, and 148)

[107] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman. ZINC: a free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012. (see pages: 90, 146)

[108] S. Islam, H. Elmekki, A. Elsebai, J. Bentahar, N. Drawel, G. Rjoub, and W. Pedrycz. A comprehensive survey on applications of transformers for deep learning tasks. *Expert Systems with Applications*, page 122666, 2023. (see page: 54)

[109] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. (see pages: 20, 22)

[110] K. Jha, S. Saha, and H. Singh. Prediction of protein–protein interaction using graph neural networks. *Scientific Reports*, 12(1):1–12, 2022. (see page: 2)

[111] K. Jha, S. Saha, and H. Singh. Prediction of protein–protein interaction using graph neural networks. *Scientific Reports*, 12(1):8360, May 2022. ISSN 2045-2322. doi: 10.1038/s41598-022-12201-9. (see page: 101)

[112] J. Jia and A. R. Benson. Residual correlation in graph neural network regression. In *ACM International Conference on Knowledge Discovery and Data Mining*, 2020. (see page: 89)

[113] J. Jia and A. R. Benson. Residual Correlation in Graph Neural Network Regression. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 588–598, New York, NY, USA, Aug. 2020. Association for Computing Machinery. ISBN 978-1-4503-7998-4. doi: 10.1145/3394486.3403101. (see page: 105)

[114] B. Jiang and Z. Zhang. Incomplete graph representation and learning via partial graph neural networks. *arXiv preprint arXiv:2003.10130*, 2020. (see pages: 19, 30)

[115] X. Kan, W. Dai, H. Cui, Z. Zhang, Y. Guo, and C. Yang. Brain network transformer. *Advances in Neural Information Processing Systems*, 35:25586–25599, 2022. (see page: 3)

[116] A. Kazi, S. krishna, S. Shekarforoush, K. Kortuem, S. Albarqouni, and N. Navab. Self-attention equipped graph convolutions for disease prediction. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 1896–1899, 2019. doi: 10.1109/ISBI.2019.8759274. (see page: 3)

[117] A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2022. doi: 10.1109/TPAMI.2022.3170249. (see pages: 3, 33, 35, 36, 38, 39, 117, 118, 119, and 121)

[118] A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. (see pages: 23, 24, and 30)

[119] B.-H. Kim, J. C. Ye, and J.-J. Kim. Learning dynamic graph representation of brain connectome with spatio-temporal attention. *Advances in Neural Information Processing Systems*, 34:4314–4327, 2021. (see page: 3)

[120] J. Kim, T. D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong. Pure Transformers are Powerful Graph Learners, Oct. 2022. (see page: 99)

[121] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. https://arxiv.org/abs/1412.6980v9, Dec. 2014. (see page: 105)

[122] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. (see pages: 23, 30, 114, 123, and 148)

[123] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017. (see pages: 39, 50)

[124] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. (see pages: 128, 129)

[125] D. Knoke and S. Yang. *Social Network Analysis*. SAGE Publications, Inc., 2008. ISBN 978-1-4129-8586-4. doi: 10.4135/9781412985864. (see page: 101)

[126] D. Knoke and S. Yang. *Social network analysis*. SAGE Publications, 2019. (see page: 2)

[127] W. Kool, H. Van Hoof, and M. Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR, 2019. (see pages: 23, 30)

[128] D. M. Kreindler and C. J. Lumsden. The effects of the irregular sample and missing data in time series analysis. *Nonlinear dynamics, psychology, and life sciences*, 2006. (see page: 19)

[129] V. La Gatta, V. Moscato, M. Pennone, M. Postiglione, and G. Sperlí. Music recommendation via hypergraph embedding. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. (see page: 63)

[130] K. Lakshminarayan, S. A. Harp, R. P. Goldman, T. Samad, et al. Imputation of missing data using machine learning techniques. In *KDD*, volume 96, 1996. (see pages: 19, 29)

[131] N. W. Landry, M. Lucas, I. Iacopini, G. Petri, A. Schwarze, A. Patania, and L. Torres. XGI: a Python package for higher-order interaction networks. *Journal of Open Source Software*, 8(85):5162, 2023. (see page: 85)

[132] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989. (see page: 54)

[133] J. Lee. *Introduction to topological manifolds*, volume 202. Springer Science & Business Media, 2010. (see page: 34)

[134] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019. (see pages: 80, 127, and 135)

[135] J. Li, C. Hua, J. Park, H. Ma, V. Dax, and M. J. Kochenderfer. Evolvehypergraph: Group-aware dynamic relational reasoning for trajectory prediction. *arXiv preprint arXiv:2208.05470*, 2022. (see page: 63)

[136] K. Li and J. Malik. Implicit maximum likelihood estimation, 2018. (see page: 42)

[137] P. Li and O. Milenkovic. Inhomogeneous hypergraph clustering with applications. *Advances in neural information processing systems*, 30, 2017. (see pages: 79, 126)

[138] Q. Li, Z. Han, and X.-M. Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, Jan. 2018. (see page: 99)

[139] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015. (see pages: 1, 14)

[140] M. Lisa and H. Bot. My Research Software, 12 2017. URL https://github.com/github-linguist/linguist. (see page: 146)

[141] R. J. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019. (see page: 25)

[142] X. T. Liu, J. Firoz, A. Lumsdaine, C. Joslyn, S. Aksoy, B. Praggastis, and A. H. Gebremedhin. Parallel algorithms for efficient computation of high-order line graphs of hypergraphs. In *International Conference on High Performance Computing, Data, and Analytics*, 2021. (see page: 85)

[143] A. Mackinnon. The use and reporting of multiple imputation in medical research–a review. *Journal of internal medicine*, 268(6):586–593, 2010. (see page: 19)

[144] M. M. Mayfield and D. B. Stouffer. Higher-order interactions capture unexplained complexity in diverse communities. *Nature ecology & evolution*, 1(3): 0062, 2017. (see page: 1)

[145] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval*, 3(2):127–163, July 2000. ISSN 1573-7659. doi: 10.1023/A:1009953814988. (see page: 105)

[146] L. Meng, H. Li, B.-C. Chen, S. Lan, Z. Wu, Y.-G. Jiang, and S.-N. Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318, 2022. (see page: 54)

[147] X. Miao, Y. Wu, L. Chen, Y. Gao, and J. Yin. An experimental survey of missing data imputation algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 2022. (see page: 20)

[148] M. Montagna, S. Scardapane, and L. Telyatnikov. Topological deep learning with state-space models: A mamba approach for simplicial complexes. *arXiv preprint arXiv:2409.12033*, 2024. (see pages: 7, 98)

[149] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. TUDataset: a collection of benchmark datasets for learning with graphs. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020. (see page: 90)

[150] K. A. Murgas, E. Saucan, and R. Sandhu. Hypergraph geometry reflects higher-order dynamics in protein interaction networks. *Scientific Reports*, 12 (1):20879, 2022. (see page: 1)

[151] B. Muzellec, J. Josse, C. Boyer, and M. Cuturi. Missing data imputation using optimal transport. In *International Conference on Machine Learning*, pages 7130–7140. PMLR, 2020. (see page: 25)

[152] S. K. Narang, A. Gadde, and A. Ortega. Signal processing techniques for interpolation in graph structured data. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5445–5449. IEEE, 2013. (see page: 19)

[153] A. Nazabal, P. M. Olmos, Z. Ghahramani, and I. Valera. Handling incomplete heterogeneous data using vaes. *Pattern Recognition*, 107:107501, 2020. (see pages: 19, 29)

[154] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010. (see pages: 45, 50)

[155] E. Pan and Z. Kang. Multi-view contrastive graph clustering. *Advances in neural information processing systems*, 34:2148–2159, 2021. (see page: 3)

[156] T. Papamarkou, T. Birdal, M. Bronstein, G. Carlsson, J. Curry, Y. Gao, M. Hajij, R. Kwitt, P. Liò, P. Di Lorenzo, V. Maroulas, N. Miolane, F. Nasrin, K. N. Ramamurthy, B. Rieck, S. Scardapane, M. T. Schaub, P. Veličković, B. Wang, Y. Wang, G.-W. Wei, and G. Zamzmi. Position Paper: Challenges and Opportunities in Topological Deep Learning. https://arxiv.org/abs/2402.08871v1, Feb. 2024. (see pages: 107, 109)

[157] T. Papamarkou, T. Birdal, M. Bronstein, G. Carlsson, J. Curry, Y. Gao, M. Hajij, R. Kwitt, P. Liò, P. Di Lorenzo, et al. Position: topological deep learning

is the new frontier for relational learning. *arXiv preprint arXiv:2402.08871*, 2024. (see pages: 4, 84, 93, and 96)

[158] M. Papillon, M. Hajij, A. Myers, F. Frantzen, G. Zamzmi, H. Jenne, J. Mathe, J. Hoppe, M. Schaub, T. Papamarkou, et al. ICML 2023 topological deep learning challenge: design and results. In *Topological, Algebraic and Geometric Learning Workshop*, 2023. (see pages: 84, 97)

[159] M. Papillon, S. Sanborn, M. Hajij, and N. Miolane. Architectures of topological deep learning: A survey on topological neural networks, 2023. (see pages: x, 2, 4, 10, 51, 84, 85, and 93)

[160] M. Papillon, G. Bernárdez, C. Battiloro, and N. Miolane. Topotune: A framework for generalized combinatorial complex neural networks. *arXiv preprint arXiv:2410.06530*, 2024. (see page: 4)

[161] M. Papillon, S. Sanborn, M. Hajij, and N. Miolane. Architectures of Topological Deep Learning: A Survey of Message-Passing Topological Neural Networks, Feb. 2024. (see pages: 101, 107)

[162] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training Recurrent Neural Networks, Feb. 2013. (see page: 106)

[163] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020. (see pages: 79, 143)

[164] B. Peters, V. Niculae, and A. F. T. Martins. Sparse sequence-to-sequence models, 2019. (see page: 36)

[165] O. Platonov, D. Kuznedelev, M. Diskin, A. Babenko, and L. Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*, Sept. 2022. (see page: 105)

[166] O. Platonov, D. Kuznedelev, M. Diskin, A. Babenko, and L. Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: are we really making progress? *arXiv preprint arXiv:2302.11640*, 2023. (see page: 89)

[167] L. Qiao, L. Zhang, S. Chen, and D. Shen. Data-driven graph construction and graph learning: A review. *Neurocomputing*, 312:336–351, 2018. (see page: 3)

[168] M. Raboh, R. Herzig, G. Chechik, J. Berant, and A. Globerson. Learning latent scene-graph representations for referring relationships. *ArXiv e-prints*, 3, 2019. (see page: 3)

[169] K. N. Ramamurthy, A. Guzmán-Sáenz, and M. Hajij. Topo-MLP: a simplicial network without message passing. In *International Conference on Acoustics, Speech and Signal Processing*, 2023. (see page: 86)

[170] T. M. Roddenberry, N. Glaze, and S. Segarra. Principled simplicial neural networks for trajectory prediction. In *International Conference on Machine Learning*, 2021. (see pages: 2, 4)

[171] T. M. Roddenberry, M. T. Schaub, and M. Hajij. Signal processing on cell complexes. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8852–8856, 2022. doi: 10.1109/ICASSP43922.2022.9747233. (see page: 123)

[172] E. Rossi, H. Kenlay, M. I. Gorinova, B. P. Chamberlain, X. Dong, and M. Bronstein. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. *arXiv preprint arXiv:2111.12128*, 2021. (see pages: 19, 30)

[173] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015. (see page: 44)

[174] B. Rozemberczki, O. Kiss, and R. Sarkar. Karate Club: an API oriented open-source Python framework for unsupervised learning on graphs. In *ACM International Conference on Information and Knowledge Management*, 2020. (see page: 85)

[175] B. Rozemberczki, C. Allen, and R. Sarkar. Multi-Scale attributed node embedding. *Journal of Complex Networks*, 9(2), 05 2021. ISSN 2051-1329. doi: 10.1093/comnet/cnab014. cnab014. (see page: 39)

[176] D. B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976. (see page: 19)

[177] A. G. Salman, B. Kanigoro, and Y. Heryadi. Weather forecasting using deep learning techniques. In *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 281–285, Oct. 2015. doi: 10.1109/ICACSIS.2015.7415154. (see page: 106)

[178] S. Sanborn, J. Mathe, M. Papillon, D. Buracas, H. J. Lillemark, C. Shewmake, A. Bertics, X. Pennec, and N. Miolane. Beyond euclid: An illustrated guide to modern machine learning with geometric, topological, and algebraic structures. *arXiv preprint arXiv:2407.09468*, 2024. (see page: 2)

[179] S. Sardellitti, S. Barbarossa, and L. Testa. Topological signal processing over cell complexes. In *Asilomar Conference on Signals, Systems, and Computers*, 2021. (see pages: 33, 34, 35, and 123)

[180] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80, 2008. (see page: 1)

[181] M. T. Schaub, A. R. Benson, P. Horn, G. Lippner, and A. Jadbabaie. Random walks on simplicial complexes and the normalized hodge 1-laplacian. *SIAM Review*, 62(2):353–391, 2020. (see page: 92)

[182] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. (see page: 115)

[183] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29(3):93–93, Sept. 2008. ISSN 2371-9621. doi: 10.1609/aimag.v29i3.2157. (see page: 105)

[184] G. Serra and M. Niepert. Learning to explain graph neural networks, 2022. (see page: 42)

[185] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation, 2019. (see page: 39)

[186] H. Shi, M. Xu, and R. Li. Deep Learning for Household Load Forecasting—A Novel Pooling Deep RNN. *IEEE Transactions on Smart Grid*, 9(5):5271–5280, Sept. 2018. ISSN 1949-3053, 1949-3061. doi: 10.1109/TSG.2017.2686012. (see page: 106)

[187] M. Śmieja, Ł. Struski, J. Tabor, B. Zieliński, and P. Spurek. Processing of missing data by neural networks. *Advances in neural information processing systems*, 31, 2018. (see page: 19)

[188] X. Song, M. Mao, and X. Qian. Auto-metric graph neural network based on a meta-learning strategy for the diagnosis of alzheimer's disease. *IEEE Journal of Biomedical and Health Informatics*, 25(8):3141–3152, 2021. doi: 10.1109/JBHI.2021.3053568. (see page: 3)

[189] I. Spinelli, S. Scardapane, and A. Uncini. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks*, 129: 249–260, 2020. (see pages: 19, 21, 25, and 30)

[190] I. Spinelli, S. Scardapane, A. Hussain, and A. Uncini. Fairdrop: Biased edge dropout for enhancing fairness in graph representation learning. *IEEE Transactions on Artificial Intelligence*, 3(3):344–354, 2022. doi: 10.1109/TAI.2021.3133818. (see page: 39)

[191] I. Spinelli, R. Bianchini, and S. Scardapane. Drop edges and adapt: A fairness enforcing fine-tuning for graph neural networks. *Neural Networks*, 167:159–167, 2023. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2023.08.002. (see page: 39)

[192] D. J. Stekhoven and P. Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012. (see page: 25)

[193] J. A. Sterne, I. R. White, J. B. Carlin, M. Spratt, P. Royston, M. G. Kenward, A. M. Wood, and J. R. Carpenter. Multiple imputation for missing data in epidemiological and clinical research: potential and pitfalls. *Bmj*, 338, 2009. (see page: 19)

[194] E. C. Strinati and S. Barbarossa. 6G networks: Beyond Shannon towards semantic and goal-oriented communications. *Computer Networks*, 190:107930, 2021. (see page: 53)

[195] E. C. Strinati, P. Di Lorenzo, V. Sciancalepore, A. Aijaz, M. Kountouris, D. Gündüz, P. Popovski, M. Sana, P. A. Stavrou, B. Soret, et al. Goal-oriented and semantic communication in 6G AI-native networks: The 6G-GOALS approach. *arXiv preprint arXiv:2402.07573*, 2024. (see page: 53)

[196] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015. (see page: 138)

[197] X. Sun, H. Cheng, B. Liu, J. Li, H. Chen, G. Xu, and H. Yin. Self-supervised hypergraph representation learning for sociological analysis. *IEEE Transactions on Knowledge and Data Engineering*, 2023. (see pages: 142, 143)

[198] H. Taguchi, X. Liu, and T. Murata. Graph convolutional networks for graphs containing missing features. *Future Generation Computer Systems*, 117:155–168, 2021. (see page: 30)

[199] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long Range Arena: A Benchmark for Efficient Transformers, Nov. 2020. (see page: 99)

[200] L. Telyatnikov and S. Scardapane. Egg-gae: scalable graph neural networks for tabular data imputation. In F. Ruiz, J. Dy, and J.-W. van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 2661–2676. PMLR, 25–27 Apr 2023. (see pages: 7, 18)

[201] L. Telyatnikov, M. S. Bucarelli, G. Bernardez, O. Zaghen, S. Scardapane, and P. Lio. Hypergraph neural networks through the lens of message passing: a common perspective to homophily and architecture design. *arXiv preprint arXiv:2310.07684*, 2023. (see pages: 7, 62, and 83)

[202] L. Telyatnikov, G. Bernardez, M. Montagna, P. Vasylenko, G. Zamzmi, M. Hajij, M. T. Schaub, N. Miolane, S. Scardapane, and T. Papamarkou. TopoBenchmarkX: A Framework for Benchmarking Topological Deep Learning, June 2024. (see page: 7)

[203] D. Teney, J. Wang, and E. Abbasnejad. Selective mixup helps with distribution shifts, but not (only) because of mixup. *arXiv preprint arXiv:2305.16817*, 2023. (see page: 74)

[204] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34: 24261–24272, 2021. (see page: 69)

[205] J. Topping, F. D. Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022. (see pages: 3, 33)

[206] H. Touvron, M. Cord, and H. Jégou. Deit iii: Revenge of the vit. In *European Conf. on Computer Vision*, pages 516–533. Springer, 2022. (see page: 58)

[207] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 2001. (see page: 25)

[208] P. Tune, M. Roughan, H. Haddadi, and O. Bonaventure. Internet traffic matrices: A primer. *Recent Advances in Networking*, 1:1–56, 2013. (see page: 43)

[209] S. Van Buuren. *Flexible imputation of missing data*. CRC press, 2018. (see page: 19)

[210] S. Van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45:1–67, 2011. (see pages: 25, 29)

[211] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. (see pages: 131, 134)

[212] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need, Aug. 2023. (see page: 99)

[213] N. Veldt, A. R. Benson, and J. Kleinberg. Combinatorial characterizations and impossibilities for higher-order homophily. *Science Advances*, 9(1):eabq3200, 2023. (see pages: 64, 65, 79, 141, and 142)

[214] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. (see pages: 1, 14, 50, 128, 129, 133, and 148)

[215] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. accepted as poster. (see page: 39)

[216] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. (see page: 30)

[217] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008. (see page: 21)

[218] J. Wang, K. Ding, Z. Zhu, and J. Caverlee. Session-based recommendation with hypergraph attention networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 82–90. SIAM, 2021. (see page: 4)

[219] L. Wang, W. Wu, F. Zhou, Z. Yang, and Z. Qin. Adaptive resource allocation for semantic communication networks. *arXiv preprint arXiv:2312.01081*, 2023. (see page: 54)

[220] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. Deep Graph Library: a graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019. (see page: 85)

[221] P. Wang, S. Yang, Y. Liu, Z. Wang, and P. Li. Equivariant hypergraph diffusion neural operators. *arXiv preprint arXiv:2207.06680*, 2022. (see page: 148)

[222] P. Wang, S. Yang, Y. Liu, Z. Wang, and P. Li. Equivariant hypergraph diffusion neural operators. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=RiTjKoscnNd. (see pages: xii, 64, 68, 72, 73, 79, 80, 127, 132, 136, and 143)

[223] X. Wang, A. Li, Z. Jiang, and H. Feng. Missing value estimation for dna microarray gene expression data by support vector regression imputation and orthogonal coding scheme. *BMC bioinformatics*, 7(1):1–10, 2006. (see pages: 19, 29)

[224] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. Heterogeneous graph attention network. In *The world wide web conference*, pages 2022–2032, 2019. (see pages: 132, 138)

[225] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. (see pages: 3, 30, and 114)

[226] Y. Wang, R. Huang, S. Song, Z. Huang, and G. Huang. Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition. *Advances in Neural Information Processing Systems*, 34:11960–11973, 2021. (see page: 54)

[227] J. Wei, Y. Wang, M. Guo, P. Lv, X. Yang, and M. Xu. Dynamic hypergraph convolutional networks for skeleton-based action recognition. *arXiv preprint arXiv:2112.10570*, 2021. (see pages: 79, 126)

[228] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl. Sampling matters in deep embedding learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2840–2848, 2017. (see page: 115)

[229] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019. (see page: 114)

[230] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020. (see page: 54)

[231] H. Wu, A. Yip, J. Long, J. Zhang, and M. K. Ng. Simplicial Complex Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(1):561–575, Jan. 2024. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2023.3323624. (see pages: 105, 107)

[232] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022. (see page: 1)

[233] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. (see page: 138)

[234] B. Wójcik, A. Devoto, K. Pustelnik, P. Minervini, and S. Scardapane. Adaptive computation modules: Granular conditional computation for efficient inference. *ArXiv*, 2023. (see page: 54)

[235] H. Xie, Z. Qin, G. Y. Li, and B.-H. Juang. Deep learning enabled semantic communication systems. *IEEE Transactions on Signal Processing*, 69:2663–2675, 2021. (see page: 54)

[236] C. Xu, M. Li, Z. Ni, Y. Zhang, and S. Chen. Groupnet: Multiscale hypergraph neural networks for trajectory prediction with relational reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6498–6507, 2022. (see pages: 51, 63, and 93)

[237] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin. Understanding mobile traffic patterns of large scale cellular towers in urban environment. *IEEE/ACM transactions on networking*, 25(2):1147–1161, 2016. (see page: 44)

[238] J. Xu, Y. Yang, C. Wang, Z. Liu, J. Zhang, L. Chen, and J. Lu. Robust network enhancement from flawed networks. *IEEE Transactions on Knowledge and Data Engineering*, 34(7):3507–3520, 2022. doi: 10.1109/TKDE.2020.3025147. (see page: 3)

[239] J. Xu, T.-Y. Tung, B. Ai, W. Chen, Y. Sun, and D. D. Gündüz. Deep joint source-channel coding for semantic communications. *IEEE Communications Magazine*, 61(11):42–48, 2023. (see pages: 54, 56)

[240] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. (see page: 148)

[241] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How Powerful are Graph Neural Networks?, Feb. 2019. (see page: 102)

[242] O. Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra. (see page: 146)

[243] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *Advances in neural information processing systems*, 32, 2019. (see pages: 63, 67, 79, 80, 126, 127, 129, 130, 136, 138, and 139)

[244] N. Yadati, V. Nitin, M. Nimishakavi, P. Yadav, A. Louis, and P. Talukdar. Nhp: Neural hypergraph link prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 1705–1714, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3411870. URL https://doi.org/10.1145/3340531.3411870. (see pages: 63, 79, and 126)

[245] C. Yang, R. Wang, S. Yao, and T. Abdelzaher. Hypergraph learning with line expansion. *arXiv preprint arXiv:2005.04843*, 2020. (see pages: 79, 126, and 138)

[246] M. Yang and E. Isufi. Convolutional learning on simplicial complexes, 2023. (see page: 148)

[247] M. Yang and E. Isufi. Convolutional Learning on Simplicial Complexes, Jan. 2023. (see pages: 105, 108)

[248] M. Yang and H.-S. Kim. Deep joint source channel coding for wireless image transmission with adaptive rate control. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5193–5197. IEEE, 2022. (see page: 54)

[249] R. Yang, F. Sala, and P. Bogdan. Efficient representation learning for higher-order data with simplicial complexes. In *Learning on Graphs Conference*, 2022. (see page: 148)

[250] Y. Yang, S. Mandt, L. Theis, et al. An introduction to neural data compression. *Foundations and Trends® in Computer Graphics and Vision*, 15(2):113–200, 2023. (see page: 44)

[251] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 40–48, New York, New York, USA, 20–22 Jun 2016. PMLR. (see page: 39)

[252] J. Yi and J. Park. Hypergraph convolutional recurrent neural network. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3366–3376, 2020. (see pages: 79, 126)

[253] J. Yoon, C. Davtyan, and M. van der Schaar. Discovery and clinical decision support for personalized healthcare. *IEEE journal of biomedical and health informatics*, 21(4):1133–1145, 2016. (see page: 19)

[254] J. Yoon, J. Jordon, and M. Schaar. Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning*, pages 5689–5698. PMLR, 2018. (see pages: 19, 25, and 29)

[255] J. Yoon, W. R. Zame, A. Banerjee, M. Cadeiras, A. M. Alaa, and M. van der Schaar. Personalized survival predictions via trees of predictors: An application to cardiac transplantation. *PloS one*, 13(3):e0194985, 2018. (see page: 19)

[256] J. Yu, H. Yin, J. Li, Q. Wang, N. Q. V. Hung, and X. Zhang. Self-supervised multi-channel hypergraph convolutional network for social recommendation. In *Proceedings of the web conference 2021*, pages 413–424, 2021. (see page: 63)

[257] T. Yuan, W. Deng, J. Tang, Y. Tang, and B. Chen. Signal-to-noise ratio: A robust distance metric for deep metric learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4815–4824, 2019. (see page: 46)

[258] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim. Graph Transformer Networks, Feb. 2020. (see page: 99)

[259] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017. (see pages: 80, 127)

[260] J. Zhang, Y. Zhu, Q. Liu, M. Zhang, S. Wu, and L. Wang. Latent structure mining with contrastive modality fusion for multimedia recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(9):9154–9167, 2023. doi: 10.1109/TKDE.2022.3221949. (see page: 3)

[261] M. Zhang, Z. Cui, S. Jiang, and Y. Chen. Beyond link prediction: Predicting hyperlinks in adjacency space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. (see pages: 63, 79, and 126)

[262] D. Zhou, J. Huang, and B. Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19, 2006. (see pages: 79, 126)

[263] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. (see pages: 1, 44, and 79)

[264] Q. Zhou, R. Li, Z. Zhao, C. Peng, and H. Zhang. Semantic communication with adaptive universal transformer. *IEEE Wireless Communications Letters*, 11(3):453–457, 2021. (see page: 54)

[265] A. Zia, A. Khamis, J. Nichols, U. B. Tayab, Z. Hayder, V. Rolland, E. Stone, and L. Petersson. Topological deep learning: a review of an emerging paradigm. *Artificial Intelligence Review*, 57(4):77, 2024. (see page: 2)

[266] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems*, 32, 2019. (see page: 30)