EXPRESSIVE AND INVARIANT GRAPH LEARNING VIA CANONICAL TREE COVER NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

While message-passing NNs (MPNNs) are naturally invariant on graphs, they are fundamentally limited in expressive power. Canonicalization offers a powerful alternative by mapping each graph to a unique, invariant representation on which expressive encoders can operate. However, existing approaches rely on a single canonical *sequence*, which flattens the structure, distorts graph distances, and restricts expressivity. To address these limitations, we introduce *Canonical Tree Cover Neural Networks* (CTNNs), which represent the graph with a canonical spanning tree cover, i.e., a small collection of canonical trees covering all edges. Each tree is then processed with an existing expressive tree encoder. Theoretically, tree covers better preserve graph distances than sequences, and on sparse graphs, the cover recovers all edges with a logarithmic number of trees in the graph size, making CTNNs strictly more expressive than sequence-based canonicalization pipelines. Empirically, CTNNs consistently outperform invariant GNNs, random samplers, and sequence canonicalizations across graph classification benchmarks. Overall, CTNNs advance graph learning by providing an efficient, invariant, and expressive representation learning framework via tree cover-based canonicalization.

1 INTRODUCTION

In graph representation learning, capturing a graph's natural symmetries (i.e., isomorphism invariance) is essential for learning and generalization. One way to enforce this invariance is to bake it directly into the architecture: message-passing neural networks (MPNNs) (Duvenaud et al., 2015; Gilmer et al., 2017; Kipf and Welling, 2017) achieve architectural invariance by iteratively aggregating neighbor embeddings, but are provably equivalent in expressive power to the 1-dimensional Weisfeiler–Leman test (Xu et al., 2019; Morris et al., 2019), suffer from oversmoothing (Li et al., 2018; Chen et al., 2020) and oversquashing (Oono and Suzuki, 2020; Di Giovanni et al., 2023), and are thus fundamentally limited. A second approach achieves invariance via sampling: random walk neural networks (RWNNs) (Wang and Cho, 2024; Tönshoff et al., 2023; Chen et al., 2025; Kim et al., 2025) sample walks and feed them into powerful sequence models, overcoming limitations in MPNN expressivity but incurring potentially prohibitive sampling costs on large datasets to achieve the invariance. Finally, canonicalization maps each graph to a unique representative, allowing any expressive, non-invariant model to operate on a fixed, invariant input, bypassing expensive sampling (Bloem-Reddy and Teh, 2020). In this work, we establish the limitations of existing canonicalization approaches on graphs and propose a new canonicalization.

Existing graph canonicalization approaches first assign labels to each node, flatten the graph into a single sequence, either via learned sorting layers (Niepert et al., 2016; Zhang et al., 2018; Grover et al., 2019) or through traversal as in canonical SMILES (Goh et al., 2017; Honda et al., 2019; Chithrananda et al., 2020), and then feed the sequence into a powerful downstream sequence model. In this work, we formally quantify how flattening into a sequence distorts graph distance. To illustrate this limitation, consider S_n, the n-node star (Figure 1, $n = 7$). Each leaf node in the graph has distance 1 to the center node, while leaf nodes in the sequence necessarily have distance $O(n)$ to the center node (stretch). Moreover, while leaves have distance 2 to each other

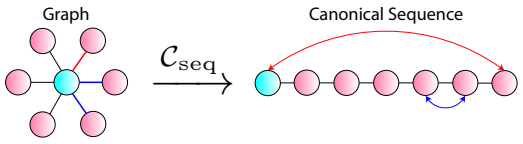


Figure 1: Canonical sequence representations introduce significant stretch and contraction.

in S_n, certain leaves have distance 1 in the sequence (contraction). Thus, the canonicalization can stretch and contract original distances, making structure harder to capture. We further establish that the reduction of the graph into a single representative limits the expressivity of the overall approach to that of its node labeler, discarding the benefits of using powerful downstream models.

To address these limitations, we propose *Canonical Tree Cover Neural Networks* (CTNNs), which construct a canonical spanning tree cover via minimum spanning tree extraction and coverage-aware edge label refinement. Each tree in the cover is processed by an existing expressive tree encoder (Tai et al., 2015) and aggregating over the cover yields an invariant representation. By leveraging tree representations and capturing structure across a set of canonical representatives, CTNNs better capture graph distances and are more expressive than sequence canonicalizations. Across a variety of graph classification tasks, CTNNs consistently outperform architecturally invariant GNNs, sampling methods, and existing canonicalization approaches. In summary, we make the following contributions:

- **Current Limitations of Canonicalizations.** We establish that sequence-based graph canonicalization methods fail to preserve graph distance and are limited in expressivity.
- **New Canonical Model: Canonical Tree Cover Neural Networks (CTNNs).** We introduce CTNNs, which construct a canonical tree cover. Each tree is then processed by existing expressive recurrent tree encoders and aggregated to obtain an invariant representation.
- **Theory: Invariance, Distance Preservation, and Expressivity Guarantees.** We prove that CTNNs produce invariant graph representations, preserve graph distance information, and exceed the expressivity of sequence-based canonicalizations and MPNNs. With universal tree encoders, CTNNs achieve universality on invariant graph functions.
- **Extensive Empirical Evaluation.** Across 8 graph classification benchmarks, CTNNs outperform architecturally invariant models, sampling approaches, and canonical baselines.

2 Background and Preliminaries

We first introduce notation and review canonical approaches on graphs, the primary family of models under investigation. These approaches typically produce a single sequence that is fed to a sequence model. We then formalize recurrent sequence models, which often outperform attention and convolution on graphs by better matching the traversal inductive bias. Despite their practical performance, however, recurrent sequence models can suffer from long graph-derived sequences. These limitations lead us to consider recurrent tree models that instead propagate information along trees, which we will later demonstrate better capture graph distance.

2.1 Notation on Graphs and Trees

Let $G = (V, E, \mathbf{X})$ be an undirected graph with $n = |V|$ nodes, $m = |E|$ edges, and node features $\mathbf{X} \in \mathbb{R}^{n \times d}$. For $v \in V$, let $\mathbf{x}_v$ denote the v-th row of $\mathbf{X}$, $\mathcal{N}(v) = \{u \in V : (u, v) \in E\}$ its neighborhood, and $\deg(v) = |\mathcal{N}(v)|$ and $d_G(u, v)$ the shortest path distance in G. A rooted tree is $T = (V, E, \mathbf{X}, r)$ with root $r \in V$. Each non-root node $v \neq r$ has a unique parent $p(v)$, and we write $C(v) = \{u \in V : p(u) = v\}$ for its children. Leaf nodes of the tree satisfy $C(v) = \varnothing$.

2.2 Message-passing Neural Networks and GNN Expressivity

Standard GNNs adopt a message-passing approach, where each layer iteratively updates a node's representation by aggregating the features of its neighbors (Gilmer et al., 2017). Formally, the initial message-passing layer can be defined as the following propagation rule at the node level for all $i \in V$,

$$f_{\text{MPNN}}(G)_i = f_{\text{agg}}(\{\mathbf{x}_j \mid j \in \hat{\mathcal{N}}(i)\}),$$

where f_{agg} is a permutation-invariant function. Because of this aggregation step, MPNNs incur fundamental expressivity limitations and cannot distinguish certain classes of non-isomorphic graphs (Xu et al., 2019). We compare the expressivity of GNNs by the pairs of graphs they can distinguish (Azizian and Lelarge, 2020), introducing the following notation. For two GNNs f_1 and f_2, we write

$$f_2 \preceq f_1 \quad \iff \quad \forall\, G, H :\ f_1(G) = f_1(H) \ \Rightarrow\ f_2(G) = f_2(H).$$

Thus, any pair indistinguishable by f_1 is also indistinguishable by f_2, so f_1 is at least as expressive as f_2. The relation is strict, $f_2 \prec f_1$, if $f_2 \preceq f_1$ and there exist graphs G, H with $f_1(G) \neq f_1(H)$

while $f_2(G) = f_2(H)$. f_1 and f_2 are equally expressive, written $f_1 \simeq f_2$, if $f_2 \preceq f_1$ and $f_1 \preceq f_2$. These relations coincide with notions of approximation power. For example, if $f_2 \prec f_1$, every target approximable by f_2 is approximable by f_1, and there exist targets approximable by f_1 but not f_2.

2.3 Canonical Approaches on Graphs

Graph canonicalization aims to obtain a unique isomorphism–invariant node labeling (McKay et al., 1981). Because computing an exact canonical labeling is as hard as the graph isomorphism problem, practical methods adopt soft approximations (e.g., GNN embeddings). After obtaining an approximate labeling, these pipelines typically flatten the graph into a single sequence either via sorting layers (Niepert et al., 2016; Zhang et al., 2018) or through traversal such as canonical SMILES (Goh et al., 2017; Honda et al., 2019), allowing expressive sequence models to process the sequence. Formally, let $\pi_V : V \to \mathbb{R}$ be a node labeling function (e.g., MPNN), $\mathcal{C}_{\text{seq}}$ be a single-sequence canonicalizer that maps the labeled graph (G, π_V) to a sequence depending only on π_V and carrying only the node features $\mathbf{X}$, and f_{seq} be a sequence model. A general sequence–based canonical model is defined as

$$f_{\text{CanSeq}}(G) = f_{\text{seq}}(\mathcal{C}_{\text{seq}}(G, \pi_V)).$$

As a concrete instance, if π_V is an MPNN, $\mathcal{C}_{\text{seq}}$ is a differentiable sorting layer, and f_{seq} is a 1D CNN, then f_{CanSeq} recovers Deep Graph Convolutional Neural Network (Zhang et al., 2018).

2.4 Recurrent Sequence and Tree Models

Recent RWNNs find that recurrence often outperforms attention and convolution by better matching the traversal inductive bias (Wang and Cho, 2024; Chen et al., 2025). Given inputs $(\mathbf{x}_t)_{t=1}^T$, initial state $\mathbf{h}_0$, and state transition map $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$, the recurrent update is defined

$$\mathbf{h}_t \;=\; \Phi\big(\mathbf{h}_{t-1}, \mathbf{x}_t\big), \qquad \text{for } t = 1, \ldots, T.$$

Recurrent models suffer on long sequences that exacerbate vanishing/exploding gradients, which motivates our use of recurrent tree models that shorten dependency paths and mitigate these instabilities. Recurrent tree models generalize sequence recurrence to rooted trees (Tai et al., 2015; Xiao et al., 2024), propagating information bottom–up from children to their parent. Given $T = (V, E, r)$ with L levels and node inputs $\{\mathbf{x}_v\}_{v \in V}$, recurrent tree models compute hidden states $\{\mathbf{h}_v\}_{v \in V}$ by applying a local transition to child states and aggregating with a permutation–invariant operator f_{agg}:

$$\mathbf{h}_v \;=\; f_{\text{agg}}(\{\Phi(\mathbf{h}_c, \mathbf{x}_v) \,|\, c \in C(v)\}) \quad \text{for } \ell = L, \ldots, 0 \text{ and all } v \text{ with } d_T(v, r) = \ell,$$

with $f_{\text{agg}}(\varnothing) = 0$ for leaves. Setting $\Phi\big(\mathbf{h}_c, \mathbf{x}_v\big)$ as a standard LSTM update recovers the Tree LSTM of Tai et al., 2015. The tree representation is taken as $\mathbf{h}_r$ at the root. In Section 4, we propose a canonicalization of graphs via spanning tree covers that can be used as input to recurrent tree models.

3 Limitations of Sequence-based Canonicalizations

In this section, we characterize the limitations of single-sequence canonicalization. First, we quantify how sequence canonicalization distorts graph structure, stretching and contracting graph distances. We next turn to expressivity and demonstrate that even when the sequence model is universal, the full canonical pipeline is no more expressive than its node labeler because it relies on a single canonical representative. Together, these limitations motivate our tree cover–based canonicalization, which better preserves distances and increases expressivity by operating on a cover of spanning trees.

3.1 Distance Distortion under Sequence Canonicalization

To formalize how sequence canonicalization fails to preserve structure, we use distortion (Matoušek, 2013), which quantifies the stretch/contraction in distance after mapping points between spaces. Intuitively, we prefer canonicalizations with lower distortion, better preserving the original distances.

Definition 3.1 (Distortion)**.** Let (X, d_X) and (Y, d_Y) be metric spaces. A mapping $f : (X, d_X) \to (Y, d_Y)$ has distortion $D \geq 1$ if there exists $r > 0$ such that for all $x, y \in X$,

$$r\, d_X(x, y) \;\leq\; d_Y\big(f(x), f(y)\big) \;\leq\; D\, r\, d_X(x, y).$$

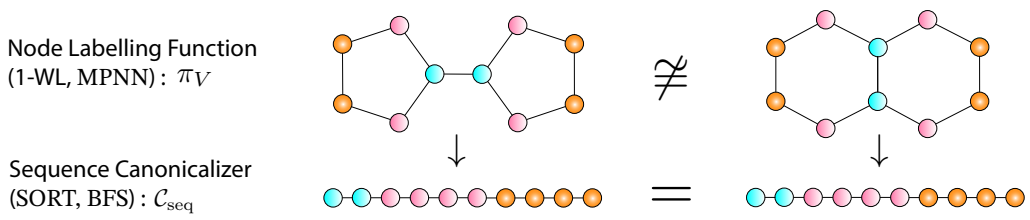


Figure 3: Sequence canonicalization is only as expressive as its labeler π_V despite using a universal downstream sequence model. f_{CanSeq} thus fails to distinguish graphs π_V fails to distinguish.

Let (G, d_G) denote a graph G with shortest–path distance d_G, and let $\mathcal{C}_{\text{seq}}(G, \pi_V)$ be its single canonical sequence under π_V. Equip $\mathcal{C}_{\text{seq}}$ with the positional distance $d_{\text{seq}}(u, v) = |\sigma(u) - \sigma(v)|$, where $\sigma : V \to \{1, \ldots, |V|\}$ is the induced ordering. The next proposition lower bounds the distortion of $\mathcal{C}_{\text{seq}}$ with the *graph bandwidth* (Díaz et al., 2002), $\varphi(G)$, which measures the smallest maximum stretch over any edge when G is laid out on a line across all orderings:

$$\varphi(G) \;=\; \min_{\sigma} \max_{(u,v)\in E} |\sigma(u) - \sigma(v)|.$$

Proposition 3.2 (Graph bandwidth lower bounds sequence distortion)**.** *Let D_{seq} be the distortion of $\mathcal{C}_{\text{seq}}(G, \pi_V)$ from (G, d_G) to the line with distance d_{seq}. Then, for any π_V, $\varphi(G) \;\leq\; D_{\text{seq}}$.*

All proofs are in Appendix A. The bandwidth lower bound gives a concrete well-studied graph metric to evaluate the distortion of $\mathcal{C}_{\text{seq}}$. Although $\varphi(G)$ is hard to compute in general, it is known for many families (Figures 1, 2): on n-node stars S_n and cliques K_n one has $\varphi(S_n) = \varphi(K_n) = \Theta(n)$, so any single–sequence canonicalization incurs the worst-case linear distortion; on complete binary trees $\varphi(T_{2,\ell}) = \Theta(2^\ell/\ell) = \Theta(n/\log n)$, and on cycles C_n and paths P_n one has $\varphi(P_n) = \varphi(C_n) = \Theta(1)$. Beyond specific families, the bound offers general insights. Given that $\varphi(G) \geq (n-1)/\operatorname{diam}(G)$, D_{seq} is at least $(n-1)/\operatorname{diam}(G)$. It is also monotone under edge addition, indicating that highly connected graphs, reflected by larger algebraic connectivity λ_2, force larger distortion. These effects negatively impact the sequence model: distorted distances make structure more difficult to capture. Importantly, any method relying on sequences, including canonicalizations and sampling approaches like RWNNs, inherits these limitations. **To address the limitations of sequences, we turn to tree representations.**

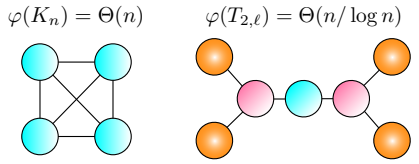


Figure 2: $\varphi(G)$ for n-node clique, K_n, and complete binary tree with ℓ levels, $T_{2,\ell}$.

3.2 Expressive Limitations of Sequence Canonicalization

Beyond the limitations of sequence representations due to distortion, we characterize the expressive limitations of the full canonical model due to relying only on a single representative. Formally, we show that f_{CanSeq} when equipped with universal f_{Seq} is only as expressive as its node labeler π_V.

Proposition 3.3 (π_V and f_{CanSeq} are equally expressive)**.** *Let f_{CanSeq} be a canonical sequence–based model with universal f_{seq} and let π_V be its labeling function. Then, $f_{\text{CanSeq}} \simeq \pi_V$.*

If π_V is an MPNN, its power matches 1-WL; consequently, f_{CanSeq} inherits 1-WL limitations and fails on the same graph families (Figure 3). Crucially, this holds even when f_{seq} is universal: once information is lost at the labeling stage, no downstream single-sequence canonicalization can recover it, limiting the expressivity of the full pipeline. **Thus, in order to address the limitations of the single labeler, we instead consider sets of labelers and canonical representatives.**

4 Canonical Tree Cover Neural Networks (CTNNs)

To address the sequence representation limitations due to distortion and the expressive limitations due to single representatives, we introduce Canonical Tree Cover Neural Networks (CTNNs), which construct a canonical spanning tree cover. In Section 5, we demonstrate that tree representations better reflect graph distances in comparison to sequences, while sets of canonical representatives that allow for complete graph reconstruction are strictly more expressive than a single representative.

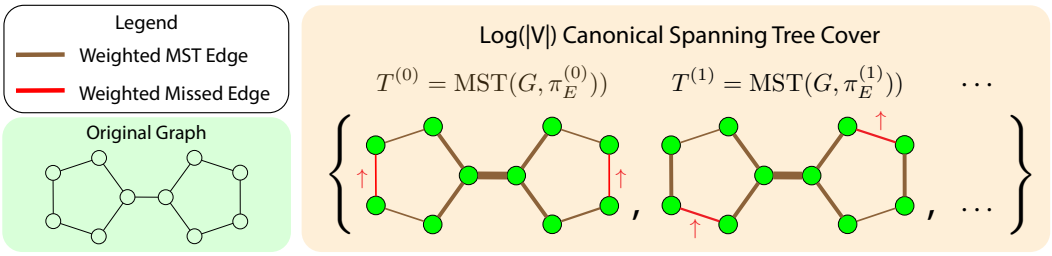


Figure 4: Canonical spanning-tree cover. At iteration k, compute $\mathrm{MST}(G, \pi_E^{(k)})$ using coverage-aware edge weights (thicker = larger magnitude weight). Edges missed in k (red) are up-weighted to bias their inclusion in $k+1$. On sparse graphs, the union of $O(\log |V|)$ trees covers all edges.

4.1 Canonical Spanning Tree Covers

To construct a canonical spanning tree cover, we leverage coverage-aware edge labelers and minimum spanning tree (MST) samplers rather than a fixed node labeler and sequence canonicalizers. By updating edge weights across rounds, later trees are biased toward edges not yet selected, yielding provable coverage across the union of sampled trees. Formally, let G be a graph and at iteration $k \in \{0, \ldots, K-1\}$ for hyperparameter K let $\pi_E^{(k)} \colon E \to \mathbb{R}$ be an edge labeler. Let $\mathcal{C}_{\text{tree}}$ be an MST extractor that maps an edge–labeled graph $\big(G, \pi_E^{(k)}\big)$ to a spanning tree $T^{(k)}$ according to weights $\pi_E^{(k)}$, setting the root node as the center of $T^{(k)}$. To promote edge coverage across the set, we update the weights by penalizing edges used in the last tree $T^{(k)}$ with hyperparameter τ. We initialize with any isomorphism-invariant node labeler π_V (e.g., degree), which biases MSTs towards edges incident to high label nodes. Formally, the update and initialization can be written:

$$\pi_E^{(k+1)}(e) \;=\; \pi_E^{(k)}(e) \;+\; \tau\, \mathbb{1}\{e \in T^{(k)}\}, \quad \pi_E^{(0)}(u,v) \;=\; -\big(\pi_V(u) + \pi_V(v)\big).$$

We refer to further implementation and pseudocode details of the construction in Appendix B.

4.2 Invariant Canonical Tree Neural Networks

Given a canonical cover of MSTs, $\mathcal{T} = \{T^{(k)}\}_{k=0}^{K-1}$, we process each tree with a recurrent tree encoder and augment it with message passing over the remaining non–tree edges to capture the local connectivity missed by each individual spanning tree. Let the residual graph be $G \backslash T^{(k)} := (V,\, E \setminus E(T^{(k)}))$ and denote f_{tree} as a recurrent tree encoder (e.g., Tree–LSTM) and f_{MPNN} an MPNN. For each k and node $i \in V$, define the per–tree representation

$$f_{\text{TreeMPNN}}\big(T^{(k)}\big)_i \;=\; f_{\text{tree}}\big(T^{(k)}\big)_i \;+\; f_{\text{MPNN}}\big(G \backslash T^{(k)}\big)_i$$

We then aggregate across the set of trees with a permutation–invariant operator f_{agg} to obtain

$$f_{\text{CTNN}}(G) \;\coloneqq\; f_{\text{agg}}\Big(\big\{\, f_{\text{TreeMPNN}}\big(T^{(k)}\big) :\; T^{(k)} = \mathcal{C}_{\text{tree}}\big(G, \pi_E^{(k)}\big),\; k = 0, \ldots, K-1 \big\}\Big).$$

Probabilistic invariance. Because CTNNs sample an MST at each iteration, we adopt probabilistic invariance (Bloem-Reddy and Teh, 2020). A randomized graph representation is invariant if its distribution is unchanged by any isomorphic graph. Formally, for a permutation $g \in \mathbb{S}_n$ acting on G by relabeling nodes, we will show that the random output $f_{\text{CTNN}}(G)$ has the same distribution as $f_{\text{CTNN}}(g \cdot G)$; consequently, the averaged predictor $\mathbb{E}\big[f_{\text{CTNN}}(G)\big]$ is an invariant function on graphs.

Theorem 4.1 (Probabilistic invariance of CTNNs). *A randomized graph representation $X(G)$ is probabilistically invariant if its distribution is unchanged under any node relabeling, i.e., $X(G) \overset{d}{=} X(g \cdot G)$ for every permutation $g \in \mathbb{S}_n$. The random output $f_{\text{CTNN}}(G)$ is probabilistically invariant:*

$$f_{\text{CTNN}}(G) \;\overset{d}{=}\; f_{\text{CTNN}}(g \cdot G) \quad \textit{for all } g \in \mathbb{S}_n.$$

Then, $\Phi(G) \coloneqq \mathbb{E}\big[f_{\text{CTNN}}(G)\big]$ is an invariant function satisfying $\Phi(G) = \Phi(g \cdot G)$ for all $g \in \mathbb{S}_n$.

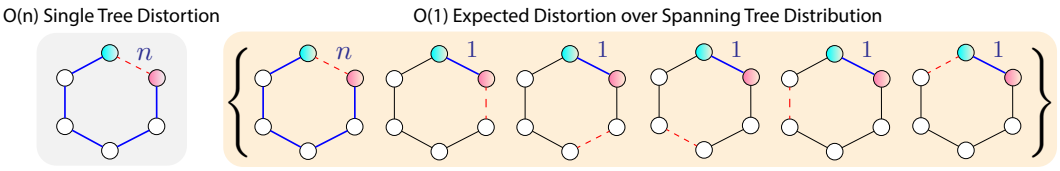


Figure 5: Single tree distortion is $O(n)$ on C_n, while expected distortion is constant over a spanning tree distribution since on average the distance between any two nodes is small.

4.3 Runtime Complexity

CTNN preprocessing is primarily dominated by constructing the K MSTs and cost of π_V. Using Kruskal's algorithm (Kruskal, 1956), the total cost is $O(K\, m \log n + \pi_V)$, which is efficient on sparse graphs where $m = O(n)$ and for inexpensive π_V (e.g., degree). A major practical advantage of canonicalization is that these trees are computed once before training and reused across epochs, eliminating on-the-fly sampling incurred by sampling approaches. The computation parallelizes naturally across graphs, and the memory cost is small ($O(Kn)$ edges per graph). Empirically, we show this preprocessing time is efficient across datasets (Appendix E.2).

5 Distortion and Expressivity Bounds for CTNNs

We first analyze distance preservation: because CTNNs aggregate over spanning trees, they yield distortion bounds that better preserve graph distance in comparison to single-sequence canonicalization. We then turn to expressivity, establishing the benefits of sets of canonical representatives. On sparse graphs our tree cover recovers the full edge set with only $O(\log m)$ trees, which has two immediate consequences for expressivity: (i) CTNNs are strictly more expressive than single–sequence canonicalizations, and (ii) when paired with universal tree encoders, CTNNs become universal.

5.1 Expected Distortion Bounds for CTNNs

We first analyze how well CTNNs preserve distances, establishing distortion bounds for $\mathcal{C}_{\text{tree}}$. Because CTNNs sample MSTs, we use probabilistic distortion (Fakcharoenphol et al., 2003).

Definition 5.1 (Expected distortion)**.** Let (X, d_X) be a metric space and let μ be a distribution on metrics $\mathcal{M}(X)$. The *expected distortion* of μ is the least $D \geq 1$ such that for some $r > 0$ and for all $x, y \in X$,

$$r\, d_X(x,y) \;\leq\; \mathbb{E}_{\rho\sim\mu}\big[\rho(x,y)\big] \;\leq\; D\, r\, d_X(x,y).$$

As a baseline, we analyze the case in which CTNN samples uniform spanning trees (USTs) (obtained when $\tau = 0, \pi_V = \mathbf{0}$), where we experimentally verify similar behavior in the general setting. In this regime, the expected tree distance between nodes u and v is upper bounded by the square root of their hitting time, the expected number of steps a random walk takes to travel from u to v:

Theorem 5.2 (UST expected distortion)**.** *Let G be a graph, and let T be a uniform random spanning tree of G. Denote by $H(u,v)$ the random walk hitting time from u to v. Then,*

$$D_{\text{UST}} = \max_{u,v} \frac{\mathbb{E}\big[d_T(u,v)\big]}{d_G(u,v)}, \quad \mathbb{E}\big[d_T(u,v)\big] \;\leq\; \sqrt{\frac{H(u,v)+H(v,u)}{2}},$$

In contrast to the bandwidth lower bound for single–sequence canonicalization, which can force worst-case distortion, the expected UST distortion aligns with random walk distance and preserves structure significantly better on sparse families. Every tree admits a unique spanning tree, so on trees $D_{\text{UST}} = 1$. By comparison, $\mathcal{C}_{\text{seq}}$ incurs distortion $\Theta(n/\log n)$ on balanced trees and $\Theta(n)$ on stars. On C_n, distortion is also constant, highlighting the benefit of averaging over trees (Figure 5), while on dense cliques K_n, $D_{\text{UST}} = \Theta(\sqrt{n})$. Despite the $\Theta(\sqrt{n})$ distortion, this remains smaller than $\mathcal{C}_{\text{seq}}$ which again incurs $\Theta(n)$ distortion. Our bounds also provide general insights: tree distances behave well in sparse graphs, where the square root of hitting times and shortest paths scale comparably. In highly dense graphs, however, shortest paths are smaller than hitting times and distortion worsens. Overall, CTNNs yield expected distortion that is small on many sparse structures where in comparison single sequences stretch distances, better capturing graph structure for downstream encoders.

5.2 Coverage and Expressivity Guarantees via MST Canonicalization

We now turn to the expressive benefits of CTNNs. Instead of relying on a single canonical representative, CTNNs build a spanning tree cover, providing downstream encoders access to full structure. We first show our coverage–aware MST scheme needs only logarithmically many trees to cover all edges on sparse graphs. We then leverage coverage to show CTNN expressivity is strictly greater than sequence–based canonicalization and establish its universality on graph functions.

Lemma 5.3 (Logarithmic spanning–tree cover)**.** *Let $G = (V, E)$ be a graph with $m = |E|$ and arboricity $\Upsilon(G)$, the minimum number of forests required to cover G. Fix any node labeler π_V with $\tau > \max_e \pi_E^{(0)}(e) - \min_e \pi_E^{(0)}(e)$. Denote $\mathcal{T} = \{T^{(k)}\}_{k=0}^{K-1}$ as the set of trees produced by a CTNN. If $K \geq \Upsilon(G) \ln m$ iterations, the union of the MSTs covers all edges: $\bigcup_{k=0}^{K-1} E\big(T^{(k)}\big) = E$.*

Importantly, on sparse graphs, arboricity is constant and CTNNs obtain full coverage with $K \geq O(\log(|V|))$. As established in Section 3, f_{CanSeq} is only as expressive as π_V. CTNNs, by contrast, operate on a tree cover, and as a result are *strictly* more expressive than f_{CanSeq} when $\pi_V \simeq f_{\mathrm{MPNN}}$.

Lemma 5.4 (f_{CanTree} is strictly more expressive than f_{MPNN} and f_{CanSeq})**.** *Suppose K satisfies Lemma 5.3. Let $\pi_V \simeq f_{\mathrm{MPNN}}$. Then, $\pi_V \prec f_{\mathrm{CanTree}}$ and hence $f_{\mathrm{CanSeq}} \prec f_{\mathrm{CanTree}}$.*

Notably, f_{CanTree} initializes π_E with π_V, but additionally leverages evolving edge weights that ensure full edge coverage across trees, allowing f_{CanTree} to surpass the expressivity of π_V. Moreover, equipped with Lemma 5.3, CTNNs can achieve universality when its tree encoder is universal.

Theorem 5.5 (CTNN Universality)**.** *Let $\mathcal{G}$ be a finite class of graphs. Assume: (i) K satisfies Lemma 5.3; (ii) the tree encoder f_{tree} and aggregation f_{agg} are universal on their domains. Then for any continuous invariant graph function $f : \mathcal{G} \to \mathbb{R}$ and any $\varepsilon > 0$, there exists a CTNN such that*

$$\sup_{G \in \mathcal{G}} \big| f_{\mathrm{CTNN}}(G) - f(G) \big| \leq \varepsilon.$$

6 Experiments and Results

Through empirical evaluation we aim to answer the following research questions, extending our theory by testing CTNNs on datasets with factors not explicitly addressed in the theoretical analysis (e.g., class imbalance), and including domain–specific canonicalizations beyond our theory, such as molecular fingerprints (Rogers and Hahn, 2010) commonly used in molecular analysis.

- **RQ1 (Discriminative performance).** How does CTNN compare to (i) invariant GNNs (MPNNs, GTs), (ii) sampling approaches (RWNN), and (iii) canonicalization baselines?
- **RQ2 (Distance distortion).** Do CTNNs reduce metric distortion relative to sequence-based canonicalizations, and does this reduction translate into improved task performance?
- **RQ3 (Ablations and sensitivity).** Which components of CTNN contribute most to performance, and how sensitive is performance to their settings?

6.1 Experimental Setup

Datasets. We evaluate on molecular and protein benchmarks, domains where canonicalization is widely adopted and frequently used in practice (Goh et al., 2017; Alley et al., 2019) and where long–range dependencies and high expressivity are critical (Dwivedi et al., 2022). For molecules, we use tasks from the **PCBA** datasets from MoleculeNet (Wu et al., 2018). For proteins, we adopt ProteinShake (Kucera et al., 2023) datasets: **SCOP**, **PFAM**, **GO MOL**, **GO BIO**. These tasks span diverse molecule and protein tasks such as molecular activity and protein structure classification. Notably, proteins are larger and denser than molecules, making structure more difficult to capture.

Baselines. We consider invariant GNNs and sampling approaches: (1) **GCN** (Kipf and Welling, 2017), (2) **GAT** (Veličković et al., 2018), (3) **GIN** (Xu et al., 2019),(4) **GT** (Dwivedi and Bresson, 2021), and (5) **RWNN** (Kim et al., 2025). We next evaluate canonicalization approaches: (6) **Fingerprint** (Rogers and Hahn, 2010), stacking an MLP on hand-crafted chemical descriptors, (7) **SMILES** (Goh et al., 2017), applying sequence models over canonical SMILES, (8) **Primary Seq.** (Alley et al., 2019), applying sequence models to the primary sequence, (9) **DGCNN** (Zhang et al., 2018), a representative sequence-based canonical approach leveraging MPNNs as π_V and sorting as $\mathcal{C}_{\mathrm{seq}}$, and (10) **RCM** (Diamant et al., 2023), applying sequence models to the ordering determined by

Table 1: Median (min, max) of model performance ($\times 100$) across 5 test splits. We highlight in **blue** the best model. "NA" indicates not applicable; "OOT" denotes training exceeds the time limit (24h).

		Molecular Benchmarks				Protein Benchmarks			
		PCBA-1030	**PCBA-1458**	**PCBA-4467**	**PCBA-5297**	**SCOP**	**PFAM**	**GO BIO**	**GO MOL**
	# Graphs	160K	200K	240K	300K	10K	25K	22K	32K
	Avg. $\lvert V\rvert$	24.29	25.05	25.27	25.19	217.5	251.3	254.5	250.1
	Avg. $\lvert E\rvert$	26.18	27.10	27.24	27.20	593.8	691.5	698.5	687.5
	Metric	AUC ↑	AUC ↑	AUC ↑	AUC ↑	ACC ↑	ACC ↑	AUC ↑	AUC ↑
MP/GT/RW	**GCN**	72.7 (70.3, 74.7)	84.9 (84.2, 85.6)	80.9 (78.6, 82.7)	91.4 (88.2, 91.7)	63.4 (62.8, 64.9)	9.3 (6.4, 11.5)	59.2 (57.9, 69.7)	60.6 (49.8, 84.5)
	GAT	71.9 (64.9, 72.8)	80.5 (79.8, 80.8)	76.5 (74.7, 80.0)	89.3 (88.1, 90.6)	58.9 (51.6, 59.9)	5.1 (2.5, 6.0)	57.0 (53.2, 58.7)	57.6 (50.3, 81.1)
	GIN	75.6 (71.3, 77.4)	85.7 (84.4, 86.4)	82.9 (81.8, 83.9)	92.2 (90.7, 92.5)	68.0 (67.9, 69.2)	20.0 (18.1, 21.0)	66.3 (59.9, 79.0)	83.7 (81.5, 85.6)
	GT	68.1 (67.9, 68.6)	81.2 (81.0, 81.5)	78.9 (77.8, 79.9)	87.7 (87.6, 88.2)	OOT	OOT	OOT	OOT
	RWNN	62.1 (62.0, 63.3)	77.0 (75.7, 77.1)	75.0 (74.6, 76.5)	80.6 (80.6, 81.1)	59.0 (58.4, 60.2)	13.5 (12.1, 14.9)	65.4 (64.9, 65.8)	76.7 (74.1, 77.3)
Canonicalization	**Fingerprint**	79.3 (78.5, 79.5)	86.7 (85.9, 88.1)	83.8 (83.2, 84.8)	92.4 (91.4, 93.2)	NA	NA	NA	NA
	SMILES	71.6 (70.2, 72.5)	84.9 (84.5, 86.4)	80.9 (80.0, 81.4)	90.2 (89.8, 90.8)	NA	NA	NA	NA
	Primary Seq.	NA	NA	NA	NA	63.0 (60.8, 63.5)	23.5 (17.4, 26.3)	74.3 (69.2, 79.5)	**85.2 (84.5, 85.8)**
	DGCNN	73.1 (72.7, 73.9)	86.3 (86.0, 86.8)	82.8 (82.1, 83.7)	91.6 (91.2, 92.1)	65.3 (64.6, 67.8)	20.8 (20.2, 23.7)	62.0 (59.7, 68.9)	84.5 (84.0, 84.7)
	RCM	77.8 (77.1, 77.9)	87.7 (87.2, 89.0)	85.3 (84.4, 85.7)	93.0 (92.6, 93.4)	57.0 (56.5, 57.8)	22.1 (16.8, 23.4)	68.4 (66.7, 69.3)	83.3 (82.6, 83.7)
	DFS SET	65.6 (60.2, 67.6)	78.7 (77.3, 80.7)	75.5 (74.8, 79.0)	83.6 (83.3, 84.0)	55.8 (54.1, 57.2)	14.3 (12.3, 14.8)	75.6 (74.4, 77.6)	84.6 (83.8, 85.9)
	CTNN (ours)	**80.6 (80.3, 81.2)**	**89.1 (88.0, 89.9)**	**86.8 (86.5, 87.4)**	**94.6 (94.2, 94.9)**	**72.0 (71.4, 72.3)**	**24.7 (20.9, 26.0)**	**78.3 (77.9, 79.4)**	84.3 (84.0, 86.0)

Table 2: Mean ± s.d. of empirical stretch and contraction across 50 random samples for canonicalizations. In comparison to all canonicalizations, CTNNs significantly reduce stretch and contraction.

	Max Stretch ↓				Max Stretch ↓			
	PCBA-1030	**PCBA-1458**	**PCBA-4467**	**PCBA-5297**	**SCOP**	**PFAM**	**GO BIO**	**GO MOL**
SMILES	18.12 ± 5.49	20.2 ± 6.02	19.32 ± 6.95	19.74 ± 6.36	NA	NA	NA	NA
Primary Seq.	NA	NA	NA	NA	172.6 ± 34.11	164.36 ± 45.55	165.72 ± 44.51	173.08 ± 37.83
DGCNN	18.96 ± 4.07	19.40 ± 4.63	19.48 ± 4.99	18.64 ± 4.02	196.44 ± 16.03	196.96 ± 15.87	192.56 ± 15.54	192.84 ± 14.62
RCM	3.38 ± 0.71	3.66 ± 1.17	3.64 ± 1.05	3.64 ± 0.86	34.68 ± 7.68	32.32 ± 7.56	33.76 ± 9.11	33.44 ± 8.71
DFS SET	18.41 ± 5.89	18.91 ± 6.59	18.77 ± 6.85	18.68 ± 6.25	211.02 ± 19.28	211.52 ± 19.06	209.45 ± 20.51	210.90 ± 16.52
CTNN (ours)	**2.23 ± 0.26**	**2.18 ± 0.22**	**2.24 ± 0.30**	**2.28 ± 0.30**	**17.85 ± 3.15**	**18.21 ± 4.72**	**17.56 ± 4.56**	**18.12 ± 4.45**
	Max Contraction ↓				**Max Contraction ↓**			
SMILES	5.32 ± 1.96	6.22 ± 2.06	5.54 ± 1.89	5.34 ± 1.86	NA	NA	NA	NA
Primary Seq.	NA	NA	NA	NA	2.72 ± 2.86	5.16 ± 5.34	4.44 ± 5.62	5.44 ± 6.31
DGCNN	12.82 ± 2.79	13.24 ± 2.76	13.16 ± 2.54	12.02 ± 2.37	16.32 ± 3.25	17.56 ± 4.85	16.04 ± 4.12	16.16 ± 4.15
RCM	4.66 ± 1.94	4.94 ± 1.98	5.50 ± 2.62	5.70 ± 2.30	12.56 ± 2.04	12.00 ± 2.60	12.16 ± 2.37	11.92 ± 2.34
DFS SET	4.92 ± 1.44	5.71 ± 1.82	5.46 ± 1.86	5.30 ± 1.49	9.22 ± 2.58	9.45 ± 2.20	8.96 ± 1.60	8.49 ± 1.86
CTNN (ours)	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**

the Cuthill-McKee algorithm. We also include (11) **DFS SET**, a set-based sequence approach. We provide a summary of the design space for all canonicalizations in Appendix C.

Training and Evaluation. For all benchmarks, we set f_{tree} as a Tree-LSTM, f_{MPNN} as a GIN, f_{agg} as SUM, $\pi_V(v) = \deg(v)$, and $\tau = 1$. For molecular datasets, we set $K = 4$, and for proteins, we use $K = 8$. Following each dataset's protocol, performance is computed as AUC or accuracy. We report median (min, max) performance over five random splits (60/20/20), which is more robust than mean and standard deviation for small sample sizes. We compute stretch as $\max_{i,j}\{d_{\text{emb}}(i,j)/d_G(i,j)\}$ and contraction as $\max_{i,j}\{d_G(i,j)/d_{\text{emb}}(i,j)\}$. For sequence canonicalizations, $d_{\text{emb}} = d_{\text{seq}}$. For DFS SET and CTNNs, we report expected distortion as the average across the sequences or trees (e.g., $\max_{i,j} \text{mean}_k\{d_G(i,j)/d_{T^{(k)}}(i,j)\}$). We provide remaining details in Appendix D.

6.2 RQ1 & RQ2: Discriminative Performance and Distance Distortion

CTNNs significantly outperform invariant GNNs, consistent with the theoretical expressivity gains established in Section 5.2 (Table 1). CTNNs also outperform RWNN, demonstrating the benefits of canonicalizaion over sampling approaches. While some canonicalizations are competitive, they depend on domain knowledge and lack generality (e.g., Fingerprint). Notably, CTNNs outperform or match all sequence-based canonicalizations, including those that are domain-driven and provide one-to-one encodings of their graphs (SMILES, Primary Seq.), allowing for maximal expressivity.

We attribute CTNNs' gains to distortion introduced by sequences (Tables 2). Across molecular and protein benchmarks, CTNNs achieve substantially smaller stretch than sequence-based canonicalizations. Crucially, trees never contract distances, obtaining optimal contraction $= 1$. In contrast, sequences exhibit both large stretch and nontrivial contraction. A noteworthy case is RCM: its ordering reduces bandwidth and lowers stretch on molecular graphs, yet it still doesn't reach CTNN performance because it incurs contraction. Moreover, on denser protein graphs its stretch dramatically

Table 3: Median (max-min) performance for ablations on benchmarks across 5 test splits. CTNN (full) obtains or matches the best performance across all datasets, supporting each design choice.

Ablation	PCBA-1030	PCBA-1458	PCBA-4467	PCBA-5297	SCOP	PFAM	GO BIO	GO MOL
Single can. tree instead of cover	79.4 (1.2)	86.2 (0.4)	85.6 (0.6)	92.2 (0.6)	67.5 (1.3)	22.0 (3.6)	69.5 (0.7)	56.2 (12.1)
MPNN instead of TreeRNN	76.7 (0.8)	85.8 (0.2)	82.9 (1.6)	91.5 (1.0)	68.2 (0.5)	18.9 (2.9)	63.1 (2.6)	82.8 (0.9)
No MPNN on residual edges	**80.9 (0.2)**	**89.2 (0.5)**	**87.0 (0.5)**	**94.6 (0.2)**	69.2 (1.3)	**26.8 (5.2)**	**77.6 (1.5)**	61.8 (16.9)
CTNN (full)	**80.6 (0.9)**	**89.1 (1.9)**	**86.8 (0.9)**	**94.6 (0.7)**	**72.0 (0.9)**	**24.7 (5.1)**	**78.3 (1.5)**	**84.3 (2.0)**

increases, underscoring a fundamental limitation of single sequence canonical representatives. While DFS SET, which leverages sets of sequences, can improve performance in comparison to a single sequence (e.g., GO BIO), it still underperforms relative to CTNNs across most benchmarks because it also incurs significant stretch and contraction, indicating sets of sequences do not capture distances as well as sets of trees. Collectively, these results align with our theory: canonical spanning-tree covers preserve graph distances significantly better than sequences, enabling stronger downstream models.

6.3 RQ3: Ablations and Sensitivity

Ablations. We evaluate three CTNN variants to isolate what contributes most to its performance (Table 3). (i) Replacing the cover with a single canonical tree reduces edge coverage, limits expressivity, and can increase distortion by collapsing to a single representative. Thus, it underperforms across all benchmarks, especially in protein graphs where multiple trees significantly increase coverage and reduce distortion on average. (ii) Replacing the TreeRNN with an MPNN is equivalent to a standard message-passing encoder on the full graph, reintroducing 1-WL expressivity limits, and significantly drops performance across all benchmarks. (iii) Removing the processing of residual edges leaves performance largely unaffected on sparse molecules, where few edges remain after MST extraction, but drops performance on denser proteins, where residual edge processing can help capture local signals. Overall, CTNN (full) obtains or matches best performance across datasets, and the analysis highlights that (a) the canonical tree cover and (b) expressive tree encoder are the primary drivers of performance, while the residual MPNN provides complementary gains on denser graphs.

Sensitivity. We also conduct sensitivity analyses for different choices of number of trees, K, node labeler, π_V, and penalty, τ (Appendix E.1). Increasing K yields consistent gains: edge coverage rises rapidly, average distortion decreases, and performance improves. These results align with our theory that only a small number of trees is needed for full coverage on sparse graphs and additional trees better capture original graph distances on average, resulting in increased performance for larger K. CTNN is also robust to node labeler π_V: degree, closeness centrality (CC), and 1-WL are close in performance, with CC and 1-WL offering improvements at higher cost. In the main experiments, we default to degree for its efficiency. CTNN is also stable across the penalty τ, where coverage, distortion, and accuracy follow similar trends across choices of τ.

7 Conclusion

In this work, we developed the first theoretical analysis of sequence-based canonicalization for graphs, establishing that sequences distort structure and that single-representative approaches are constrained by the expressivity of their labelers. This analysis covered canonicalizations widely used in practice such as domain-driven sequences including SMILES (Goh et al., 2017; Honda et al., 2019) and primary protein sequences (Alley et al., 2019; Rao et al., 2019), learnable orderings based on GNNs and differentiable sorting (Niepert et al., 2016; Zhang et al., 2018), and algorithmic orderings that optimize bandwidth (Cuthill and McKee, 1969; Diamant et al., 2023). Motivated by this analysis, we introduced *Canonical Tree Cover Neural Networks*, which construct canonical spanning-tree covers and leverage expressive tree encoders. CTNNs are provably invariant, preserve graph distances, and are more expressive than sequence canonicalizations. Empirically, CTNNs outperform invariant GNNs, sampling approaches, and canonicalization baselines on molecular and protein benchmarks.

Our coverage and expressivity guarantees rely on sparsity assumptions, and thus, characterizing CTNNs in dense regimes remains open. Despite the focused scope, CTNNs consistently maintain advantages in our experiments, highlighting the value of spanning-tree covers over sequences. More broadly, our results underscore the importance of canonical representations that respect underlying graph geometry. By leveraging canonical tree covers, CTNNs offer an expressive, invariant, and efficient framework for learning on sparse graphs.

REFERENCES

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, 2015.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

Francesco Di Giovanni, T Konstantin Rusch, Michael M Bronstein, Andreea Deac, Marc Lackenby, Siddhartha Mishra, and Petar Veličković. How does over-squashing affect the power of gnns? *arXiv preprint arXiv:2306.03589*, 2023.

Yuanqing Wang and Kyunghyun Cho. Non-convolutional graph neural networks. In *Advances in Neural Information Processing Systems*, 2024.

Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Walking out of the weisfeiler leman hierarchy: Graph learning beyond message passing. *Transactions in Machine Learning Research*, 2023.

Dexiong Chen, Till Hendrik Schulz, and Karsten Borgwardt. Learning long range dependencies on graphs via random walks. In *International Conference on Learning Representations*, 2025.

Jinwoo Kim, Olga Zaghen, Ayhan Suleymanzade, Youngmin Ryou, and Seunghoon Hong. Revisiting random walks for learning on graphs. In *International Conference on Learning Representations*, 2025.

Benjamin Bloem-Reddy and Yee Whye Teh. Probabilistic symmetries and invariant neural networks. *Journal of Machine Learning Research*, 21(90):1–61, 2020.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*, 2019.

Garrett B Goh, Nathan O Hodas, Charles Siegel, and Abhinav Vishnu. Smiles2vec: An interpretable general-purpose deep neural network for predicting chemical properties. *arXiv preprint arXiv:1712.02034*, 2017.

Shion Honda, Shoi Shi, and Hiroki R Ueda. Smiles transformer: Pre-trained molecular fingerprint for low data drug discovery. *arXiv preprint arXiv:1911.04738*, 2019.

Seyone Chithrananda, Gabriel Grand, and Bharath Ramsundar. Chemberta: large-scale self-supervised pretraining for molecular property prediction. *arXiv preprint arXiv:2010.09885*, 2020.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*, 2020.

Brendan D McKay et al. Practical graph isomorphism. 1981.

Yicheng Xiao, Lin Song, Jiangshan Wang, Siyu Song, Yixiao Ge, Xiu Li, Ying Shan, et al. Mambatree: Tree topology is all you need in state space model. *Advances in Neural Information Processing Systems*, 37:75329–75354, 2024.

Jirı Matoušek. Lecture notes on metric embeddings. *ETH Zürich*, 2013.

Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.

Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, 2003.

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

Ethan C Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature methods*, 16(12):1315–1322, 2019.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. In *Advances in Neural Information Processing Systems*, 2022.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Tim Kucera, Carlos Oliver, Dexiong Chen, and Karsten Borgwardt. Proteinshake: Building datasets and benchmarks for deep learning on protein structures. In *Advances in Neural Information Processing Systems*, 2023.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. In *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

Nathaniel Lee Diamant, Alex M Tseng, Kangway V Chuang, Tommaso Biancalani, and Gabriele Scalia. Improving graph generation by restricting graph bandwidth. In *International Conference on Machine Learning*, pages 7939–7959. PMLR, 2023.

Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. Evaluating protein transfer learning with tape. *Advances in neural information processing systems*, 32, 2019.

Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172, 1969.

Russell Lyons and Yuval Peres. *Probability on trees and networks*, volume 42. Cambridge University Press, 2017.

László Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2(1-46):4, 1993.

A Omitted Mathematical Proofs

A.1 Distance Distortion under Sequence Canonicalization

Proposition A.1 (Bandwidth lower-bounds sequence distortion)**.** *Let $G = (V, E)$ be a connected, unweighted graph with shortest-path metric d_G. Let $\varphi(G) := \min_\sigma \max_{\{u,v\}\in E} |\sigma(u) - \sigma(v)|$ be the bandwidth of G. Then for every ordering π,*

$$\varphi(G) \;\le\; D_{\mathrm{seq}}(\pi).$$

Proof. For an injective ordering $\pi : V \to \{1, \dots, n\}$, define the sequence distance $d^\pi_{\mathrm{seq}}(u, v) := |\pi(u) - \pi(v)|$. The (two-sided) distortion can be written

$$D_{\mathrm{seq}}(\pi) \;:=\; \frac{\max\limits_{u\neq v} \dfrac{d^\pi_{\mathrm{seq}}(u,v)}{d_G(u,v)}}{\min\limits_{u\neq v} \dfrac{d^\pi_{\mathrm{seq}}(u,v)}{d_G(u,v)}}.$$

Define $\rho_\pi(u, v) := d^\pi_{\mathrm{seq}}(u, v)/d_G(u, v)$ for $u \neq v$, so that $D_{\mathrm{seq}}(\pi) = \frac{\max \rho_\pi}{\min \rho_\pi}$.

For any edge $\{u, v\} \in E$, $d_G(u, v) = 1$, hence $\rho_\pi(u, v) = |\pi(u) - \pi(v)|$. Therefore,

$$\max_{u\neq v} \rho_\pi(u, v) \;\ge\; \max_{\{u,v\}\in E} |\pi(u) - \pi(v)| \;=\; \varphi(\pi).$$

Let x, y be the two adjacent vertices in π; then $d^\pi_{\mathrm{seq}}(x, y) = 1$ while $d_G(x, y) \ge 1$, so

$$\min_{u\neq v} \rho_\pi(u, v) \;\le\; \rho_\pi(x, y) = \frac{1}{d_G(x, y)} \;\le\; 1.$$

Combining the two bounds proves the claim.

$$D_{\mathrm{seq}}(\pi) = \frac{\max \rho_\pi}{\min \rho_\pi} \;\ge\; \frac{\varphi(\pi)}{1} \;\ge\; \varphi(G),$$

□

A.2 Expressive Limitations of Sequence Canonicalization

Proposition A.2 (π_V and f_{CanSeq} are equally expressive)**.** *Let f_{CanSeq} be a canonical sequence–based model with universal f_{seq} and let π_V be its labeling function. Then, $f_{\mathrm{CanSeq}} \simeq \pi_V$.*

Proof. Let $G_i = (V_i, E_i, X_i)$ for $i \in \{1, 2\}$ with $G_1 \ncong G_2$. Let $\pi_V : V_i \to \mathbb{R}$ be a node labeler and define the *augmented* features $\tilde{\mathbf{x}}_v := (\mathbf{x}_v, \pi_V(v))$. Assume the augmented multisets coincide:

$$\{\{\tilde{\mathbf{x}}_v : v \in V_1\}\} \;=\; \{\{\tilde{\mathbf{x}}_v : v \in V_2\}\}.$$

Consider a single-sequence canonicalizer $\mathcal{C}_{\mathrm{seq}}$ that outputs a permutation of V and the corresponding sequence of per-node feature vectors, without adding structural annotations and whose ordering rule is a deterministic function of $\{\tilde{\mathbf{x}}_v\}_{v\in V}$ (e.g., a stable sort by a fixed key in $\tilde{\mathbf{x}}_v$ with deterministic tie-breaking depending only on $\tilde{\mathbf{x}}_v$). Because the two graphs have the same multiset of keys, and the ordering depends solely on these keys (and not on E_i), the resulting ordered lists of features are identical:

$$\mathcal{C}_{\mathrm{seq}}(G_1, \pi_V) \;=\; \mathcal{C}_{\mathrm{seq}}(G_2, \pi_V).$$

(If ties occur, the tie-breaking is the same function of $\tilde{\mathbf{x}}$; when two items share identical $\tilde{\mathbf{x}}$, they are indistinguishable in the output sequence, so any permutation within such ties yields the same feature sequence.) Hence, there exist non-isomorphic graphs that collide under such $\mathcal{C}_{\mathrm{seq}}$ that no f_{seq} can distinguish regardless of its expressivity. □

Example (DGCNN / Sort). Let $\mathcal{C}_{\text{seq}} = \text{Sort}$ be a stable sort that orders vertices by a fixed key computed from $\tilde{\mathbf{x}}_v = (\mathbf{x}_v, \pi_V(v))$ with deterministic tie-breaking depending only on $\tilde{\mathbf{x}}_v$. If $\{\{(\mathbf{x}_v, \pi_V(v)) : v \in V_1\}\} = \{\{(\mathbf{x}_v, \pi_V(v)) : v \in V_2\}\}$ and $G_1 \ncong G_2$, then $\text{Sort}(G_1, \pi_V) = \text{Sort}(G_2, \pi_V)$. This covers the DGCNN setting where $\pi_V \simeq f_{\text{MPNN}}$ provides the sort keys; the sort-based canonicalization cannot separate G_1 and G_2 beyond what is already encoded in the augmented multiset.

A.3 Invariant Canonical Tree Neural Networks

We first introduce definitions for probabilistic invariance for random trees and covers.

Definition A.3 (Probabilistic invariance for random trees)**.** Let $\mathcal{A}$ be a randomized procedure that, on input a graph G, outputs a (labeled) spanning tree $T_{\mathcal{A}}(G)$. We say $\mathcal{A}$ is *probabilistically invariant* if for every pair of isomorphic graphs $G \cong_\pi H$ with isomorphism $\pi : V(G) \to V(H)$,

$$\pi\big(T_{\mathcal{A}}(G)\big) \overset{d}{=} T_{\mathcal{A}}(H).$$

Equivalently, $T_{\mathcal{A}}(G) \overset{d}{=} \pi^{-1}\big(T_{\mathcal{A}}(H)\big)$.

Definition A.4 (Probabilistic invariance for tree covers)**.** Let $\mathcal{A}$ output a (multi)set or sequence of trees $\mathcal{T}_{\mathcal{A}}(G) = (T^{(0)}, \dots, T^{(K-1)})$ on G. We call $\mathcal{A}$ *probabilistically invariant* if for every isomorphism $G \cong_\pi H$,

$$\pi\big(\mathcal{T}_{\mathcal{A}}(G)\big) \overset{d}{=} \mathcal{T}_{\mathcal{A}}(H),$$

where π acts elementwise on the sequence (and, for an unordered cover, equality in distribution is taken after forgetting order).

Lemma A.5 (MST is probabilistically invariant)**.** *Let $G = (V, E)$ be an undirected graph. Let $w : E \to \mathbb{R}$ be an isomorphism–invariant base weight (so $w(g \cdot e) = w(e)$ for all $g \in \mathbb{S}_{|V|}$), and let $\zeta : E \to (0, 1)$ assign i.i.d. continuous tie–breakers to edges. Run Kruskal's algorithm (Algorithm 2) with lexicographic keys $k(e) = (w(e), \zeta(e))$ and let $X_{\text{MST}}(G) = (e_0, \dots, e_{|V|-2})$ be the resulting edge sequence. Then, for every $g \in \mathbb{S}_{|V|}$,*

$$g \cdot X_{\text{MST}}(G) \overset{d}{=} X_{\text{MST}}(g \cdot G) \quad \textit{(equivalently, } X_{\text{MST}}(G) \overset{d}{=} g^{-1} \cdot X_{\text{MST}}(g \cdot G)\textit{)}.$$

Proof. We prove by induction on t that the t-th edge in Kruskal's sequence has the same *pushforward* conditional law on G and on $g \cdot G$.

For a prefix $\mathbf{x} = (e_0, \dots, e_{t-1})$ valid for Kruskal on G, let $\mathcal{C}(G; \mathbf{x})$ be the component partition (union–find state) after processing $\mathbf{x}$. Define the *admissible set*

$$A(G; \mathbf{x}) := \{\, e = \{u, v\} \in E : \ u, v \text{ lie in different components of } \mathcal{C}(G; \mathbf{x}) \,\},$$

and the *frontier* of minimum–base–weight admissible edges

$$F(G; \mathbf{x}) := \{\, e \in A(G; \mathbf{x}) : \ w(e) = \min_{e' \in A(G; \mathbf{x})} w(e') \,\}.$$

Under Kruskal with keys (w, ζ), the next edge e_t is the unique minimizer of ζ over $F(G; \mathbf{x})$ (if $|F| = 1$ the choice is deterministic). Since the ζ's are i.i.d. continuous, conditional on $\mathbf{x}$ the edge e_t is *uniform* on $F(G; \mathbf{x})$.

Base case ($t = 0$). Here $A(G; \emptyset) = E$ and $F(G; \emptyset) = \{e \in E : \ w(e) = \min_{e' \in E} w(e')\}$. Because w is isomorphism–invariant, $F(g \cdot G; \emptyset) = g \cdot F(G; \emptyset)$. The next edge is uniform on the respective frontier; pushing this uniform forward by g yields

$$g \cdot X_{\text{MST}}(G)[0] \overset{d}{=} X_{\text{MST}}(g \cdot G)[0].$$

Induction step. Assume for some $t \geq 0$ that the prefixes satisfy

$$g \cdot X_{\text{MST}}(G)[:t] \overset{d}{=} X_{\text{MST}}(g \cdot G)[:t].$$

Fix any realization $\mathbf{x} = (e_0, \dots, e_{t-1})$ of this prefix on G, and let $g\mathbf{x} = (g \cdot e_0, \dots, g \cdot e_{t-1})$ be the corresponding prefix on $g \cdot G$. Relabeling preserves adjacency, hence components map as $\mathcal{C}(g \cdot G; g\mathbf{x}) = g \cdot \mathcal{C}(G; \mathbf{x})$ and therefore

$$A(g \cdot G; g\mathbf{x}) = g \cdot A(G; \mathbf{x}), \qquad F(g \cdot G; g\mathbf{x}) = g \cdot F(G; \mathbf{x}).$$

Conditional on $\mathbf{x}$, the next edge on G is uniform on $F(G; \mathbf{x})$. Pushing this distribution forward by g gives a uniform distribution on $g \cdot F(G; \mathbf{x}) = F(g \cdot G; g\mathbf{x})$, which is exactly the conditional law of the next edge on $g \cdot G$ given $g\mathbf{x}$:

$$g \cdot \big(X_{\mathrm{MST}}(G)[t] \,\big|\, \mathbf{x}\big) \overset{d}{=} X_{\mathrm{MST}}(g \cdot G)[t] \,\big|\, g\mathbf{x}.$$

Averaging over all realizations $\mathbf{x}$ (which, by the induction hypothesis, have matching laws under G and $g \cdot G$ after applying g to the G prefix) yields

$$g \cdot X_{\mathrm{MST}}(G)[t] \overset{d}{=} X_{\mathrm{MST}}(g \cdot G)[t].$$

By induction for $t = 0, 1, \dots, |V| - 2$, we conclude $g \cdot X_{\mathrm{MST}}(G) \overset{d}{=} X_{\mathrm{MST}}(g \cdot G)$, equivalently $X_{\mathrm{MST}}(G) \overset{d}{=} g^{-1} \cdot X_{\mathrm{MST}}(g \cdot G)$. □

Theorem A.6 (Probabilistic invariance of BUILDCANONICALTREECOVER)**.** *Fix any isomorphism–invariant node labeler π_V (i.e., $\pi_V(g \cdot u) = \pi_V(u)$ for all $g \in \mathbb{S}_{|V|}$) and define base edge weights $\pi_E^{(0)}(u, v) = -(\pi_V(u) + \pi_V(v))$. For $k \geq 0$ let the iterative weights update be*

$$\pi_E^{(k+1)}(e) \;=\; \pi_E^{(k)}(e) \;+\; \tau\, \mathbf{1}\{e \in T^{(k)}\}, \qquad \tau > 0,$$

where $T^{(k)}$ is the tree returned by KRUSKALMST *(Algorithm 2) on $(G, \pi_E^{(k)})$ with i.i.d. continuous exchangeable tie–breakers $\zeta : E \to (0, 1)$ reused across all rounds. Let $\mathcal{T}(G) = (T^{(0)}, \dots, T^{(K-1)})$ be the random sequence of trees produced by* BUILDCANONICALTREECOVER *(Algorithm 1). Then, for every permutation $g \in \mathbb{S}_{|V|}$,*

$$g \cdot \mathcal{T}(G) \overset{d}{=} \mathcal{T}(g \cdot G) \quad \textit{(equivalently, } \mathcal{T}(G) \overset{d}{=} g^{-1} \cdot \mathcal{T}(g \cdot G)\textit{).}$$

Proof. We prove by induction on k that, together with the evolving weights, the next tree is distributionally equivariant under relabeling.

For clarity, write the round–k weights as a function of the history

$$\Pi^{(k)}(G) \;:=\; \pi_E^{(k)}(\,\cdot\,; T^{(0)}, \dots, T^{(k-1)}),$$

and let g act on edge–indexed objects by $(g \cdot f)(e) = f(g^{-1} \cdot e)$. The induction claim is

$$g \cdot T^{(k)}(G) \overset{d}{=} T^{(k)}(g \cdot G) \quad \text{and} \quad g \cdot \Pi^{(k+1)}(G) \overset{d}{=} \Pi^{(k+1)}(g \cdot G). \tag{$\star_k$}$$

Base case ($k = 0$). Since π_V is isomorphism–invariant, so is $\pi_E^{(0)}$: $\pi_E^{(0)}(g \cdot e) = \pi_E^{(0)}(e)$. By Lemma A.5 (KruskalMST probabilistic invariance), we have $g \cdot T^{(0)}(G) \overset{d}{=} T^{(0)}(g \cdot G)$. The update $\pi_E^{(1)} = \pi_E^{(0)} + \tau\, \mathbf{1}\{\cdot \in T^{(0)}\}$ is isomorphism–equivariant, hence $g \cdot \Pi^{(1)}(G) \overset{d}{=} \Pi^{(1)}(g \cdot G)$. Thus ($\star_k$) holds for $k = 0$.

Induction step. Assume ($\star_k$) holds for all $s < k$. Couple the two runs on G and $g \cdot G$ by reusing the same exchangeable tie–breakers ζ. By the induction hypothesis, the joint law of the prefixes

$$\big(g \cdot T^{(0)}(G), \dots, g \cdot T^{(k-1)}(G),\; g \cdot \Pi^{(k)}(G)\big)$$

equals the joint law of

$$\big(T^{(0)}(g \cdot G), \dots, T^{(k-1)}(g \cdot G),\; \Pi^{(k)}(g \cdot G)\big).$$

Conditioned on these prefixes, the round–k Kruskal call on each graph uses (isomorphism–equivariant) weights and the same i.i.d. continuous tie–breakers, so Lemma A.5 applies *conditionally* and yields

$$g \cdot \big(T^{(k)}(G) \,\big|\, T^{(<k)}(G), \Pi^{(k)}(G)\big) \overset{d}{=} T^{(k)}(g \cdot G) \,\big|\, T^{(<k)}(g \cdot G), \Pi^{(k)}(g \cdot G).$$

Averaging over the (matched) prefixes gives $g \cdot T^{(k)}(G) \overset{d}{=} T^{(k)}(g \cdot G)$. Finally, the weight update $\Pi^{(k+1)} = \Pi^{(k)} + \tau\, \mathbf{1}\{\cdot \in T^{(k)}\}$ is isomorphism–equivariant, so $g \cdot \Pi^{(k+1)}(G) \overset{d}{=} \Pi^{(k+1)}(g \cdot G)$. Thus $(\star_k)$ holds for round k.

By induction for $k = 0, 1, \ldots, K-1$ we conclude $g \cdot \mathcal{T}(G) \overset{d}{=} \mathcal{T}(g \cdot G)$. □

Having established $\mathcal{T}(G)$ is probabilistically invariant, it follows that $f_{\mathrm{CTNN}}(G)$, a deterministic function on $\mathcal{T}(G)$, is also probabilistically invariant.

A.4 Expected Distortion Bounds for CTNNs

We introduce two identities leveraging the electrical interpretation along networks: (i) the probability an edge appears in a uniform random spanning tree due to Kirchhoff, and (ii) a commute-time identity relating commute time and effective resistance.

Theorem A.7 (Kirchoff's Effective Resistance Formula (Lyons and Peres, 2017))**.** *Let $G = (V, E)$ be an unweighted connected graph, T a uniform spanning tree of G, and let $i^{(u\to v)} = (i_e^{(u\to v)})_{e\in E}$ denote the unit electrical $u \to v$ flow (so $\sum_{e\in E}(i_e^{(u\to v)})^2 = R_{\mathrm{eff}}(u, v)$). For any undirected edge e, if $P_T(u, v)$ is the unique u–v path in T, then*

$$\Pr\left[e \in E(P_T(u, v))\right] \;=\; \left|\, i_e^{(u\to v)} \,\right|.$$

In particular, for a single edge $e = \{a, b\}$, $\Pr[e \in T] = R_{\mathrm{eff}}(a, b)$.

Theorem A.8 (Commute-time and effective resistance (Lovász, 1993))**.** *For a simple random walk on G with $m = |E|$, the commute time satisfies*

$$C_{uv} \;:=\; H(u, v) + H(v, u) \;=\; 2m\, R_{\mathrm{eff}}(u, v).$$

Theorem A.9 (UST distance bound and expected distortion)**.** *Let G be an unweighted connected graph and let T be a uniform spanning tree of G. Then for every $u, v \in V$,*

$$\mathbb{E}\big[d_T(u, v)\big] \;\le\; \sqrt{\frac{H(u, v) + H(v, u)}{2}} \;=\; \sqrt{\frac{C_{uv}}{2}}.$$

Consequently, the expected UST distortion

$$D_{\mathrm{UST}} \;:=\; \max_{u\neq v} \frac{\mathbb{E}[d_T(u, v)]}{d_G(u, v)}$$

obeys

$$D_{\mathrm{UST}} \;\le\; \max_{u\neq v} \frac{\sqrt{C_{uv}/2}}{d_G(u, v)}.$$

Proof. Fix $u, v \in V$. Since T is a tree, there is a unique u–v path $P_T(u, v)$, and

$$d_T(u, v) \;=\; \sum_{e\in E} \mathbf{1}\{\, e \in E(P_T(u, v))\,\}.$$

Taking expectations and using the transfer–current fact,

$$\mathbb{E}[d_T(u, v)] = \sum_{e\in E} \Pr[e \in E(P_T(u, v))] = \sum_{e\in E} \left|\, i_e^{(u\to v)} \,\right|.$$

Apply Cauchy–Schwarz:

$$\sum_{e\in E} \left|\, i_e^{(u\to v)} \,\right| \;\le\; \sqrt{\Big(\sum_{e\in E} 1\Big)\Big(\sum_{e\in E} (i_e^{(u\to v)})^2\Big)} \;=\; \sqrt{m\, R_{\mathrm{eff}}(u, v)}\,.$$

Finally, use $C_{uv} = 2m\, R_{\mathrm{eff}}(u, v)$ to obtain $\mathbb{E}[d_T(u, v)] \le \sqrt{C_{uv}/2}$. Dividing by $d_G(u, v)$ and taking the maximum over $u \neq v$ yields the stated distortion bound. □

A.5 Coverage and Expressivity Guarantees via MST Canonicalization

Lemma A.10 (Logarithmic spanning–tree cover)**.** *Let $G = (V, E)$ be a graph with $m = |E|$ and arboricity $\Upsilon(G)$. Fix any node labeler π_V with $\tau > \max_e \pi_E^{(0)}(e) - \min_e \pi_E^{(0)}(e)$. Let Algorithm 1 produce trees $\mathcal{T} = \{T^{(k)}\}_{k=0}^{K-1}$. Then, after $K \geq \Upsilon(G) \ln m$ iterations, the union of the MSTs covers all edges:*

$$\bigcup_{k=0}^{K-1} E\Big(T^{(k)}\Big) \;=\; E.$$

Proof. Let $\Upsilon = \Upsilon(G)$. By the definition of arboricity, there exists a partition of the edges into Υ forests, $E = \dot{\bigcup}_{j=1}^{\Upsilon} E(F_j)$. For each j, fix a (witness) spanning tree $\widetilde{T}_j \supseteq F_j$. Let $U_k \subseteq E$ be the set of *uncovered* edges after k rounds and set $u_k := |U_k|$. For each j, define $u_{k,j} := |U_k \cap E(F_j)|$, so that $\sum_{j=1}^{\Upsilon} u_{k,j} = u_k$. By the pigeonhole principle, there exists $j^\star$ with $u_{k,j^\star} \geq u_k/\Upsilon$. Choose $\tau > \max_e b_e - \min_e b_e$. Then, for any $k \geq 1$,

$$\min_{e \in E \setminus U_k} \pi_E^{(k)}(e) \;\geq\; \max_{e \in U_k} \pi_E^{(k)}(e),$$

i.e., every seen edge is strictly more expensive than every unseen edge. Hence, minimizing total weight over spanning trees is equivalent to *maximizing* the number of unseen edges $|T \cap U_k|$ (any exchange that replaces a seen edge by an unseen edge strictly reduces cost). Since $\widetilde{T}_{j^\star}$ contains all edges of $F_{j^\star}$, it achieves $|\widetilde{T}_{j^\star} \cap U_k| = u_{k,j^\star}$. Therefore the MST $T^{(k)}$ satisfies

$$|T^{(k)} \cap U_k| \;\geq\; u_{k,j^\star} \;\geq\; \frac{u_k}{\Upsilon}.$$

Consequently,

$$u_{k+1} \;=\; u_k - |T^{(k)} \cap U_k| \;\leq\; \Big(1 - \frac{1}{\Upsilon}\Big) u_k.$$

Iterating yields $u_K \leq u_0\big(1 - \frac{1}{\Upsilon}\big)^K \leq m\, e^{-K/\Upsilon}$. Choosing $K \geq \Upsilon \ln m$ gives $u_K < 1$, hence $U_K = \varnothing$ and $\bigcup_{k=0}^{K-1} E(T^{(k)}) = E$, as claimed. □

Lemma A.11 (f_{CanTree} is strictly more expressive than f_{MPNN} and f_{CanSeq})**.** *Suppose K satisfies Lemma 5.3. Let $\pi_V \simeq f_{\mathrm{MPNN}}$. Then, $\pi_V \prec f_{\mathrm{CanTree}}$ and hence $f_{\mathrm{CanSeq}} \prec f_{\mathrm{CanTree}}$.*

Proof. We first show $f_{\mathrm{CanSeq}} \preceq f_{\mathrm{CanTree}}$ and then prove strictness, i.e., $f_{\mathrm{CanSeq}} \prec f_{\mathrm{CanTree}}$.

Step 1: $f_{\mathrm{CanSeq}} \preceq f_{\mathrm{CanTree}}$. Fix $G = (V, E, X)$ and any spanning tree T of G. For a node $v \in V$, let $C_T(v)$ be its children in T, $p_T(v)$ its parent, and $N_G(v)$ its 1–hop neighbors in G. Consider a tree-aware per-node update of the form

$$\mathbf{h}_v^{(T)} \;=\; f_{\mathrm{agg}}\Big(\{\!\{\mathbf{x}_u : u \in C_T(v)\}\!\} \;\uplus\; \{\!\{\mathbf{x}_{p_T(v)}\}\!\} \;\uplus\; \{\!\{\mathbf{x}_u : u \in N_G(v) \setminus (C_T(v) \cup \{p_T(v)\})\}\!\}\Big),$$

where f_{agg} is an *injective*, permutation-invariant function over multisets (e.g., a DeepSets). Note that $\mathbf{h}_v^{(T)}$ can be recovered by a TreeMPNN layer setting $\mathbf{h}_c = \mathbf{0}$ in f_{tree} for all children c. By construction, the multiset union inside f_{agg} simplifies to the full neighbor multiset:

$$\{\!\{\mathbf{x}_u : u \in C_T(v)\}\!\} \uplus \{\!\{\mathbf{x}_{p_T(v)}\}\!\} \uplus \{\!\{\mathbf{x}_u : u \in N_G(v) \setminus (C_T(v) \cup \{p_T(v)\})\}\!\} \;=\; \{\!\{\mathbf{x}_u : u \in N_G(v)\}\!\}.$$

Hence

$$\mathbf{h}_v^{(T)} \;=\; f_{\mathrm{agg}}\big(\{\!\{\mathbf{x}_u : u \in N_G(v)\}\!\}\big),$$

which emulates a GIN/1-WL update at v. Since f_{agg} is injective on multisets, this matches the expressivity of a GIN step; in particular, any single-sequence model bounded by the same node labeler π_V (and hence by GIN/1-WL) can be simulated by a suitable choice of f_{agg} within f_{CanTree}. Thus $f_{\mathrm{CanSeq}} \preceq f_{\mathrm{CanTree}}$.

Step 2: Strictness. We construct G, H with $G \not\cong H$ such that $f_{\mathrm{CanSeq}}(G) = f_{\mathrm{CanSeq}}(H)$ but $f_{\mathrm{CanTree}}(G) \neq f_{\mathrm{CanTree}}(H)$. Let $G = C_n \sqcup C_n$ (two disjoint n-cycles; $2n$ vertices, 2 components) and $H = C_{2n}$ (one $2n$-cycle). Both are 2-regular, so 1-WL/GIN (and thus any $\pi_V \simeq f_{\mathrm{CanSeq}}$ that is 1-WL-bounded) yields identical color multisets; hence $f_{\mathrm{CanSeq}}(G) = f_{\mathrm{CanSeq}}(H)$.

Now compare the multisets of spanning trees/forests:

$$\mathcal{T}(G) = \{\!\{\, P_n \sqcup P_n \,\}\!\}^{\,n^2}, \qquad \mathcal{T}(H) = \{\!\{\, P_{2n} \,\}\!\}^{\,2n},$$

since each cycle C_m has exactly m spanning trees, all paths P_m. Thus every spanning forest of G is a disjoint union of two n-paths (multiplicity n^2), whereas every spanning tree of H is a single $2n$-path (multiplicity $2n$). Choose a per-tree readout $\rho(T)$ that is injective on tree isomorphism types (e.g., map $P_n \sqcup P_n$ and P_{2n} to distinct representations), and a global pooling over spanning trees that is an injective multiset aggregator (e.g., DeepSets). Then

$$f_{\mathrm{CanTree}}(G) \neq f_{\mathrm{CanTree}}(H).$$

Combining Steps 1 and 2 shows $f_{\mathrm{CanSeq}} \prec f_{\mathrm{CanTree}}$. □

To prove universality, we equip CTNNs with anonymous labels, a strategy used in (Wang and Cho, 2024; Kim et al., 2025). Intuitively, anonymous labels allow injectivity for tree covers on graphs, while the overall construction remains isomorphism-invariant in distribution.

Definition A.12 (Anonymous labels)**.** Given a graph $G = (V, E, X)$, draw i.i.d. tags $z_v \sim \mathrm{Unif}[0, 1]$ for $v \in V$ (and set $\tilde{\mathbf{x}}_v := (\mathbf{x}_v, z_v)$). Use the *same* tag assignment $z = \{z_v\}_{v \in V}$ for all trees in the CTNN cover of G. This yields a labeled cover $\mathcal{T}_z(G) = \{T^{(k)}(G), z\}_{k=0}^{K-1}$. Because the tag distribution is i.i.d. and continuous, the construction remains permutation-invariant *in distribution*.

Lemma A.13 (Labeled covers are separating a.s.)**.** *Let K satisfy Lemma 5.3 so that $\bigcup_k E(T^{(k)}(G)) = E(G)$. With probability 1 over the draw of z (ties occur with prob. 0), the map $G \mapsto \mathcal{T}_z(G)$ is separating up to isomorphism on any finite class $\mathcal{G}$.*

Proof. Almost surely, all node tags are distinct. Let $\sigma_z : V \to \{1, \ldots, |V|\}$ be the order on vertices induced by sorting tags (i.e., $z_{\sigma_z^{-1}(1)} < \cdots < z_{\sigma_z^{-1}(|V|)}$). Form the edge list

$$\mathrm{Canon}(G; z) \;:=\; \{\!\{\, \{\, \sigma_z(u),\, \sigma_z(v) \,\} \;:\; \{u, v\} \in \textstyle\bigcup_k E\big(T^{(k)}(G)\big) \,\}\!\}$$

which is isomorphic to the edge list of G because the cover union is $E(G)$. Hence if $G \not\cong H$ then $\mathrm{Canon}(G; z) \neq \mathrm{Canon}(H; z)$, i.e., the cover is separating. For a finite $\mathcal{G}$ this holds simultaneously for all $G \in \mathcal{G}$ with probability 1. □

Theorem A.14 (CTNN Universality with anonymous labels)**.** *Let $\mathcal{G}$ be a finite class of graphs. Assume: (i) K satisfies Lemma 5.3; (ii) the per-tree encoder f_{tree} is universal on labeled trees and the multiset aggregator f_{agg} is universal on finite multisets. Then for any continuous permutation-invariant graph function $f : \mathcal{G} \to \mathbb{R}$ and any $\varepsilon > 0$, there exists a CTNN such that, for every draw of anonymous labels z used consistently across the cover,*

$$\sup_{G \in \mathcal{G}} \big|\, f_{\mathrm{CTNN}}(G; z) - f(G) \,\big| \;\leq\; \varepsilon.$$

Proof. By Lemma 5.3 the cover edges union to $E(G)$; by Lemma A.13, for every z the labeled cover is separating on $\mathcal{G}$ via $\mathrm{Canon}(G; z)$. Thus the invariant target f factors as

$$f(G) \;=\; F\big(\, \{\!\{\, \rho(T) : T \in \mathcal{T}_z(G) \,\}\!\} \,\big)$$

for some continuous permutation-invariant F on multisets of tree representations and some ρ on labeled trees (e.g., any encoding that reproduces $\mathrm{Canon}(G; z)$). By universality, f_{tree} can approximate ρ arbitrarily well on the finite support of observed labeled trees, and f_{agg} can approximate F on finite multisets of tree representations. Because $\mathcal{G}$ is finite, the composition error is uniformly bounded by ε after suitable parameter choices. □

B Additional Model Details

B.1 Algorithms for Canonical Spanning-Tree Covers

Algorithm 1 outlines our procedure for building a K–tree canonical cover. We initialize the edge labeler $\pi_E^{(0)}$ using the negative sum of endpoint degrees for each edge, which prioritizes edges incident to high-degree nodes. For rounds $t = 0, \ldots, K-1$, we construct an MST with respect to the current edge weights using Kruskal's algorithm (Algorithm 2): edges are stably sorted (random tie–breaking) and scanned in nondecreasing order, adding an edge if it does not create a cycle. After forming the tree $T^{(t)}$, we update the labeler to encourage coverage in subsequent rounds: edges *not* selected in $T^{(t)}$ receive an additive penalty (controlled by τ) that increases their priority in the next MST. Finally, we choose a canonical root as the tree center via the standard two–BFS routine (Algorithm 3): one BFS finds an endpoint of a longest path, and a second BFS from that endpoint finds the opposite endpoint; the center(s) of this path serve as the root. This procedure runs in $O(m \log n)$ per round for the MST and $O(n)$ for root selection, and returns the K trees with their canonical roots.

Algorithm 1: BuildCanonicalTreeCover: iterative MST cover with root selection

Input: Graph $G = (V, E)$; node labeler $\pi_V : V \to \mathbb{R}$; iterations K; step $\tau > 0$; tiny $\varepsilon > 0$
Output: Tree cover $\mathcal{T} = \{T^{(k)}\}_{k=0}^{K'-1}$ and roots $\mathcal{R} = \{r^{(k)}\}_{k=0}^{K'-1}$

(Initialization)
  foreach $e = \{u, v\} \in E$ **do**
    $w_e^{(0)} \leftarrow -\big(\pi_V(u) + \pi_V(v)\big)$ `/* base edge weights` $\pi_E^{(0)}$ `*/`
  Draw i.i.d. tie–breakers $\zeta : E \to (0, 1)$ (fixed across rounds)
  $S_0 \leftarrow \varnothing$ `/* covered edges so far */`
  $\mathcal{T} \leftarrow \varnothing, \mathcal{R} \leftarrow \varnothing$
for $k = 0$ **to** $K - 1$ **do**
  `// 1) Minimum spanning tree with lexicographic keys`
  $T^{(k)} \leftarrow \textsc{KruskalMST}\big(G,\ e \mapsto \big(w_e^{(k)}, \zeta(e)\big)\big)$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{T^{(k)}\}$; $S_{k+1} \leftarrow S_k \cup E(T^{(k)})$
  `// 2) Canonical root: tree center via two BFS passes`
  $r^{(k)} \leftarrow \textsc{TreeCenter}\big(T^{(k)}, \pi_V\big)$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{r^{(k)}\}$
  `// 3) Edge-weight update (penalize edges just used)`
  foreach $e \in E$ **do**
    if $e \in E(T^{(k)})$ **then** $w_e^{(k+1)} \leftarrow w_e^{(k)} + \tau$
    else $w_e^{(k+1)} \leftarrow w_e^{(k)}$
  if $|S_{k+1}| = |E|$ **then break**
return $(\mathcal{T}, \mathcal{R})$

Algorithm 2: KruskalMST with exchangeable tie–breakers

Input: Undirected graph $G = (V, E)$; base edge weights $w : E \to \mathbb{R}$; tie–breakers $\zeta : E \to (0, 1)$ i.i.d.
Output: A spanning tree T of G
$T \leftarrow \varnothing$; initialize disjoint–set D with $\textsc{MakeSet}(v)$ for all $v \in V$
if $E = \varnothing$ **then return** T
`// I.i.d. continuous tie-breakers ζ m ake keys distinct w.p. 1.`
Sort edges E by nondecreasing key $k(e) := \big(w(e), \zeta(e)\big)$
`// Union-find tracks connected components of the partial forest.`
for $e = \{u, v\}$ *in E in the above order* **do**
  if $\textsc{Find}_D(u) \neq \textsc{Find}_D(v)$ **then**
    $T \leftarrow T \cup \{e\}$; $\textsc{Union}_D(u, v)$
    if $|T| = |V| - 1$ **then return** T
return T

Algorithm 3: TREECENTER: root selection by two BFS passes

Input: Tree $T = (V_T, E_T)$; tie–breaker ranking on vertices (e.g., (π_V, ID))
Output: Root $r \in V_T$ (a center of T)
Pick canonical start $s \in V_T$ (minimizing the tie–breaker);
$u \leftarrow \text{BFS\_FARTHEST}(T, s)$; $v \leftarrow \text{BFS\_FARTHEST}(T, u)$;
$P \leftarrow$ unique path from u to v in T;
if $|P|$ *odd* **then** $r \leftarrow$ middle vertex of P
else $r \leftarrow$ the nearer of the two middle vertices under the tie–breaker
return r

B.2 ALGORITHMS FOR RECURRENT TREE NEURAL NETWORKS

We provide discussion and implementation details of the recurrent tree neural network (Algorithm 4).

Algorithm 4: BITREELSTMFORWARD: Bidirectional child–sum Tree-LSTM forward pass

Input : $x \in \mathbb{R}^{N \times D}$ node features; rooted tree $T = (V, E_T, r)$ in COO form with $(\mathsf{row}, \mathsf{col}) = (\text{parent}, \text{child})$; arrays $\mathsf{parent}[v]$, $\mathsf{depth}[v] \in \{0, \dots, L\}$
Output : $h \in \mathbb{R}^{N \times 2H}$: concat. of bottom-up and top-down hidden states
Parameters: bottom-up $W_{iou} \in \mathbb{R}^{D \times 3H}, U_{iou} \in \mathbb{R}^{H \times 3H}, W_f \in \mathbb{R}^{D \times H}, U_f \in \mathbb{R}^{H \times H}$; top-down $W^{\downarrow}_{iou}, U^{\downarrow}_{iou}, W^{\downarrow}_f, U^{\downarrow}_f$ of matching shapes.
Init: $h^{\uparrow} \leftarrow 0_{N \times H}$, $c^{\uparrow} \leftarrow 0_{N \times H}$, $h^{\downarrow} \leftarrow 0_{N \times H}$, $c^{\downarrow} \leftarrow 0_{N \times H}$.
Bucket nodes by depth: $\mathcal{V}_\ell \leftarrow \{v \in V : \mathsf{depth}[v] = \ell\}$ for $\ell = 0, \dots, L$.

/ Bottom-up pass (children → parent): process parents from deepest to root. */*
for $\ell = L$ **to** 0 **do**
  $P \leftarrow \mathcal{V}_\ell$ `// parents at depth` ℓ
  $E_\ell \leftarrow \{(u \leftarrow v) \in E_T : u \in P\}$ `// edges with parent at depth` ℓ
  `// Aggregate child states with` scatter_add `(child-sum` $f_{\text{agg}} = \sum$`)`
  $h_{\text{sum}} \leftarrow 0_{N \times H}$;
  $h_{\text{sum}}[u] \mathrel{+}= \sum_{(u \leftarrow v) \in E_\ell} h^{\uparrow}[v]$
  `// Per-edge forgets and summed transformed cell contributions`
  $f_{uv} \leftarrow \sigma\big(W_f x[u] + U_f h^{\uparrow}[v]\big)$ for $(u \leftarrow v) \in E_\ell$
  $c_{\sim} \leftarrow 0_{N \times H}$;
  $c_{\sim}[u] \mathrel{+}= \sum_{(u \leftarrow v) \in E_\ell} f_{uv} \odot c^{\uparrow}[v]$
  `// Node-level gates and updates for all` $u \in P$
  for $u \in P$ **do**
    $[i, o, \tilde{u}] \leftarrow \text{split}_3\big(W_{iou} x[u] + U_{iou} h_{\text{sum}}[u]\big)$;
    $i \leftarrow \sigma(i),\ o \leftarrow \sigma(o),\ \tilde{u} \leftarrow \tanh(\tilde{u})$;
    $c^{\uparrow}[u] \leftarrow i \odot \tilde{u} + c_{\sim}[u]$;
    $h^{\uparrow}[u] \leftarrow o \odot \tanh\big(c^{\uparrow}[u]\big)$

/ Top-down pass (parent → children): propagate from root to leaves. */*
for $\ell = 1$ **to** L **do**
  $V \leftarrow \mathcal{V}_\ell$ `// children at depth` ℓ
  for $v \in V$ **do**
    $p \leftarrow \mathsf{parent}[v]$ `// unique parent`
    $[i, o, \tilde{u}] \leftarrow \text{split}_3\big(W^{\downarrow}_{iou} x[v] + U^{\downarrow}_{iou} h^{\downarrow}[p]\big)$;
    $i \leftarrow \sigma(i),\ o \leftarrow \sigma(o),\ \tilde{u} \leftarrow \tanh(\tilde{u})$;
    $f \leftarrow \sigma\big(W^{\downarrow}_f x[v] + U^{\downarrow}_f h^{\downarrow}[p]\big)$;
    $c^{\downarrow}[v] \leftarrow i \odot \tilde{u} + f \odot c^{\downarrow}[p]$;
    $h^{\downarrow}[v] \leftarrow o \odot \tanh\big(c^{\downarrow}[v]\big)$

return $h \leftarrow \text{concat}\big(h^{\uparrow},\ h^{\downarrow}\big)$ `//` $[N, 2H]$

Parallel bottom–up / top–down passes. For each canonical tree we precompute three arrays: `edge_index_tree` (row=parent, col=child), `parent`[v] (unique parent), and `depth`[v] (distance to root). The bidirectional forward pass runs in two levelwise sweeps that are parallel across all nodes at the same depth. In the bottom–up pass (leaves $\rightarrow$ root), we bucket nodes by `depth` and use `scatter_add` to implement the child–sum aggregator and edgewise forget contributions in a single batched operation over all edges whose parent is at the current depth. In the top–down pass (root $\rightarrow$ leaves), each node reads its parent's state via `index_select` and applies the same batched gating. This organization avoids Python loops over edges and exploits segmented reductions on the GPU; it only iterates over depth buckets. The forward pass can be efficiently batched across trees by treating the full batch as a collection of disjoint graphs, whose edges are stored in COO format.

Tree encoder. We implement a *bidirectional child–sum Tree–LSTM* layer with two parameter sets (children$\rightarrow$parent and parent$\rightarrow$children). Each direction computes input/output/update gates via a single linear projection that yields $3H$ channels per node and applies elementwise nonlinearity; edgewise forget gates are computed in parallel across all incident edges at a depth. The output feature per node is the concatenation of the two hidden states, making the layer stackable (we use residual/normalization as in standard practice when beneficial).

Complexity and memory. Each direction runs in $O(|V|)$ time and $O(|V|)$ memory for node states. For a cover of K trees, the bidirectional layer cost is $O(K|V|)$. In practice we batch trees along the sample dimension, so the work parallelizes across graphs and across trees.

C Design Space of Canonical Approaches

As shown in Table 4, we organize canonicalization methods along six axes: (i) whether they rely on domain knowledge; (ii) the node labeler π_V; (iii) the edge labeler π_E; (iv) the canonicalizer (ordering/selection rule); (v) the induced representation (vector/sequence/tree); and (vi) whether they use a set of canonical elements per graph (vs. a single representative), together with the downstream encoder. Table 4 situates common pipelines: domain-driven approaches (Fingerprint, SMILES, Primary Seq.) produce a single canonical vector or sequence; graph-agnostic orderings (DGCNN/SortPooling, RCM) also yield a single sequence from graph-derived ranks. DFS Set employs multiple sequences but remains sequence-based. In contrast, CTNN is the only approach that (a) uses an edge labeler to drive a coverage-aware canonicalization and (b) represents each graph by a tree cover, a set of canonical trees obtaining full coverage. This design is domain-agnostic, preserves distances more faithfully than sequences, and increases expressivity by operating on a set rather than a single representative.

Table 4: Design space for graph canonicalization. "Set" indicates whether the method uses multiple canonical elements per graph (e.g., a cover of trees) rather than a single canonicalization. "Domain Knowledge" indicates reliance on domain-specific information (e.g., chemistry rules).

Approach	Domain Knowledge	Node Labeler	Edge Labeler	Canonicalizer	Representation	Set	Backbone
Fingerprint	Yes	NA	NA	Handcrafted chemical descriptors	Vector	No	MLP
SMILES	Yes	Atom canonical ranks	NA	Canonical SMILES algorithm	Sequence	No	RNN/TRSF
Primary Seq.	Yes	NA	NA	Identity	Sequence	No	RNN
DGCNN	No	MPNN	No	Differentiable sort (SortPooling)	Sequence	No	1D CNN
RCM	No	Degree	No	Reverse Cuthill–McKee ordering	Sequence	No	RNN
DFS Set	No	Degree	No	Sorted DFS	Sequence Set	Yes	RNN
CTNN (full)	No	Degree	Coverage-aware	Minimum Spanning Tree	Tree Cover	Yes	TreeMPNN

D Additional Experimental Details

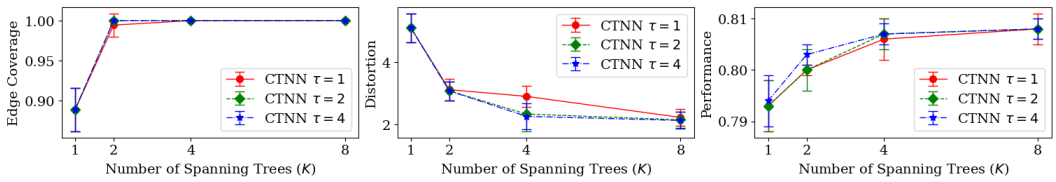
Training and Hyperparameter Selection. All models are trained by minimizing the binary cross-entropy loss on binary classification tasks and the negative log-likelihood loss on multiclass classification tasks. Training is performed for a maximum of 200 epochs with early stopping patience set to 15 epochs based on validation performance. The best-performing model on the validation set is selected for evaluation on the test set. We perform a grid search over the following hyperparameters for models where applicable:

- Number of layers: {1, 2, 3, 4}
- Learning rate: {0.05, 0.01, 0.005, 0.001}
- Batch size: {64, 128, 512, 1024}
- Hidden dimension: {64, 128, 256}
- Global pooling: {`mean`, `sum`, `max`}
- Sequence model: {`GRU`, `LSTM`, `Transformer`}
- Number of sequences/trees K: {1, 4, 8}
- Coverage penalty τ: {1, 2, 4}

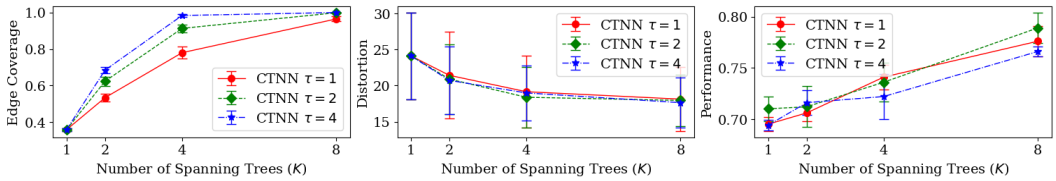
All models are optimized using the Adam optimizer.

E Extended Results

We include additional analyses on (i) sensitivity to the node labeler π_V, (ii) the coverage penalty τ, and (iii) the number of trees K. We find CTNN to be robust to the choice of π_V and τ, while increasing K consistently increases coverage, reduces distortion, and improves performance. We further conduct a sensitivity analyses to the choice of sequence models for SMILES, comparing performance when f_{seq} is a LSTM or transformer. Here, we find recurrence outperforms attention aligning with recent findings in RWNN studies. We lastly report preprocessing runtimes, confirming that CTNN's preprocessing is efficient and negligible with respect to training times for all datasets.



(a) CTNN coverage, distortion, and performance across τ and K for PCBA-1030 molecular dataset.



(b) CTNN coverage, distortion, and performance across τ and K for GO BIO protein dataset.

Figure 6: Sensitivity of CTNN to the number of trees K and coverage penalty τ on PCBA-1030 (top) and GO BIO (bottom). Coverage rises rapidly with K; distortion decreases monotonically with K; and performance improves. Trends are similar across τ, indicating robustness, with larger τ yielding slightly higher coverage at fixed K on proteins. Error bars denote standard deviation over samples.

Table 5: Median (min, max) test AUC on molecular datasets. We test CTNN with three node labelers π_V (Degree, Closeness Centrality, 1–WL). CTNN is robust to π_V and typically outperforms baselines; CC or 1–WL often yields the best scores, while Degree serves as a competitive, low-cost default.

		Small MoleculeNet Molecular Benchmarks (AUC ↑)					
		CLINTOX	**SIDER**	**BACE**	**TOXCAST**	**BBBP**	**TOX21**
# Graphs		1.5K	1.5K	1.5K	1.7K	2K	6K
Avg. $\lvert V\rvert$		26.1	33.6	34.1	17.1	23.9	16.4
Avg. $\lvert E\rvert$	**Node Labeler** π_V	28.0	35.4	36.9	17.5	26.0	16.9
GCN	NA	62.4 (56.9, 74.7)	64.2 (62.4, 70.3)	59.2 (53.9, 64.3)	59.8 (52.4, 66.7)	73.9 (68.9, 81.4)	67.5 (63.1, 71.9)
GAT	NA	62.1 (55.8, 65.9)	63.6 (61.0, 67.1)	60.8 (52.0, 75.1)	60.6 (56.9, 69.1)	77.5 (74.1, 82.8)	68.2 (65.0, 72.5)
GIN	NA	59.7 (54.1, 72.4)	66.5 (64.0, 69.9)	59.9 (51.4, 71.8)	55.7 (38.8, 60.8)	75.3 (49.4, 85.3)	66.9 (64.6, 73.4)
GT	NA	57.1 (46.5, 73.5)	64.3 (57.9, 69.0)	67.1 (57.6, 75.7)	68.4 (60.7, 74.1)	75.8 (62.6, 84.0)	67.8 (64.8, 73.9)
SMILES	Atom Ranks	62.5 (45.7, 68.6)	61.5 (57.6, 66.4)	76.5 (68.4, 80.3)	65.7 (58.3, 70.7)	71.9 (65.5, 75.3)	71.3 (66.4, 73.8)
DGCNN	GCN	60.1 (27.6, 69.6)	**65.5 (62.9, 68.9)**	67.2 (64.1, 74.8)	71.3 (67.5, 75.6)	75.0 (42.8, 86.4)	75.2 (71.4, 77.2)
RCM	Degree	70.7 (48.6, 87.0)	63.1 (57.3, 67.7)	76.3 (73.3, 81.2)	70.9 (68.4, 74.9)	84.3 (75.1, 89.1)	76.0 (72.4, 79.9)
CTNN	Degree	82.7 (56.8, 89.9)	64.1 (62.8, 67.3)	82.4 (79.7, 86.5)	75.7 (70.0, 78.0)	**88.4 (83.7, 91.6)**	80.9 (79.6, 84.9)
CTNN	CC	84.3 (80.1, 92.0)	64.8 (61.0, 68.9)	82.7 (75.0, 87.4)	76.2 (71.5, 79.1)	88.3 (84.1, 92.3)	81.1 (78.9, 82.8)
CTNN	1-WL	**84.8 (76.4, 91.0)**	65.1 (63.3, 68.5)	**83.5 (80.5, 86.7)**	**76.5 (71.4, 80.8)**	86.8 (82.6, 92.0)	**81.4 (78.7, 84.5)**

E.1 Sensitivity Analyses

Sensitivity to K. We vary the number of trees K on PCBA-1030 (Figure 6a) and GO BIO (Figure 6b). Edge coverage increases rapidly with K, reaching full coverage by K=4 on PCBA-1030 and by K=8 on GO BIO, consistent with the theory that only a small number of trees is needed on sparse graphs. Distortion decreases monotonically as K grows, indicating that additional trees better preserve original graph distances. Task performance likewise improves with K, showing the practical value of the canonical tree cover.

Sensitivity to τ. We test coverage penalty $\tau = \{1, 2, 4\}$ on the same datasets (Figure 6). Across benchmarks, coverage, distortion, and accuracy follow similar trends for different τ, indicating robustness to the choice of penalizer. For proteins, larger τ yields higher coverage at a fixed K, as heavier penalties bias the MST toward previously unseen edges. Overall, CTNN's behavior is stable across τ, while K primarily controls the coverage–distortion–accuracy tradeoff.

Sensitivity to π_V. We assess CTNN's dependence on the node labeler π_V using small-molecule benchmarks and include MPNNs and Graph Transformer (GT) as invariant architectural baselines as well as canonical sequence baselines (Table 5). We compare three labelers: degree, closeness

centrality (CC), and 1-Weisfeiler–Lehman (1-WL). Overall, CTNN is robust to the choice of π_V: CC or 1-WL typically achieve the best scores, while degree is slightly behind but competitive across datasets. This reflects a natural trade-off: more informative labelers can yield slight gains at higher preprocessing cost. Concretely, degree runs in $O(m)$, 1-WL in $O(t\,m)$ for t refinement rounds, and CC in $O(nm)$ via all-pairs BFS. In our main experiments, we default to the inexpensive degree labeler for efficiency, noting that CC or 1-WL can be used when improvements justify the added cost.

Sensitivity to f_{seq}. We evaluate the sensitivity of the sequence encoder f_{seq} by comparing attention (Transformer) and recurrence (LSTM) on canonical SMILES, and include a graph Transformer (GT) that operates directly on molecular graphs (Table 6). We report the analysis on the molecular benchmarks, whereas training analogous models on the larger, denser protein graphs did not converge within 24 hours. Results demonstrate that attention and recurrence perform comparably on SMILES and are similar to the GT baseline. In all cases, however, these models underperform in comparison to CTNNs, which maintain a clear performance advantage. This occurs since attention relies on the sequential positional encoding and recurrence relies on the linear ordering, which both incur distortion and fail to capture graph distances

Table 6: Transformers and LSTMs achieve comparable AUC across PCBA datasets, indicating that attention and recurrence perform similarly on the canonical sequence. GTs also perform comparably to both. Values are median (min, max) over splits. CTNNs outperform all models.

		Molecular Benchmarks (AUC ↑)			
Approach	**Backbone**	**PCBA-1030**	**PCBA-1458**	**PCBA-4467**	**PCBA-5297**
GT	Transformer	68.1 (67.9, 68.6)	81.2 (81.0, 81.5)	78.9 (77.8, 79.9)	87.7 (87.6, 88.2)
SMILES	Transformer	71.9 (71.5, 72.3)	84.4 (83.7, 84.5)	82.4 (81.5, 82.5)	88.9 (88.3, 89.4)
SMILES	LSTM	71.9 (71.2, 72.5)	84.9 (84.5, 85.9)	81.1 (80.0, 81.4)	90.2 (90.0, 90.3)
CTNN	TreeMPNN	**80.6** (80.3, 81.2)	**89.1** (88.0, 89.9)	**86.8** (86.5, 87.4)	**94.6** (94.2, 94.9)

E.2 Runtime Analyses

We report per-graph preprocessing time for CTNNs when constructing $K{=}8$ trees with a degree-based node labeler $\pi_V(v){=}\deg(v)$ (Table 7). On molecular graphs the cost is in the milliseconds, and on protein graphs it is on the order of tenths of a second. In practice, this preprocessing is parallelizable across graphs and is computed once and reused over all training epochs, making it a small fraction of end-to-end training time. Overall, CTNN preprocessing is efficient for the datasets considered.

Table 7: CTNN preprocessing time per graph to construct $K{=}8$ canonical spanning trees using degree labeler ($\pi_V(v) = \deg(v)$). We report dataset sizes and average graph statistics; times (seconds) are averaged over all graphs. Overall, CTNN preprocessing is efficient across all datasets.

Dataset	**# Graphs**	**Avg.** $\lvert V\rvert$	**Avg.** $\lvert E\rvert$	**Avg.** $\deg(v)$	**Time (sec)**
PCBA-1030	160K	24.3	26.2	2.2	0.004
GO MOL	32K	250.1	687.5	5.4	0.093
PFAM	25K	251.3	691.3	5.4	0.121