Rapid and Precise Topological Comparison with Merge Tree Neural Networks

Yu Qin, Brittany Terese Fasy, Carola Wenk, and Brian Summa

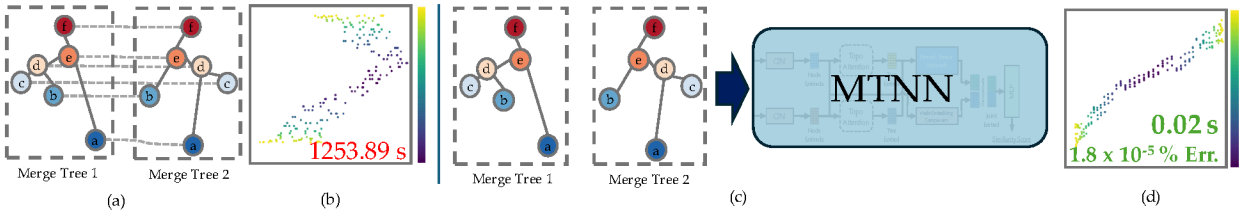

Fig. 1: (a) Merge tree distances, where trees are matched and edited, are computationally heavy and slow to compute. This leads to analyses that can be a bottleneck in scientific workflows. (b) Here, we show a Multidimensional scaling (MDS) visualization of a test dataset and the significant runtime necessary to produce a pair-wise distance matrix. (c) Rather than directly compute tree distances, we treat merge tree comparison as a learning task using our novel Merge Tree Neural Network (MTNN) to calculate the similarity. (d) MDS of the same data as (b) using MTNN. This reduces comparison times by orders of magnitude (5, here) with extremely low error added compared to the state-of-the-art merge tree distance.

Abstract—Merge trees are a valuable tool in the scientific visualization of scalar fields; however, current methods for merge tree comparisons are computationally expensive, primarily due to the exhaustive matching between tree nodes. To address this challenge, we introduce the Merge Tree Neural Network (MTNN), a learned neural network model designed for merge tree comparison. The MTNN enables rapid and high-quality similarity computation. We first demonstrate how to train graph neural networks, which emerged as effective encoders for graphs, in order to to produce embeddings of merge trees in vector spaces for efficient similarity comparison. Next, we formulate the novel MTNN model that further improves the similarity comparisons by integrating the tree and node embeddings with a new topological attention mechanism. We demonstrate the effectiveness of our model on real-world data in different domains and examine our model's generalizability across various datasets. Our experimental analysis demonstrates our approach's superiority in accuracy and efficiency. In particular, we speed up the prior state-of-the-art by more than $100\times$ on the benchmark datasets while maintaining an error rate below 0.1%.

Index Terms—Computational topology, merge trees, graph neural networks.

1 Introduction

A fundamental challenge in scientific visualization is analyzing and visualizing data, such as scalar fields, at speeds that support real-time exploration. Consider Topological Data Analysis (TDA) [17], which is now a critical component in many visualization pipelines due to its ability to extract and succinctly summarize structural information from complex datasets. TDA has been shown to effectively visualize [11, 23] and analyze a wide range of applications in chemistry [9, 21], neuroscience [31], social networks [2], material science [23], energy [41], and medical domains [28, 32, 51], to name a few. However, TDA approaches can be quite slow to compute due to their computational expense. Therefore, they are difficult to use in scenarios where a quick answer is needed for an analysis.

In particular, *Merge Trees* (MTs) [8, 17] serve as expressive topological descriptors that capture the complex structure of data, encoding the evolution of topological features with a history that records how their components merge. Despite being a powerful tool in various domains, the computational cost associated with distance calculations for merge trees has been a significant limitation. This is due to the core operation of computing the distance on merge trees needing to compare the optimal matching between the nodes of the merge trees. This matching is known to be NP-hard [1, 10]. Given the importance, yet great difficulty, of computing merge tree distances, a variety of metrics and pseudo-metrics have been proposed, such as functional distortion distance [8], edit distance [46], interleaving distance [20, 33], and distances based on branch decomposition and matching [42]. While many of these distance metrics are computationally challenging, it is worth noting that they are not universally NP-hard [8]. Some metrics may have polynomial-time algorithms under specific conditions. However, these methods usually require complicated design and implementation based on discrete optimizations. Despite not being exponential, the time complexity is usually still polynomial or sub-exponential in the number of nodes in the merge trees.

In this work, we take a different approach. To address the challenge of fast comparisons, we re-frame it as a *learning* task. Once the similarities of merge trees become learnable, we reduce the comparison runtime by avoiding the need to compute matchings between two MTs. To do so, we first need an encoder to map merge trees to a vector space, and learn this embedding model to position similar merge trees closely and dissimilar ones far apart in this space. We focus our investigation on graph neural networks (GNNs) as our primary encoder, a technique extensively studied for graph embeddings yet unexplored for merge trees.

In the following, we introduce the first merge tree neural network (MTNN) that uses GNNs to learn merge tree similarity. In particular, we design a neural network model that maps a pair of merge trees to a similarity score. At the training stage, the parameters involved in this model are learned by minimizing the difference between a predicted and a true similarity score, which is the interleaving distance on labeled

- *Yu Qin is with Tulane University. E-mail: yqin2@tulane.edu.*
- *Brittany Terese Fasy is with Montana State University. E-mail: brittany.fasy@montana.edu.*
- *Carola Wenk is with Tulane University. E-mail: cwenk@tulane.edu.*
- *Brian Summa is with Tulane University. E-mail: bsumma@tulane.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

merge trees [53] in this work. At the test stage, we obtain a predicted similarity score by feeding the learned model with any pair of unseen merge trees.

To effectively learn both the structural and topological information of merge trees, we need to design the neural network architecture carefully. Traditional GNNs only encode structural information of graphs, not merge trees. We introduce two key strategies to address this gap: firstly, we employ the graph isomorphism network (GIN) as our encoder, which excels in identifying structural distinctions in merge trees due to its design for graph isomorphism tasks. Secondly, we develop an innovative topological attention mechanism designed to identify and highlight important nodes within a merge tree. This mechanism re-weights nodes based on their ground truth MT distance and nodes' topological significance (persistence).

We conduct our experiments on five datasets: a synthetic point cloud, two datasets from time-varying flow simulations, one from a repeating pattern flow simulation, and one of 3D shapes. On all datasets, MTNN significantly enhances efficiency, accelerating the runtime by over 100 times while maintaining a low error rate when compared to a target MT similarity metric [53].

Our contributions can be summarized as follows:

- The first neural network model for merge tree similarity (MTNN);
- A novel topological-based attention mechanism for GNNs;
- An evaluation that shows MTNN's extremely precise($< 0.1\%$ mean squared error) and fast ($> 100\times$ speedup) similarity measures;
- An evaluation of the generalizability of trained models; and
- An open-source implementation for reproducibility.[1]

The paper is organized as follows: Section 2 discusses related work, focusing on the computation of distance between merge trees and the application of learning methods for graph similarity. Section 3 provides the necessary preliminary background. In Section 4, we examine the using GNNs for merge trees. Our proposed model, MTNN, is detailed in Section 5. Section 6 presents our results, and Section 7 concludes.

2 Related Work

Our approach is informed by two main areas of research: (1) the computation of distances between merge trees and (2) learning graph similarity using graph neural networks (GNNs). This section discusses the related works on merge tree distance and learning graph similarity using GNNs. For the definitions of merge trees, their distances, and GNNs, please refer to Section 3.

2.1 Merge Trees Distance

Many distances on merge trees exist: the interleaving distance [20, 33], the edit distance [7, 16], the functional distortion distance [8], and the universal distance [6]. While these distances are valuable for their stability and discriminatory capabilities, computing these distances often becomes impractical due to their NP-hard nature [1, 10].

Recent efforts aim to define more computationally feasible distances for merge trees. For instance, the edit distance on merge trees, as discussed in [46, 47], identifies the optimal edit operations between merge trees and has been experimentally shown to be more discriminative than both the bottleneck and one-Wasserstein [17] distances. In the [47], they introduced the local merge tree edit distance, specifically designed to analyze local similarities within merge trees. Despite the work in [49] demonstrating greater discriminative power than the previously proposed edit distances for merge trees, by employing a new set of improved edit operations, the computational cost is significantly higher, which hinders their practical application.

The interleaving distance requires an initial labeling of the merge trees, followed by the identification of the optimal matching between the labeled merge trees. This process, which establishes a computable metric known as the labeled interleaving distance, is introduced in [20]. Yan et al. [54] adapted this distance for practical use and incorporated geometric information in the labeling strategy in [53]. Similarly, Curry et al. [15] employed the Gromov–Wasserstein distance to label merge trees and compute the labeled interleaving distance.

Branch decomposition trees (BDTs) represent another method that first converts merge trees into BDTs (i.e., transferring edges of merge trees as nodes in a new tree), and then finds pairwise matching between these transformed trees [8, 36, 37, 42, 43, 50]. This process adds extra computational steps, further increasing complexity.

Applications utilizing merge trees distances, such as sketching [29] or encoding merge trees [36, 37], primarily operate in the original space, necessitating optimal matching between merge trees or their BDT variants. Machine learning has been used for merge trees in the work of Pont et al. [36], who applied neural networks to merge trees for compression and dimensionality reduction. Our work focuses on fast comparison. The previous work also uses a novel but classic auto-encoder with merge trees and BDTs as input, requiring a transformation step. In addition, their model is not designed to be generalizable across datasets, and their use of Wasserstein distance in training is computationally more intensive than our approach.

In summary, existing work either proposes a rigorous definition of distance with theoretical guarantees but with NP-hard computation or describes a similarity measure with practical applications that still require optimal matching between merge trees. No existing work focuses on mapping merge trees to vector space for efficient comparison. The closest approach is the work of Qin et al. [40], who map topological persistence diagrams to a hash for fast comparison. Merge trees are a more expressive and more complex topological abstraction than these diagrams. In our work, we utilize GNNs to map the merge trees to a vector space and re-frame merge tree comparison as a *learning* task.

2.2 Learning Graph Similarity and Dissimilarity

Graph (dis)similarity computation is a fundamental problem in graph theory. The graph edit distance (GED) [19] is a widely recognized metric between graphs, defining the minimum number of edit operations required to transform one graph into another. Despite its popularity, GED is known to be an NP-hard problem [56].

With the advancements in graph neural networks (GNNs), it is now possible to encode graphs into vector spaces effectively [24, 27, 52]. This capability allows GNNs to compute a similarity or dissimilarity score quickly. These methods employ an end-to-end framework to learn the graph representation that can, after training, map pairs of graphs to a similarity score.

A common approach in this area is the use of a Siamese neural network architecture [14], which processes each graph independently, but in parallel, to aggregate information. A feature fusion mechanism then captures the inter-graph similarities, and a multi-layer perceptron (MLP) is applied for regression analysis. This method is typically trained in a supervised manner using the mean squared error (MSE) loss against a ground truth similarity scores.

Many GNN-based approaches for learning graph similarity have great promise due to the competitive performance in both efficiency and efficacy [4, 5, 30, 39]. For example, the graph matching network (GMN) [30] is the first deep graph similarity model, which computes the similarity between two given graphs by a cross-graph attention mechanism. SimGNN [4] turned the graph similarity task into a regression task, and leveraged the graph convolutional network (GCN) [27] layers with self-attention-based mechanism on the model.

However, despite the growing popularity of GNN-based methods for graph similarity, their application to topological descriptors like merge trees has not been explored. Topological deep learning is a rapidly evolving field, although it primarily focuses on using topological features to enhance deep learning models [25, 34, 57]. Our perspective focuses on the inverse: designing deep learning models specifically for topological descriptors. Our research establishes the initial connection between GNNs and merge trees, potentially laying the groundwork for future developments in machine learning and TDA.

3 Preliminaries

In this section, we outline the foundational concepts of this work, beginning with merge trees induced by scalar fields and the common

[1] https://osf.io/2n8dy

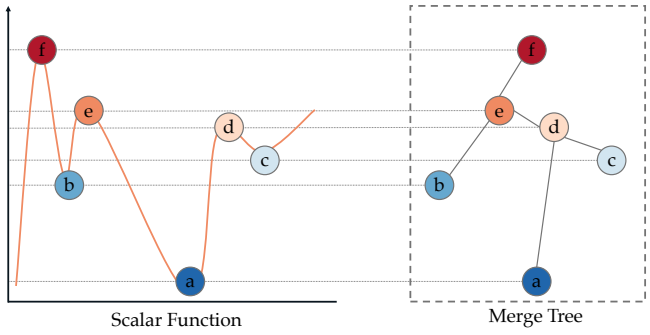

Fig. 2: An illustration of merge tree. On the left, a scalar function is represented by an orange line, highlighting the critical points, and showcasing how these points merge and connect topologically. On the right, the corresponding merge tree of sub-level set filtration.

distances used for topological comparisons. Then, we describe graph neural networks (GNNs), which serves as the core architecture for encoding merge trees for topological comparison.

3.1 Scalar Fields and Merge Trees

Consider a dataset represented as $(\mathbb{X}, f)$, where $\mathbb{X}$ denotes a topological space, and $f : \mathbb{X} \to \mathbb{R}$ denotes a scalar function, which is a continuous real-valued function. This function f assigns a real number to each point in $\mathbb{X}$, reflecting a characteristic of interest within the dataset. This is often referred to as a *scalar field*.

An equivalence relation $\sim_f$ is defined on $\mathbb{X}$, where $x \sim_f y$ if both x and y belong to the same connected component of the sub-level set $\mathbb{X}_a$ for a threshold value a in f. Consequently, two points are equivalent under $\sim_f$ if they exhibit a shared characteristic under the threshold a.

This leads to the definition of $\mathscr{T}_f^- := (\mathbb{X}_a, f)$ as the *join tree* for the dataset $(\mathbb{X}, f)$. The *join tree* encapsulates how dataset components merge as the threshold a varies. The *split tree* $\mathscr{T}_f^+ := (\mathbb{X}_a, f)$ is similarly constructed using super-level set to depict component splits with changing thresholds. Each of these two trees is called a *merge tree* $\mathscr{T}_f := (\mathbb{X}_a, f)$, illustrating the evolution of topological features in relation to f. In this work, we use join trees to capture the connectivity of sub-level sets, with the global maximum being the tree's root. Example merge tree for a 1D function is shown in Fig. 2.

Persistence Persistence is a quantity derived from persistent homology [17] that tracks the lifetime of a topological feature. This is often important in analysis. For instance, low persistence features are often deemed to be noise, while high persistence features are considered to encode important topological properties. In the context of our work, persistence is used to track the evolution of connected components of a sub-level set of a scalar function. A persistence pair (b,d) represents a topological feature that is born in the sub-level set $\mathbb{X}_b$ and dies going into the sub-level set $\mathbb{X}_d$. b and d correspond to the critical points m and s where $b = f(m)$ and $d = f(s)$. In our join trees, a feature is born at a minimum and dies when it merges with an older, in terms of f, feature. The *persistence* of such a pair is defined as the difference in function value at the two critical points, $d - b$.

3.2 Distance on Merge Trees and Graphs

Below, we first define graph edit distance, which GCNs primarily focus on reproducing. We then describe how edit distance on merge trees differs. Finally, we describe the distance we use as our ground truth, the interleaving distance.

Graph Edit Distance The graph edit distance (GED) [19] between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the minimum cost of transforming G_1 into G_2 using node and edge insertions, deletions, and substitutions. It's defined as:

$$D_e(G_1, G_2) := \min_{S \in O} \left\{ \sum_{op \in S} \delta(op) \right\},$$

where O represents the set of all valid sequences of graph edit operations that transform G_1 into G_2, S is a specific sequence of graph edit operations from O, and $\delta(op)$ is the cost function that assigns a non-negative real number to each edit operation op in the sequence S. This operation could be the insertion, deletion, or substitution of a node or an edge.

Each operation op has an associated cost, and $\sum_{op \in S} \delta(op)$, the total cost of a sequence S, is the sum of the costs of its individual operations. The GED is then determined by finding the sequence S with the minimum total cost that transforms G_1 into G_2.

Edit Distance on Merge Trees The edit distance between merge trees builds on tree edit distance, which is a specific case of GED where the cost operation is only on nodes. Given two merge trees, denoted as $\mathscr{T}_1$ and $\mathscr{T}_2$, it is defined as [46]:

$$D_e(\mathscr{T}_1, \mathscr{T}_2) := \min_{S \in O} \{\delta(S)\},$$

where O represents the set of valid tree edit operations, and S represents a sequence of these tree edit operations that transform $\mathscr{T}_1$ into $\mathscr{T}_2$. The cost function δ assigns a non-negative real number to each edit operation, which is defined as [46]:

$$\delta(m \to s) = \min \left\{ \begin{array}{c} \max(|b_m - b_s|, |d_m - d_s|), \\ \frac{(|d_m - b_m| + |d_s - b_s|)}{2} \end{array} \right\}$$
$$\delta(m \to \lambda) = \frac{|d_m - b_m|}{2}$$
$$\delta(\lambda \to s) = \frac{|d_s - b_s|}{2}$$

where λ denotes the empty set and m and s are nodes in $\mathscr{T}_1$ and $\mathscr{T}_2$, respectively. The first cost is for a node relabel operation from m to s. Next is deleting a node m. The last cost is adding a node s.

This is similar to GED, but the cost is formulated to account for topological persistence. Consider nodes $m \in \mathscr{T}_1$ and $s \in \mathscr{T}_2$, each node encodes a topological feature, (b_m, d_m) and (b_s, d_s). In particular, b_m is the function value where m is born. d_m is the function value where m dies. For sublevel sets, $b_m < d_m$. This distance computation is shown in Fig. 3.

Interleaving Distance on Merge Trees To compute the interleaving distance on merge trees, the trees need to be labeled first. A labeled merge tree, denoted as $T = (\mathscr{T}, \pi)$, including a merge tree $\mathscr{T}$ with a labeling $\pi : [a] \to V_{\mathscr{T}}$ where $[a]$ is the set of labels, $\{1, ..., a\}$, and $V_{\mathscr{T}}$ is the set of merge tree vertices [20]. π only needs to be surjective since a vertex can have multiple labels. The interleaving distance on labeled merge trees is calculated based on the induced matrix $T_M(\mathscr{T}, \pi)$. This matrix also can be referred to as the least common ancestor (LCA) matrix and defined as:

$$T_M(i, j) = f(LCA(\pi(i), \pi(j))),$$

where $f(LCA(\cdot))$ denotes the function value of LCA of a pair of vertices with labels i and j, $1 \le i, j \le a$. Given two labeled merge trees $T_1 = (\mathscr{T}_1, \pi_1)$ and $T_2 = (\mathscr{T}_2, \pi_2)$ that share the same set of labels, $[a]$, the interleaving distance between labeled merge trees is defined as:

$$D_i(T_1, T_2) = \|T_M^1 - T_M^2\|_\infty,$$

where $\|T_M\| = \max_{ij} |T_{Mij}|$ is the L_∞ norm. In [53], they proposed geometry-aware labeling strategies and, for brevity, we refer the reader to their paper for those details. But, at a high level, their labeling minimizes a cost function that accounts for the geometric structure of the tree along with the function value differences between nodes in the tree. In this way, topological persistence is encoded in the labeling strategy. The interleaving distance is used as the ground truth merge tree distance in our training.

3.3 Graph Neural Networks (GNNs)

Given a graph $G = (V, E)$ with nodes V and edges E, where each node $v \in V$ has an initial feature vector $h_v^{(0)}$, GNNs update the feature representation of each node by leveraging the structural context provided by the graph [24]. In particular, each node aggregates features from its neighbors (**Message Function**), updates its own features (**Update Function**), and finally produces an embedding (**Embedding**) that represents either the node's or the entire graph's comprehensive features. As Fig. 4 shows, the message function is responsible for aggregating information from a node's neighbors, which is then integrated into the node's feature vectors through the update function.

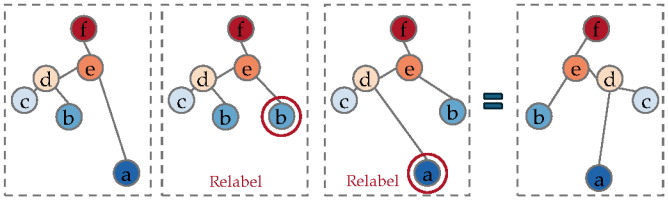

Fig. 3: Edit distance on merge trees example. The transformation from merge tree 1 to merge tree 2 needs two node relabel operations, where the relabeled node is highlighted in the red circle.

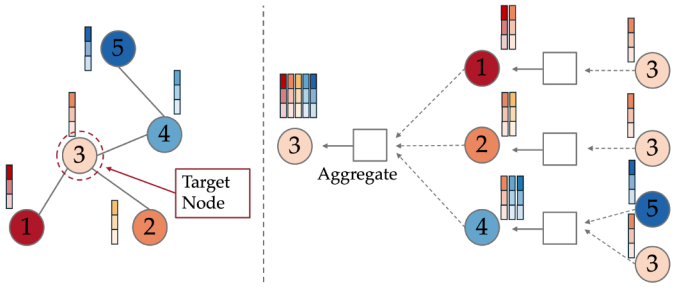

Fig. 4: Example of the GNNs process for a single node. Left: a graph with node features. Right: message function and update function for the target node 3. This node receives information from its directly connected neighbors, nodes 1, 2, and 4. Notably, node 4 further communicates a message from its adjacent node 5. In the end, the feature of node 3 is updated by aggregating all incoming messages.

Message Function: The message function aggregates features from neighboring nodes. For a node v, the message function M can be defined as:

$$m_v^{(l+1)} = \sum_{u \in \mathcal{N}(v)} M(h_v^{(l)}, h_u^{(l)}, e_{vu}),$$

where $h_v^{(l)}$ is the feature vector of node v at layer l, $\mathcal{N}(v)$ denotes the neighbors of v, and e_{vu} represents the edge features between nodes v and u.

Update Function: The update function U integrates the aggregated messages with the node's current state to compute its new state:

$$h_v^{(l+1)} = U(h_v^{(l)}, m_v^{(l+1)}),$$

where $h_v^{(l+1)}$ is the updated feature vector of node v at layer $l+1$.

Embedding: After processing through L layers of the GNN, the final embedding z_v for a node v is obtained, which can be used for downstream tasks like classification or clustering:

$$z_v = h_v^{(L)}.$$

For a whole graph embedding, an aggregation function (e.g., sum, mean, or max) is applied over all node embeddings to produce a single vector representing the entire graph. GNNs do not always generalize across different graph sizes, especially from small to large graphs, as noted in [55]. Therefore, the number of layers is determined by the graph's size and complexity. For large graphs, careful tuning and techniques like hierarchical pooling or attention mechanisms are essential for scalability. Through these mechanisms, GNNs offer a powerful framework for learning from graph-structured data, which we adapt to the context of merge trees for our topological comparison.

4 GNNs Meet Merge Trees

This section describes our methodology for adapting GNNs to merge trees. While GNNs have demonstrated effectiveness in learning on graph-structured data, the application to merge trees is not directly translatable due to unique topological characteristics inherent to merge trees. See Fig. 5. To address this, we enrich the node features with topological information. Specifically, we assign an attribute to each node derived from its scalar function value. Moreover, our objective goes beyond learning the structure of merge trees; we are focused on a more complicated task, learning merge tree distance. To this end, we employ GCN similarity [4], a model initially designed for graph similarity learning, adapting it to the context of merge trees. This adaptation allows us to establish a baseline for our approach. More details are given below.

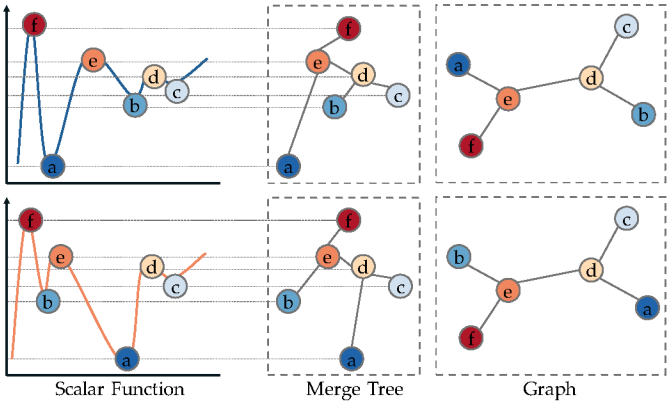

Fig. 5: An illustration of two merge trees and their graph structure. From left to right: critical points (dots) in two scalar functions, showcasing how these points merge and connect topologically. In the middle are corresponding merge trees, highlighting structural differences. Finally, the graph structure of the two merge trees is identical but differs in function value labels.

4.1 Graph Convolutional Network (GCN) Similarity

Following [4], the computation of the merge tree similarity has the following steps: (1) a GNN transforms the node of each tree into a vector, this is called node embedding; (2) a tree embedding is computed using attention-based aggregation; (3) a joint embedding is obtained by comparing node and tree-level embeddings to capture a comprehensive similarity between the merge trees. (4) Finally, the joint embedding is fed into a fully connected neural network to get the final similarity score. Below, we provide more details on these steps.

Initialization The inputs to our approach are merge trees, stored as adjacency matrices. The nodes of a merge tree are weighted with the normalized ($[0,1]$) function value. The ground truth distance is also normalized.

Node Embedding Computation For the first step, we compute a node embedding using a graph convolutional network (GCN) [27], a type of GNN that updates the node features by aggregating features from their neighbors. The formula for node level updates in a GCN can be expressed as:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_v \hat{d}_u}} h_u^{(l)} W^{(l)} + b^{(l)}\right),$$

where $h_v^{(l+1)}$ is the feature vector of node v at layer $l+1$; σ is a non-linear activation function, such as $\mathrm{ReLU}(x) = max(0,x)$; $\hat{d}_v$ is the degree of node v, plus 1 for self-loops; $W^{(l)}$ is the weight matrix at layer l; and $b^{(l)}$ is the bias at layer l. Given a pair of merge trees $\mathcal{T}_i$, $\mathcal{T}_j$, We obtain the node embedding h_{iv} and h_{jv} for each node of $\mathcal{T}_i$ and $\mathcal{T}_j$ via the GCN.

Tree Embedding For tree embeddings, we use attention-based aggregation. Attention-based aggregation is designed to assign weights to each node based on a given similarity metric. $h_n \in \mathbb{R}^m$ is the embedding of n-th node, where m is the size of embedding. The *global context* $c \in \mathbb{R}^m$ is obtained as

$$c = \tanh\left(\frac{1}{|V|} \sum_{n=1}^{|V|} h_n W_c\right), \quad (1)$$

where $|V|$ is the number of nodes and $W_c \in \mathbb{R}^{m \times m}$ is a learnable weight matrix. Next, each node should receive attention relative to this global context. The attention-weight embedding for a node is

$$h_n^* = \sum_{n=1}^{|V|} \mu(h_n^T c) h_n,$$

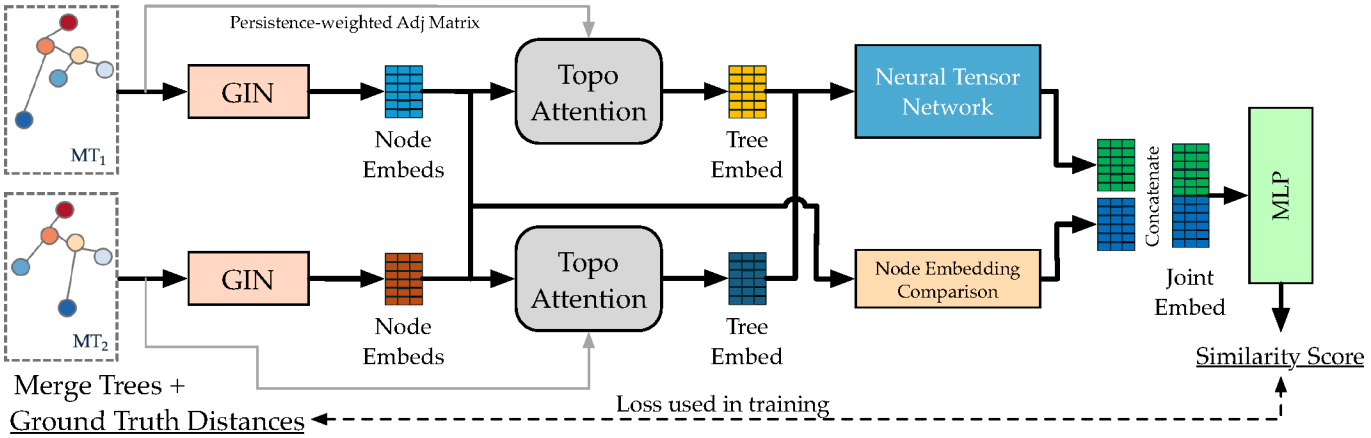

Fig. 6: The architecture of our merge tree neural network (MTNN) operating on a pair of merge trees ($MT_{1,2}$). First, adjacency matrices for each merge tree with node feature are weighted by function value and are fed into a GIN. This produces node embeddings for both. Next, our topological attention produces tree embeddings from the node embeddings. A persistence-weighted adjacency matrix is an additional input for this step. The node embeddings are then compared, and tree embeddings are fed into a neural tensor network. These outputs are concatenated to form a joint embedding. This joint embedding is fed into a multi-layered perceptron (MLP) to produce a similarity score. This score is compared to the ground truth distance (normalized) in the training loss function.

where $\mu(\cdot)$ is the sigmoid function. The tree-level embedding H^* is aggregated from all re-weighted node embeddings, h^*. $H^* = \sum_{n=1}^{|V|} h^*$, where $|V|$ is the number of nodes in merge tree.

Joint Embedding Computation For step three, the joint embedding is obtained by comparing both tree-level and node-level embeddings. We first describe how to compare the tree-level embedding using neural tensor networks (NTNs) following [44]:

$$D_{tree}(H_i^*, H_j^*) = \sigma(H_i^{*T} W_t^{[1:K]} H_j^* + V_w[H_i^* \cdot H_j^*] + b_t),$$

where D_{tree} is the tree-level vector that encodes similarity, $\sigma(\cdot)$ is an activation function, $W_t^{[1:K]} \in \mathbb{R}^{m \times m \times K}$ is a weight tensor, $[H_i^* \cdot H_j^*]$ is concatenation operation between H_i^*, H_j^*, $V_w \in \mathbb{R}^{K \times 2m}$ is a weight vector, and $b_t \in \mathbb{R}^K$ is a bias vector. Here, K is a hyperparameter for the size of D_{tree}. In summary, the NTN provides a measure of similarity between two tree embeddings.

To assess node-level similarities, we compare the node embeddings from two trees, $\mathcal{T}_i$ and $\mathcal{T}_j$. Letting N_i and N_j be the number of nodes in $\mathcal{T}_i$ and $\mathcal{T}_j$, respectively, this comparison yields an $N_i \times N_j$ similarity matrix. Specifically, the similarity matrix D_{node} is calculated as $D_{node} = \sigma(H_i H_j^T)$, where $H_i \in \mathbb{R}^{N_i \times m}$ and $H_j \in \mathbb{R}^{N_j \times m}$ are each the matrix of node embeddings for trees $\mathcal{T}_i$ and $\mathcal{T}_j$. $\sigma(\cdot)$ is an activation function that normalizes the scores into the same range, $(0,1)$. To address the discrepancy in node counts between trees, we zero-pad the trees with fewer nodes to match the node count of the trees with more nodes. Specifically, we append zero or null nodes to the end of the node list in the smaller trees. This approach ensures that the size difference between trees is accounted for in the similarity assessment without altering the original structure of the trees..

While zero-padding ensures that the node count between trees in each pair is the same, the overall size between different pairs can still differ. This is because the trees are padded only within each comparison pair to match the larger tree in that specific pair. However, the number of nodes in the larger tree can vary from one pair to another. Following the methodology in [4], we convert this matrix into a histogram $\text{hist}(D_{node}) \in \mathbb{R}^B$, where B is a predefined number of bins. This histogram representation standardizes the similarity matrix's size, facilitating comparison across tree pairs.

It is important to note that converting the similarity matrix D_{node} into a histogram standardizes the similarity matrix's size, facilitating comparison across tree pairs. However, this conversion can introduce distortions and artifacts because histograms are not continuous differential functions and do not support backpropagation.

To address this, we primarily rely on tree-level embedding comparisons to update model weights. The histogram-based node embedding comparisons are used to supplement the tree-level features, adding extra performance gains to our model. This approach ensures that the model benefits from the standardized representation of histograms while maintaining effective training through continuous tree-level embeddings.

Final In the final step, we combine the tree-level similarity $D_{tree}(H_i^*, H_j^*)$ with the histogram $\text{hist}(D_{node})$ as the joint embedding. A fully connected neural network then processes this embedding to produce the ultimate similarity score between the merge trees pair in the $[0,1]$ range. This approach serves as the baseline model in our study, with its performance detailed in Section 6.

4.2 Graph Isomorphism Network (GIN) Similarity

While our results with GCN similarity on merge tree comparison are encouraging, the variation in the number of nodes between two merge trees plays a significant role in distinguishing them. To emphasize the differences in node count, we further improved the model by replacing the standard GCN with a graph isomorphism network (GIN) for the node embedding computation. As presented in [52], GIN is adept at capturing distinct graph structures. For instance, it can reproduce the Weisfeiler–Lehman test for graph isomorphism.

The update rule for a GIN is:

$$h_v^{(l+1)} = \text{MLP}^{(l)}\left((1+\varepsilon^{(l)}) \cdot h_v^{(l)} + \sum_{u \in \mathcal{N}(v)} h_u^{(l)}\right),$$

where $h_v^{(l+1)}$ is node v's feature vector at layer $l+1$, $\text{MLP}^{(l)}$ denotes a multi-layer perceptron, and $\varepsilon^{(l)}$ is a tunable parameter that balances the node's own features with its neighbors'. The summation aggregates the features of node v's neighbors, enhancing the representation with local structural information.

By deploying a GIN, we enriched the model with a more complicated understanding of node differences across merge trees. In our work, we incorporated three GIN layers to optimize node feature learning and report the performance in Section 6. The result shows that this approach improves performance significantly over the GCN model.

However, it is crucial to note that GIN's capacity for graph isomorphism may not fully capture the subtleties of MT comparisons. For instance, two merge trees might be structurally the same yet distinct in a topological sense due to differences in the function values of their nodes. We extend GIN to a more customized network for merge trees to address this. This is our final approach, called a merge tree neural network (MTNN).

5 Merge Tree Neural Networks (MTNN) Similarity

As mentioned earlier, effectively capturing the topological characteristics of merge trees is crucial for learning their similarities using GNNs.

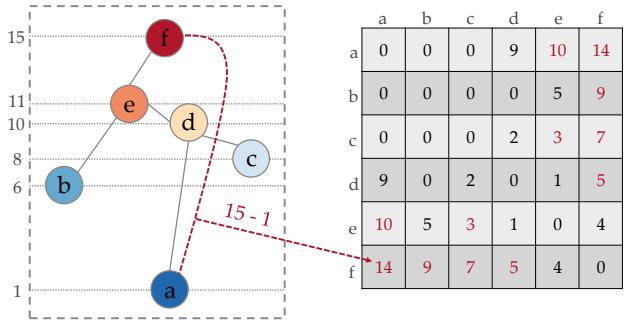

Fig. 7: An example of our persistence-weighted adjacency matrix. Each entry in the matrix represents the function value difference between connected nodes in the merge tree. Values highlighted in red denote entries for nodes that are not directly connected in the merge tree but are included to capture the persistence pairs. This enhancement allows for the inclusion of broader topological connections, ensuring the full topological characteristics of the merge tree are represented.

We have previously incorporated the function value for the nodes in a merge tree and applied a GIN to emphasize the node differences across merge trees. However, we have not yet addressed the inclusion of another important topological measure: the *persistence* of features in merge trees. To integrate persistence, we use a persistence-weighted adjacency matrix for tree embedding, which we detail in Section 5.1. Following that, in Section 5.2, we describe how to apply this weighted matrix in our model effectively, and we outline the learning process in Section 5.3. An overview of the model is provided in Fig. 6.

5.1 From Persistence to Edge Features

In the context of a merge tree, $\mathscr{T}$, each edge connecting nodes s and m represents the merging of critical points as the scalar value increases, denoted as $(f(s), f(m))$, which we utilize as an edge feature in our model. This approach, while useful, captures only incomplete topological information, as certain topological features span across multiple edges forming a path in the merge tree, For example, a persistence pair (b,d) represents the function values at the birth (b) and death (d) of a feature, where b is the function value at the local minimum and d is the function value at the saddle point where this feature merges with another.

To summarize the full topological characteristics of a merge tree, we introduce a persistence-weighted edge adjacency matrix $\hat{E}$. This matrix is constructed by considering the function value differences between connected nodes and their connections to neighboring nodes. Specifically, each entry in the matrix represents the function value difference between nodes, recording all paths. This enhancement allows us to extend our analysis from direct pairs like $(f(s), f(m))$ to broader connections, capturing $(f(s), f(\mathscr{D}(m)) \cup f(m))$, which includes node m and its descendants, $\mathscr{D}(m)$. Therefore, the complete topological information is encoded, as the persistence pair has been added. Fig. 7 illustrates how we accomplish this construction.

To integrate this persistence-weighted matrix, we tested two methods: (1) Incorporating it into GNN architectures like GIN or GCN; (2) Utilizing it in an attention-based aggregation to map node embeddings to tree embeddings.

Option (1) introduces modified edges (referred to as "pseudo" edges) to include *persistence* information. Consequently, this approach updates node features by utilizing neighbor features via the adjacency matrix. However, this modification changes the original merge tree structure in the message updates function of GNNs due to the introduction of pseudo edges.

To preserve the original merge tree structure while incorporating *persistence* information, we choose method (2). This approach leverages the persistence-weighted matrix in an attention-based aggregation process, mapping node embeddings to tree embeddings (Sec. 5.2). Here, node re-weighting is informed by their persistence, but not exclusively so. We train the weight matrix to consider both the persistence information and the overall merge tree distance (Sec. 5.3), ensuring a balanced integration of topological features.

5.2 Topological Attention

This section outlines the design of topological attention, leveraging the persistence-weighted adjacency matrix $\hat{E}$.

Building on the attention-based aggregation discussed in Section 4.1's Tree Embedding, we reformulate the *global context* vector c, replacing Eq.1. We integrate the topological information as:

$$Norm = \sum_{k=1}^{|V|} \sum_{l \in \mathscr{N}(k)} \hat{e}_{kl},$$

$$c = \tanh\left(\frac{1}{|V|} \sum_{n=1}^{|V|} \left(\frac{\sum_{u \in \mathscr{N}(n)} \hat{e}_{un}}{Norm} \right) h_n W_c \right),$$

where $u \in \mathscr{N}(n)$ is a neighbor node of n, $\hat{e}_{un} \in \hat{E}$ is the persistence-weighted edge feature of u and n, h_n is the embedding of n-th node, $|V|$ is the number of nodes, and W_c is a learnable weight matrix . To further break down the the formula above, we have the local weighting factor for each node v, the sum $\sum_{u \in \mathscr{N}(v)} \hat{e}_{uv}$ computes the total edge weight connected to v. By dividing this sum by the normalization term $Norm$, we normalize the local weighting with respect to the total edge weights in the tree, ensuring the scale of the features remains consistent. Then, we use the same calculation as the previous attention-based aggregation. Note that c is used to compute h^* which in turn is used to compute the term, H^*, used in training.

5.3 MTNN Learning

Training of the MTNN uses a Siamese network architecture, utilizing GINs as encoders to transform input merge trees into node embeddings. We generate joint embeddings by combining node and tree embeddings, where the tree embedding is derived using a topological attention-based aggregation. This aggregation reweights nodes according to their topological features. Subsequently, we employ an MLP-based regression network to map the joint embedding to the ground truth similarity score between the merge trees. The model is trained to minimize the Mean Squared Error (MSE) loss, defined as:

$$L = \frac{1}{D} \sum_{i,j \in D} (\text{MLP}(H^*_{ij}) - s_{ij})^2$$

Here, MLP denotes the MLP-based regression network, and D is the set of all training merge trees pairs, H^*_{ij} is the joint embedding and s_{ij} is the ground truth MT distance.

Table 1: Number of merge trees (MTs), the simplification threshold, τ, employed (same as used in previous work), and their range in node counts after simplification.

Dataset	# of MTs	τ	# of Nodes
MT2k	2000	0.1	[8,191]
Corner Flow	1500	0.2	[24,30]
Heated Flow	2000	0.06	[12,27]
Vortex Street	1000	0.05	[56,62]
TOSCA	400	0.01	[12,78]

6 Results

While our proposed approach can be generalized to different merge tree distances, we have chosen the state-of-the-art distance metric from [53] as our ground truth. This metric is noted for its efficient computation and is accompanied by an accessible open-source implementation. As mentioned, this distance is normalized to fall within the $[0,1]$ range to coincide with a similarity score. To compare the quality of our MTNN similarity in reproducing [53], we use Mean Squared Error (MSE) as our evaluation metric.

6.1 Datasets

We evaluate MTNN on five datasets: MT2k, Corner Flow, Heated Flow, Vortex Street, and TOSCA. All datasets, except MT2k, were chosen because they have previously been used in merge tree distance research [46, 53]. Interestingly enough, three [53] are from a similar

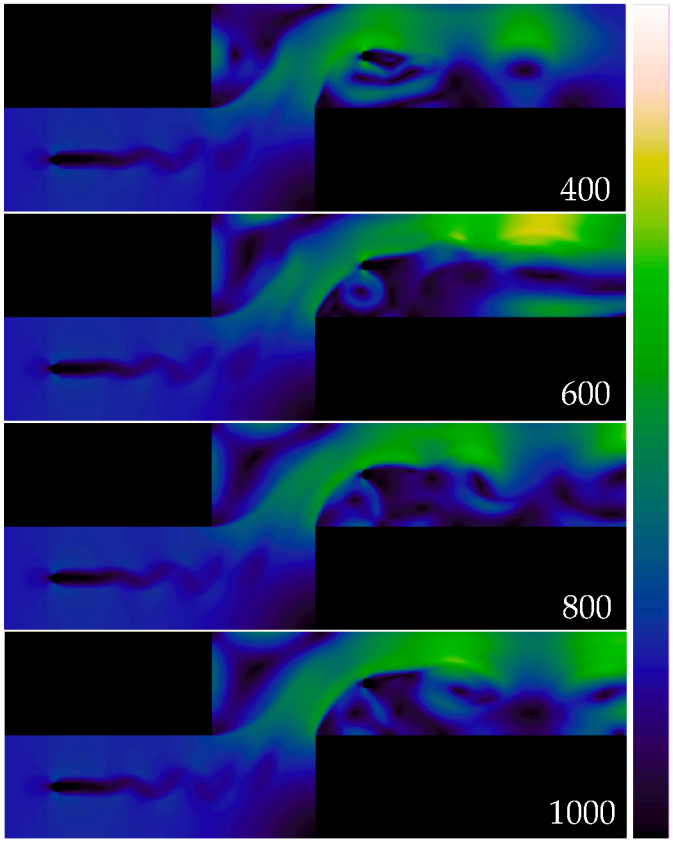

Fig. 8: Time steps of the 2D viscous Corner Flow simulation dataset.

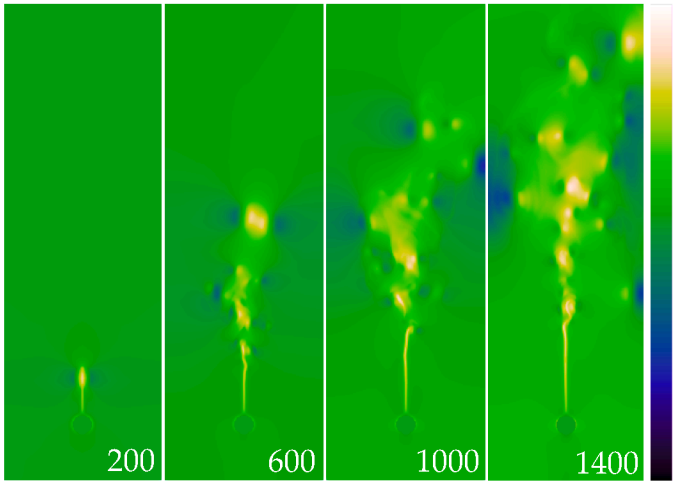

Fig. 9: Time steps of the 2D Heated Flow simulation dataset.

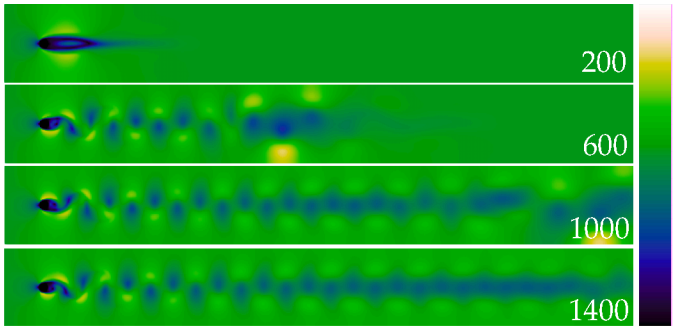

Fig. 10: Time steps of the 2D von-Kármán Vortex Street dataset.

domain: 2D flow simulations. This allows us to test the general applicability of a trained MTNN model across the same domain. Following the previous work, each we de-noise each merge tree for a pre-determined threshold τ as follows: given a node pair (s,m), where s is a local minimum and m is its emerging saddle point, the persistence is computed for each pair, $p = f(m) - f(s)$, where $f(s)$ and $f(m)$ are the function values at node s and m, respectively. If p is less than τ, then node s and its connecting edge are removed with the children of s being directly connected to m. Nodes are processed in reverse order of persistence. This results in the merge tree having fewer nodes and edges, with only the significant features remaining. More information on merge tree simplification can be found in [17,46,53]. The number of merge trees and range of node counts after simplification for each are summarized in Table 1.

As mentioned, all merge trees used are *join trees.*, although the approach could have also easily used *split trees*. The comparison uses a simplified tree based on the same parameters of previous work. Finally, we used the standard random split, 80% and 20%, of all the merge trees as training and testing sets for each dataset.

MT2k This dataset is 2000 synthetic 3D point clouds with two distinct classes. The first features three noisy tori, and the second class contains three noisy tori plus one noisy sphere. Each point cloud is constructed by synthetically sampling 100 points from the respective geometric shapes. Random noise is added during the sampling to ensure the uniqueness of each data point. Corresponding merge trees are constructed for each point cloud to represent their topological features. We apply a persistence threshold ($\tau = 0.1$) in the merge tree simplification process.

Corner Flow This dataset is 1500 time steps from a simulation of 2D viscous flow around two cylinders [3, 38]. Following [53], we generate a set of merge trees from the vertical component of the velocity vector fields. We also use the same persistence threshold ($\tau = 0.2$) for merge tree simplification as the previous work. See Fig. 8.

Heated Flow This dataset is from a simulation of the 2D flow created by a heated cylinder using a Boussinesq Approximation [22, 38]. Following [53], we convert each time instance of the flow into a scalar field using the magnitude of the velocity vector. The persistence simplification threshold used here is the same as the previous work ($\tau = 0.06$). See Fig. 9.

Vortex Street This is an ensemble of 2D regular grids [22, 38], each with a scalar function defined on the vertices to represent flow turbulence behind a wing, creating a 2D von-Kármán vortex street. Following [53], we use the velocity magnitude field to generate merge trees and apply a persistence simplification with the threshold ($\tau = 0.05$) to the merge trees. See Fig. 10.

TOSCA This dataset [12] contains a collection of different, non-rigid shapes of animals and humans. Following [46], we compute the average geodesic distance field on the surface mesh. Like the previous work, persistence simplification is used with the threshold ($\tau = 0.01$). See Fig. 11.

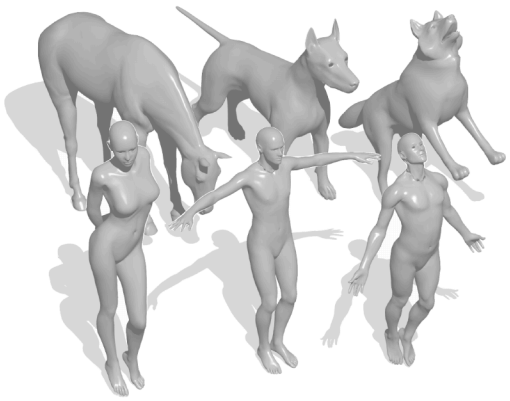
Fig. 11: Example 3D models from the TOSCA dataset.

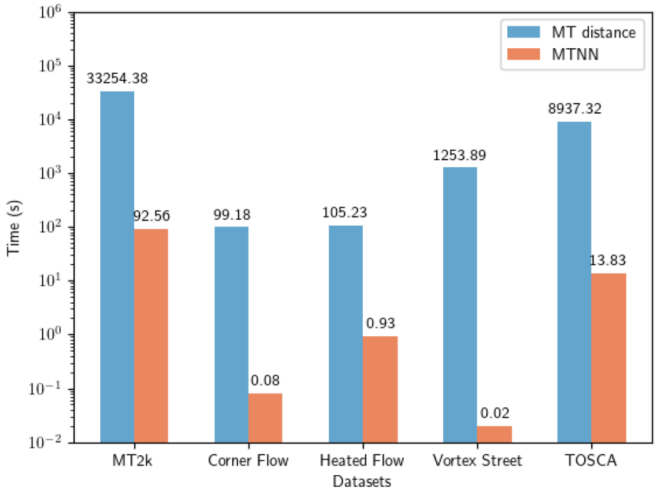

Fig. 12: Comparison of computational times for merge tree distance [53] and MTNN across different datasets. Times are presented in **log scale**. (Runtime for merge tree distance is computed with 16 cores)

6.2 Implementation Details

Merge trees are computed using TTK [48] and Paraview [45] is used for dataset visualization. Our implementation utilizes PyTorch [35]. We employ a three-layer GIN [52] as the encoder network with ReLU activation function with all weights initialized randomly. The output dimensions for the 1st, 2nd, and 3rd GIN layers are 64, 32, and 16, respectively. In the NTN layer, we specify $K = 16$. Following the approach in [4], we use 16 histogram bins for pairwise node embedding comparisons. For training, we choose the Adam optimizer [26], setting the learning rate to 0.001 and the weight decay to 0.0005. The model is trained with a batch size of 128 over 100 epochs. All datasets are converted into standard dataloaders compatible with PyTorch Geometric (PyG) [18] for GNN processing. To present each technique in the best light, we computed all timings for our ground truth distance [53] on a machine with a 16-Core (8P/8E) Intel I9-12900K CPU @ 5/4 GHz with 32 GB memory, as the approach benefits most from multiple cores. MTNN timings are computed on a machine equipped with a six-core Intel i7-6800K CPU @ 3.50GHz, 68GB of memory, and an Nvidia 3090Ti GPU, since it benefits most from a better GPU.

6.3 Evaluation Results

We evaluate the proposed approach from two perspectives: effectiveness and efficiency. For effectiveness, we compute the distance between the predicted similarity and our ground truth distance [53] using mean squared error (MSE), the standard for learned graph similarity validation. For efficiency, we record the wall time needed to compute the full pairwise similarity matrix for the testing set.

Effectiveness Table 2 provides quality results for our machine learning approaches. This includes the initial GCN, improved GIN, and our final MTNN model with topological attention. This study focused on models trained and tested on the same (split) dataset. For readability, we have scaled each by 10^3. Since both our ground truth and similarity output are in the $[0, 1]$ range, the error is $< 0.1\%$. Therefore the MSE between our approach and the ground truth is extremely low. The table also shows that the GIN model consistently outperforms our initial GCN model. Finally, our MTNN model in this scenario is comparable to or better than our GIN formulation. Our MTNN approach outperforms GIN in cases of higher GIN error, but has diminishing returns as the error lowers.

Table 3 explores the generalizability of our models. As before, all error results are scaled by 10^3 for readability. First, we tested the quality of our similarity measure in a scenario where the model is trained on merge trees from one data type but is applied to other types. For this, we used a model trained on the synthetic MT2k dataset. The table shows that the error is quite low even though we use merge trees from a 3D point cloud on very different data (i.e., 2D flows and 3D shapes). MT2k is also a fairly limited and constrained data source. The quality of these results shows that the MTNN model has the potential to be generally applied.

Next we tested how a model trained on a mix of *like* data performs. In this scenario, we trained on the combination of the Corner Flow, Heated Flow, and Vortex Street training sets, each sampled equally by 800. This model was then applied to our test sets. As is shown, the error is very low for our vector field datasets. This means that our model seems to generalize well across fields from this domain. In addition, it performs quite well on the 3D shape dataset but not as well as the MT2k-trained model. This makes intuitive sense since TOSCA and MT2k are both geometric 3D datasets. Therefore, with these results, we can postulate that this model can be generalized with low error. However, for the lowest error possible, it is best to train on data in the same domain but not necessarily from the same source (e.g., simulation).

Efficiency As stated in the beginning, this work aims to create an approach that can compute merge tree similarity, not only faster than the state-of-the-art, but also at speeds that could potentially support real-time analysis. Fig. 12 provides our runtimes compared to the fast merge tree distance calculation of [53]. Also, as detailed in the introduction to this section, we chose machines from our available hardware on which each technique performs best (i.e., more cores vs. a better GPU). Timings are based on the total time to compute all pairwise distances for each test dataset. Note that the times are plotted in log scale. As this figure shows, not only are we significantly faster, we are orders of magnitude faster ([2-5], 3 median). In addition, the time to compute similarity for a single pair of merge trees is so small that it can be considered negligible. Therefore MTNNs can potentially support interactive queries.

Table 2: MSE error for test datasets for GCN, GIN, and our MTNN networks. Each dataset was trained on a standard split of the **source data**. All results are scaled by 10^3 for readability, as our errors are extremely low. This means that all trained networks reproduce the similarity very close to the ground truth. Our MTNN network improves on the quality of the reproduction, but not in cases where GIN error is already extremely low.

Dataset	GCN	GIN	MTNN
MT2k	0.53819	0.18996	**0.10408**
Corner Flow	65.23393	**0.00145**	0.00312
Heated Flow	36.36125	0.00468	**0.00263**
Vortex Street	470.82582	**0.00008**	0.00018
TOSCA	12.60986	0.01542	**0.00986**

Table 3: MSE error for our datasets on models trained on other data. On the left is the error of applying the MT2k-trained model. On the right is a new model trained with a mix of Corner Flow, Heated Flow, and Vortex Street. Our topological attention are most effective in this scenario, leading to the lowest error values for each model (bold). Our synthetic MT2k performs best on the 3D shape dataset, while the model trained on the 3 separate flow datasets is best for the flow data (underlined). All results are scaled by 10^3 for readability, so all models return a low error with most $< 0.1\%$.

	Trained on MT2k			Trained on CF + HF + VS		
	GCN	GIN	MTNN	GCN	GIN	MTNN
Corner Flow	83.93713	0.03507	**0.01795**	17.20086	1.60534	**0.00862**
Heated Flow	57.93813	0.16013	**0.00501**	32.83913	0.23754	**0.00573**
Vortex Street	541.03814	113.24254	**3.78652**	178.93201	63.92713	**0.17532**
TOSCA	27.9381	3.17674	**0.2016**	37.8729	8.03914	**1.63408**

Table 4: Training times for different datasets (hours).

Dataset	MT2k	Corner Flow	Heated Flow	Vortex Street	TOSCA
Training Time (h)	4.28	2.86	3.83	1.93	3.95

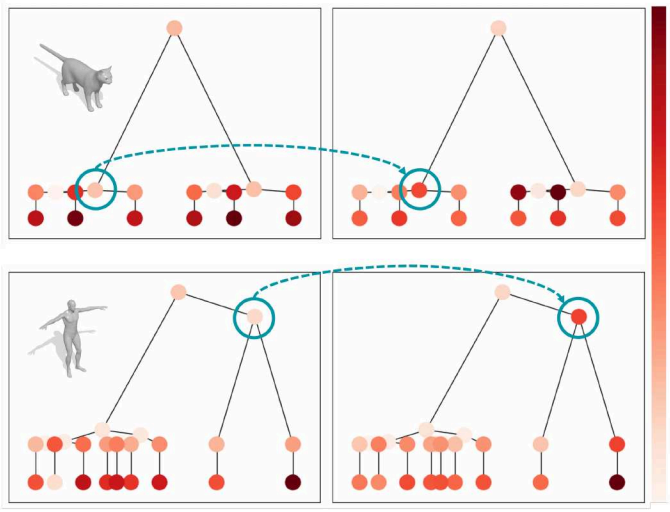

Fig. 13: Topological attention on two merge trees from the TOSCA dataset. Each row displays the merge tree visualized by the relative importance, with example inserts within the merge trees. Left: node importance without topological attention, using the GIN model. The node colors indicate their relative importance when comparing the two examples (the darker with red, the higher the weight). Right: node importance with topological attention using the MTNN model. The GIN model already emphasizes structural differences between the two merge trees. But, the introduction of topological attention further emphasizes these differences, re-weighing the nodes in the high persistence feature highlighted in cyan.

Table 4 shows the training times for each dataset using our approach. This demonstrates the efficiency of our method, which is comparable to other GNN methods. Note that these timings are insignificant when compared to the time needed to compute the ground truth distances for training. Additionally, our model is potentially generalizable across different datasets, as shown in Table 3, possibly mitigating training costs.

6.4 Further Analysis and Visualization

To further assess our method, we visualize the effect our topological attention mechanism has on merge trees. Fig. 13 gives an example pair from TOSCA. Attention weights on merge tree nodes are displayed in red, with the intensity indicating the weight magnitude: the darker the red, the higher the weight. This figure shows node importance with (left) and without (right) our topological attention mechanism.

As highlighted in cyan, the high persistence feature in the human model increases in weight due to our attention approach, which makes intuitive sense. It is interesting to see that the saddle point associated with the global maximum obtains more weight after applying topological attention in both the human and cat models. For the cat model, the left saddle point associated with the global maximum becomes more important, and the overall leaf node becomes less important. In the human model, the right saddle point gains more importance, and only the leaf node of the left subtree becomes less significant. These observations indicate that topological attention is highly data-dependent. This dependency demonstrates the usefulness of topological attention, as it adapts to the unique characteristics of each dataset, highlighting critical features that might otherwise be overlooked.

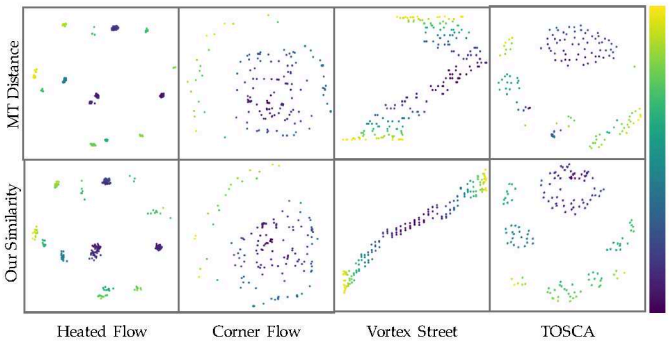

Fig. 14: Results from multidimensional scaling (MDS) analysis. Two MDS maps represent each dataset. On the top are the maps created using a similarity matrix based on the ground truth merge tree (MT) distance [53]. On the bottom are maps generated from a similarity matrix of MTNN. The coloring indicates the mean distance, showing how close or far points are from each other in the MDS space. Comparing the two maps shows similar patterns in colors and point arrangements. This similarity indicates that MTNN effectively learns and reflects the relationships between points.

The extremely low error, as detailed in Table 2, demonstrates that our method accurately learns pairwise similarities, motivating us to explore how well it reproduces the entire similarity matrix. We applied multidimensional scaling (MDS) [13] to the similarity matrix constructed from our ground truth merge tree distance [53] and the matrix formed from MTNN. MDS maps these matrices into 2D space, which we can visualize side-by-side. We assigned colors to each point in the MDS plots based on their average distance to other points. If MTNN captures the similarity of our ground truth, we would expect to see similar color patterns and point arrangements. While there are some differences in the visualization, on the whole, as demonstrated in Fig. 14, the structures and patterns are indeed similar.

7 Conclusion

In this study, we addressed the challenge of topological comparison, focusing specifically on merge trees. By reconceptualizing this problem as a *learning* task, we introduced merge tree neural network (MTNN) that employ graph neural networks (GNNs) for comparisons. Our approach is both fast and precise leading to comparisons that are orders of magnitude faster and extremely low in added error when compared to the state-of-the-art.

MTNN is the first deep learning model specifically designed for merge trees comparison, combining modified GNNs with novel topological attention to jointly learn the distance metric and topological properties of merge trees. MTNN introduces several key technical advancements over standard GNNs like GCN: (1) MTNN addresses node count discrepancies in merge trees comparison by employing GIN for node embedding computation, emphasizing the differences in node counts; (2) MTNN enhances the representation of topological information by initializing node features with function values derived from the merge tree; (3) MTNN incorporates topological attention by explicitly integrating persistence information into the aggregation function, enabling the model to capture and utilize the intrinsic topological properties of merge trees more effectively.

While our method represents a significant advancement in merge tree comparisons, it also presents challenges, such as the need for data to train. However, our experiments demonstrate the model's potential for generalization, which can mitigate this issue in its practical application. In addition, our test datasets only needed training sets of size in the hundreds or low thousands, which is relatively low for accurately trained models.

Future directions for our work include applying our merge tree comparisons to various analytical tasks in scalar field analysis, such as fast topological clustering or error assessment in approximated scalar functions. A key focus of our future work will be exploring more applications and datasets that could benefit from the use of merge trees. Our work opens up a new direction at the intersection of machine learning and topological data analysis. We believe that our work will encourage further investigations and developments in this emerging field, driving advancements in both theoretical understanding and practical applications.

Acknowledgements

This work was supported by NIH R01GM143789, DOE ASCR DE-SC0022873, NSF CCF 2107434, and NSF CCF 2046730.

References

[1] P. K. Agarwal, K. Fox, A. Nath, A. Sidiropoulos, and Y. Wang. Computing the gromov-hausdorff distance for metric trees. *ACM Transactions on Algorithms (TALG)*, 14(2):1–20, 2018. doi: 10.1145/3185466 1, 2

[2] K. Almgren, M. Kim, and J. Lee. Extracting knowledge from the geometric shape of social network data using topological data analysis. *Entropy*, 19(7):360, 2017. doi: 10.3390/e19070360 1

[3] I. Baeza Rojo and T. Günther. Vector field topology of time-dependent flows in a steady reference frame. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Scientific Visualization)*, 2019. doi: 10.1109/tvcg.2019.2934375 7

[4] Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pp. 384–392, 2019. doi: 10.1145/3289600.3290967 2, 4, 5, 8

[5] Y. Bai, H. Ding, K. Gu, Y. Sun, and W. Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3219–3226, 2020. doi: 10.1609/aaai.v34i04.5720 2

[6] U. Bauer, X. Ge, and Y. Wang. Measuring distance between Reeb graphs. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, pp. 464–473, 2014. doi: 10.1145/2582112.2582169 2

[7] U. Bauer, C. Landi, and F. Mémoli. The Reeb graph edit distance is universal. *Foundations of Computational Mathematics*, pp. 1–24, 2021. 2

[8] K. Beketayev, D. Yeliussizov, D. Morozov, G. H. Weber, and B. Hamann. *Measuring the distance between merge trees*. Springer, 2014. doi: 10.1007/978-3-319-04099-8_10 1, 2

[9] H. Bhatia, A. G. Gyulassy, V. Lordi, J. E. Pask, V. Pascucci, and P.-T. Bremer. Topoms: Comprehensive topological exploration for molecular and condensed-matter systems. *Journal of Computational Chemistry*, 39(16):936–952, 2018. doi: 10.1002/jcc.25181 1

[10] B. Bollen, P. Tennakoon, and J. A. Levine. Computing a stable distance on merge trees. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1168–1177, 2022. doi: 10.1109/tvcg.2022.3209395 1, 2

[11] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2010. doi: 10.1109/tvcg.2010.253 1

[12] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. *Numerical Geometry of Non-Rigid Shapes*. Springer Science & Business Media, 2008. doi: 10.1007/978-0-387-73301-2_12 7

[13] J. D. Carroll and P. Arabie. Multidimensional scaling. *Measurement, Judgment and Decision Making*, pp. 179–250, 1998. doi: 10.1016/b978-012099975-0.50005-1 9

[14] D. Chicco. Siamese neural networks: An overview. *Artificial Neural Networks*, pp. 73–94, 2021. doi: 10.1007/978-1-0716-0826-5_3 2

[15] J. Curry, H. Hang, W. Mio, T. Needham, and O. B. Okutan. Decorated merge trees for persistent topology. *Journal of Applied and Computational Topology*, 6(3):371–428, 2022. doi: 10.1007/s41468-022-00089-3 2

[16] B. Di Fabio and C. Landi. The edit distance for Reeb graphs of surfaces. *Discrete & Computational Geometry*, 55:423–461, 2016. doi: 10.1007/s00454-016-9758-6 2

[17] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Soc., 2010. doi: 10.1090/mbk/069 1, 2, 3, 7

[18] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. arXiv preprint arXiv:1903.02428, 2019. 8

[19] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13:113–129, 2010. doi: 10.1007/s10044-008-0141-y 2, 3

[20] E. Gasparovic, E. Munch, S. Oudot, K. Turner, B. Wang, and Y. Wang. Intrinsic interleaving distance for merge trees. arXiv preprint arXiv:1908.00063, 2019. 1, 2, 3

[21] D. Günther, R. A. Boto, J. Contreras-Garcia, J.-P. Piquemal, and J. Tierny. Characterizing molecular interactions in chemical systems. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2476–2485, 2014. doi: 10.1109/tvcg.2014.2346403 1

[22] T. Günther, M. Gross, and H. Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 36(4):141:1–141:11, 2017. doi: 10.1145/3072959.3073684 7

[23] A. Gyulassy, P.-T. Bremer, R. Grout, H. Kolla, J. Chen, and V. Pascucci. Stability of dissipation elements: A case study in combustion. *Computer Graphics Forum*, 33(3):51–60, 2014. doi: 10.1111/cgf.12361 1

[24] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017. 2, 3

[25] F. Hensel, M. Moor, and B. Rieck. A survey of topological machine learning methods. *Frontiers in Artificial Intelligence*, 4:681108, 2021. doi: 10.3389/frai.2021.681108 2

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Third International Conference on Learning Representation*, 2015. 8

[27] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016. 2, 4

[28] P. Lawson, A. B. Sholl, J. Q. Brown, B. T. Fasy, and C. Wenk. Persistent homology for the quantitative evaluation of architectural features in prostate cancer histology. *Scientific Reports*, 9(1):1–15, 2019. doi: 10.1038/s41598-018-36798-y 1

[29] M. Li, S. Palande, L. Yan, and B. Wang. Sketching merge trees for scientific visualization. In *2023 Topological Data Analysis and Visualization (TopoInVis)*, pp. 61–71. IEEE, 2023. doi: 10.1109/topoinvis60193.2023.00013 2

[30] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning*, pp. 3835–3845. PMLR, 2019. 2

[31] D. Maljovec, B. Wang, P. Rosen, A. Alfonsi, G. Pastore, C. Rabiti, and V. Pascucci. Rethinking sensitivity analysis of nuclear simulations with topology. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, Apr. 2016. doi: 10.1109/pacificvis.2016.7465252 1

[32] Z. Meng, D. V. Anand, Y. Lu, J. Wu, and K. Xia. Weighted persistent homology for biomolecular data analysis. *Scientific Reports*, 10(1):1–15, 2020. doi: 10.1038/s41598-019-55660-3 1

[33] D. Morozov, K. Beketayev, and G. Weber. Interleaving distance between merge trees. *Discrete and Computational Geometry*, 49(22-45):52, 2013. 1, 2

[34] T. Papamarkou, T. Birdal, M. M. Bronstein, G. E. Carlsson, J. Curry, Y. Gao, M. Hajij, R. Kwitt, P. Lio, P. Di Lorenzo, et al. Position: Topological deep learning is the new frontier for relational learning. In *Forty-First International Conference on Machine Learning*, 2024. 2

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019. 8

[36] M. Pont and J. Tierny. Wasserstein auto-encoders of merge trees (and persistence diagrams). *IEEE Transactions on Visualization and Computer Graphics*, 2023. doi: 10.1109/tvcg.2023.3334755 2

[37] M. Pont, J. Vidal, and J. Tierny. Principal geodesic analysis of merge trees (and persistence diagrams). *IEEE Transactions on Visualization and Computer Graphics*, 29(2):1573–1589, 2022. doi: 10.1109/tvcg.2022.3215001 2

[38] S. Popinet. Free computational fluid dynamics. *ClusterWorld*, 2(6), 2004. 7

[39] C. Qin, H. Zhao, L. Wang, H. Wang, Y. Zhang, and Y. Fu. Slow learning and fast inference: Efficient graph similarity computation via knowledge distillation. *Advances in Neural Information Processing Systems*, 34:14110–14121, 2021. 2

[40] Y. Qin, B. T. Fasy, C. Wenk, and B. Summa. A domain-oblivious approach for learning concise representations of filtered topological spaces for clustering. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):302–312, 2021. doi: 10.1109/TVCG.2021.3114872 2

[41] Y. Qin, G. Johnson, and B. Summa. Topological guided detection of extreme wind phenomena: Implications for wind energy. In *2023 Workshop on Energy Data Visualization (EnergyVis)*, pp. 16–20. IEEE, 2023. doi: 10.1109/energyvis60781.2023.00010 1

[42] H. Saikia, H.-P. Seidel, and T. Weinkauf. Extended branch decomposition graphs: Structural comparison of scalar data. In *Computer Graphics Forum*, vol. 33, pp. 41–50. Wiley Online Library, 2014. doi: 10.1111/cgf.12360 1, 2

[43] H. Saikia and T. Weinkauf. Global feature tracking and similarity estimation in time-dependent scalar fields. In *Computer Graphics Forum*, vol. 36, pp. 1–11. Wiley Online Library, 2017. doi: 10.1111/cgf.13163 2

[44] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in Neural Information Processing Systems*, 26, 2013. 5

[45] A. H. Squillacote, J. Ahrens, C. Law, B. Geveci, K. Moreland, and B. King. *The Paraview Guide*, vol. 366. Kitware Clifton Park, NY, 2007. 8

[46] R. Sridharamurthy, T. B. Masood, A. Kamakshidasan, and V. Natarajan. Edit distance between merge trees. *IEEE Transactions on Visualization and Computer Graphics*, 26(3):1518–1531, 2018. doi: 10.1109/tvcg.2018.2873612 1, 2, 3, 6, 7

[47] R. Sridharamurthy and V. Natarajan. Comparative analysis of merge trees using local tree edit distance. *IEEE Transactions on Visualization and Computer Graphics*, 29(2):1518–1530, 2021. doi: 10.1109/tvcg.2021.3122176 2

[48] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The topology toolkit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2017. doi: 10.1109/TVCG.2017.2743938 8

[49] F. Wetzels, M. Anders, and C. Garth. Taming horizontal instability in merge trees: On the computation of a comprehensive deformation-based edit distance. In *2023 Topological Data Analysis and Visualization (TopoInVis)*, pp. 82–92. IEEE, 2023. doi: 10.1109/topoinvis60193.2023.00015 2

[50] F. Wetzels, H. Leitte, and C. Garth. Branch decomposition-independent edit distances for merge trees. In *Computer Graphics Forum*, vol. 41, pp. 367–378. Wiley Online Library, 2022. doi: 10.1111/cgf.14547 2

[51] K. Xia and G.-W. Wei. Persistent homology analysis of protein structure, flexibility, and folding. *International Journal for Numerical Methods in Biomedical Engineering*, 30(8):814–844, 2014. doi: 10.1002/cnm.2655 1

[52] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826, 2018. 2, 5, 8

[53] L. Yan, T. B. Masood, F. Rasheed, I. Hotz, and B. Wang. Geometry aware merge tree comparisons for time-varying data with interleaving distances. *IEEE Transactions on Visualization and Computer Graphics*, 2022. doi: 10.1109/tvcg.2022.3163349 2, 3, 6, 7, 8, 9

[54] L. Yan, Y. Wang, E. Munch, E. Gasparovic, and B. Wang. A structural average of labeled merge trees for uncertainty visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):832–842, 2019. doi: 10.1109/tvcg.2019.2934242 2

[55] G. Yehudai, E. Fetaya, E. Meirom, G. Chechik, and H. Maron. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*, pp. 11975–11986. PMLR, 2021. 4

[56] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009. doi: 10.14778/1687627.1687631 2

[57] 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. doi: 10.1007/s10462-024-10710-9 2