Towards Distillation Guarantees under Algorithmic Alignment for Combinatorial Optimization

Thien Le
SEAS, Harvard University
Cambridge, MA
thien_le@seas.harvard.edu

Melanie Weber
SEAS, Harvard University
Cambridge, MA
mweber@seas.harvard.edu

Abstract

Distillation transfers knowledge from a large model trained on broad data to a smaller, more efficient model suitable for deployment. In structured prediction settings, prior knowledge about the task can guide the choice of a target architecture that is algorithmically aligned with the underlying problem. Building on recent learning-theoretic analyses of decision-tree (DT) distillation (Boix-Adsera, 2024), we study when distillation succeeds for combinatorial optimization tasks. We focus on the case where the target model is a graph neural network whose architecture is aligned with a dynamic programming (DP) algorithm for the task. Assuming that the source model is sufficiently rich, formalized through the linear representation hypothesis (LRH) (Elhage et al., 2022; Park et al., 2024), we show that the distillation problem can be solved efficiently in the complexity parameters of the DP transition function, represented as a DT. Our results provide a rigorous sufficient condition for successful distillation in the flavour of algorithmic alignment.

1 Introduction

Modern machine learning often benefits from incorporating useful structure into the learner, as demonstrated by a wide range of models, from convolutional architectures in vision to graph neural networks (GNNs) for relational data (Krizhevsky et al., 2012; LeCun et al., 2015; Goodfellow et al., 2016; Gilmer et al., 2017; Kipf and Welling, 2017). At the same time, the success of general-purpose models has made it natural to ask whether such structure can instead be learned from data and subsequently transferred to smaller models (Bachmann et al., 2023; Brehmer et al., 2025; Hinton et al., 2015; Jiao et al., 2020). The "learn first, distill later" paradigm offers a middle ground: train a large source model, then distill its knowledge into a target model whose architecture embodies the desired inductive bias.

In this work, we ask when the knowledge learned by a large neural network on graph algorithmic tasks can be distilled into a smaller graph model whose architecture is aligned with the underlying algorithm. We answer this question affirmatively for a class of local-iteration graph algorithms, a dynamic-programming abstraction that encompasses simple reachability-type computations. In particular, we show that if the learned source representation linearly exposes the elementary components of the algorithm, then PAC-distillation (Boix-Adsera, 2024) into an algorithmically aligned GNN is tractable in the complexity parameters of the local transition rule, under the restricted parameter regime made explicit below. This provides a rigorous sense in which the target architecture not only compresses the source model, but also leverages the algorithmic structure of the task.

Algorithm 1 Local-iteration algorithm $\mathcal{A}^l[g]$

Input Initialization vector INIT, graph $G = ([n], E)$
Output $\{0, 1\}$ classification

```
for all v ∈ [n] do
    h_{v,0} ← INIT(v)
end for
for t = 1 to l do
    for all v ∈ [n] do
        h_{v,t} ← g((h_{u,t−1})_{u∈N(v)}, G, v)        ▷ h_{v,t} denotes state of v after iteration t
    end for
end for
return h_{n,l}
```

1.1 Contribution of this paper

We focus on the setting in which the ground truth is computed by a *local-iteration algorithm* (Algorithm 1). This is a dynamic programming (DP) algorithm whose states are indexed by pairs $(t, v) \in [l] \times V$, where t denotes an iteration of message-passing and v is a vertex of the input graph $G = (V = [n], E)$. We assume that the DP transition function is represented by a small decision tree of depth r, for instance, in the DP that solves graph reachability.

The central question that we address in this paper is as follows:

> Suppose the local transition rule of a local-iteration graph algorithm is given by a decision tree g. Can a large model trained on tasks generated by this algorithm be tractably distilled into a smaller graph model, with complexity controlled by the complexity of g?

We make the following contributions:

1. We give a sufficient condition, based on a LRH for the source network, that enables distillation of certain local algorithms. This condition, which we call *local-iteration alignment* (Theorem 3.3), requires the source representation to linearly represent simple functions of the form $\mathcal{A}^l[b]$, where b is a single conjunction (Figure 1, right).

2. We prove that under this condition, distillation from the large source network to a GNN is tractable in a restricted structural regime (Theorem 4.2). In contrast, efficiency is not guaranteed under algorithmic mismatch, such as when learning the same local-iteration computation with ordinary decision trees (Theorem 4.1).

3. We give an explicit (ϵ, δ)-distillation algorithm for the framework (Algorithm 2). It draws $\mathrm{poly}(1/\epsilon, \log(1/\delta), n, l, r)$ samples and runs in time $\mathrm{poly}(s^n, 2^{nlr}, 1/\epsilon, \log(1/\delta))$ where n is the size of the graph, s and r are the size and depth of the true inner tree and l is the number of rounds of message-passing.

We complement our theoretical results with experiments that primarily examine whether the proposed LRH assumption emerges in a learned source model, together with a small end-to-end implementation of the proposed distillation algorithm.

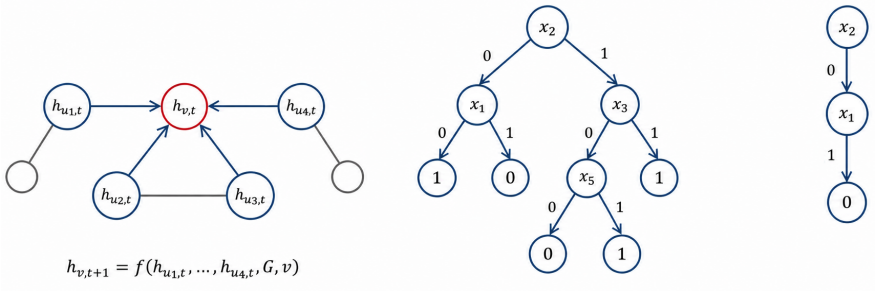

Figure 1: **Illustration of Algorithm 1.** (Left) A GNN that aggregates neighboring information at each iteration. Its architecture reflects the inductive bias of our local-iteration algorithms. (Middle) In our setting, the aggregation function g is represented by a decision tree. (Right) A root-prefix path in the decision tree. The path in this example can be viewed as a degenerate decision tree that outputs 0 (the leaf constant) iff $\neg x_2 \wedge x_1$ (x_2 is negated because it takes the edge with label 0, x_1 takes the edge labeled 1 and thus not negated). Our hypothesis requires the source model to be rich enough to linearly represent simple local-iteration algorithms whose aggregation functions correspond to such root-prefixed paths

1.2 Related works (See Extended Background Section A for more details)

Distillation. Model distillation asks whether the behavior of a large trained model can be transferred to a smaller target model (Hinton et al., 2015; Jiao et al., 2020). We adopt the PAC-distillation framework of Boix-Adsera (2024), which assumes that the source model exposes useful latent features through a LRH (Elhage et al., 2022; Park et al., 2024). This hypothesis is motivated by the empirical observation that neural representations often encode meaningful concepts as linear directions, as seen for example in word embeddings and debiasing applications (Mikolov et al., 2013; Bolukbasi et al., 2016; Manzini et al., 2019; Liang et al., 2020; Chuang et al., 2023). In our setting, the relevant linear features are not arbitrary concepts: they are simple local-iteration subroutines.

Neural Combinatorial Optimization. Many graph optimization algorithms, such as Bellman-Ford, iteratively propagate local information across the graph, much like the layers of a message-passing GNN. An aligned model can therefore reuse this algorithmic scaffold and focus on learning the local update rule (Xu et al., 2020). This perspective has motivated theoretical work on algorithmic alignment (Valiant, 1984; Dudzik and Veličković, 2022; Dudzik et al., 2024) as well as practical neural heuristics (Kahng et al., 2024; Nerem et al., 2025; He and Vitercik, 2025; Gasse et al., 2019). We study the same principle in a distillation setting: rather than asking whether a GNN can learn an algorithm from scratch, we ask when a trained source model can be efficiently distilled into a GNN.

2 Preliminaries

In general, we denote by $\mathcal{X}$ the input set and by $\mathcal{Y}$ the label set. In this paper, we focus on binary classification, i.e., $\mathcal{Y} = \{0, 1\}$. We will write $[n] := \{1, 2, \ldots, n\}$. We define $\mathcal{G} = \mathcal{G}_n$ as the space of all simple labeled graphs on n vertices. For some dimension m, we say that a neural network defines an (abstract) latent representation $\varphi : \mathcal{X} \to \mathbb{R}^m$.

Following standard PAC-learning notation, we will usually denote by $\mathcal{C}$ the concept class (class of possible ground truths), $\mathcal{H}$ the hypothesis class (output range of the learning algorithm), and $\mathcal{D}_c \in \mathcal{P}(\mathcal{X} \times \mathcal{Y})$ the input distribution for some ground truth $c \in \mathcal{C}$. When there is a vector of

many inputs elements $S \in \mathcal{X}^N$ we apply c pointwise and write $c(S) := (c(S_i)_{i\in[N]})$. In the setting of distillation, we have a source class $\mathcal{F}$ and a target class $\mathcal{H}$.

PAC learning In the traditional PAC learning framework (Valiant, 1984), the *concept class* is (ϵ, δ)-learnable with n samples if there is an algorithm $\mathcal{A}$ such that for any distribution $\mathcal{D}$ over the input and any concept in $c \in \mathcal{C}$,

$$\Pr_{S\sim\mathcal{D}^n}[\text{error}_{c,\mathcal{D}}(\mathcal{A}(S, c(S)) \leq \epsilon] \geq 1 - \delta. \tag{1}$$

Here, the error function is the 0-1 population risk: $\text{error}_{c,\mathcal{D}}(f) := \Pr_{\mathcal{D}}[f(x) \neq c(x)]$.

PAC-distillation PAC-distillation (Boix-Adsera, 2024) is a relaxation of PAC learning in which one assumes access to a successful source model class $\mathcal{F}$ to train a target class $\mathcal{H}$ by finding an algorithm $\mathcal{A}$ such that for any distribution $\mathcal{D}$ on $\mathcal{X}$, any source $f \in \mathcal{F}$,

$$\Pr_{S\sim\mathcal{D}^n}[\text{error}_{f,\mathcal{D}}(\mathcal{A}(S, f)) \leq \epsilon] \geq 1 - \delta. \tag{2}$$

Such an algorithm is said to (ϵ, δ)-distill $\mathcal{F} \to \mathcal{H}$. Note that since the algorithm has access to the successful model f, giving a PAC distillation algorithm is easier than giving a PAC algorithm since one can just use f to query labels and simulate PAC learning. The advantage of this framework is to sidestep some of the hardness results of PAC learning by leveraging structures in the class $\mathcal{F}$, such as the LRH that is widely observed in practice (Elhage et al., 2022; Bolukbasi et al., 2016). Concretely, Boix-Adsera (2024) was able to obtain a PAC-distillation algorithm for decision trees in time $\text{poly}(d, 2^r)$ (Theorem A.2) even though it is open whether they are PAC-learnable in less than $d^{O(r)}$ time (Bary-Weisberg et al., 2020)

In practice, $\mathcal{F}$ can be thought of as large, pre-trained neural networks that have achieved low errors on some tasks, and the target class $\mathcal{H}$ can be understood as a function class with inductive bias that can more efficiently represent the ground truth, for example, invariant neural networks such as CNNs or GNNs. Distillation then asks if there are efficient algorithms to find a good representation of the ground truth in the target class.

Algorithmic alignment and LRH In this paper, we view algorithmic alignment through the lens of the LRH, formally defined as follows:

Definition 2.1 (τ-LRH (Boix-Adsera, 2024))**.** Fix a source neural network $f : \mathcal{X} \to \mathcal{Y}$ and let $\varphi : \mathcal{X} \to \mathbb{R}^m$ be the latent representation of f. Let $\mathcal{Z}$ be a set of functions $z : \mathcal{X} \to \mathcal{Y}$. For any $\tau > 0$, we say that f satisfies τ-LRH for features $\mathcal{Z}$ if for all $z \in \mathcal{Z}$, there exists $w \in \mathbb{R}^m$ such that $\|w\| \leq \tau$ and $\langle w, \varphi(x)\rangle = z(x)$ for all $x \in \mathcal{X}$.

In our setting, we think of f as a large foundation model and aim to train a small structured model, specifically a GNN, given minimal access to f, under a guaranty that their abstract latent representation[1] is rich enough to capture the target ground truths (a combinatorial optimization task) with just another linear layer. Surprisingly, we show that distillation is theoretically possible in this setting.

[1]The penultimate layer of a neural net, or the collection of all its activations, can act as the network's latent representation. Since the latent dimension m appears in the complexity bounds, one should choose the smallest latent representation that still satisfies τ-LRH in order to optimize these bounds.

3 Structured distillation framework

All graphs in this section are labeled and have n vertices; we denote this collection by $\mathcal{G}$. For notational convenience, we assume n is a power of 2.

We begin by giving precise definitions of some concepts informally discussed earlier:

Definition 3.1 (Neural networks that compute graph algorithms)**.** A neural network $\nu : \mathcal{X} \times \mathcal{G} \to \mathcal{Y}$ *computes the graph algorithm* $\mathcal{A}$ if ν agrees with A on all inputs of size n. It is *efficient* if it can be evaluated in polynomial time in n.

Definition 3.2 (Local-iteration algorithm)**.** Denote by $\mathcal{M}(\{0,1\})$ the set of multisets of elements in $\{0,1\}$. For any multiset-function $g : \mathcal{M}(\{0,1\}) \times \mathcal{G} \times [n] \to \{0,1\}$, let $\mathcal{A}^l[g] : \{0,1\}^n \times \mathcal{G} \to \{0,1\}$ be the graph-input algorithm that computes Algorithm 1.

For the remainder of the paper, we assume that the stopping time l is fixed and known *a priori*. In this setting, each vertex has a one-bit hidden representation, which is updated by g at each round of aggregation. Although some of our results extend to multi-bit hidden representations, we focus on the one-bit case for simplicity.

We will consider the following specialization of τ-LRH, which is intuitive in the context of graph algorithms:

Definition 3.3 (Local-iteration alignment)**.** Fix a source neural network $\nu \in \mathcal{F}$ and let $\varphi : \mathcal{X} \to \mathbb{R}^m$ be the latent representation of ν. Let Z be a set of 'key features' $z : \{0,1\}^n \times \mathcal{G} \to \{0,1\}$. For any $\tau > 0$, we say that ν satisfies τ-local-iteration alignment for Z if for all $z \in Z$, there exists a $w \in \mathbb{R}^m$ with $\|w\| \leq \tau$ and $\langle w, \varphi(x) \rangle = \mathcal{A}^l[z](x)$ for all inputs x.

Lastly, we define decision trees and root-prefix paths:

Definition 3.4. A decision tree $T : \{0,1\}^d \to \{0,1\}$ is a labeled rooted binary tree with leaves labeled 0 or 1 and internal vertices labeled by *literals*[2] of its input variables $x_1 \dots x_d$. For each input $x \in \{0,1\}^d$, x takes a root-prefix path to arrive at some leaf that specifies the output of the tree on that input.

Definition 3.5. In a decision tree T of depth r rooted at ROOT, a *root-prefix path* S is a clause, i.e., a tuple of literals, $S = (p_1, \dots, p_{r'})$ for some $r' \leq r$, where each $p_i \in \{x_1, \dots, x_d, \neg x_1, \dots, \neg x_d\}$, such that $p_1 = \text{ROOT}$ and S forms a path from the root to some vertex in T. The path need not reach a leaf.

3.1 Concept class, source class and target class

We consider the following concept class, i.e., the collection of possible ground truths:

$$\mathcal{C}_{s,r} = \{\mathcal{A}^l[T] \mid \ T \text{ is a decision tree with depth } r \text{ and size } s\} \tag{3}$$

We address the conditions that ensure that T is a well-defined aggregator, as well as how different choices of T generalize the setting of Algorithm 1, in the next subsection.

To define the source class, we first specify which features are linearly represented by the source functions. Following Boix-Adsera (2024), for a decision tree T, we take the features of T to be its root-prefix path conjunctions: $Z'_T := \{\bigwedge_{p_i \in S} p_i \mid S \text{ is a root-prefix path in } T\}$.

[2] a *literal* of a Boolean variable x_i is either x_i or $\neg x_i$

The corresponding features for local-iteration algorithms are then

$$Z_T := \left\{ \mathcal{A}^l[b] \mid b \in Z'_T \right\}. \tag{4}$$

We postulate that these loops over prefix-path conjunctions are simply representable by the neural network's latent representation.

Let $\mathcal{F}^{\tau}_{s,r}$ denote the class of source networks that implicitly compute $\mathcal{A}^l[T]$ for some decision tree T of size s and depth r satisfying τ-local-iteration alignment with features Z_T:

$$\mathcal{F}^{\tau}_{s,r} = \{f \mid \exists T \text{ such that } f \text{ implicitly computes } \mathcal{A}^l[T]\}. \tag{5}$$

The target class is the same as the concept class. In practice, this class is a subset of GNNs, so the distillation process can be viewed as distilling learned neural networks into GNNs. Conceptually, $\mathcal{C}_{s,r}$ is the class of possible ground-truth functions, not the architecture itself. In our setting the two coincide operationally because every element of $\mathcal{C}_{s,r}$ is computed by a local-iteration procedure that can be represented by the aligned GNN target class.

On the aggregation function T. When using a decision tree T as an aggregation function, some well-definedness issues must be addressed.

First, by Theorem 3.2, the input to the aggregator g may have variable length, since different vertices can have different numbers of neighbors. Because the graphs are simple, g takes at most n inputs. This can be handled in several ways. For example, one can define n different trees T^i, for $i \in [n]$, one for each input length, and pass an additional $\log n$ bits indicating which subtree T^i to use. This increases the depth by at most an additive $\log n$ factor over the maximum depth of the trees T^i.

Second, Theorem 3.2 requires a multiset function, whereas decision trees take ordered tuples as input. In practice, this corresponds to using node-ids to break symmetry, making our problem more general to study from a statistical learning perspective (Kiani et al., 2024).

Third, our setup also applies to the more general case in which T is non-local; for example, $T : \{0,1\}^d \times \mathcal{G} \times [n]$ may take as input the full list of hidden representations h, together with the current node-id. In this global setting, one can decouple d from n by allowing Algorithm 1 to include extra input bits that are passed through to T. This lets us fix n without also fixing the complexity parameters of the decision tree T.

Finally, by the law of currying, our set-up in Algorithm 1 works even without parameter sharing restriction. In other words, we can have a separate tree T_v for each vertex v, and compose these per-vertex trees with a vertex selector tree of size n and depth $\log n$ (Figure 2). Therefore, although using bounded depth (and size) T as an aggregator loses some expressivity compared to using a neural network, our setting allows for other generalizations, namely symmetry breaking with node-id and the use of per-vertex aggregators.

On the key-features Z_T. The 'simple' features that are linearly representable by the source network (Equation (4)) take the form of the template $\mathcal{A}^l$ applied to a conjunction $\bigwedge_{i=1}^{s'} p_i$ for some root-prefix path $p = (p_1, \ldots, p_{s'})$. A conjunction can be viewed as a degenerate decision tree that is a single path: it outputs 1 exactly when all literals on the path are satisfied. By our parameterization in the previous remark, if $s' \leq \log n$, then the hidden representations h are all a partition of the vertex set; otherwise, the feature checks a specific conjunction of bits in the previous hidden representation. Such functions are expected to be learnable by our source class in the aligned regime, and we experimentally test this in Section 5 'Validating the LRH'.

4 Efficient distillation

4.1 GNNs are more efficient than decision trees

The following simple fact gives a separation between GNNs and decision trees:

Lemma 4.1. *There exists a simple decision tree* $T : \{0,1\}^n \times \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ *that can be evaluated in polynomial time in* n *such that, for some constant* l, $\mathcal{A}^l[T]$ *can be evaluated in polynomial time, but it cannot be represented by decision trees of polynomial size.*

The proof considers the 2-reachability DP: Given a graph on $n \geq 2$ vertices, is there a path of length at most 2 that connects the vertex labeled 1 and n? The full proof is in Appendix B. As a result, without the for-loop structure, one cannot just convert the concept class $\mathcal{C}_{s,r}$ into the class of efficient decision trees. This forms a type of algorithmic mismatch which we resolved in the next section, using GNNs.

4.2 Distillation under τ-LRH

We are now ready to state the main theorem.

Theorem 4.2. *For any* $\epsilon, \delta \in (0,1)$*, there is an algorithm that* (ϵ,δ)*-distills from* $\mathcal{F}^{\tau}_{s,r}$ *to* $\mathcal{C}_{s,r}$ *and runs in time polynomial in* $s^n, m, 1/\epsilon, d, 2^{nrl}, \log(1/\delta), \tau, B$ *and uses a number of source-model samples polynomial in* $1/\epsilon, s, n\log(d/\delta), \log(\tau B)$*, where* m *is the latent representation dimension and* $B = \max_x \|\varphi(x)\|$.

Efficiency. The sample complexity analysis of the algorithm follows from a Hoeffding bound and is polynomial in the stated parameters. The time complexity should be interpreted more narrowly. The algorithm is efficient only in the restricted regime where (i) the number of GNN iterations l is a constant, (ii) the depths of the true decision trees are logarithmic or otherwise small, and (iii) the number of vertices n is fixed. Thus the theorem is a complexity-theoretic positive result for a structured setting, not a claim of practical scalability for arbitrary graph sizes or combinatorial instances. This restriction is nevertheless meaningful in comparison with a naive reduction to ordinary decision-tree distillation: unrolling the local-iteration computation and then applying Theorem A.2 would lead to a dependence exponential in the depth of the unrolled tree, on the order of 2^{r^l} in the worst case. Our algorithm instead uses the local iterative structure directly. Removing the fixed-n dependence, or reducing it under symmetry, parameter sharing, or equivariance constraints, is an important open direction; such restrictions are also motivated by evidence that learning under invariances can yield exponential savings (Kiani et al., 2024).

Improvement over (Boix-Adsera, 2024). Comparatively, a naive way to use the decision-tree distillation algorithm of Boix-Adsera (2024) in our setting is to first unroll the entire l-step local-iteration algorithm into a single ordinary decision tree, and then apply their algorithm to this flattened tree. Indeed, if the inner decision tree has depth r, each rounds of unrolling adds r depths per node to the unrolled tree. Thus the fully unrolled computation have depth $O(r^l)$ and their algorithm naively runs in time $2^{O(r^\ell)}$, which is not polynomial even with our restricted setting. We exploit the iterative structure directly in the proof to circumvent this issue.

In the remainder of this section, we describe a GNN distillation that admits the complexity guarantees in Theorem 4.2. A full proof of the theorem can be found in Appendix C.

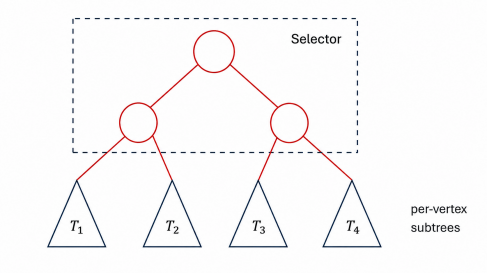

Figure 2: The global decision tree T can be decomposed into a vertex Selector, followed by per-vertex subtrees at its leaves. This generalizes traditional GNNs by allowing for different aggregation schemes based on node statistics or id.

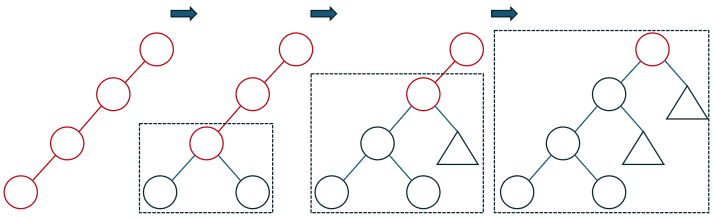

Figure 3: The dynamic program that optimizes for the inner decision in the second phase of Algorithm 2. Circles represent decision-tree nodes while triangles represent subtrees. The base cases are root-prefix path candidates collected in the first phase; the DP then optimizes subtrees of increasing sizes rooted at the ends of candidate root-prefix paths.

GNN distillation algorithm Algorithm 2, operates in two phases. It first builds a set of paths (conjunctions) that is a superset of all root-prefix paths in the true tree T, with high probability. This is done using LRH-type assumptions to linearly probe which paths would be likely to appear in the true decision tree. Then it stitches these paths together efficiently using a modification of a classical tree building DP (Mehta and Raghavan, 2002).

The algorithm has two key subroutines. The LinearProbe$(q, \varphi, B, \tau, \epsilon, \delta, \mathcal{D})$ subroutine comes from Lemma 3.7 of (Boix-Adsera, 2024). It runs in polynomial time and draws polynomially many samples. Intuitively, it tests whether the candidate Boolean function q is easy to read out linearly from the source representation φ. It returns true w.p. $\geq 1-\delta$ if there is a $w \in \mathbb{R}^m$ with $\|w\| \leq \tau$ and $\mathbb{E}_x[(\langle w, \varphi\rangle - q)^2] \leq 2\epsilon$, and returns false w.p. $\geq 1-\delta$ if for all such w, the expectation is at least 2ϵ. Maximization over decision trees in the final step is done with a DP modified from (Guijarro et al., 1999; Mehta and Raghavan, 2002). After Phase 1, we know a pool of plausible root-prefix path fragments, but not how to connect them into a tree. This DP stitches the fragments into the tree that optimizes the objective val; the objective is chosen so that maximizing it corresponds to minimizing the 0-1 risk of the candidate local-iteration algorithm. In details:

Phase 1. We start by initializing the collection $\mathcal{S}$ with root-prefix paths of the vertex selector tree selector of size n and depth $\log n$ (as discussed in Section 3.1). The loop in Lines 2–4 adds the selector prefixes that identify which per-vertex subtree T_u is used. Since the selector decides which of the n per-vertex subtrees $T_1, \ldots, T_n$ processes the input, each such prefix is a root-prefix path of the global true tree T. After this initialization, we recursively grow each surviving path by one valid literal at a time. For a candidate path S, the $\bigwedge_{p\in S} p$ computes the conjunction of all literals in S. If the corresponding local-iteration feature $\mathcal{A}^l[\bigwedge_{p\in S} p]$ cannot be linearly represented by the source network with a low-norm coefficient vector, as checked by LinearProbe, then the path is pruned. The key technical analysis is to prove both sides of this filtering step: all true root-prefix paths pass with high probability, while not too many spurious paths can pass. The latter is a packing argument showing that the source representation cannot simultaneously contain too many low-norm linear readouts for unrelated candidate Boolean functions.

Phase 2. We set up a DP algorithm that builds up our estimated inner decision tree. This step is more involved than that of (Boix-Adsera, 2024), because the same input to $\mathcal{A}^l[\widehat{T}]$ can traverse the global tree $\widehat{T}$ along different paths depending on which vertex u is being updated. To resolve this issue, we build each of the n per-vertex subtrees $T_1, \ldots, T_n$ simultaneously. Finally, before

Algorithm 2 GNN distillation algorithm

Input Neural network ν, random samples from $\mathcal{D}$, depth bound $R \in \mathbb{N}$, error parameters $\epsilon, \delta > 0$
Output A GNN that computes $\mathcal{A}^l[\widehat{T}]$

/* *Phase 1: Collecting root-prefix paths* */
$\mathcal{S}_0 \leftarrow \emptyset$
for $u = 1$ to n **do**
    $\mathcal{S}_0 \leftarrow \mathcal{S}_0 \cup \{p \mid p \text{ is a root-prefix path of Selector leading to } u\}$
end for
for $i = 1$ to R **do**
    $\mathcal{P}_{i-1} \leftarrow \left\{ S \in \mathcal{S}_{i-1} \middle| \textsc{LinearProbe}\left(\mathcal{A}^l\left[\bigwedge_{p\in S} p\right], \varphi, B, \tau, 2^{-il-3}, \frac{\delta}{2|\mathcal{S}_{i-1}|R} \right) = \text{true} \right\}$
    $\mathcal{S}_i \leftarrow \bigcup_{S\in\mathcal{P}_{i-1}} \bigcup_{j=1}^{d} \{S \cup \{x_j\},\ S \cup \{\neg x_j\}\}$
end for
$\mathcal{S} \leftarrow \bigcup_{j=0}^{R} \mathcal{S}_j$

/* *Phase 2: Estimate the true tree using paths from* $\mathcal{S}$ */
$\mathcal{S} := \bigsqcup_{u=0}^{n} \mathcal{S}^u$ decomposing paths by the per-vertex subtree T_u identified by their selector prefix
for all $S_1 \in \mathcal{S}^1, \ldots, S_n \in \mathcal{S}^n$ **do**
    $\hat{v}_{S_1,\ldots,S_n} \leftarrow \mathbb{E}_{y\sim\mathcal{D}}\left[\left(\prod_{u=1}^{n} \prod_{p\in S_u} p(y)\right)(2\nu(y) - 1)\right] \pm \epsilon/|\mathcal{S}|$
end for
return $\arg\max_{\widetilde{T}_1,\ldots,\widetilde{T}_n} \mathrm{val}(\widetilde{T}_1, \ldots, \widetilde{T}_n, \hat{v})$, where $\widetilde{T}_i$ ranges over decision trees with $Z'_{\widetilde{T}_i} \subseteq \left\{ \bigwedge_{p\in S} p \;\middle|\; S \in \mathcal{S}^i \right\}$.

setting up the DP objective, recall that each root-prefix path S of size more than $\log n$ contains a $\log n$-length selector prefix that identifies which subtree T_u the path comes from. Thus, Line 11 decomposes $\mathcal{S} := \bigsqcup_{u=0}^{n} \mathcal{S}^u$, where $\mathcal{S}^0$ contains the selector-only paths and $\mathcal{S}^u$ contains paths from per-vertex subtree T_u. Lines 12–14 then estimate local correlation statistics $\hat{v}_{S_1,\ldots,S_n}$ for tuples of retained paths, one from each candidate set. The tree-building DP uses these statistics to jointly infer all per-vertex trees.

Valuation function. Now we can set up the DP objective. For a candidate tree $\widetilde{T}$ of depth r and size s with per-vertex subtrees $\widetilde{T}_i$ for each $i \in [n]$ and for any weight function $u : \bigotimes_i \mathcal{S}_i \to \mathbb{R}$, define the valuation function:

$$\mathrm{val}(\widetilde{T}_1, \ldots, \widetilde{T}_n, u) := \sum_{S_i \in \mathrm{Leaves}(T_i), i \in [n]} (2\widetilde{T}(S_1, \ldots, S_n) - 1) u_{S_1,\ldots,S_n}, \tag{6}$$

where $\widetilde{T}(S_1, \ldots, S_n)$ is computes $\mathcal{A}[\widetilde{T}]$ after replacing $\widetilde{T}_i$ with S_i only in the first layer. Using $u := v$– the exact expectation in Line 15, we claim to get: $\mathrm{val}(\widetilde{T}_1, \ldots, \widetilde{T}_n, v) = \mathbb{E}_x \left[(2\nu(x) - 1)(2\widetilde{T}(x) - 1) \right]$, which negatively correlates with the 0-1 loss for the estimated trees. In Algorithm 2, one uses a Hoeffding bound to obtain $\hat{v}$ as an empirical approximation to v. We can now describe the DP that estimates our tree. The states of the DP are indexed by tuples $(S'_1, \ldots, S'_n, s'_1, \ldots, s'_n) \in \mathcal{S}^n \times [s]^n$ where s is the size upper bound for the true decision tree. At state (S'_i, s'_i), the transition computes the optimal subtree of size s'_i rooted at the end of path S'_i for each $i \in [n]$. The tree that optimizes 0-1 loss can then be queried at the final DP state. A simplification of this procedure is shown in Figure 3.

5 Experiments

We fix the number of vertices to be $n = 6$ and the number of message passing rounds to be $l = 6$. For each depth $r \in \{2, 3, 4, 5\}$, we generate an independent uniformly random decision tree T_u of depth r for each vertex u. These per-vertex trees, together with a fixed vertex selector, define the ground truth local-iteration algorithm $\mathcal{A}^l[T_1, \ldots, T_n]$. We use a 5-layer ResNet with width 1000 as our large unstructured source model and train it on uniformly sampled Boolean graph inputs under classification loss using stochastic gradient descent. We choose these dimensions so that the largest task in the experiment, depth-5 per-vertex trees, can be fit by the source model.

Validating the LRH. After training the source model, we collect all hidden activations as the representation φ. For every root-prefix conjunction b appearing in the true per-vertex trees, we train a linear probe w so that $\langle w, \varphi \rangle$ approximates $\mathcal{A}^l[b]$. The probe is trained for 100 iterations with Adam on squared loss using uniformly sampled graph inputs. To test the low-norm form of the assumption, we also run projected gradient descent with a prescribed norm bound on w. Table 1 reports the resulting train and test errors. Low error under the norm constraint is evidence that the trained source model satisfies the local-iteration alignment condition used by

End-to-end distillation. We next evaluate the full algorithmic pipeline by implementing a heuristics of Algorithm 2. For each depth $r \in \{2, 3, 4, 5\}$ and each probe number $k \in \{10, 50, 100, 200\}$, Phase 1 starts from the vertex-selector prefixes and probes candidate clauses of the form $\mathcal{A}^l[\wedge_{p \in S} p]$. We use a practical top-k probe search: at each depth, we keep the k clauses with lowest validation probe error and branch only from those clauses. Phase 2 then ranks the collected clauses by probe error, keeps a bounded number per vertex, and runs the tree-building DP over this pruned set. Because explicitly materializing $\hat{v}_{S_1,\ldots,S_n}$ has size $\prod_u |\mathcal{S}^u|$, which is already prohibitive for $n = 6$, the implementation estimates the phase-2 objective by Monte Carlo agreement with the source

depth (norm)	# of conj.	source acc.	average training err	average testing err
2 (∞)	42	1.000	0.1982	0.2058
2 (0.001)	42	1.000	1.898	1.935
3 (∞)	90	1.000	0.3155	0.3298
3 (0.001)	90	1.000	1.258	1.270
4 (∞)	186	0.885	0.0875	0.0971
4 (0.001)	186	0.885	0.4680	0.4690
5 (∞)	378	0.784	0.0526	0.0602
5 (0.001)	378	0.784	0.2164	0.2177

Table 1: Linearly probing $\mathcal{A}^l[b]$ where b is a single conjunction in the true tree. First column includes the true depth, and a norm upper bound for projected descent.

depth	k	source acc.	distil. acc.	# probes	probe frac.
2	10	1.000	0.754	360	0.208
2	50	1.000	0.745	1041	0.600
2	100	1.000	0.736	1508	0.870
2	200	1.000	0.757	1734	1.000
3	10	1.000	0.909	557	0.045
3	50	1.000	0.898	2002	0.163
3	100	1.000	0.911	3355	0.273
3	200	1.000	0.900	5198	0.423
4	10	0.881	0.652	736	0.012
4	50	0.881	0.646	2856	0.048
4	100	0.881	0.634	5096	0.085
4	200	0.881	0.679	8642	0.144
5	10	0.791	0.667	894	0.004
5	50	0.791	0.686	3665	0.017
5	100	0.791	0.661	6649	0.031
5	200	0.791	0.675	11729	0.055

Table 2: End-to-end Algorithm 2. Here k is the top-k probe size used in Phase 1.

network. We report the accuracy of the pipeline in Table 2 and some more detailed diagnostics in the Appendix (Table 3).

6 Conclusion and discussion

In this paper, we extend the work of Boix-Adsera (2024) in PAC-distillation to study the well-known 'learn first, distill later' paradigm, where a large multipurpose model is trained on a variety of tasks, then distilled to more structured models that have built-in algorithmic alignment properties, such as GNNs for learning DP algorithms. We showed that although some DP algorithms have a local transition rule representable by a small decision tree, the full DP computation cannot itself be represented by an efficient decision tree – a case of misalignment. On the other hand, the local iteration structure of a GNN with decision tree aggregation allows for distillation from a large, learned neural network that exhibits a certain kind of linear representability. We also propose an algorithm that is tractable when the number of GNN iterations is fixed, the depth of the inner decision tree is small, and the number of vertices in the input graphs is fixed.

Several exciting questions and future directions arise from this work. We conjecture that in a setting as general as ours (as described in Section 3.1), an exponential lower bound in the number of vertices n could be proven. One can also study more restricted settings, such as shared transition trees or equivariant parameterizations, to obtain a better dependence on n. Finally, one could analyze distillation algorithms that are popular in practice, such as reinforcement learning or teacher-student supervised learning.

Acknowledgments

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) under agreement no. HR0011-25-3-0205. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. MW acknowledges partial support from an Alfred P. Sloan Fellowship in Mathematics and the AI2050 program at Schmidt Sciences (Grant G-25-69786).

References

Gregor Bachmann, Sotiris Anagnostidis, and Thomas Hofmann. Scaling MLPs: A tale of inductive bias. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=R45A8eKcax.

Galit Bary-Weisberg, Amit Daniely, and Shai Shalev-Shwartz. Distribution free learning with local queries. In Aryeh Kontorovich and Gergely Neu, editors, *Proceedings of the 31st International Conference on Algorithmic Learning Theory*, volume 117 of *Proceedings of Machine Learning Research*, pages 133–147. PMLR, 08 Feb–11 Feb 2020. URL https://proceedings.mlr.press/v117/bary-weisberg20a.html.

Alberto Bietti, Luca Venturi, and Joan Bruna. On the sample complexity of learning under invariance and geometric stability. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Enric Boix-Adsera. Towards a theory of model distillation, 2024. URL https://arxiv.org/abs/2403.09053.

Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4356–4364, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Johann Brehmer, Sönke Behrends, Pim De Haan, and Taco Cohen. Does equivariance matter at scale? *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=wilNute8Tn.

Ching-Yao Chuang, Varun Jampani, Yuanzhen Li, Antonio Torralba, and Stefanie Jegelka. Debiasing vision-language models via biased prompts, 2023. URL https://arxiv.org/abs/2302.00070.

Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20635–20647. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8248b1ded388fcdbbd121bcdfea3068c-Paper-Conference.pdf.

Andrew Joseph Dudzik, Tamara von Glehn, Razvan Pascanu, and Petar Veličković. Asynchronous algorithmic alignment with cocycles. In Soledad Villar and Benjamin Chamberlain, editors, *Proceedings of the Second Learning on Graphs Conference*, volume 231 of *Proceedings of Machine Learning Research*, pages 3:1–3:17. PMLR, 27–30 Nov 2024. URL https://proceedings.mlr.press/v231/dudzik24a.html.

Bryn Elesedy. Provably strict generalisation benefit for invariance in kernel methods. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish,

Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. *Exact combinatorial optimization with graph convolutional neural networks*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning (book). In *MIT Press*, 2016.

David Guijarro, Víctor Lavín, and Vijay Raghavan. Exact learning when irrelevant variables abound. *Inf. Process. Lett.*, 70(5):233–239, June 1999. ISSN 0020-0190. doi: 10.1016/S0020-0190(99)00063-0. URL https://doi.org/10.1016/S0020-0190(99)00063-0.

Yu He and Ellen Vitercik. Primal-dual neural algorithmic reasoning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=iBpkzB5LEr.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.372. URL https://aclanthology.org/2020.findings-emnlp.372/.

Andrew B. Kahng, Robert R. Nerem, Yusu Wang, and Chien-Yi Yang. Nn-steiner: a mixed neural-algorithmic approach for the rectilinear steiner minimum tree problem. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i12.29200. URL https://doi.org/10.1609/aaai.v38i12.29200.

Bobak Kiani, Thien Le, Hannah Lawrence, Stefanie Jegelka, and Melanie Weber. On the hardness of learning under symmetries. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ARPrtuzAnQ.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25. Curran Associates, Inc., 2012.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Paul Pu Liang, Irene Mengze Li, Emily Zheng, Yao Chong Lim, Ruslan Salakhutdinov, and Louis-Philippe Morency. Towards debiasing sentence representations. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5502–5515, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.488. URL https://aclanthology.org/2020.acl-main.488/.

Thomas Manzini, Lim Yao Chong, Alan W Black, and Yulia Tsvetkov. Black is to criminal as Caucasian is to police: Detecting and removing multiclass bias in word embeddings. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 615–621, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1062. URL https://aclanthology.org/N19-1062/.

Dinesh P. Mehta and Vijay Raghavan. Decision tree approximations of boolean functions. *Theor. Comput. Sci.*, 270(1-2):609–623, 2002. URL https://doi.org/10.1016/S0304-3975(01)00011-1.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Robert R. Nerem, Samantha Chen, Sanjoy Dasgupta, and Yusu Wang. Graph neural networks extrapolate out-of-distribution for shortest paths, 2025. URL https://arxiv.org/abs/2503.19173.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Behrooz Tahmasebi and Stefanie Jegelka. The exact sample complexity gain from invariances for kernel regression. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=6iouUxI45W.

Robert Tarjan. Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 114–121, 1971. doi: 10.1109/SWAT.1971.10.

L. G. Valiant. A theory of the learnable. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, page 436–445, New York, NY, USA, 1984. Association for Computing Machinery. ISBN 0897911334. doi: 10.1145/800057.808710. URL https://doi.org/10.1145/800057.808710.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJxbJeHFPS.

A Extended Background

Graph machine learning Graph machine learning is a testbed for graph-based inductive biases that may allow for exponential gains in learning efficiency. Informally, symmetry constraints of graph functions, in terms of vertex permutations, induce certain sparsity structures in the function space, making learning more data-efficient (Bietti et al., 2021; Elesedy, 2021; Tahmasebi and Jegelka, 2023). However, learning graph neural networks and other equivariant networks is still computationally hard in the worst case, requiring, for example, exponentially or superpolynomially many queries in the correlation statistical queries model of learning (Kiani et al., 2024). Understanding which settings exactly give rise to quantitative benefits for learning is an important and active area of research.

More specifically for graphs, a graph neural network (GNN) (Gilmer et al., 2017; Kipf and Welling, 2017) is a deep-learning parameterization of the space of functions on graphs, potentially of different sizes.

Graph neural networks The main architecture we consider as the target class is that of graph neural networks with strong inductive bias for graph datasets. In particular, we are interested in message passing neural network (MPNN) (Theorem A.1), in which each node aggregates neighboring information and processes them with a neural network to form a new latent representation in each round. After a fixed number of rounds, the network outputs a learned representation for each vertex of the graph, or combines them together to form a single representation for the whole graph, depending on the specific tasks. While we will only consider decision tree aggregators, neural networks can (arguably efficiently) emulate decision trees, since they are universal approximators.

Definition A.1 (Message-passing neural network)**.** Let $G = (V, E)$ be a graph with node features $x_v \in \mathcal{X}$ for $v \in V$. An l-layer message-passing neural network (MPNN) consists of an initialization map $\iota : \mathcal{X} \to \mathbb{R}^{d_0}$, message maps $M_t : \mathbb{R}^{d_{t-1}} \times \mathbb{R}^{d_{t-1}} \to \mathbb{R}^{q_t}$, update maps $U_t : \mathbb{R}^{d_{t-1}} \times \mathbb{R}^{q_t} \to \mathbb{R}^{d_t}$, and a permutation-invariant aggregation operator AGG_t on multisets. It computes hidden states

$$h_v^{(0)} = \iota(x_v), \qquad m_v^{(t)} = \mathrm{AGG}_t(\{M_t(h_v^{(t-1)}, h_u^{(t-1)}) : u \in \mathcal{N}(v)\}),$$

$$h_v^{(t)} = U_t(h_v^{(t-1)}, m_v^{(t)}), \qquad t = 1, \ldots, l.$$

A node-level MPNN outputs $\rho_{\mathrm{node}}(h_v^{(L)})$, while a graph-level MPNN outputs

$$\rho_{\mathrm{graph}}(\mathrm{READOUT}(\{h_v^{(L)} : v \in V\})),$$

where READOUT is permutation invariant. If the message, update, and readout maps are neural networks, we call the resulting architecture an MPNN.

Combinatorial optimization with graph ML One proposed area where GNNs could have strong inductive bias with the learning task is that of using neural networks to learn combinatorial optimization. It is observed (Xu et al., 2020) that the loop structure of an MPNN closely follows that of local graph algorithms, such as Bellman-Ford for shortest path. As such, Xu et al. (2020) argues that the neural network used in the aggregation operation of an MPNN only had to learn a simple function of its inputs, and not the actual for-loop structure, thus decreasing the sample complexity of learning from supervised examples produced by such algorithms. Although the original paper provided a theoretical justification for this phenomenon through PAC learning (Valiant, 1984), a tighter analysis of what constitutes such *algorithmic alignment* has drawn many

follow-up investigations (Dudzik and Veličković, 2022; Dudzik et al., 2024). Nevertheless, the idea that learning architecture should be built to resemble a potential algorithmic paradigm, such as dynamic programming, is intuitive and has been the inspiration for many neural heuristics that are widely successful in practice (Kahng et al., 2024; Nerem et al., 2025; He and Vitercik, 2025; Gasse et al., 2019).

PAC-distillation keystone result To give a taste of the results that can be obtained from this framework, we restate a result from Boix-Adsera (2024). Recall that in the Boolean setting, a decision tree has vertices labeled by some literals of the input bits and each vertex is only reachable by inputs that satisfy the conjunction of literals on the path from the root to said vertex.

Theorem A.2 (Theorem 3.6 of (Boix-Adsera, 2024))**.** *Let $\mathcal{F}$ be the set of neural networks f that implicitly compute a decision tree $T : \{0,1\}^d \to \{0,1\}$ of depth r and size s such that f satisfies τ-LRH for features $\mathcal{Z}_T := \{\bigwedge_{p\in S} p : S$ is a path of literals from the root of T to any of its vertices$\}$. Let $\mathcal{H}$ be the set of decision trees with depth r and size s. Then for any $\epsilon, \delta \in (0,1)$, there is an algorithm that (ϵ,δ)-distills from $\mathcal{F}$ to $\mathcal{H}$ that runs in polynomial time in $d, m, 1/\epsilon, s, 2^r, \log(1/\delta), \tau$ and B and takes polynomially many samples in $1/\epsilon, s, \log(d/\delta), \log(\tau B)$ where $B \geq \max_x \|\varphi(x)\|$.*

Remark A.3. This is an unexpected result, as it is unknown if PAC-learning a decision tree can take less than $d^{O(r)}$ time (Bary-Weisberg et al., 2020). On the other hand Theorem A.2 shows that PAC-distillation takes only $\text{poly}(d, 2^r)$ time.

B Proof of Lemma 4.1

Proof. Consider the combinatorial problem of deciding, for a labeled graph, whether the first and last vertex is connected with a path of length at most 2, or 2-reachability.

The classic dynamic programming (DP) algorithm for this problem runs in time $O(n)$ (Tarjan, 1971).

However, any decision tree that correctly solves this problem on all labeled graphs of size n must have exponential size. To see this, we bound the number of leaves of a correct tree (which in turn bounds the order of its size since a decision tree is binary).

Consider the subset of graphs on the n vertices labeled by $[n]$ where the only possible edges are $(1,n)$ and $(1,v)$, (v,n) for all $v \in V\backslash\{1,n\}$. There are $2^{2(n-2)+1}$ such graphs. Among them, graphs that fail to have a path of size at most 2 between 1 and n does not have the $(1,n)$ edge and for each other v, have one of the 3 configurations out of 4 possible choices of presence/absence of the pair $(1,v)$, (v,n). This counts to $(3/4)^{n-2}/2$ fraction of the total number of graphs.

Now, each 0-leaf (leaf that outputs 0 for the DT) of a correct DT on these inputs fixes a certain presence/absence of some edges on the path from the DT's root to it. Once certain variables are fixed, all other variables are free to range between 0 and 1 and the output of the DT is still 0. This means that $(1,n)$ must always be included in the fixed variables, and so is at least one in each pair $(1,v)$, (v,n). Thus, each 0-leaf accounts for at most a fraction of $2^{-(n-1)}$ of the total number of graphs.

Therefore, the number of leaves must be at least $(3/4)^{n-2}/2/2^{-(n-1)}$, which is exponential in n. □

C Proof of Theorem 4.2

C.1 Notations and definitions

We will set up some notation for this particular proof and also remind the readers of previously defined notation:

- **The input space:** Since we are using a graph neural network to emulate an algorithm of the form $\mathcal{A}^l[T]$ for some decision tree T, there is a difference between the input of $\mathcal{A}^l[T]$ and that of T. The former takes as input an initialization feature in $\{0,1\}^n$ a graph adjacency matrix $\{0,1\}^{n\times n}$ and we write $\mathcal{X}_{\mathcal{A}}$ for this input space. T itself has an input consisting of the previous representation of each layer $\{0,1\}^n$, a graph adjacency matrix $\{0,1\}^{n\times n}$ and additionally the index of a vertex v and we reserve $\mathcal{X} := \{0,1\}^d$ where $d = \Omega(n^2)$.
- **Logical notation:** for some input bit x_j, $j \in [d]$, a *literal* is x_j or its negation $\neg x_j$. A *clause* $S = (p_1, \ldots, p_s)$ is an ordered tuple of s literals and we define $\mathrm{AND}_S(x) := \bigwedge_{p\in S} p$ the conjunction of literals in S. A *non-degenerate* k-clause is a clause S such that $|S| = k$ and each variable appears at most once in S.
- **Decision trees:** Given a decision tree T, recall that Z'_T is the set of clauses each corresponds to a path in T with one end-point being the root (i.e. a *root-prefix path*). We include the trivial path $\emptyset$ with $\mathrm{AND}_\emptyset = \textsc{true}$ in this collection. We also denote by Z_T the collection of all $\mathcal{A}^l[\mathrm{AND}_S]$ functions for each $S \in Z'_T$.

In our algorithm, which is an extension of that in (Boix-Adsera, 2024), we use some subroutines from the original paper.

Lemma C.1 (Lemma 3.7 (Boix-Adsera, 2024))**.** *Given a function $g : \mathcal{X} \to [-1,1]$, a representation map $\varphi : \mathcal{X} \to \mathbb{R}^m$ with norm bounded by $B \geq \max_x \|\varphi(x)\|$, $\tau, \epsilon, \delta > 0$ and an input distribution $\mathcal{D}$, there is a subroutine $\textsc{LinearProbe}(g, \varphi, B, \tau, \epsilon, \delta, \mathcal{D})$ that runs in time poly$(1/\epsilon, \log(1/\delta), \tau, B, m)$ and draws poly$(1/\epsilon, \log(1/\delta), \tau, B)$ samples from $\mathcal{D}$ such that:*

- *If there is a $w \in \mathbb{R}^m$ with $\|w\| \leq \tau$ and $\mathbb{E}[(w\cdot\varphi(x) - g(x))^2] \leq \epsilon$, then $\textsc{LinearProbe}$ returns true with probability $1-\delta$.*
- *If there is no $w \in \mathbb{R}^m$ with $\|w\| \leq \tau$ and $\mathbb{E}[(w\cdot\varphi(x) - g(x))^2] \leq \epsilon$, then $\textsc{LinearProbe}$ returns false with probability $1-\delta$.*

C.2 Proof of Theorem 4.2

We are now ready to start the proof. Assume that there is a true decision tree T. We first show that the paths collection from the distillation algorithm contains all root prefix paths in the true tree: $\mathcal{S} \supseteq Z_T$ with high probability.

$\mathcal{S}$ contains all root-prefix paths of the true tree From the guarantees of Theorem C.1, it suffices to show that any clause $S \in Z'_T$ is checked by $\textsc{LinearProbe}$ and thus added to $\mathcal{S}$ with high probability. Assume to the contrary that this is not true; in other words, there is a root-prefix path $S \in Z'_T$ of length i that was not added to $\mathcal{S}_i$. Recall that whenever $\textsc{LinearProbe}$ accepted a clause S', we added all possible extension of S' to our collection $\mathcal{S}$. The fact that S was not included means that either it was not checked by $\textsc{LinearProbe}$ or it was checked and then rejected. In the former case, this means that $S \neq \emptyset$ (since the $\emptyset$ is always checked) and the root-prefix path

corresponds to S' parent was not included in the previous set $\mathcal{S}_{i-1}$. We can then use induction on this parent node instead. In the latter case, S was checked but LinearProbe returns false, which occurs with probability $1 - \frac{\delta}{2|\mathcal{S}_i|R}$ because of our linear representation hypothesis and Theorem C.1.

A union bound at each layer suffices to argue that all length i clauses in Z'_T are in $\mathcal{S}$ with probability $1 - \frac{\delta}{2R}$ for each i. Another union bound over all i then concludes that $Z'_T \in \mathcal{S}$ with probability $1 - \delta/2$.

Now we argue that $|S_i| \leq \text{poly}(2^{\Theta(il)}, \tau, B, d)$

$\mathcal{S}_i$ size is upper-bounded The key way we control $\mathcal{S}_i$ size is by arguing that only the $\mathcal{A}^l[\text{AND}_S]$'s that can be linearly represented by the source network are kept while the rest are pruned. Furthermore, there cannot be too many of these $\mathcal{A}^l[\text{AND}_S]$ kept, at the same time. A naïve approach uses the following lemma from (Boix-Adsera, 2024).

Lemma C.2 (Lemma 3.8 (Boix-Adsera, 2024))**.** *Let $\mathcal{S}$ be a collection of non-degenerate k-clauses. Let $\mathcal{G} = \{AND_S \mid S \in \mathcal{S}\}$. If φ approximately satisfies τ-LRH w.r.t. $\mathcal{G}$:*

$$\forall g \in \mathcal{G}, \exists w \in \mathbb{R}^m, \|w\| \leq \tau \text{ and } \mathbb{E}_{x \sim U[\{0,1\}^d]}[(w \cdot \varphi(x) - g(x))^2] \leq 2^{-k-2}, \tag{7}$$

then $|\mathcal{S}| \leq 2^{3k+4}\tau^2 \mathbb{E}_x \|\varphi(x)\|^2$

We want to strengthen this to the following:

Lemma C.3 (Packing with for-loops)**.** *Let $\mathcal{S}$ be a collection of non-degenerate k-clauses. If φ approximately satisfies local iteration alignment w.r.t. $\mathcal{S}$:*

$$\forall S \in \mathcal{S}, \exists w \in \mathbb{R}^m, \|w\| \leq \tau \text{ and } \mathbb{E}_{x \sim U[\{0,1\}^d]}[(w \cdot \varphi(x) - \mathcal{A}^l[[AND_S]](x))^2] \leq 2^{-\Theta(kl)}, \tag{8}$$

then $|\mathcal{S}| \leq 2^{\Theta(kl)}\tau^2 \mathbb{E}_x \|\varphi(x)\|^2$.

Proof. To demonstrate the structure of $\mathcal{A}^l[\text{AND}_S]$, we will perform a loop unrolling. Recall that the input to the inner tree has three parts: some bits to specify the vertex, which we will denote $x_{v,1} \dots x_{v,\log n}$; some bits to query the graph adjacency matrix $x_{e,1} \dots x_{e,\binom{n}{2}}$ and some bits to query the DP table $x_{dp,1} \dots x_{dp,n}$. Fix a k-clause S. Let its index set be $\{v_1, \dots, v_a; e_1, \dots, e_b; dp_1, \dots, dp_c\}$ where $a + b + c = k$ and denote by z_I the literal for x_I for some I in the index set. We have, for some initialization vector and graph adjacency A:

$$\mathcal{A}^l[\text{AND}_S](\textsc{Init}, A) := h_{n,l}(\textsc{Init}, A) \tag{9}$$

$$= \bigwedge_{i \in [a]} z_{v_i}(n) \wedge \bigwedge_{j \in [b]} z_{e_j}(A) \wedge \bigwedge_{u \in [c]} z_{dp_u, l-1}(\textsc{Init}, A) \tag{10}$$

Now, the conjunctions $\bigwedge_{i \in [a]} z_{v_i}$ defines a bipartition of the vertex set into two parts,. Denote by $c_{S,v} \in \{0, 1\}$ the indicator function for the set carved out by $\bigwedge_{i \in [a]} z_{v_i}$ and remark that we know $c_{S,v}$ even before seeing any input (and thus they are constants w.r.t. $\mathcal{A}^l[\text{AND}_S]$). Furthermore, we write $\bigwedge_{j \in [b]} z_{e_j}(A)$ as $S(A)$ since this quantity depends only on the input graph. Thus:

$$\mathcal{A}^l[\text{AND}_S](\textsc{Init}, A) = c_{S,n} \wedge S(A) \bigwedge_{u \in [c]} z_{dp_u, l-1}(\textsc{Init}, A) \tag{11}$$

$$= c_{S,n} \wedge S(A) \wedge \bigwedge_{u \in [c]^+} c_{S,dp_u} \wedge S(A) \wedge \bigwedge_{u_2 \in [c]} h_{dp_{u_2}, l-2}(\textsc{Init}, A)$$

$$\wedge \bigwedge_{u \in [c]^-} \neg \left(c_{S,dp_u} \wedge S(A) \wedge \bigwedge_{u_2 \in [c]} h_{dp_{u_2}, l-2}(\textsc{Init}, A) \right) \tag{12}$$

At layer i of the for-loop, evaluating an entry asks for c evaluation of the previous layer, naïvely giving us c^l evaluations when completely unrolling all l layers of $\mathcal{A}^l$. However, because the evaluations are only at the indices $dp_1, \ldots, dp_c$ which are determined by S, we do not need to fill out the whole DP table but only at these c points. Thus, $\mathcal{A}^l[\mathrm{AND}_S](\textsc{Init}, A)$ depends on $\textsc{Init}$ only through the c bits and on A only through $S(A)$ which looks at the b bits of A. Thus, $\mathcal{A}^l[\mathrm{AND}_S]$ is a function of $(b+c)$ bits ($\leq k$ bits) of its input, i.e., a $(b+c)$-junta.

Having established that $\mathcal{A}^l[\mathrm{AND}_S]$ are functions of at most k bits, we can use the same packing bound based on Fourier-analytic arguments of (Boix-Adsera, 2024).

Recall that τ-local-iteration alignment implies that: for every $S \in Z'_T$, there is a $w_S \in \mathbb{R}^m$ with $\|w_S\| \leq \tau$ such that, $\langle w, \varphi(x)\rangle = \mathcal{A}^l[\mathrm{AND}_S(x)]$ for all $x \in \{-1,1\}^d$ (note that here we use the domain $\{\pm 1\}^d$ that works better with Fourier analytic arguments of boolean functions). Collect all such w into the rows of a matrix $W \in \mathbb{R}^{|\mathcal{S}|\times m}$ and $\varphi(x)$ into the columns of some $\Phi \in \mathbb{R}^{m\times 2^d}$. We can derive the packing bound from the fact that orthogonal projection to the row span of $V \in \mathbb{R}^{\binom{d}{k}\times 2^d}, V_{A,x} = \chi_A(x)$ for all $A \in \binom{[d]}{k}$ and χ_A being the parity function of indices in A, has large norm. Let $P \in \mathbb{R}^{2^d\times 2^d}$ be the orthogonal projection to this subspace of low degree polynomials in $L^2(\{\pm 1\}^d)$ and $P^\top$ the projection to the orthogonal subspace

First, we want to compute the coefficient of the degree k term of $\mathcal{A}^l[\mathrm{AND}_S](\textsc{Init}, A)$ when written as $\{\pm 1\}^d$-polynomial gives:

$$\mathcal{A}^l[\mathrm{AND}_S](\textsc{Init}, A) = h_{n,l} \tag{13}$$

$$= 2^{-(c+2)+1}(1 + c_{S,n}) \cdot (1 + S(A)) \cdot \prod_{u\in[c]} (1 + \sigma h_{dp_u,l-1}(\textsc{Init}, A))) - 1 \tag{14}$$

where σ is either 1 or -1 depending on the sign of the literal. Thus the leading term is of degree k has coefficient $\pm 2^{-\Theta(kl)}$. Therefore,

$$[W\Phi P_k]_{S,x} = \pm 2^{-\Theta(kl)}\chi_{I(S)}, \tag{15}$$

where $I(S)$ is the set of indices of the input that appears in literals of S.

Therefore,

$$W\Phi\Phi^\top W^\top \succeq W\Phi PP^\top\Phi^\top W^\top = 2^{d-\Theta(kl)}I, \tag{16}$$

and one conclude that $|\textsc{det}(W\Phi\Phi^\top W^\top)| \geq 2^{(d-\Theta(kl))|\mathcal{S}|}$.

Using τ-local-iteration alignment, we get the following for free:

Lemma C.4 (Claim B.4 (Boix-Adsera, 2024)). *We have:* $\textsc{det}(W\Phi\Phi^\top W^\top) \leq (2^d(\mathbb{E}\|\varphi\|^2)^2\tau^2/|\mathcal{S}|)^{|\mathcal{S}|}$.

□

Combining the two gives:

$$2^{(d-\Theta(kl))|\mathcal{S}|} \leq (2^d(\mathbb{E}\|\varphi\|^2)^2\tau^2/|\mathcal{S}|)^{|\mathcal{S}|}, \tag{17}$$

which gives $|\mathcal{S}| \leq 2^{\Theta(kl)}(\mathbb{E}\|\varphi\|^2)^2\tau^2$.

Finally, we give details on the DP procedures that stitch together all the root-prefix paths to find the true tree based on 0-1 loss.

Dynamic programming algorithm to infer final tree For this section, recall that the concept class dictates that the decision tree can be different when processing different vertices of the graph. To this end, we let t_i be the true decision tree for vertex i and treat them as a different decision tree. To get back the full tree, we simply add a $\log n$-depth index selector at the beginning of the estimated tree.

The problem is reduced to inferring t_i's simultaneously. Recall that the set of possible clauses for tree i is $\mathcal{S}_i$. For some weight function $u : \bigotimes_i \mathcal{S}_i \to \mathbb{R}$, define the valuation function:

$$\mathrm{val}(\widetilde{T}_1, \ldots, \widetilde{T}_n, u) := \sum_{S_i \in \mathrm{Leaves}(T_i), i \in [n]} (2\widetilde{T}(S_1, \ldots, S_n) - 1) u_{S_1, \ldots, S_n}, \tag{18}$$

where $\widetilde{T}(S_1, \ldots, S_n)$ is defined as the computation of the template function $\mathcal{A}[\widetilde{T}]$ by replacing $\widetilde{T}_i$ with S_i in only the first layer. In other words, even though we do not have any input x to evaluate $\mathcal{A}[\widetilde{T}]$ at, we can still use the leaf node of the paths $S_1, \ldots, S_n$ to determine the hidden representation $h_{v,1}$ of the first layer, for each v's. Determining all $h_{v,1}$ allows us to compute $\mathcal{A}[\widetilde{T}]$ by passing it to the next layers.

Recall that the weight function for our dynamic program is:

$$v : \bigotimes_i \mathcal{S}_i \to \mathbb{R}, v_{S_1, \ldots, S_n} := \mathbb{E}_{x \sim \mathcal{D}} \left[(2 f_\theta(x) - 1) \prod_i \mathrm{AND}(S_i) \right]. \tag{19}$$

Using this weight function, one gets a negative correlation of the 0-1 loss:

$$\mathrm{val}(\widetilde{T}_1, \ldots, \widetilde{T}_n, v) = \sum_{S_i \in \mathrm{Leaves}(T_i), i \in [n]} (2\widetilde{T}(S_1, \ldots, S_n) - 1) \mathbb{E}_{x \sim \mathcal{D}} \left[(2 f_\theta(x) - 1) \prod_i \mathrm{AND}(S_i) \right] \tag{20}$$

$$= \mathbb{E}_{x \sim \mathcal{D}} \left[(2 f_\theta(x) - 1) \left(\sum_{S_i \in \mathrm{Leaves}(T_i), i \in [n]} (2\widetilde{T}_i(S_i) - 1) \prod_i \mathrm{AND}(S_i) \right) \right]. \tag{21}$$

Since for each input x in the domain, by the definition of a decision tree, exactly one path S_i is traversed for each tree at node i. For these correct path, $\widetilde{T}(S_1, \ldots, S_n) = \widetilde{T}(x)$. Thus we have:

$$\mathrm{val}(\widetilde{T}_1, \ldots, \widetilde{T}_n, v) = \mathbb{E}_{x \sim \mathcal{D}} \left[(2 f_\theta(x) - 1)(2\widetilde{T}(x) - 1) \right], \tag{22}$$

which is the $0 - 1$ loss for the estimated trees.

In our algorithm, we use Hoeffding inequality to approximate v with random sampling. Note that this step requires, in the worst case, with probability at least $1 - \delta/2$, approximating $|\mathcal{S}|^n$ entries of v naively with error at most $\epsilon / \prod_i s_i$ where s_i is a bound on the number of leaves of t_i (naively $|\mathcal{S}|$), and runs in time $\mathrm{poly}(m', n)$ where $m' = \mathrm{poly}(1/\epsilon, \log(|S|^n/\delta))$ is the number of draws to obtain the Hoeffding bound. When choosing $R = r$ in the algorithm, $|\mathcal{S}|$ is of order $2^{O(lr)} (\mathbb{E}\|\varphi\|)^2 \tau^2$ based on the previous bound on $|\mathcal{S}|$ and the approximation of v is done in $\mathrm{poly}(n, l, r, 1/\epsilon, \log(1/\delta))$.

Finally, we can run the dynamic program that computes for each $S_1 \in \mathcal{S}_1$, each tree size $s_1' = 0..s_1$, for each $S_2 \in \mathcal{S}_2$, each tree size $s_2' = 0..s_2$, etc. the best subtrees $\widetilde{T}_i$ of size s_i' rooted at the end of the clause S_i, for all $i \in [n]$. The runtime of this DP is computed as $|\mathcal{S}|^n \cdot s^n \cdot \mathrm{poly}(n, l, r, 1/\epsilon, \log(1/\delta)) = \mathrm{poly}(2^{nlr}, (\mathbb{E}\|\varphi\|)^2 \tau^2, s^n, n, l, r, 1/\epsilon, \log(1/\delta))$.

D Experiments

We report more detailed diagnostics and statistics for our end-to-end implementation of the algorithm in Table 3.

depth	k	paths / vertex	candidate trees / vertex	source agreement
2	10	68/47/88 107/25/25	2/2/2/2/2/2	0.753
2	50	174/159/214 189/201/68	3/3/3/3/3/2	0.748
2	100	200/200/200 200/200/126	2/3/2/2/3/2	0.755
2	200	200/200/200 200/200/210	2/2/2/2/2/2	0.755
3	10	127/86/147 147/25/25	1/1/1/1/1/1	0.903
3	50	235/223/250 247/252/125	1/1/1/1/1/1	0.906
3	100	302/281/293 242/295/209	1/1/1/1/1/1	0.900
3	200	342/313/336 294/312/319	1/1/1/1/1/1	0.907
4	10	180/104/219 183/25/25	2/1/1/1/1/1	0.645
4	50	246/258/345 254/324/212	1/1/1/1/1/2	0.649
4	100	432/391/399 330/294/255	1/1/1/2/1/2	0.652
4	200	424/477/377 424/476/347	1/1/1/2/1/2	0.701
5	10	212/120/277 209/25/25	2/2/2/2/2/2	0.736
5	50	276/303/330 278/368/217	2/3/3/3/3/2	0.740
5	100	444/434/413 319/305/275	3/3/6/2/3/2	0.748
5	200	535/526/475 416/490/379	4/4/5/3/3/2	0.739

Table 3: Diagnostic statistics for Phase 2. "Paths / vertex" is the number of clauses retained for each per-vertex subtree after the validation-error pruning step, and "candidate trees / vertex" is the number of candidate trees generated by the phase-2 DP before the final product search. Source agreement is the agreement between the reconstructed local-iteration algorithm and the trained source network.

D.1 Details on Experimental Setups

All experiments were implemented in Python (v3.12.13). Neural-network models were built using PyTorch (v2.10.0+cu128), graph construction and synthetic benchmark generation handled through NetworkX (v3.6.1). Numerical computation and data processing used NumPy (v2.2.6), SciPy (v1.15.2). We also use tqdm (v4.67.3) and graphviz (v0.21). Models were trained using the optimization and initialization routines provided by the default PyTorch stack, with Adam used as the optimizer in our learned pipelines.

We ran the experiments on Google Colab, and the hardware configuration used is summarized in Table 4.

Table 4: Hardware specifications.

Component	Specification
Architecture	x86_64
OS	Ubuntu 22.04.5 LTS
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz
GPU	NVIDIA A100-SXM4-40GB
GPU memory	40960 MiB (approximately 141 GB)
RAM	83.47 GiB

Table 5 lists the licenses of the main software libraries used in the experiments.

Table 5: Main software licenses.

Software	License
PyTorch	BSD-3-Clause
NetworkX	BSD-3-Clause
NumPy	BSD-3-Clause
SciPy	BSD-3-Clause
tqdm	MPL-2.0 and MIT
graphviz	Eclipse Public License - v 2.0

E LLM Usage Disclosure

We used an LLM to help with code writing and with polishing the paper text.