ON GRAPH TRANSFORMER EXPRESSIVITY: A MONADIC SECOND-ORDER LOGIC PERSPECTIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

We characterize the expressive power of graph transformers by showing that transformers express *monadic second-order logic* (MSO) for bounded-treewidth graphs. MSO is a powerful logic for graph-related tasks, as it allows to decide many problems for graphs with bounded treewidth in linear fixed-parameter-tractable (FPT) time, such as connectivity, planarity, and p-colorability for fixed p. The expressive power of general transformers is often quantified by their ability to simulate certain formal languages in the context of natural language processing. In this paper, we focus on graph representation learning and MSO: We show that transformers with $\log n$-precision and average-hard attention can produce distinct embeddings for bounded-treewidth graphs that differ in MSO-definable properties in near-linear FPT time if the tree decomposition is given as input, and in near-cubic FPT time if the tree decomposition is not given. We establish this expressivity lower bound by showing that transformers have sufficient expressive power to compute bounded-width tree decompositions and simulate tree automata for graphs equipped with simple structural encodings. Our proofs are constructive and provide upper bounds on the required size of the transformers, scaling with the size of the input graph, the treewidth as well as the length of the MSO formula.

1 INTRODUCTION

The expressivity analysis of graph learning algorithms is commonly conducted through the lens of Weisfeiler-Leman (WL) tests, a hierarchy of increasingly expressive color refinement algorithms which test graph isomorphism. Xu et al. (2019) and Morris et al. (2019) have shown that the expressive power of message-passing graph neural networks (GNNs) can be characterized by the 1-dimensional WL test. Transformers can, very broadly, be seen as GNNs with some attention mechanism as an aggregation function (Groß et al., 2023; Sáez de Ocáriz Borde, 2024). However, in practice graph transformer architectures often use additional tools such as positional or structural encodings which increase their expressive power (Black et al., 2024). Indeed, graph transformers can be aligned with the k-WL expressivity hierarchy (Müller & Morris, 2024) and, given sufficiently expressive positional encodings, even be universal (Rosenbluth et al., 2024).

In this paper, we propose to characterize graph transformer expressivity from a novel perspective, following the line of research that investigates the ability of general transformers, including large language models (LLMs), to reason on graph-structured data (de Luca & Fountoulakis, 2024; Sanford et al., 2024; Ye et al., 2024). We show that for certain sparse graph classes, *monadic second-order logic* (MSO) provides a lower bound on graph transformer expressivity. MSO is a powerful logic for graph-related tasks, as it allows to solve problems that are NP-hard for general graphs in linear fixed-parameter tractable (FPT) time for graphs of bounded treewidth. Indeed, our results allow us to conclude that *graph transformers are able to distinguish non-isomorphic graphs of bounded treewidth that differ in any graph property which is definable in MSO in FPT time*. We provide concrete constructions that provide upper bounds and are non-uniform; the size of the transformer grows with the size of the input graph. For our theoretical analysis, we consider average hard attention and $\log$-precision that grows with the size of the input graph. To inject the graph structure, we employ a simple structural encoding of the input graph, i.e., tokens that represent nodes and tokens that represent edges. This setting is complementary to (Ahvonen et al., 2024; 2025), who showed that graph transformers *without* any positional or structural encoding are equivalent to graded modal logic with counting, which is strictly less expressive than MSO. Graph properties that are definable in MSO

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

include p-colorability for fixed p, connectivity, and planarity; see Figure 1a for two connected, planar graphs that differ in their chromatic number – i.e., for G_1 two colors suffice, while G_2 requires three colors – and therefore can be distinguished by the MSO-definable property 2-colorability. Bounded treewidth graphs include a wide variety of natural graph classes such as outerplanar graphs, which comprise the majority of molecular benchmark datasets (Bause et al., 2024, Table 1). Thus, our results provide a theoretical explanation for the success of transformer-based models for, e.g., drug design (Cofala & Kramer, 2021; Mao et al., 2024; Lu et al., 2023; Pirnay et al., 2024).

Main contributions. Our main contributions are:

- We are the first to establish a connection between graph transformer expressivity and MSO by showing that graph transformers with a *basic structural encoding* can express MSO for treewidth-bounded graphs.
- We initiate the study of the *parameterized complexity* of graph transformers by showing that, given a *nice* and *balanced* tree decomposition, graph transformers can express MSO in FPT $\mathcal{O}(n \log n)$ time.
- We show that graph transformers can compute optimal-width tree decompositions in FPT $\mathcal{O}(n^3 \log n)$ time for bounded-treewidth graphs.
- We provide sufficient upper bounds on the graph transformer width and depth required to express MSO via explicit constructions.

Structure of the paper. The remainder of the paper is structured as follows. Section 2 discusses related work; Section 3 introduces notation and preliminaries. Section 4 presents our main result (Theorem 4.3) on the expressive power of transformers to express MSO-definable properties, their ability to compute tree decompositions (Theorem 4.4), and upper bounds on the size of the architecture (Theorem 4.5). Section 5 discusses the implications of our result in the context of graph machine learning, and contextualizes our contribution within the existing literature. Section 6 summarizes our contribution, addresses limitations and considers directions for future work.

2 Related work

In this section, we give an overview on related work on graph transformer expressivity.

WL tests and relation to MPNNs. Graph transformers have been shown to be successful for many graph classification tasks, often surpassing standard message-passing neural networks (MPNNs) (Müller et al., 2024), whose expressive power is upper bounded by the Weisfeiler Leman (WL) color refinement algorithm (Xu et al., 2019; Morris et al., 2019). In general, graph transformers without additional positional encodings have expressive power equivalent to set-based architectures (cf. DeepSets (Zaheer et al., 2017) and are thus less expressive than standard MPNNs or higher-order variants (Zhou et al., 2024), whereas with sufficiently expressive structural encodings graph transformers can become universally expressive (Müller et al., 2024). Recently, several architectures with expressivity guarantees aligned with the WL-hierarchy have been proposed. Ying et al. (2021), Kreuzer et al. (2021) and Rampášek et al. (2022) propose graph transformers with structural encodings and positional encodings that surpass MPNN expressivity. Black et al. (2024) provide a theoretical investigation into the discriminative power of different positional encodings, while Zhu et al. (2023) propose a WL test for graph transformers to study the expressive power of structural encodings. Kim et al. (2022) proposed a hierarchy of graph transformers that aligns with k-IGNs (Maron et al., 2019), which (Müller & Morris, 2024) build upon to propose an even more expressive hierarchy. Müller et al. (2024) showed that the EdgeTransformer (Bergen et al., 2021) is at least as expressive as 3-WL, creating another direct link to classical MPNN expressivity analysis. A recent result by Stoll et al. (2025) shows that the expressive power of graph transformers with different types of positional encodings can be characterized by the generalized-distance WL test (Zhang et al., 2023). Cai et al. (2023) show that MPNNs with virtual nodes can simulate certain graph transformers. Nikolentzos et al. (2025) compare the expressive power of graph transformers with MPNNs. Rosenbluth et al. (2024) showed that MPNNs endowed with the same structural information have equivalent expressive power. In the uniform setting, i.e., one network for graphs of all sizes, Rosenbluth et al. (2024) show that MPNNs and graph transformers are incomparable in expressive power.

Logical characterizations. Ahvonen et al. (2025) precisely quantify the expressive power of *naked* graph transformers, i.e., no structural or positional encodings, in the uniform setting via graded modal logic (GML) with global counting. Barcelo et al. (2024) show that Unique Hard Attention (UHAT) transformer expressivity is lower bounded by first-order logic with modular counting. Müller et al. (2024) utilize the connection of the EdgeTransformer and 3-WL to formalize systematic generalization through first-order logic statements. Inspired by the zero-one convergence law for formal languages, a new line of research studies the expressive power of probabilistic graph classifiers (Adam-Day & Ceylan, 2023; Adam-Day et al., 2024). For their expressivity analysis, (Adam-Day et al., 2024) develop an aggregate term language, which captures several graph transformer architectures and for which they prove asymptotically almost sure convergence.

General transformers on graph reasoning tasks. Sanford et al. (2024); de Luca & Fountoulakis (2024); Ouyang et al. (2024) investigate the ability of general transformers on graph reasoning tasks; Ma et al. (2025) show that plain transformers with some additional encodings perform well at graph tasks, and Merrill & Sabharwal (2025a) show that transformers with $\mathcal{O}(\log n)$ depth are able to decide graph connectivity. Sanford et al. (2024) is the work closest to our analysis. They focus on global graph properties relevant for graph reasoning tasks and provide bounds dependent on the arboricity of a graph, whereas we focus on MSO-definable properties and bounded-treewidth graphs.

3 PRELIMINARIES

In this section, we introduce transformers and several concepts from graph theory, logics, automata theory and complexity theory that are relevant for the remainder of this paper.

Transformers. The transformer architecture consists of alternating blocks of *attention heads* and multi-layer perceptrons (MLPs) and is used for sequence-to-sequence tasks. *Attention* is defined as the softmax function applied element-wise to a linear projection of *queries*, *keys*, and *values*. More formally, a *single attention head* $f_{att} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ that takes feature matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ with sequence length n and feature dimension d as input is defined as

$$f_{att}(\mathbf{X}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_K}}\right)\mathbf{V}, \tag{1}$$

where $\mathbf{Q} = \mathbf{X}\mathbf{W}_\mathbf{Q}$, $\mathbf{K} = \mathbf{X}\mathbf{W}_\mathbf{K}$, and $\mathbf{V} = \mathbf{X}\mathbf{W}_\mathbf{V}$ with learnable weight matrices $\mathbf{W}_\mathbf{Q}, \mathbf{W}_\mathbf{K} \in \mathbb{R}^{d\times d_K}$ and $\mathbf{W}_\mathbf{V} \in \mathbb{R}^{d\times d}$.

Multi-head attention is defined as the concatenation of multiple independent copies of a single attention head followed by a linear layer, which we will denote as f_{att}. An L-layer transformer f_{tf} is defined as the composition of attention heads and MLPs, i.e.,

$$f_{\text{tf}} = f^{(L)}_{\text{mlp}} \circ f^{(L)}_{\text{att}} \circ f^{(L-1)}_{\text{mlp}} \circ \ldots \circ f^{(1)}_{\text{att}} \circ \pi, \tag{2}$$

where π is a *positional encoding*, i.e., a fixed-size vector representation that indicate the positions of the sequence elements. In the context of graph representation learning, graph transformers are transformers that are adapted to operate on graph-structured input data. This usually entails encoding the structure of the graph as positional encodings, such as Laplacian eigenvectors or random-walk based encodings. Please refer to Müller et al. (2024); Black et al. (2024) for an overview of commonly used positional encodings.

Graph theory. We consider undirected graphs. A graph $G = (V, E)$ consists of vertex set V and edge set $E \subseteq V \times V$. Two graphs G, G' are *isomorphic* if there exists a mapping $\psi : V(G) \to v(G')$ that is bijective and edge-preserving. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, can be defined in terms of a tree decomposition. A *tree decomposition* of a graph $G = (V_G, E_G)$ is a pair (T, B), where $T = (V_T, E_T)$ is a tree, and $B = \{B_i \mid i \in V_T\}$ is a family of subsets of V_G, such that (i) $\bigcup_{i\in V_T} B_i = V_G$, (ii) every edge of G has both of its endpoints in some B_i with $i \in V_T$, and (iii) for all $i, j, k \in V_T$, if j lies on the path from i to k in T, then $B_i \cap B_k \subseteq B_j$. The *treewidth* of a tree decomposition is $\max_{i\in V_T} |B_i| - 1$. The treewidth of G is the minimum treewidth among all possible tree decompositions of G; we call (T, B) *balanced* if its depth is bounded by $\mathcal{O}(\log n)$. A *nice* tree decomposition has at most two children per node, and adjacent bags differ by at most

one; please find more details in Theorem A.1 in the Appendix. Intuitively, the treewidth of a graph specifies how tree-like it is; e.g., trees have treewidth 1, cycles have treewidth 2, and complete graphs of size n have treewidth $n-1$; refer to Figures 1b and 1c for an example of a graph and its tree decomposition. A related notion to treewidth is *cliquewidth*, which characterizes the complexity of constructing a graph given a finite set of labels and certain operations to combine nodes with different labels. Cliquewidth is a useful graph parameter as it captures a larger class than treewidth bounded graphs, e.g., complete graphs have cliquewidth 2, but not bounded treewidth.

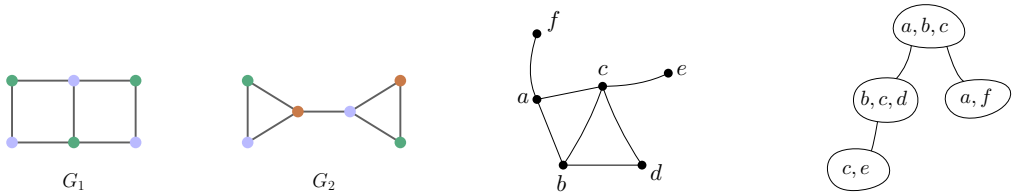


Figure 1: (a) Non-isomorphic graphs G_1 and G_2 that can be distinguished via the MSO-definable property 2-coloring, (b) outerplanar graph G, (c) a tree decomposition of width 2 of G.

Monadic second-order logic. Monadic second-order logic (MSO) is a fragment of second-order logic which allows quantification over sets. We assume familiarity with first-order logic (FO). MSO extends FO with second-order variables of arity 1. An MSO formula can contain both first-order and second-order free variables. We write $\varphi(x_1, \ldots, x_m, X_1, \ldots, X_\ell)$ to denote that x_i for $i \in [m] = \{1, \ldots, m\}$ are free first-order variables and X_j for $j \in [\ell]$ are free second-order variables. In the context of graphs, we often distinguish between MSO_1, which only allows quantification over vertices, and MSO_2, which additionally allows quantification over edges. Note that MSO_2 is more powerful than MSO_1, e.g., we can define the existence of a Hamiltonian cycle in MSO_2, but not in MSO_1. We say that a graph G *satisfies* a formula $\varphi(x_1, \ldots, x_m, X_1, \ldots, X_\ell)$, denoted as $G \models \varphi$, if there exists $v_1, \ldots, v_m \in V(G)$ and $V_1, \ldots, V_\ell \subseteq V(G)$ such that $\varphi(v_1, \ldots, v_m, V_1, \ldots, V_\ell)$ evaluates to true.

For instance, we can define 3-colorability in MSO_1 as follows:

$$\varphi_{3\text{-}color} = \exists X_1 \exists X_2 \exists X_3 \; part(X_1, X_2, X_3) \; \wedge \; \forall v \forall w Evw \implies \bigwedge_{i=1}^{3} (X_i v \wedge X_i w),$$

where Exy means that there is an edge between vertices x and y and

$$part(X_1, X_2, X_3) = \forall v X_1 v \vee X_2 v \vee X_3 v \wedge \forall v \neg (X_1 v \wedge X_2 v) \wedge \neg (X_1 v \wedge X_3 v) \wedge \neg (X_2 v \wedge X_3 v)$$

denotes that X_1, X_2, X_3 are disjoint partitions of the vertex set.

Parameterized complexity. Many NP-hard decision problems can be decided more efficiently if we can make use of the structure of the input instances. Fixed-parameter tractable (FPT) algorithms are polynomial in the input size while allowing for exponential dependence on some structural parameter.

Definition 3.1 (FPT algorithm)**.** An algorithm is *FPT* with respect to some parameter k and instance of size n, if its runtime is bounded by $g(k)\,\text{poly}(n)$ for some computable function g.

For graphs of bounded treewidth, many NP-complete decision problems can be decided in linear time, as shown in the famous meta-theorem by Courcelle (1990).

Theorem 3.2 (Courcelle 1990)**.** *For an MSO_1 sentence φ and a graph G with bounded treewidth k, one can decide whether $G \models \varphi$ in time $g(k, |\varphi|)n$ for some computable function g.*

Theorem 3.2 holds for the larger class of bounded-cliquewidth graphs. For bounded-treewidth graphs, Courcelle's theorem can also be extended to MSO_2. In fact, there are multiple variations of Courcelle's theorem, including optimization problems on graphs (Arnborg et al., 1991; Courcelle et al., 2000) and modulo counting (Courcelle, 1990).

4 TRANSFORMERS EXPRESS MONADIC SECOND-ORDER LOGIC

In this section, we present our main result: Graph transformers with a basic structural encoding can express any graph property that is definable in MSO and are thus able to decide membership for treewidth-bounded graphs for, e.g., p-colorability for fixed p and graph connectivity, in FPT $\mathcal{O}(n^3 \log n)$ time; we can reduce this to FPT $\mathcal{O}(n \log n)$ time if a balanced, nice tree decomposition of the graph is given. Our proof technique comprises a series of problem reductions and heavily relies on fundamental results in logic and automata theory. We consider $\mathcal{O}(\log n)$-precision and *average-hard attention* (Barcelo et al., 2023). We present all of our results for standard transformers and remark that we can reduce the number of layers to $\mathcal{O}(1)$ if we consider looped transformers instead. If not stated otherwise, we consider the treewidth k as a constant, and present results only in their dependence to the graph size n.

The remainder of this section is structured as follows. First, we show our main result in Section 4.1: Given any bounded treewidth graph G of size up to n and any MSO formula φ, there exists a graph transformer that can decide whether $G \models \varphi$. A crucial aspect for this result to hold for all treewidth-bounded graphs is that graph transformers can construct optimal-width tree decompositions, which we show in Section 4.2. Next, we investigate what transformer size is sufficient to express MSO-definable properties in Section 4.3, where we provide upper bounds that depend on the size of the input graph, the treewidth and the length of the MSO formula.

4.1 TRANSFORMERS EXPRESS MONADIC SECOND-ORDER LOGIC

Before we state our main theorem, we need to define how we encode our input graphs and what we mean by *expressing* an MSO-definable property. Graph transformers without *any* positional or structural encoding are not able to distinguish any pair of non-isomorphic graphs of the same size. Their expressive power has already been characterized by, e.g., (Ahvonen et al., 2025); therefore, in our setting we decided to incorporate *some* structural information by encoding nodes and edges as individual tokens, closely following the one presented in Sanford et al. (2024).

Definition 4.1 (Graph encoding)**.** Given graph $G = (V, E)$, we represent node and edge tokens as

$$x_{v_i} = (i, 0, 1, 0) \quad \text{and} \quad x_{e_{ij}} = (i, j, 0, 1),$$

and graph G as $X_G = (x_{v_1}, x_{v_2}, \ldots, x_{v_{|V|}}, x_{e_1}, x_{e_2}, \ldots, x_{e_{|E|}}) \in \mathbb{R}^{\mathcal{O}(|V|+|E|)}$.

We encode the structure directly as part of the input instead of adapting the attention mechanism, such as de Luca & Fountoulakis (2024), to keep the transformer architecture as general as possible to understand how the size of the input graph affects the size of the transformer. We argue that our encoding is the smallest amount of structural information that still represents the graph entirely. This is also demonstrated by the fact that the size of the input remains linear in the graph size, i.e., $m = \mathcal{O}(n)$ for treewidth-bounded graphs. Next, we formalize the notion of *expressibility*.

Definition 4.2 (Expressibility)**.** Let $\mathcal{G}^{n,k}$ be all graphs of treewidth k up to size n, let P be a graph property, and let $P(G) = \mathbf{1}_{[G \text{ has property } P]}$. We say that a function $f : \mathcal{G}^n \to \mathbb{R}^d$ *expresses* P if for each pair $(G, G') \in \mathcal{G}^n \times \mathcal{G}^n$, $f(G) = f(G')$ iff $P(G) = P(G')$.

For instance, the function $f(G) = \mathbf{1}_{[n \equiv 1 \bmod 2]}$ expresses the graph property *odd number of vertices* and the function $f'(G) = \mathbf{1}_{[G \models \varphi_{2\text{-}color}]}$ expresses the property of *2-colorability*; for the graphs depicted in Figure 1a, we therefore have $f(G_1) = f(G_2)$ as both graphs have an even number of vertices, but $f'(G_1) \neq f'(G_2)$ as G_2 requires at least three colors. We remark that both properties are MSO-definable, and that functions f, f' thus induce MSO-definable equivalence relations on $\mathcal{G}^{n,k}$.

We can now state our main theorem as follows.

Theorem 4.3. *Fix $k \in \mathbb{N}$. For every $n \in \mathbb{N}$ and every MSO-definable property P, there exists a graph transformer $GT : \mathcal{G}^{n,k} \to \mathbb{R}^d$ that expresses P in FPT $\mathcal{O}(n^3 \log n)$ time for any graph $G \in \mathcal{G}^{n,k}$. Given a nice, balanced tree decomposition (T, B) of width k, we only require FPT $\mathcal{O}(n \log n)$ time.*

Proof sketch. To prove Theorem 4.3, we exploit the correspondence between MSO-definability and finite automata. More specifically, we closely follow one proof method that is commonly employed

to prove Courcelle's theorem. Given some input graph G of bounded treewidth k, we first compute a tree decomposition (T, B) of width $\mathcal{O}(k)$. We then map G and (T, B) to a labeled, ordered, binary tree t that encodes the structure of G. Let φ be the MSO sentence that defines graph property P. Then, there exists a deterministic tree automaton $\mathcal{A}$ that accepts t if and only if G has property P. The remainder of the proof consists of showing that, given graph G and treewidth k, there exists a graph transformer that (i) constructs a width-k tree decomposition (T, B) of G in FPT $\mathcal{O}(n^3 \log n)$ time, (ii) constructs a labeled, ordered, binary tree t, given a tree decomposition (T, B) of depth $\mathcal{O}(d_t)$, in FPT $\mathcal{O}(nd_t)$ time, and (iii) simulates a run of a deterministic tree automaton on a tree t of depth $\mathcal{O}(\log n)$ in FPT $\mathcal{O}(n \log n)$ time. Please find more details on $(i) - (iii)$ and the full proof in Section A.6. □

The main technical novelty of our contribution lies in $(i) - (ii)$, i.e., showing that transformers are able to compute optimal-width tree decompositions and Σ-trees. For (iii), our results are similar to, but not subsumed, by Liu et al. (2022), who showed that transformers learn shortcuts for deterministic finite automata, and Rizvi-Martel et al. (2024), who showed that transformers are able to simulate *weighted* tree automata for *leaf-labeled* trees. Our result differs from the former by considering *tree* automata, and from the latter by assuming *general* labeled trees; see Section A.4 for more details.

We want to further point out that Theorem 4.3 is a purely *existential* result: We know that for each MSO-definable property P and each treewidth-bounded graph G, there exists *some* graph transformer, with *some* sets of weights, that decides whether G has property P. Theorem 4.3 does not provide bounds on the required size of such a transformer, nor any guarantees on whether finding this set of weights is achievable via standard optimization techniques. While we do not further address the latter issue in this paper and leave it for future work, we will provide an upper bound on the size of the transformer that is *sufficient* for Theorem 4.3 to hold in Section 4.3.

4.2 Transformers construct tree decompositions

Next, we present a crucial result for our main finding: Transformers can compute optimal-width tree decompositions in FPT $\mathcal{O}(n^3 \log n)$ time.

Theorem 4.4. *Fix $k \in \mathbb{N}$. For every $n \in \mathbb{N}$, there exists a graph transformer $GT : \mathcal{G}^{n,k} \to \mathcal{G}^{n,k}$ with $\mathcal{O}(n \log n)$ layers, embedding dimension $\mathcal{O}(\log n)$, MLP width $\mathcal{O}(1)$ and $\mathcal{O}(1)$ attention heads that computes a nice and balanced tree decomposition of width $\mathcal{O}(k)$ in FPT $\mathcal{O}(n^3 \log n)$ time.*

Proof sketch. For the proof, we show that graph transformers are able to simulate all subroutines necessary for Bodlaender's linear time algorithm (Bodlaender, 1993). The main idea of the algorithm is to classify vertices into low-degree and high-degree and call different subroutines depending on which class they fall into, to iteratively reduce the size of the input graph by a constant factor in each step to ensure linear runtime. When the reduced graph is small enough, the algorithm brute-forces its tree decomposition or returns that $\mathrm{tw}(G) > k$. Next, the recursion is unfolded and the computed tree decomposition is step-wise adapted to the original input graph G. The main difficulty of the proof is to ensure that while unfolding the recursion, the original graph can be reconstructed step by step, as this is necessary for some of the required subroutines. □

Theorem 4.4 enables us to state our main result for all treewidth-bounded graphs. An immediate consequence of Theorem 4.4 is that graph transformers can decide whether a given graph G has treewidth less or equal to some fixed k with the same time complexity.

Corollary 4.1. *Fix $k \in \mathbb{N}$ and let $\mathcal{G}^{n,k}$ be all graphs up to size n. For every $n \in \mathbb{N}$, there exists a graph transformer $GT : \mathcal{G}^{n,k} \to \mathbb{R}^d$ that decides whether graph G has treewidth at most k in FPT $\mathcal{O}(n^3 \log n)$ time.*

4.3 An upper bound for sufficient transformer size

Finally, we now provide precise bounds on the *sufficient* size of a transformer to realize the expressive power of the graph transformer in Theorem 4.3.

Theorem 4.5. *Let $G \in \mathcal{G}^{n,k}$, (T, B) be its tree decomposition of width k, fix MSO-definable property P and let $s = g(k, P)$ for some computable function g. Then, there exists a graph transformer*

$GT : \mathcal{G}^{n,k} \to \mathbb{R}^d$ with $\mathcal{O}(\log n)$ layers, embedding dimension $\mathcal{O}(s \log n)$, MLP width $\mathcal{O}(s^3)$ and $\mathcal{O}(1)$ attention heads that expresses P.

Proof sketch. We first transform (T, B) and G into a labeled, ordered, binary tree t that encodes the graph structure. Let φ be the MSO formula expressing property P, and let φ^* be its translation such that $G \models \varphi$ iff $t \models \varphi^*$. Then, there exists an automaton $\mathcal{A}_{\varphi^*}$ with $g(k, |\varphi|)$ states that accepts or rejects t. As shown in Theorem A.4, we can construct a transformer with $\mathcal{O}(s)$ embedding dimension, $\mathcal{O}(1)$ attention heads, attention width $\mathcal{O}(s)$, MLP of width $\mathcal{O}(s^3)$, $\mathcal{O}(\log n)$ layers that simulates $\mathcal{A}_{\varphi^*}$. □

An important implication of Theorem 4.5 is that, given a nice and balanced tree decomposition, only the depth of the transformer scales logarithmically with the size of the input graph. This means that if we consider a looped transformer, our parameter count is independent of the graph size, and only depends on the treewidth k and the length of the MSO sentence that defines P. We remark that the bounds that stem from our constructive proof are a worst-case analysis, and often larger than the unique minimal deterministic automaton; see Table 1 (*right*) for automata constructions for several MSO-definable properties with a manageable number of states for graphs of small cliquewidth.

5 Implications for graph representation learning & beyond

In this section, we discuss the implications of our results in the wider context of graph representation learning. Consistent with existing work, we define the expressivity of graph learning algorithms as their ability to distinguish between pairs of non-isomorphic graphs. Our results imply that graph transformers, given that they are sufficiently large, have sufficient expressive power to learn graph equivalence classes induced by MSO-definable properties, and thus might be able to distinguish pairs of non-isomorphic graphs that differ in MSO-definable properties.

In the context of graph classification and regression, bounded treewidth is a reasonable assumption, as, e.g., the vast majority of molecules is outerplanar (see Bause et al. 2024, Table 1 for instance) and thus has treewidth $k \leq 2$. Our results can even be extended to bounded-cliquewidth graphs if we consider the term expression of the graph as given; alternatively, we can also use parse-trees based on a rank decomposition as an approximation (Ganian & Hlinený, 2008). We formalize this as follows.

Corollary 5.1. *Fix $k \in \mathbb{N}$ and let $\mathcal{G}^{n,k}$ be all graphs of cliquewidth k up to size n. For every $n \in \mathbb{N}$ and every MSO$_1$-definable property P, there exists a graph transformer $GT : \mathcal{G}^{n,k} \to \mathbb{R}^d$ that expresses P in FPT $\mathcal{O}(n^2 \log n)$ time for any graph $G \in \mathcal{G}^n$, given a balanced term expression of width k.*

Proof. Bounded-cliquewidth graphs admit MSO$_1$ model-checking over width-k term expressions similar to bounded-treewidth graphs (Courcelle & Engelfriet, 2012). Since Theorem A.4 shows that graph transformers can simulate the resulting tree automata within the stated bounds, the claim follows for balanced term expressions. □

Graphs with bounded cliquewidth include, e.g., complete graphs, whose cliquewidth is 2.

Next, we briefly highlight some interesting MSO properties that graph transformers can express, providing a theoretical explanation for the experimental results on connectivity, Hamiltonian paths and perfect matchings in, e.g., Wang et al. (2023); Sanford et al. (2024); Ouyang et al. (2024); Zhang et al. (2024).

Corollary 5.2. *Graph transformers can express each of the following properties for treewidth-bounded graphs: (i) connectivity, (ii) p-colorability for fixed p, (iii) minor inclusion, (iv) planarity, (v) Hamiltonicity, and (vi) perfect matching.*

Proof. This follows from the fact that connectivity, p-colorability for fixed p, minor inclusion (and thus planarity), Hamiltonicity, and perfect matching are MSO-definable properties (Courcelle & Engelfriet, 2012). □

To provide some more intuition for Corollary 5.2, let us consider the following example. Figure a shows two graphs G, G' of size $2n$ which form a specific instance of the following construction: For odd n, G is made of two cycle graphs C_{n+1} which share an edge and G' is made of two cycle graphs C_n which are connected by a single edge. where G and G' are always 1-WL-equivalent. However, the two graphs can be distinguished by graph transformers since G is 2-colorable and G' is 3-colorable, and p-colorability is an MSO-definable property.

Space considerations. In general, the size bounds for transformers obtained from Theorem 4.5 can be impractically large due to their nonelementary dependence on the quantifier depth of formula. However, for some MSO-definable properties that can be expressed with sentences that have little quantifier nesting, the resulting automata may have manageable size. Indeed, in Table 1 (*left*) we provide upper bounds for tree automata for some useful MSO-expressible graph properties; please refer to Courcelle & Engelfriet (2012) for more details on the computation. We remark that minor inclusion characterizes planarity. Note that the complexity of the automata is even smaller if we have a bound on the degree, which often is the case for real-life graphs such as, e.g., molecules or transport networks. In Table 1 (*right*) we present the actual number of states for manually designed automata for several of these properties, and observe that they have a feasible number of states for graphs of small cliquewidths.

Table 1: Some useful MSO-definable graph properties and the number of states in the corresponding deterministic tree automata for cliquewidth bounded graphs; for *minor inclusion*, we show bounds for non-deterministic automata. *Left.* Asymptotic upper bounds *Right.* Hand-crafted automata for graphs of small cliquewidth; values taken from Courcelle & Durand (2010); Langer (2013).

Graph property	# of states, $k = \mathrm{cw}$
Connectivity	$2^{2^{\Theta(k)}}$
Connectivity with max degree d	2^{dk^2}
p-colorability	$2^{\mathcal{O}(pk)}$
Minor inclusion[4]	$2^{2^{\mathcal{O}(k)p}+\mathcal{O}(k)p^2}$
Minor inclusion with max degree d[4]	$2^{\mathcal{O}(dk^2p+kp)}$
Path(x, y)	2^{k^2+2}

Graph property	$\mathrm{cw} = 2$	$\mathrm{cw} = 3$
connected	11	-
2-colorable	8	56
3-colorable	21	-
3-clique	11	41
4-clique	21	135
vertex cover ≤ 3	63	414
vertex cover ≤ 4	111	1037
path(x, y)	12	128

Comparison to message-passing graph neural networks. We briefly contextualize our result with respect to the expressive power of standard message-passing graph neural networks (MPNNs). MPNNs are upper bounded in expressive power by the color refinement algorithm 1-WL; for instance, they cannot distinguish the pair of non-isomorphic graphs presented in Figure 1a. However, their higher-order variants k-GNNs, whose expressivity is limited by the k-WL hierarchy, are complete for treewidth-bounded graphs, i.e., k-WL can distinguish all graphs of treewidth $k-1$. While this is a stronger result than expressing MSO-definable properties, the complexity of k-WL scales exponentially. Indeed, if we require a bound on the number of iterations, the best known upper bound is that $(3k+6)$-WL[1] can distinguish all non-isomorphic pairs, and thus also all MSO-definable properties, in $\mathcal{O}(\log n)$ iterations (Levet et al., 2024), leading to an overall runtime of $\mathcal{O}(n^{\mathcal{O}(k)} \log n)$. Our result, on the other hand, implies that graph transformers can represent all equivalence classes induced by MSO-definable properties in $\mathcal{O}(n^3 \log n)\ g(k)$ for some computable function g.

Contextualization with general transformer expressivity results. We briefly discuss our results in light of existing work, that shows that transformers are able to simulate Turing machines (Merrill & Sabharwal, 2024; Li & Wang, 2025; 2026). Tree decomposition decision problems are in LOGCFL (=SAC^1) which is equivalent to the class of all decision problems recognized by an alternating Turing machine (ATM) using logarithmic space and having polynomially sized accepting computation trees.

[4] for minor of size p

[1] And $(4k+3)$ for $k \leq 3$ (Grohe & Verbitsky, 2006)

Elberfeld et al. (2012) show that MSO-definable problems are solvable by AC^0 for decision problems and TC^0 for counting problems when restricted to input structures of *bounded treedepth* and solvable by NC^1 for decision problems and $\#\mathrm{NC}^1$ for counting problems when a tree decomposition of bounded width is part of the input. Merrill & Sabharwal (2025b) show that padded transformers with $\mathcal{O}(\log^d n)$ looping on input sequences of length n can be precisely characterized by FO-uniform TC^d, implying that $\mathcal{O}(\log n)$ looping is sufficient to reach $\mathrm{TC}^1 \supseteq \mathrm{NC}^1$ expressive power. However, decision-class inclusions alone do not give an explicit transformer construction of a tree decomposition. Our construction addresses this search aspect by producing a decomposition in FPT $\mathcal{O}(n^3 \log n)$ time. Merrill & Sabharwal (2025a) also show that looped transformers can express regular languages, which is equivalent to MSO-expressibility on words. Sanford et al. (2024) show that transformers with logarithmic depth can solve graph connectivity.

6 Discussion & Conclusion

Our main goal was to provide novel expressivity results for transformers that have direct applicability to learning problems on graphs. For this, we analyzed the expressive power of graph transformers by showing that they are able to capture MSO-expressible properties for treewidth bounded graphs. MSO is a powerful fragment of second-order logic and can be used to define many interesting graph properties, such as connectivity and p-colorability for fixed p. Our results imply that transformers can produce distinct embeddings for treewidth-bounded graphs that differ in MSO-definable properties. Our contribution provides further evidence that transformers are good at capturing *global* graph properties, such as graph connectivity or Hamiltonicity, while MPNNs are limited to local properties. This provides a theoretical explanation for the empirical success of many graph transformers.

- (i) MSO is a natural logic to reason about graphs, and can be extended to higher-order structures,
- (ii) we provide precise constructive bounds on the size of the required transformer, and finally,
- (iii) we initiate a new direction of research to describe the expressive power of transformers through their ability to realize algorithmic meta-theorems.

Limitations. We next highlight the limitations of our contribution and propose possible ways to address them. We study the expressive power of graph transformers in the non-uniform setting for the restricted graph class of *treewidth bounded* graphs, and for the larger class of cliquewidth bounded graphs, respectively, for all MSO_1-definable properties. We consider $\mathcal{O}(\log n)$ precision and average-hard attention, which is an idealized setting commonly used for theoretical analyses. Furthermore, the upper bounds we provide on the size of the transformer have, while polynomial in the size of the graph, impractically large constants and are thus mainly interesting from a theoretical point of view. While there are likely smaller automata (see, e.g., Table 1), and therefore smaller transformers, for many MSO-definable properties, it is unclear whether gradient-based optimization techniques are generally able to find the weights that realize them. Finally, our tree decomposition construction is not permutation-invariant, i.e., different vertex orderings of the same graph might result in different tree decompositions. We believe this is an important shortcoming to mention, although this does not affect our main result.

Future work. We believe our contribution opens several interesting directions for future work. First of all, we should be able to generalize our results from undirected graphs to arbitrary relational structures by working with their *Gaifman graphs*. Next, we would like to extend this type of expressivity analysis to further restricted graph classes; for instance, existing results on transformer expressivity being characterized by first-order logic suggest that constant-depth transformers could be particularly powerful for sparse graph classes, such as nowhere dense or bounded expansion graphs. Our current results provide an upper bound on the required size of a transformer that is lower bounded in expressivity; it would be interesting to investigate whether this bound is tight. Finally, a systematic experimental evaluation with state-of-the-art graph transformers of varying sizes for graphs that differ in MSO-expressible properties would help to further close the gap between our theoretical contribution and practically deployed machine learning models.

AI USE STATEMENT

In this work, we used generative AI tools to assist with technical aspects of the research, including drafting and refining portions of proofs, checking proof structure, receiving feedback on mathematical exposition, and improving the clarity of selected sentences. We also used generative AI tools for feedback during the final revision stage, including identifying unclear passages, suggesting improvements to readability, and helping refine presentation.

We did not use generative AI tools to originate the main research idea, determine the high-level direction of the work, develop the central conceptual contribution, or write the main narrative of the paper. The core ideas, research direction, and overall structure were developed by the authors without the use of generative AI. Apart from a small number of sentences that were rewritten with the help of AI, the main paper text was written by the authors. Tasks such as generating synthetic datasets, designing experiments, interpreting empirical results, translation, cleaning or reformatting datasets, and qualitative or thematic data analysis are not applicable to this work.

We have reviewed all AI-assisted work. In particular, all AI-assisted proof drafts, mathematical suggestions, and exposition edits were manually checked by the authors for correctness, consistency with the paper's arguments, and fit with the intended claims. Any AI-generated or AI-suggested mathematical content was independently verified before inclusion, and no AI output was incorporated without authors review and revision. We take responsibility for the final content of this work, including text, claims, proofs, and any other material produced with the aid of generative AI.

We ensured that the content of this submission was not used for further training of AI to not bias the reviewing process.

Finally, we have written this statement with AI.

REPRODUCIBILITY STATEMENT

All our theoretical statements are supported by proofs, which can be found in Section A, and/or pointers to existing literature.

REFERENCES

Sam Adam-Day and Ismail Ceylan. Zero-one laws of graph neural networks. *Advances in Neural Information Processing Systems*, 36:70733–70756, 2023.

Sam Adam-Day, Michael Benedikt, Ismail Ilkan Ceylan, and Ben Finkelshtein. Almost surely asymptotically constant graph neural networks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical characterizations of recurrent graph neural networks with reals and floats. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Veeti Ahvonen, Maurice Funk, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Expressive power of graph transformers via logic. *arXiv preprint arXiv:2508.01067*, 2025.

Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. ISSN 0196-6774.

Pablo Barcelo, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. Logical languages accepted by transformer encoders with hard attention, 2023.

Pablo Barcelo, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir Podolskii. Logical languages accepted by transformer encoders with hard attention. In *The Twelfth International Conference on Learning Representations*, 2024.

Franka Bause, Fabian Jogl, Patrick Indri, Tamara Drucks, Nils Morten Kriege, Thomas Gärtner, Pascal Welke, and Maximilian Thiessen. Maximally expressive GNNs for outerplanar graphs. In *TMLR*, 2024.

Leon Bergen, Timothy O'Donnell, and Dzmitry Bahdanau. Systematic generalization with edge transformers. *Advances in Neural Information Processing Systems*, 34:1390–1402, 2021.

Mitchell Black, Zhengchao Wan, Gal Mishne, Amir Nayyeri, and Yusu Wang. Comparing graph transformers via positional encodings, 2024.

Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In J. van Leeuwen (ed.), *Graph-Theoretic Concepts in Computer Science*, pp. 1–10, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. ISBN 978-3-540-46076-3.

Hans L Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pp. 226–234, 1993.

Hans L Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.

Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between mpnn and graph transformer, 2023.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Tim Cofala and Oliver Kramer. Transformers for molecular graph generation. In *ESANN*, 2021.

Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. ISSN 0890-5401.

Bruno Courcelle and Irène Durand. Tractable constructions of finite automata from monadic second-order formulas. Logical Approaches to Barriers in Computing and Complexity, Greifswald, Germany, February 17–20, 2010, 2010. `https://www.labri.fr/perso/courcell/Conferences/ExpoGreifs2010.pdf`, last accessed: 2026-05-05.

Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

Bruno Courcelle, Johann A Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

Artur Back de Luca and Kimon Fountoulakis. Simulation of graph algorithms with looped transformers, 2024.

Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Springer, 2006.

Robert Ganian and Petr Hlinenỳ. Automata approach to graphs of bounded rankwidth. *IWOCA'08*, pp. 4–15, 2008.

Martin Grohe and Oleg Verbitsky. Testing graph isomorphism in parallel by playing a game, 2006.

Joschka Groß, Gerrit Großmann, and Verena Wolf. Elucidating the relationship between transformers and gnns. 2023.

Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners, 2022.

Stephan Kreutzer. Algorithmic meta-theorems. *Finite and algorithmic model theory*, 379:177–270, 2011.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21618–21629. Curran Associates, Inc., 2021.

Alexander Johannes Langer. *Fast algorithms for decomposable graphs*. PhD thesis, Aachen, Techn. Hochsch., Diss., 2013, 2013.

Michael Levet, Puck Rombach, and Nicholas Sieger. Canonizing Graphs of Bounded Rank-Width in Parallel via Weisfeiler-Leman. In Hans L. Bodlaender (ed.), *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 32:1–32:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-318-8. doi: 10.4230/LIPIcs.SWAT.2024.32. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SWAT.2024.32.

Qian Li and Yuyi Wang. Constant bit-size transformers are turing complete. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 62273–62292. Curran Associates, Inc., 2025. doi: 10.52202/085713-2083. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/59d501de2e283999feb3bbca259ca4d1-Paper-Conference.pdf.

Qian Li and Yuyi Wang. Efficient turing machine simulation with transformers. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 112470–112485, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/b6cd2650926d332c86a84c48529cc421-Paper-Conference.pdf.

Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004.

Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

Hao Lu, Zhiqiang Wei, Xuze Wang, Kun Zhang, and Hao Liu. Graphgpt: A graph enhanced generative pretrained transformer for conditioned molecular generation. *International Journal of Molecular Sciences*, 24(23), 2023. ISSN 1422-0067.

Liheng Ma, Soumyasundar Pal, Yingxue Zhang, Philip H. S. Torr, and Mark Coates. Plain transformers can be powerful graph learners, 2025.

Jiashun Mao, Jianmin Wang, Amir Zeb, Kwang-Hwi Cho, Haiyan Jin, Jongwan Kim, Onju Lee, Yunyun Wang, and Kyoung Tai No. Transformer-based molecular generative model for antiviral drug design. *Journal of Chemical Information and Modeling*, 64(7):2733–2745, 2024. PMID: 37366644.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

William Merrill and Ashish Sabharwal. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a.

William Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b.

Gary L Miller and John H Reif. Parallel tree contraction and its application. In *FOCS*, volume 26, pp. 478–489, 1985.

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 *The Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 4602–4609. AAAI Press, 2019.

Luis Müller and Christopher Morris. Aligning transformers with weisfeiler-leman. In *Forty-first International Conference on Machine Learning*, 2024.

Luis Müller, Daniel Kusuma, Blai Bonet, and Christopher Morris. Towards principled graph transformers. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 126767–126801. Curran Associates, Inc., 2024. doi: 10.52202/079017-4026.

Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers, 2024.

Giannis Nikolentzos, Dimitrios Kelesis, and Michalis Vazirgiannis. On the theoretical expressive power of graph transformers for solving graph problems. *Neural Networks*, pp. 108112, 2025.

Sheng Ouyang, Yulan Hu, Ge Chen, and Yong Liu. Gundam: Aligning large language models with graph understanding, 2024.

Jonathan Pirnay, Jan G. Rittig, Alexander B. Wolf, Martin Grohe, Jakob Burger, Alexander Mitsos, and Dominik G. Grimm. Graphxform: Graph transformer for computer-aided molecular design with application to extraction, 2024.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Michael Rizvi-Martel, Maude Lizaire, Clara Lacroce, and Guillaume Rabusseau. Simulating weighted automata over sequences and trees with transformers. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li (eds.), *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pp. 2368–2376. PMLR, 02–04 May 2024.

Eran Rosenbluth, Jan Tönshoff, Martin Ritzert, Berke Kisin, and Martin Grohe. Distinguished in uniform: Self attention vs. virtual nodes, 2024.

Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding transformer reasoning capabilities via graph algorithms, 2024.

Timo Stoll, Luis Müller, and Christopher Morris. Generalizable insights for graph transformers in theory and practice. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Haitz Sáez de Ocáriz Borde. Elucidating graph neural networks, transformers, and graph transformers. 02 2024.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 30840–30861. Curran Associates, Inc., 2023.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations*, 2019.

Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. Language is all a graph needs, 2024.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation?, 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Bohang Zhang, Shengjie Luo, Liwei Wang, and Di He. Rethinking the expressive power of gnns via graph biconnectivity. *arXiv preprint arXiv:2301.09505*, 2023.

Qifan Zhang, Xiaobin Hong, Jianheng Tang, Nuo Chen, Yuhan Li, Wenzhong Li, Jing Tang, and Jia Li. Gcoder: Improving large language model for generalized graph problem solving. *arXiv preprint arXiv:2410.19084*, 2024.

Cai Zhou, Rose Yu, and Yusu Wang. On the theoretical expressive power and the design space of higher-order graph transformers, 2024.

Wenhao Zhu, Tianyu Wen, Guojie Song, Liang Wang, and Bo Zheng. On structural expressive power of graph transformers. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 3628–3637, 2023.

A Appendix

Please find notation, the precise setting of our transformer architecture and the full proofs for all theoretical results in this section.

A.1 Notation and preliminaries

We use $\oplus$ to denote vector concatenation. We denote matrices $\mathbf{X}, \mathbf{Y}$ by bold upper case letters, and vectors $\mathbf{x}, \mathbf{y}$ by bold lower case letters. $\mathbf{I}$ denotes the identity matrix. We write $\mathbf{X}_i$ to denote the vector $\mathbf{x}_i$ in row i of matrix $\mathbf{X}$, and $\mathbf{X}_{ij}$ to denote the element in row i and column j. We write $\mathbf{e}_i = (0, \ldots, 1, \ldots, 0)$ to denote the i-th standard basis vector.

Definition A.1 (Nice tree decomposition)**.** A *nice* tree decomposition is a rooted tree decomposition that has exactly four different types of nodes: (i) *Leaf* nodes, i.e., nodes without any children (ii) *Introduce* nodes whose bags contain exactly one vertex more than their single child node's bag, (iii) *Forget* nodes whose bags contain exactly one vertex less than their single child node's bag, and (iv) *Join* nodes that have exactly two children whose bags contain exactly the same vertices as the bag of the join node itself. The bags of leaf nodes and the root node are empty.

A.2 Formal transformer model

In this section, we introduce the formal transformer model that we use throughout our theoretical analyses.

The transformer architecture consists of alternating blocks of *attention heads* and multi-layer perceptrons (MLPs) and is used for sequence-to-sequence tasks. A *single attention head* $f_{\text{att}} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ that takes feature matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ with sequence length n and feature dimension d as input is defined as

$$f_{\text{att}}(\mathbf{X}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_K}}\right)\mathbf{V}, \tag{3}$$

where $\mathbf{Q} = \mathbf{X}\mathbf{W}_\mathbf{Q}$, $\mathbf{K} = \mathbf{X}\mathbf{W}_\mathbf{K}$, and $\mathbf{V} = \mathbf{X}\mathbf{W}_\mathbf{V}$ with learnable weight matrices $\mathbf{W}_\mathbf{Q}, \mathbf{W}_\mathbf{K} \in \mathbb{R}^{d\times d_K}$ and $\mathbf{W}_\mathbf{V} \in \mathbb{R}^{d\times d_V}$. In general, the attention mechanism computes scores for all pairs of tokens. However, if the input structure has some sparsity constraint, we are able to restrict the attention mechanism to a subset of tokens, which we refer to as *sparse attention* (Child et al., 2019). This can be realized via *masking*. We define a mask as

$$\mathbf{M}^{h,\ell}(X) \in \{0, -\infty\}^{N\times N}$$

where $N = n + m = \mathcal{O}(n)$ denotes the sequence length, and compute masked attention as

$$f_{\text{att}}(\mathbf{X}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_K}} + \mathbf{M}(\mathbf{X})\right)\mathbf{V}.$$

Multi-head attention is defined as the concatenation of multiple independent copies of a single attention head followed by a linear layer. An L-layer transformer f_{tf} is defined as the composition of (one or multiple) attention heads and MLPs, i.e.,

$$f_{\text{tf}} = f_{\text{mlp}}^{(L)} \circ f_{\text{att}}^{(L)} \circ f_{\text{mlp}}^{(L-1)} \circ \ldots \circ f_{\text{att}}^{(1)} \circ \pi, \tag{4}$$

where π is a *positional encoding*, i.e., a fixed-size vector representation that indicate the positions of the sequence elements.

For simplicity of exposition of the proof, we use masking when appropriate to exploit structural assumptions on the input. We stress that every structural mask is computable from the input representation alone and thus does not make our formal transformer model computationally more powerful. We remark that, equivalently, we could implement structural masks by adding sufficiently large negative attention biases. The magnitudes of these biases are polynomially bounded in n, and thus compatible with our $\mathcal{O}(\log n)$ precision assumption. For our theoretical analyses we consider *averaging hard attention* (AHAT) which returns the uniform average of positions with maximum attention scores. We do not consider residual layers or layer normalization to keep our transformer model as simple as possible. We consider finite precision with $\mathcal{O}(\log n)$ bits for input graphs of size $|V| = n$.

In some our proofs we will consider looped transformers as defined in Merrill & Sabharwal (2025a), as they allow for a more elegant proof and are more parameter-efficient. We do not distinguish between initial layers, repeated layers, and final layers, but assume that all layers are repeated.

A.3 Base encoding and transformer primitives

In this section, we introduce our base encoding and collect transformer primitives necessary for our proofs.

Base encoding. Given N tokens, we encode their positions i as $i^* = (\omega_i, \omega_i^2, 1)$ with $\omega_i = \frac{i}{N+1}$. More specifically, we encode node positions x_{v_i} as (i^*) and edge positions $x_{v_iv_j}$ as (i^*, j^*, p^*_{ij}), where p^*_{ij} denotes the position of edge v_iv_j in the overall graph encoding. Additionally, we will often make use of bit flags or pointers; we introduce them when applicable and remark that each token has a constant size independent of the graph size.

Copying tokens. Assuming we do not have residual connections, we can copy a state from layer h_{i-1} to layer h_i by using one attention head f_{copy} with

$$\mathbf{W_Q} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \ \mathbf{W_K} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \text{ and } \mathbf{W_V} = \mathbf{I}.$$

This way $\mathbf{Q}_j\mathbf{K}_i^T = 2\omega_j\omega_i - \omega_j^2 - \omega_i^2 = -(\omega_i - \omega_j)^2$, which is maximized when $i = j$, and thus leads to each token selecting itself, and then copying its state with $\mathbf{V}$.

Selecting tokens based on bit flags. In our proofs, we often make use of bit flags, i.e., token positions that are either set to 1 if a certain action is required, and 0 otherwise. We can select tokens based on some bit flag $b_i = 1$ by using an MLP f_{select} that acts as a binary gate. Assume that b_i is at position k for each token x_i, i.e., $x_{i,k} = b_i$. Then, we can use the linear map $\mathbf{e}_k^T\mathbf{x}_i$ to extract b_i. If we want to gate a bounded coordinate $z \in [-B, B]$ by the bit b_i, i.e. output $b_i z$, this can be implemented by a constant-width ReLU MLP via

$$g_B(z, b_i) = ReLU(z) - ReLU(z - Bb_i) - ReLU(-z) + ReLU(-z - Bb_i).$$

Indeed, if $b_i = 0$ this equals 0, and if $b_i = 1$ this equals z. Applying this coordinate-wise allows us to select or zero-out an entire token vector.

Applying transformation only on selected tokens. In our proofs, we often make use of bit flags, i.e., token positions that are either set to 1 if a certain action is required, and 0 otherwise. Assume we have a token with bit flag $b_i \in \{0, 1\}$ and we need to apply a function B only when $b_i = 1$, and copy the input otherwise. Suppose a token has bit flag $b_i \in \{0, 1\}$ and we want to apply a map T only when $b_i = 1$, while copying the input otherwise. For bounded activations, this can be done as

$$\mathbf{x}_i \mapsto \mathbf{x}_i + b_i\big(T(\mathbf{x}_i) - \mathbf{x}_i\big),$$

where the multiplication by b_i is implemented coordinate-wise by the gating MLP defined above. In particular, if T is an affine map or a finite lookup table on one-hot encodings, the resulting operation is implementable by a constant-depth ReLU MLP.

Counter. Assume each token additionally stores an active bit a_i, with exactly one token active. To shift the active bit from node v_i to the next node v_{i+1}, every node v_j attends to its predecessor v_{j-1}. Let

$$\mathbf{q}_j = \left(2\left(\omega_j - \frac{1}{N+1}\right), -1, -\left(\omega_j - \frac{1}{N+1}\right)^2\right), \qquad \mathbf{k}_i = (\omega_i, \omega_i^2, 1).$$

Then

$$\mathbf{q}_j^\top \mathbf{k}_i = -\left(\omega_i - \omega_j + \frac{1}{N+1}\right)^2,$$

which is maximized exactly when $i = j - 1$. The value matrix $\mathbf{W_V}$ projects the active bit a_i of the attended predecessor into a scratch coordinate. An MLP then sets this scratch coordinate as the new active bit a'_j. Boundary cases, e.g. $j = 1$, are handled by masking or by a dummy start token. For edge tokens, the same construction applies after assigning each edge token a fixed token-order positional encoding $p_e^* = (\omega_e, \omega_e^2, 1)$. This only adds constantly many coordinates.

Select edges based on nodes. Suppose node tokens carry a bit b_i and we want every edge token $e = (v_i, v_j)$ to know whether its parent or child endpoint is selected. This is done by one attention head from edges to nodes. For selecting edges by parent, the query of edge $e = (v_i, v_j)$ is computed from the parent position i^*, and node keys are computed from node positions:

$$\mathbf{q}_e = (2\omega_i, -1, -\omega_i^2), \qquad \mathbf{k}_{v_r} = (\omega_r, \omega_r^2, 1).$$

Then

$$\mathbf{q}_e^\top \mathbf{k}_{v_r} = -(\omega_i - \omega_r)^2,$$

so the edge selects precisely its parent node v_i. The value matrix projects the selected node bit b_i into an edge scratch coordinate. For selecting edges by child, the same construction is used with the child position j^* instead of i^*.

Select nodes based on edges. Conversely, suppose edge tokens carry a bit b_{ij} and we want nodes to gather this information from incident edges. A node v_i attends to edges whose stored parent or child coordinate equals i^*. For example, to select outgoing left edges, use a mask that allows v_i to attend only to the edge (v_i, v_L) with $\ell_{iL} = 1$, if it exists. The edge key is computed from its parent coordinate:

$$\mathbf{k}_{(v_p, v_j)} = (\omega_p, \omega_p^2, 1, \ell_{pj}),$$

and the left-child query of node v_i is

$$\mathbf{q}_{i,L} = (2\omega_i, -1, -\omega_i^2, C),$$

where $C > 1$ separates left from right edges. The right-child query is

$$\mathbf{q}_{i,R} = (2\omega_i, -1, -\omega_i^2, -C).$$

The value matrix projects the desired edge coordinates, e.g. a stored child state or edge bit, into node scratch coordinates.

A.4 Transformers simulate deterministic tree automata

In this section, we prove that transformers can simulate deterministic tree automata for *labeled* trees. The ability of transformers to simulate deterministic automata has already been shown by Rizvi-Martel et al. (2024); Liu et al. (2022); Merrill & Sabharwal (2023). Closest to our work is Rizvi-Martel et al. (2024, Thm. 3), who showed that for every *weighted* tree automaton $\mathcal{A}$ with s states, there exists a transformer with $\mathcal{O}(s)$ embedding dimension, $\mathcal{O}(s)$ attention width, $\mathcal{O}(s^3)$ MLP width, $\mathcal{O}(1)$ attention heads and $\mathcal{O}(d)$ depth which simulates $\mathcal{A}$ for *leaf-labeled* trees up to depth d. In comparison to Rizvi-Martel et al. (2024), our result holds for *unweighted* tree automata on *labeled ordered binary* trees, i.e., trees where all nodes are labeled.

We briefly introduce *tree automata*, partially following the notation of Rizvi-Martel et al. (2024).

Tree automata. We consider trees with at most two children. Let Σ be a finite nonempty alphabet, i.e., a finite set of symbols. A binary tree domain dom is a prefix closed subset of $\{0,1\}^*$ such that if $si \in \mathrm{dom}$ for $s \in \{0,1\}^*$ and $i \in \{0,1\}$ then $sj \in \mathrm{dom}$ for all $j < i$. Every nonempty tree domain has node r, which is the root node. A tree t over Σ is a mapping $t : \mathrm{dom} \to \Sigma$. We denote by $\mathcal{T}_\Sigma$ the set of all such Σ-trees, i.e., the smallest set such that $\Sigma \subset \mathcal{T}_\Sigma$ and $\sigma(t_0, t_1) \in \mathcal{T}_\Sigma$ for all $\sigma \in \Sigma$ and $t_0, t_1 \in \mathcal{T}_\Sigma$.

Definition A.2 (Deterministic finite tree automaton (DTA))**.** A DTA $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ consists of a finite alphabet Σ, a finite set of states Q, a dedicated start state q_0, a set of final states $F \subseteq Q$ and a transition function $\delta : Q^2 \times \Sigma \to Q$. $\mathcal{A}$ computes a function $f_\mathcal{A} : \mathcal{T}_\Sigma \to Q$ defined by $f_\mathcal{A}(t) = \mu(r)$, where the mapping $\mu : \mathcal{T}_\Sigma \to Q$ is recursively defined as

$$\mu(\sigma) = \delta(q_0, q_0, \sigma) \text{ for all } \sigma \in \Sigma \text{ and } \mu(\sigma(t_0, t_1)) = \delta(\mu(t_0), \mu(t_1), \sigma) \text{ for all } t_0, t_1 \in \mathcal{T}_\Sigma.$$

In case of only one child node, we have

$$\mu(\sigma(t_0)) = \delta(q_0, \mu(t_0), \sigma) = \delta(\mu(t_0), q_0, \sigma) \text{ for all } t_0 \in \mathcal{T}_\Sigma.$$

We say that t is accepted by $\mathcal{A}$ if $f_A(t) \in F$. The tree language recognized by $\mathcal{A}$ consists of the trees accepted by $\mathcal{A}$.

Next, we define our notion of *simulation*.

Definition A.3. We say that a function $f : \mathcal{T}_\Sigma \to \mathbb{R}^{d'}$ for $d' \geq 1$ *simulates* a DTA $\mathcal{A}$, if it holds that for any two trees $t, t' \in \mathcal{T}_\Sigma$, $f(t) = f(t')$ iff $f_\mathcal{A}(t) = f_\mathcal{A}(t')$.

Note that our definition of simulation is slightly weaker than the one defined in Rizvi-Martel et al. (2024), but sufficient for our setting.

Next, we prove the main theorem of this section.

Theorem A.4. *For every n and every DTA $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ there exists a graph transformer $GT_{\mathcal{A},n} : \mathbb{R}^{\mathcal{O}(n)} \to \mathbb{R}^{|Q|}$ with $\mathcal{O}(\log n)$ layers, embedding dimension $\mathcal{O}(|Q| + |\Sigma|)$, MLP width $\mathcal{O}(|Q|^2|\Sigma|)$ and $\mathcal{O}(1)$ attention heads which simulates $\mathcal{A}$ on all balanced binary trees of size n in FPT $\mathcal{O}(n \log n)$ time.*

Proof. Given balanced binary Σ-tree t as input, we first compute a suitable encoding $X_t \in \mathbb{R}^{\mathcal{O}(n)}$ for t. Next, we show how we can use MLPs and attention heads to select specific nodes layer-wise and implement state transitions to traverse the tree in a bottom-up manner. Finally, we explicitly construct a transformer $GT_\mathcal{A}$ that simulates $\mathcal{A}$.

Positional encoding. We encode a tree $t \in \mathcal{T}_\Sigma$ via a sequence of node and edge tokens, similarly to Sanford et al. (2024). Each of these tokens has size $\mathcal{O}(|\Sigma| + |Q|)$. More specifically, we encode each node v_i and edge $v_i v_j$ as

$$x_{v_i} = (i^*, 0, 1, 0, c_i, 0) \ \oplus \ \sigma_{v_i} \ \oplus \ q_{v_i},$$

and each edge as $v_i v_j$

$$x_{v_i v_j} = (i^*, j^*, 0, 1, 0, \ell_{ij}) \ \oplus \ \sigma_{v_i v_j} \ \oplus \ q_{v_i v_j},$$

where the first two positions denote compact representations of either the node index, or the index of the endpoints of the edge as defined in the previous section, the third position is 1 if the token is a node, and 0 for edges, the fourth position is 1 if the token is an edge, and 0 for nodes, c_i is a bit flag that indicates that a node has been already processed, $\ell_{ij} = 1$ iff the edge connects parent i to a left child j, $\sigma_{v_i} \in \{0,1\}^{|\Sigma|}$ is a one-hot encoding of the node label $\sigma \in \Sigma$, and $q_{v_i} \in \{0,1\}^{|Q|}$ is a one-hot encoding of the current state $q \in Q$. Leaf nodes v_l have state q_0, i.e., $q_{v_l} = (1, \ldots, q_{|Q|})$; internal nodes v_i initially have $q_{v_i} = \mathbf{0}$. Edges are directed from the parent node to the child nodes. Initially, each node has $c = 0$. The bit flag c_i as well as all entries of $\sigma_{v_i v_j}$ are set to 0 for each edge as we do not have edge features in our setting. As we have n nodes and $n - 1$ edges, and we consider $|\Sigma|$ and $|Q|$ as constants, the overall encoding $X_t = (x_v \mid v \in V(t)) \ \oplus \ (x_e \mid e \in E(t))$ of tree t has size $\mathcal{O}(n)$. We additionally reserve scratch coordinates on edge tokens for a copied child state and child processed bit, denoted $(\tilde{q}_{ij}, \tilde{c}_{ij})$, and scratch coordinates on node tokens for the gathered left and right child information, denoted (q_L, c_L) and (q_R, c_R). This increases the embedding dimension only by $\mathcal{O}(|Q|)$.

Attention heads select nodes for state transitions. To select nodes for state transitions, we require two steps: First, we (i) query edges to obtain their child nodes and copy their state information into the edge token. Next, we (ii) gather the children state information for each parent node via the edges. We use masking to exploit the tree structure. For (i), we have layer $2h-1$ with one attention head that selects for each edge v_iv_j its child node v_j. Given our positional encoding, this can be done by setting $\mathbf{q}_j = (2\omega_j, -1, -\omega_j^2)$ and $\mathbf{k}_j = (\omega_j, \omega_j^2, 1)$, which maximizes the attention score $\mathbf{q}_j^T\mathbf{k}_i = 2\omega_j\omega_i - \omega_i^2 - \omega_j^2 = -(\omega_i - \omega_j)^2$ when $i = j$. Finally, the value matrix $\mathbf{W_V}$ projects the state coordinates and processed bit of the selected child node into the edge scratch coordinates, i.e., $\mathbf{v}_{ij} = (q_j, c_j)$. All other coordinates are preserved by the copying primitive from the previous section. For (ii), we have the subsequent layer $2h$ with two attention heads, one for each child. For parent v_i and children $v_{i,L}$ and $v_{i,R}$ we set the key $\mathbf{k}_i = (\omega_i, \omega_i^2, 1, \ell_i)$ and queries $\mathbf{q}_{i,L} = (2\omega_i, -1, -\omega_i^2, C)$ for left children and $\mathbf{q}_{i,R} = (2\omega_i, -1, -\omega_i^2, -C)$ for right children,where $C > 1$ is some constant that decides between left and right children. In case node v_i does not have any children, i.e., is a leaf node, we have the key $\mathbf{k}_i = (\omega_i, \omega_i^2, 1, 0)$, that is, itself. We ensure that nodes either choose their outgoing edges, or – in case they have no edges – themselves, by setting masks $M_{i,L}(u) = 0 \iff u = v_iv_j$ is the left outgoing edge of v_i, and if no such edge exists, then $M_{i,L}(u) = 0$ only for $u = v_i$. All other entries are $-\infty$. Analogously, $M_{i,R}(u) = 0 \iff u = v_iv_j$ is the right outgoing edge of v_i, with fallback to $u = v_i$ if no right child exists. The value matrix of the left head writes the selected value into the scratch coordinates (q_L, c_L), and the value matrix of the right head writes the selected value into (q_R, c_R). For an edge token, the value is the stored child information $(\tilde{q}_{ij}, \tilde{c}_{ij})$. For a fallback node token, the value is defined as $(q_0, 1)$, so missing children are treated as already processed children in the start state.

MLPs implement state transitions. Similarly to Liu et al. (2022), we represent our transition function δ in matrix-vector notation, i.e., $[\delta(\cdot,\cdot,\sigma)]_{|Q|q_i+q_j,q'} = \mathbf{1}\{\delta(q_i,q_j,\sigma) = q'\}$ and $e_{\delta(q_i,q_j)} = \delta(\cdot,\cdot,\sigma)\; e_{q_i} \otimes e_{q_j}$, where $\otimes$ denotes the Kronecker product. We can implement δ with a two-layer ReLU activated MLP_δ of width $\mathcal{O}(|\Sigma|\;|Q|^2)$. To do so, we encode each input instance (q_i, q_j, σ) as concatenated one-hot encoding $x_{q_i,q_j,\sigma} = (e_{q_i}, e_{q_j}, e_\sigma) \in \mathbb{R}^{2|Q|+|\Sigma|}$. In the hidden layer, we have one unit for each $(q_i, q_j, \sigma) \in \mathbb{R}^{|Q|^2\times|\Sigma|}$, e.g.,

$$h_{i,j,\sigma}(x) = ReLU(\langle x, e_{q_i} \oplus e_{q_j} \oplus e_\sigma\rangle - 2.5).$$

This ensures that there is only one unit that outputs 1 for each distinct input. In the final layer, we only require a linear readout function that computes $\mathbf{1}\{\delta(q_i,q_j,\sigma) = q'\}$, e.g.,

$$y_{q'}(x) = \sum_{q_i\in Q}\sum_{q_j\in Q}\sum_{\sigma\in\Sigma} h_{i,j,\sigma}(x)\mathbf{1}\{\delta(q_i,q_j,\sigma) = q'\}.$$

MLPs select and update nodes. As we only want to update the current tree layer in each transformer layer, we slightly adapt MLP_δ defined above. First, we need to make sure we only select nodes that are in the current layer as indicated by our bit flag c, and whose children have already been processed as well, as indicated by bit flags c_L and c_R. For exact one-hot outputs, we use the Boolean indicator $\chi_k(z) = ReLU(z-(k-1)) - ReLU(z-k)$, which equals 1 iff $z = k$ on integer inputs. Then, we define

$$h_{i,j,\sigma}(x) = \chi_5\big(\langle q_L, e_{q_i}\rangle + \langle q_R, e_{q_j}\rangle + \langle \sigma_v, e_\sigma\rangle + c_L + c_R + (1-c_v)\big)\,.$$

This is 1 exactly when the current node is unprocessed, both children are processed, and their states and the node label are (s_i, s_j, σ). Next, we add hidden units that deal with the nodes who already have been processed or whose children have not yet been processed), e.g.,

$$h_{q'} = ReLU(\langle q_{v_i}, e_{q'}\rangle - (c_L + c_R - c_i) - 0.5),$$

which is positive iff the current node is not ready to be processed. We update our output layer as follows

$$y_{q'}(x) = \sum_{q_i\in Q}\sum_{q_j\in Q}\sum_{\sigma\in\Sigma} h_{i,j,\sigma}(x)\mathbf{1}\{\delta(q_i,q_j,\sigma) = q'\} + h_{q'}$$

to compute the new states and

$$y_c(x) = c_i + (1-c_i)\,ReLU(c_L + c_R - 1).$$

Since $c_i, c_L, c_R \in \{0,1\}$, this sets c_i to 1 exactly when the node was previously unprocessed and both children are processed, and otherwise leaves it unchanged. The multiplication by $(1-c_i)$ is implemented by the gating primitive.

Traversing the tree. The main idea of the traversal is that for each layer of the tree, (i) edges query their child nodes and copy the child's state information, and then (ii) parents query their edges (containing the childrens' states) and perform the state transition. To know which node is processed in which layer, we use our flag c. Initially, each flag c is set to 0. We next distinguish between the first layer (leaf nodes) and every subsequent layer (internal nodes).

- **Leaf nodes.** We identify leaf nodes through their absence of edges. The fallback value for each missing child is $(q_0, 1)$. Hence for a leaf node we have $q_L = q_R = q_0$ and $c_L = c_R = 1$. Since initially $c_i = 0$, the readiness condition is satisfied, so MLP_δ computes $q' = \delta(q_0, q_0, \sigma)$ and the processed-bit update sets $c_i = 1$.
- **Internal nodes.** For an internal node, the two child states are first copied into the outgoing edge tokens and then gathered into the parent scratch coordinates. If both children have already been processed, then $c_L = c_R = 1$, so MLP_δ applies $q' = \delta(q_L, q_R, \sigma)$. If one child is missing, the corresponding fallback value is $(q_0, 1)$, giving the single-child transitions $\delta(q_0, q_R, \sigma)$ or $\delta(q_L, q_0, \sigma)$. If some existing child has not yet been processed, then the readiness condition fails and the parent state and flag are copied unchanged.

By induction over the height of the tree, after h rounds exactly all nodes of height at most h have been processed and store the DTA state of their rooted subtree. Since the input trees are balanced, their height is $\mathcal{O}(\log n)$. Thus after $\mathcal{O}(\log n)$ rounds the root stores $f_{\mathcal{A}}(t)$. □

Remark A.5. If we allow our transformer to be *looped*, we obtain the same complexity bounds as in Theorem A.4 but with only $\mathcal{O}(1)$ layers, as we can traverse the tree in $\mathcal{O}(\log n)$ iterations.

A.5 Transformers compute tree decompositions

In this section, we show that a transformer can compute a tree decomposition for a treewidth bounded graph G. This could be trivially done by constructing a transformer large enough to simply enumerate all possible tree topologies for a tree decomposition of G. Instead, we want to be able to bound the time and space complexity of the constructed transformer to be confined to $\text{poly}(n)$, considering the treewidth k as a constant. To do so, we will show that transformers are able to simulate all individual steps of Bodlaender's FPT linear-time algorithm (Bodlaender, 1993). Finally, we will show how to construct a graph transformer that implements all these individual steps.

We will present our result for a *looped* transformer; this leads to a more parameter-efficient model and allows for a cleaner proof. However, we remark that we can unroll this looped transformer, which requires $\mathcal{O}(n \log n)$ iterations into a non-looped transformer with $\mathcal{O}(n \log n)$ layers, and thus our result holds for non-looped transformers as well.

In the following, we will first (i) present Bodlaender's linear time algorithm. Next, we show that (ii) several subroutines necessary for this algorithm can be simulated by a (looped) transformer. Finally, we (iii) construct the transformer consisting of these subroutines.

A.5.1 Bodlaender's linear time algorithm

We next detail the linear-time algorithm by Bodlaender (Bodlaender, 1993). The main idea of the algorithm is to classify vertices into low-degree and high-degree and call different subroutines depending on which class they fall into, to iteratively reduce the size of the input graph by a constant factor in each step to ensure linear runtime. When the reduced graph is small enough, the algorithm brute-forces its tree decomposition and subsequently unfolds the recursion by modifying the tree decomposition until we obtain the tree decomposition of the input graph. We present the algorithm in Section A.5.1; when the recursion ends, the tree decomposition of the input graph is iteratively constructed by undoing the operations to the input graph and adjusting the tree decomposition accordingly. We will describe these steps in the following and refer the reader to Bodlaender (1993) for more details on the algorithm, including proofs of correctness.

To understand the algorithm presented in Section A.5.1, we first need to provide some definitions.

Definition A.6. For a fixed constant $d \in \mathbb{N}$, we say that a vertex v has *low-degree* if it has degree $deg(v) \leq d$. We say that v is *friendly* if it is connected to at least one other low-degree vertex.

Please see (Bodlaender, 1993, Section 2) for more details on d; we consider this constant as given in our setting.

Definition A.7. Given graph $G = (V, E)$, let the *improved* graph $G' = (V, E')$ be the graph obtained by adding an edge (u, v) to E for all pairs $u, v \in V$ s.t. u and v have at least $k+1$ common neighbors of low degree in G.

Definition A.8. We say that a vertex v is *simplicial* if its neighbors form a clique in G.

Definition A.9. We say that a vertex in G is *I-simplicial* if it is simplicial in the improved graph of G, it is of low degree in G and it is not a friendly vertex in G.

Algorithm 1 Tree Decomposition

Input: $G = (V, E)$, $k \in \mathbb{N}$
if $|V|$ is of constant size **then**
  Brute-force compute the tree decomposition (T, B) of G
  if (T, B) has width $> k$ **then**
    return G has treewidth larger than k
  else
    return (T, B)
  end if
end if
if $|E| > k|V| - \frac{1}{2}k(k+1)$ **then**
  return G has treewidth larger than k
end if
Compute number of friendly vertices n_f
if $n_f \geq |V|/(4k^2 + 12k + 16)$ **then**
  Find a maximal matching $M \subseteq E$
  Compute $G' = (V', E')$ by contracting every edge in M
else
  Compute improved graph of G
  if improved graph of G contains I-simplicial vertex v with $\deg(v) \geq k + 1$ **then**
    return G has treewidth larger than k
  else
    Compute G' by deleting all I-simplicial vertices and their adjacent edges from G
  end if
end if
return Algorithm 1 with input G', k

We next explain how to adapt the tree decomposition constructed for G' by undoing the two types of operations: (1) Finding a maximal matching and contracting all edges in this matching, and (2) computing the improved graph and removing all I-simplicial vertices. For (1) we obtain a $(2k+1)$-width tree decomposition (T, B) from (T', B') by simply replacing each contracted vertex v' by its original vertices u, w. Next, the algorithm by Bodlaender uses a different algorithm (Bodlaender & Kloks, 1996) that returns an optimal-width tree decomposition given a non-optimal one of width k' and needs $\mathcal{O}(n)f(k')$ time. For (2), let (T', B') be the tree decomposition of $G'[V \setminus SL]$ where SL is the set of I-simplicial vertices that was removed and G' is the improved graph of G. Since all neighbors of each node in SL are in a clique, we know they all have to be in some bag $b_i \in B'$. For each node $v \in SL$, we add a new leaf node with bag b_v to b_i that contains v and all its neighbors to obtain (T, B) of G.

A.5.2 Transformers can simulate required subroutines

In this section, we show that transformers can perform several subroutines required to simulate the algorithm described in the previous section. We re-use one existing result from Sanford et al. (2024); all results are constructive and provide bounds on the required size of the respective transformer. We first define a common input encoding, of which we will use variants for the subroutines and Bodlaender's algorithm.

Input encoding. We encode the input graph as a sequence of node and edge tokens. Each node and edge encoding has some additional flags to denote whether vertices have been removed or edges have been contracted. More specifically, we encode each node v_i as

$$x_{v_i} = (i^*, 0, 0, 1, 0, a_i, p_i, r_i)$$

and each edge v_iv_j as

$$x_{v_iv_j} = (i^*, j^*, p^*_{ij}, 0, 1, a_{ij}, d_{ij}, s_{ij})$$

where the first two positions for nodes and the first three positions for edges denote compact representations of their positions as previously described. The fourth position is 1 if the token is a node, and 0 for edges, the fifth position is 1 if the token is an edge, and 0 for nodes. For nodes, a_i denotes whether the node is currently active, and p_i, r_i are pointers necessary for the edge contraction subroutine. The field p_i stores the immediate parent of v_i in the contraction forest, i.e. the vertex into which v_i was contracted, and r_i stores the current representative of v_i. Initially $p_i = r_i = i^*$. If v_j is contracted into v_i, then p_j is set to i^* and the representative of every vertex in the contracted component is updated to r_i; a_i are initialized to 1 for all nodes, p_i and r_i are initialized to i^* for all nodes. For edges, a_{ij} denotes whether the edge is currently active, d_{ij} denotes whether it is a duplicate edge, and s_{ij} denotes whether it is a self-loop. In addition to the actual $m = |E|$ edges in the graph, we also have $nk^2 - m$ initially empty edge tokens, with an additional recursion step timestamp, that denotes when they were added to use during the construction of the improved graph. We encode the tree decomposition (T, B) as $X_{TD} = (x_{t_i} \mid t_i \in V(T)) \oplus (x_{t_it_j} \mid t_it_j \in E(T))$ where

$$x_{t_i} = (i^*, 0, 1, 0) \oplus (r_i, f_i, i_i, j_i, l_i) \oplus x_{b_i}$$

and

$$x_{t_it_j} = (i^*, j^*, 0, 1).$$

The flag r_i indicates whether a node is the root nodes, and f_i, i_i, j_i, l_i denote whether nodes are *forget*, *introduce*, *join* or *leaf* nodes. Bag encodings $x_{b_i} \in \mathbb{R}^{2k+1}$ store the index of each node in bag b_i. To distinguish between graph nodes/edges and tree decomposition nodes/edges, we add an additional entry $\delta_{i,G} \in \{0,1\}^2$ with $\delta_{i,G} = (1,0)$ if x_i is a graph token, and $\delta_{i,G} = (0,1)$ if x_i is a tree decomposition token; we add constantly many 0 entries to ensure that $|x_{t_i}| = |x_{t_{ij}}| = |x_{v_i}| = |x_{v_iv_j}|$. Finally, we also add one global status bit r_{rej}, copied as an additional coordinate of every token and initialized to 0. All subsequent updates are gated by $1 - r_{\text{rej}}$: if $r_{\text{rej}} = 1$, the transformer only copies the current configuration and propagates the reject bit.

Next, we show that we can perform maximal matching with transformers and our input encoding.

Lemma A.1 (Maximal matching)**.** *For every $n \in \mathbb{N}$ there exists a graph transformer $GT : \mathcal{G}^n \to \mathcal{G}^n$ with $\mathcal{O}(n)$ depth, $\mathcal{O}(1)$ attention heads, $\mathcal{O}(1)$ embedding dimension, and $\mathcal{O}(1)$ width MLP that computes a maximal matching for a graph $G \in \mathcal{G}^n$ in $\mathcal{O}(n^3)$ time.*

Proof. We encode G as X_G; for ease of presentation, we only denote the first four entries and two additional bit flags m_i, c_i to indicate whether the edge/node is in the matching yet, and whether an edge is currently being processed, i.e., $x_{v_i} = (i^*, 0, 1, 0, \ldots, 0, m_i)$ for each $v_i \in V(G)$ and $x_{v_iv_j} = (i^*, j^*, 0, 1, \ldots, c_{ij}, m_{ij})$. Here $m_i = 1$ iff node v_i is already incident to a chosen matching edge, and $m_{ij} = 1$ iff edge v_iv_j has been chosen for the matching. The flag $c_{ij} = 1$ marks the unique edge currently being processed. To compute a maximal matching, we now need to iteratively add an edge and check whether its endpoints are not in the matching yet. We use one attention head that starts at the first edge position in X_G and iteratively moves edge by edge until the last position in X_G. Next, we do the following check: We check whether either i^* or j^* is already matched, i.e., whether $m_i = 1$ in x_{v_i} or $m_j = 1$ in x_{v_j}. If yes we do nothing, and $m_{ij} = 0$ and we move on to the next edge. If $m_i = m_j = 0$ we set $m_{ij} = m_i = m_j = 1$. For the active edge $e = (v_i, v_j)$, we use our edge selection primitive to copy the endpoint flags m_i and m_j into edge scratch coordinates. An MLP on the edge token computes $a_{ij} = c_{ij}(1 - m_i)(1 - m_j)$, sets $m_{ij} = m_{ij} \vee a_{ij}$, and stores a_{ij} as an accepted-edge flag. Next, using the primitive "select nodes based on edges", each node gathers whether it is incident to the unique edge with $a_{ij} = 1$. Its MLP updates $m_i = m_i \vee a_{\text{inc}}$. Finally, the counter primitive shifts the active flag c_{ij} to the next edge in the fixed edge order. By induction over the edge order, after processing the first t edges, the selected edges form a matching, and every selected edge was added only when both endpoints were previously unmatched. After all edges have been processed, consider any edge v_iv_j not in the matching. When v_iv_j was processed, it

was not added, so at least one of v_i, v_j was already matched. Hence, no edge can be added to the final matching, so the matching is maximal. □

The following theorem from Sanford et al. (2024) shows that single-layer transformers can perform simple graph retrieval tasks such as node/edge counts. We point out that Sanford et al. (2024) encode the graph similarly to us (node and edge tokens) with the same asymptotic complexity.

Theorem A.10 (Sanford et al. 2024, Thm. 36, slightly re-written as $|E| = \mathcal{O}(n)$ in our setting)**.** *For any graph retrieval task (including node count, edge count, edge existence, and node degree) there exists a single-layer single-headed transformers with embedding dimension $\mathcal{O}(\log n)$ and $\mathcal{O}(\log n)$ bit-precision that solves the task.*

An important step in Bodlaender's algorithm is the computation of an optimal-width tree decomposition, given a tree decomposition of non-optimal width (see Theorem A.12). As part of this subroutine, we first compute a *nice* tree decomposition.

Proposition A.11. *For every $n \in \mathbb{N}$ there exists a graph transformer $GT : \mathcal{G}^n \to \mathcal{G}^n$ with $\mathcal{O}(n)$ depth, $\mathcal{O}(1)$ attention heads, $\mathcal{O}(1)$ embedding dimension, and $\mathcal{O}(1)$ width MLP that, given a tree decomposition of width k and size $\mathcal{O}(n)$, constructs a nice tree decomposition of width k.*

Proof. We encode the tree decomposition (T, B) as X_{TD}, as defined above. For the proof, we first provide a high-level overview about the algorithm, and then show how each required operation can be implemented with our transformer primitives. To obtain a nice tree decomposition from a given tree decomposition, we do the following: We root the tree at an abritrary node; w.l.o.g., we root the tree at node t_1 by setting $r_1 = 1$. Next, we traverse the tree top-down. For each node t_i with more than two children $(t_1, \dots t_d)$, we replace them by a chain of binary children, i.e., we introduce new nodes $t_{i,1}, \dots, t_{i,d-1}$ each with bag $B_{i,j} = B_i$ for $1 \leq j \leq d-1$. For all connected nodes $t_i \neq t_j$, we introduce a path of introduce/forget nodes. For each leaf and the root node, we add at most $k+1$ forget/introduce bags, such that the bags of the leaf nodes and the root node are empty.

We now explain how the above transformations are implemented with the available primitives. First, we root T at t_1. During this traversal, each tree-edge token $x_{t_i t_j}$ obtains an orientation flag indicating which endpoint is the parent and which is the child. If necessary, the endpoint fields are swapped so that every active tree-edge token is represented as $(p^*, c^*, 0, 1)$, where p is the parent of c. We add temporary fields q_i and p_i to each bag token, where q_i indicates whether t_i has already been reached and p_i stores its parent. Initialize $q_1 = 1$ and $q_i = 0$ for $i \neq 1$. For $\mathcal{O}(n)$ iterations, each tree-edge token $t_i t_j$ reads q_i, q_j from its endpoints. If $q_i = 1$ and $q_j = 0$, it orients itself from t_i to t_j and writes $p_j = i^*$; symmetrically, if $q_j = 1$ and $q_i = 0$, it orients itself from t_j to t_i and writes $p_i = j^*$. Then every newly reached node sets its visited flag to 1. Since T is a tree, the reached vertices always induce a connected subtree. For each bag token t_i, attention over incident oriented tree-edges allows it to count its children and to write their identifiers into $\mathcal{O}(1)$ scratch fields, since k is fixed and the introduction of the binary chain is performed locally. If t_i has children $c_1, \dots, c_d$ with $d > 2$, we allocate fresh bag tokens $t_{i,1}, \dots, t_{i,d-2}$ from the yet unused tokens reserved for the tree decomposition. Each fresh token copies the bag field x_{b_i} of t_i. We then deactivate the original edges $(t_i, c_2), \dots, (t_i, c_d)$ and create a binary chain/tree of copies of t_i, reattaching the old children so that every node has at most two children. Since all new bags are identical to B_i, this preserves the tree-decomposition properties and does not increase width. Next consider an oriented tree edge (p, c) with bags B_p and B_c. Using constant-width MLPs, since k is fixed, the transformer computes the two constant size sets $B_p \setminus B_c$ and $B_c \setminus B_p$. It allocates at most $2k+2$ fresh bag tokens for this edge. These tokens form a path replacing (p, c). Along the first part of the path, vertices of $B_p \setminus B_c$ are forgotten one by one; along the second part, vertices of $B_c \setminus B_p$ are introduced one by one: $B_p \to \cdots \to B_p \cap B_c \to \cdots \to B_c$. The old edge token (p, c) is deactivated and replaced by the newly allocated tree-edge tokens connecting this path. Finally, the root bag and leaf bags are normalized in the same way. For the root, we allocate a path above the old root that forgets all vertices of the root bag, ending in an empty root bag. For each leaf, we allocate a path below the leaf that forgets all vertices, ending in an empty leaf bag. Since every bag has size at most $k+1$, only $\mathcal{O}(k) = \mathcal{O}(1)$ new bags are needed per root/leaf. Every original tree edge is replaced by a path of length $\mathcal{O}(k)$, and every high-degree node is replaced by $\mathcal{O}(\deg(t_i))$ copies. Hence the total number of new tree decomposition tokens is $\mathcal{O}(k|T|) = \mathcal{O}(n)$ for fixed k. □

Next, we show how a transformer can simulate the algorithm by Bodlaender & Kloks (1996), which computes an optimal tree decomposition from a non-optimal tree decomposition. We will need this as subroutine for the constant size case (when the graph is small enough to brute-force it), and for the step where we undo the matching in the tree decomposition. We remark that we only call this subroutine in our main algorithm when the non-optimal width is $\mathcal{O}(k)$, i.e., either to reduce a $(2k+1)$-width tree decomposition to k or to brute-force a tree-decomposition for $n = \mathcal{O}(k)$ with the tree decomposition that consists of one node containing all vertices.

Theorem A.12. *For every $n \in \mathbb{N}$ there exists a looped GT: $\mathcal{G}^n \to \mathcal{G}^n$ with $\mathcal{O}(n)$ layers, $\mathcal{O}(1)$ embedding width and $\mathcal{O}(1)$ attention heads that computes a tree decomposition of optimal width k, given a graph $G \in \mathcal{G}^n$ with treewidth k and a non-optimal tree decomposition (T', B') of G of width $k' = \mathcal{O}(k)$.*

Proof. Let k be the target width and let k' be the width of the given tree decomposition (T', B'). First, we transform (T', B') into a nice tree decomposition (T, B) as in Theorem A.11. For a node $t \in V(T)$, let G_t be the subgraph represented by the subtree below t, and let B_t be the boundary bag at t. The Bodlaender-Kloks algorithm stores, at each t, a bounded set of signatures describing how a width-k decomposition of G_t can interact with the boundary B_t. Let $\Sigma_{k,k'} = \{\gamma_1, \ldots, \gamma_m\}$ be the finite universe of reduced non-redundant signatures for target width k and input width k'. By the Bodlaender–Kloks bound, we may assume $m \leq \Gamma_{k,k'} = f(k')g(k)$ for some computable functions f, g. Moreover, each signature has encoding size at most $\Gamma_{k,k'}$. Since $k' = \mathcal{O}(k)$ and k is fixed, both m and the size of each signature are constants independent of n. A signature γ_i can be viewed as a canonical reduced characteristic: it is a constant-size tree-like object whose bags refer only to positions in the current boundary bag B_t, together with the finite characteristic informatin used by Bodlaender-Kloks to compare partial decompositions under domination. Thus the signature only stores boundary positions and constant-size integer labels. At each node t, we keep a set $A_t \subseteq \Sigma_{k,k'}$ of surviving signatures, whose transition depends on the node type.

- If t is a leaf, then A_t contains only the empty signature.
- If t is an introduce node with child c, candidates for A_t are obtained by extending signatures from A_c to the larger boundary bag.
- If t is a forget node with child c, candidates are obtained by deleting the forgotten boundary position from signatures in A_c and then applying the Bodlaender-Kloks reduction to obtain the canoncial representative.
- If t is a join node with children c_1, c_2, candidates are obtained by combining pairs of signatures from A_{c_1} and A_{c_2}.

After candidate generation, we prune dominated signatures. Let $Dom^+(\delta, \gamma)$ mean that δ strictly dominates γ in the Bodlaender-Kloks domination order. Then, we keep $A_t = \{\gamma \in C_t : \text{there is no } \delta \in C_t \text{ with } Dom^+(\delta, \gamma)\}$. By the correctness of the Bodlaender-Kloks characteristic-set construction, this pruning preserves extendability to a width-k tree decomposition. At the root, a surviving accepting signature exists iff the graph has treewidth at most k. If no such signature exists, we set $r_{\text{rej}} = 1$. To construct the actual decomposition, we store backpointers during the bottom-up computation and then perform a top-down backtrace from an accepting root signature. We now explain how the above dynamic program is implemented by the transformer. For every nice node t and every signature $\gamma_i \in \Sigma_{k,k'}$, we reserve one state token $S_{t,i}$. The index i is only a constant-size identifier for the precomputed signature γ_i. Since $m = \mathcal{O}_{k,k'}(1)$, this gives only $\mathcal{O}(n)$ state tokens. The token $S_{t,i}$ stores constant-size fields that contain a bit flag to indicate that γ_i survives at t abd backpointers: the local rule used and the child signature token, or the pair of child signature tokens in the case of a join node. For each nice node t, the transformer first computes the constant-size local pattern p_t: the node type, the introduced or forgotten bag position, the correspondence between parent and child bag positions, and the adjacency pattern on the relevant constant-size boundary. Since $|B_t| \leq k' + 1$ and k' is fixed, p_t has constant size. For fixed k, k', the universe $\Sigma_{k,k'}$, the set of local patterns p_t, the set of local rule templates r, and the domination relation are all finite and constant-size. Hence the leaf, introduce, forget, join, and pruning operations can be represented by fixed finite tables. The MLP at a node t takes as input p_t and the active bits of the relevant child states, consults these fixed tables, and outputs the candidate bits, the pruned active bits, and the first valid backpointer witness in a fixed

ordering. Since the number of signatures and candidate witnesses is constant, this is a constant-size Boolean function and is implemented by a constant-size MLP.

□

Next, we want to briefly show that a transformer can also balance a tree decomposition. For this, we give G and tree decomposition (T, B) of optimal width as input, and output balanced (T', B').

Theorem A.13. *For every $n \in \mathbb{N}$ there exists a graph transformer $GT : \mathcal{G}^n \to \mathcal{G}^n$ with $\mathcal{O}(n)$ depth, $\mathcal{O}(1)$ attention heads, $\mathcal{O}(1)$ embedding dimension, and $\mathcal{O}(1)$ width MLP that, given a nice tree decomposition of width k, constructs a balanced tree decomposition of width $\mathcal{O}(k)$.*

Proof. Let (T, B) be the given rooted nice tree decomposition of G of width k. We apply the balancing algorithm of Bodlaender (1989, Thm. 4.1) to the decomposition tree T, which is based on a parallel tree-contraction method by Miller & Reif (1985). The algorithm constructs a new balanced decomposition tree $\widehat{T}$ whose bags are obtained from constant-size collections of bags of the old decomposition. The algorithm proceeds by rake-and-compress tree contraction. It constructs a sequence of rooted trees $T = T_0, T_1, \ldots, T_r$ where $|V(T_r)| = 1$ and $r = \mathcal{O}(\log n)$. Each active node $v \in V(T_i)$ represents a connected cluster $\varphi(v, i) \subseteq V(T)$ of old decomposition nodes. Initially, $\varphi(v, 0) = \{v\}$. Each contraction step consists of the following two operations.

- *Rake.* Every active leaf $u \neq root$ is contracted into its active parent $p(u)$. Thus the cluster of $p(u)$ is enlarged by $\varphi(u, i)$, and u becomes inactive.
- *Compress.* After the rake step, consider every maximal unary chain, i.e. every maximal directed path of active nodes in which each node has exactly one active child. On each such chain, contract alternating disjoint parent-child pairs. If u and v are such a pair, they are replaced by one active representative whose cluster is $\varphi(u, i) \cup \varphi(v, i)$.

Each step removes a constant fraction of the active nodes. Thus, after $\mathcal{O}(\log n)$ steps only one active cluster remains. We now recall how Bodlaender (1989) obtains the balanced tree decomposition from this contraction history. Every rake or compress event creates one node a of the new decomposition tree $\widehat{T}$. Associated with this event is a constant-size set $C_a \subseteq V(T)$ of old decomposition nodes. This set consists of the relevant attachment nodes of the clusters involved in the contraction. Bodlaender We know from Bodlaender (1989) that there is an absolute constant c such that $|C_a| \leq c$ for every event a, and that the sets C_a form a balanced decomposition of the old tree T of depth $\mathcal{O}(\log n)$. The final bags for the graph G are obtained by replacing every old decomposition-node index $x \in C_a$ by its old bag B_x. Namely, for every $a \in V(\widehat{T})$, define $\widehat{B}_a = \bigcup_{x \in C_a} B_x$. Since $|C_a| \leq c$ and $|B_x| \leq k + 1$, we get $|\widehat{B}_a| \leq c(k + 1)$, so the width of $(\widehat{T}, \widehat{B})$ is at most $c(k + 1) - 1 = \mathcal{O}(k)$. We now explain the transformer implementation. The transformer does not store all sets $\varphi(v, i)$ explicitly. Instead, each active cluster is represented by one current representative token. The token stores only constant-size information: an active bit, its current parent pointer, whether it has zero, one, or at least two active children, its unique child if it has exactly one, and the constant-size attachment information needed for Bodlaender's local contraction templates. Leaves are identified by testing whether the active-child count is zero. Unary chain nodes are identified by testing whether the active-child count is one. Pairing alternating vertices on each unary chain is done by pointer jumping/list ranking along the chain, keeping only the parity of the distance from the chain head. This takes $\mathcal{O}(\log n)$ transformer rounds per contraction phase, and therefore $\mathcal{O}(\log^2 n) \subseteq \mathcal{O}(n)$ rounds overall. Whenever a rake or compress contraction is performed, the transformer creates a constant-size event/output token. This token stores the constant-size set $C_a \subseteq V(T)$ prescribed by Bodlaender's contraction template and pointers to the event tokens that become its children in $\widehat{T}$. Since every event deactivates at least one cluster, the total number of event tokens is $\mathcal{O}(n)$. Finally, each event token a expands its set $C_a = \{x_1, \ldots, x_s\}, \qquad s \leq c$, to the graph bag $\widehat{B}_a = \bigcup_{j=1}^{s} B_{x_j}$. Because both c and k are fixed constants, this copying and duplicate removal is a constant-size local computation. The output edges of $\widehat{T}$ are copied from the stored event-token pointers. Thus the transformer uses $\mathcal{O}(n)$ tokens, $\mathcal{O}(n)$ depth, constant-size local computations, and outputs a balanced tree decomposition of width $\mathcal{O}(k)$ and depth $\mathcal{O}(\log n)$. □

Next, we show how a graph transformer can construct a Σ-tree, i.e., a labeled, ordered, binary tree encoding the graph structure and the tree decomposition.

Theorem A.14. *For every $n \in \mathbb{N}$ there exists a graph transformer $GT : \mathcal{G}^n \to \mathcal{G}^n$ with $\mathcal{O}(n)$ depth, $\mathcal{O}(1)$ attention heads, $\mathcal{O}(1)$ embedding dimension, and $\mathcal{O}(1)$ width MLP that, given a nice, balanced tree decomposition of width k, constructs a Σ-tree of width $\mathcal{O}(k)$.*

Proof. Given a nice balanced tree decomposition (T, B) of width $K - 1$, where $K = \mathcal{O}(k)$, we construct the corresponding Σ-tree as follows. For every node $t \in V(T)$, we assign the vertices of B_t injectively to ports $\{1, \dots, K\}$, padding unused ports with $\perp$. For every parent-child edge tc of T, we store the partial relabeling map $\rho_{t,c} : \{1, \dots, K\} \to \{1, \dots, K\}$ defined by $\rho_{t,c}(i) = j$ iff $B_t[i] = B_c[j] \neq \perp$. Since K is constant, this map has constant-size encoding. It remains to encode the edges of G. For every edge $uv \in E(G)$, the set $T_{uv} = \{t \in V(T) : u, v \in B_t\}$ is nonempty and connected. Therefore it has a unique node $h(uv)$ closest to the root. We store the edge uv only at $h(uv)$, as an edge between the two ports containing u and v. Equivalently, at a node t and ports i, j, we set the local edge bit to 1 iff $B_t[i]B_t[j] \in E(G)$ and the parent of t does not contain both $B_t[i]$ and $B_t[j]$. Finally, we delete the original vertex identifiers from the output. The graph vertices are represented by equivalence classes of port occurrences connected by the relabeling maps $\rho_{t,c}$. For every graph vertex v, the bags containing v form a connected subtree of T. Hence all port occurrences representing v are connected by relabeling maps, and no relabeling map connects two distinct graph vertices. Thus the equivalence classes of port occurrences are in bijection with $V(G)$. Every graph edge uv is stored exactly once. Conversely, an edge label is only created when the corresponding pair is an actual edge of G. Therefore the Σ-tree represents exactly G and remains balanced and nice. The size of the alphabet is finite because $K = \mathcal{O}(k)$ is fixed. □

A.5.3 Constructing the transformer that constructs tree decompositions

In this section, we provide the general construction of a transformer that computes a tree decomposition by showing that a transformer can simulate each respective part of the linear-time algorithm proposed by Bodlaender (1993).

Theorem A.15. *Given a graph $G = (V, E)$ with $|V| = n$ and treewidth $\mathrm{tw}(G) = k$, there exists a* looped *graph transformer with $\mathcal{O}(1)$ layers, embedding dimension $\mathcal{O}(\log n)$, MLP width $\mathcal{O}(1)$ and $\mathcal{O}(1)$ heads that can compute its tree decomposition (T, B) of width k with $\mathcal{O}(n \log n)$ iterations in FPT $\mathcal{O}(n^3 \log n)$.*

Proof. We first give a general overview about the proof. The main difficulty in simulating a recursive algorithm is that we need to keep track of all changes done to the original graph, and to adapt the constructed tree decomposition in each unfolding step by undoing the operations that led to the smaller graph. In order to do so, we keep a scratchpad of size $\mathcal{O}(n)$ that keeps track of the operations and the affected nodes and edges.

As before, we encode the tree decomposition as

$$X_{TD} = (x_{t_i} \mid t_i \in V(T)) \oplus (x_{t_i t_j} \mid t_i t_j \in E(T)).$$

Each bag token is encoded as

$$x_{t_i} = (i^*, 0, 1, 0) \oplus (r_i, f_i, \iota_i, j_i, l_i) \oplus x_{b_i}$$

where r_i marks the root bag and f_i, ι_i, j_i, l_i indicate forget, introduce, join and leaf bags in case the tree decomposition is nice; otherwise we set $r_i = f_i = \iota_i = j_i = l_i = 0$. The bag field $x_{b_i} \in \mathbb{R}^{2k+2}$ stores the vertex indices contained in the bag, using 0 for empty slots. A tree-edge token is encoded as

$$x_{t_i t_j} = (i^*, j^*, 0, 1).$$

Since k is fixed, each bag field has constant dimension, and we allocate $\mathcal{O}(nk)$ bag slots, hence $\mathcal{O}(n)$ tree decomposition tokens. To keep track of the graph operations during the recursion, we utilize a scratchpad of size $\mathcal{O}(n)$, where for each graph reduction step (i.e., each recursion step) we store the recursion number, the type of operation, and which edges got contracted/which nodes where removed. More specifically,

$$x_{scratchpad} = (i^*, r_1, r_2) \ \oplus \ x_{G'}$$

where the first entry denotes the recursion step (encoded as (ω_i, ω_i^2), as before), the next two entries define the type of operation done to the current graph, i.e., $r_1 = 1$ and $r_2 = 0$ for edge contraction and $r_1 = 0$ and $r_2 = 1$ for removing simplicial vertices, and $x_{G'}$ stores the affected object of the operation. For a contraction step it stores pairs (i^*, j^*) for every matched edge $v_i v_j$ contracted at recursion depth t. For a deletion step it stores the deleted vertex i^*. All remaining information needed to undo the operation is recoverable from the persistent graph tokens: inactive vertices and edges are not deleted, but only marked inactive, and contractions are represented by the pointer forest (p_i, r_i). Every auxiliary edge created during the construction of the improved graph carries a timestamp and an auxiliary-edge flag. Hence, when unfolding recursion level t, all auxiliary edges created at level t can be deactivated again. Finally, we also have some immutable token for the desired treewidth k and the constants d and c_2.

Implementing subroutines. In order to implement the steps required for Bodlaender's algorithm, the transformer needs to be able to perform the following operations.

(i) Compute a tree decomposition given a non-optimal tree decomposition (Theorem A.12)

(ii) Count nodes (Theorem A.10)

(iii) Count edges (Theorem A.10)

(iv) Compute the degree of a node (Theorem A.10)

(v) Compute maximal matchings (Lemma A.1)

(vi) Identify I-simplicial vertices (Section A.5.3).

Construct improved graph and identify I-simplicial vertices. To construct the improved graph, we need to identify all pairs $v_i, v_j \in V(G)$ that have at least $k+1$ common neighbors of low degree. Next, we add an edge between each such pair, resulting in at most $\mathcal{O}(k^2 n)$ new edges, to construct the improved graph G'. Then, we need to identify all vertices whose neighbors form a clique in G', are of low degree and not friendly. Recall that we say that a vertex v has low degree, if $\deg(v) \leq d$. We maintain all edge tokens in canonical order (i, j) with $i < j$, and ignore inactive, duplicate, and self-loop tokens. First, using Theorem A.10, each vertex computes its active degree, saturated at $d+1$, and sets $p_{low,v} = \mathbf{1}[\deg(v) \leq d]$. Each low-degree vertex then stores its active neighbors in constant-size fields $u_1, \ldots, u_d$ for $u_i \in N(v)$. This is done by scanning the active edge tokens once: when the current edge (u, w) is incident to a low-degree vertex v, the opposite endpoint is written into the first empty neighbor slot of v. Next, we create a linear-size scratchpad of records. For every active edge (u, w), we add an original-edge record $((u, w), \perp, \text{orig})$. For every low-degree vertex v and every unordered pair of stored neighbors $u, w \in N(v)$, we add a witness record $((u, w), v, \text{wit})$ again with $u < w$. Since each low-degree vertex has at most $d = \mathcal{O}_k(1)$ neighbors, it creates only $\binom{d}{2} = \mathcal{O}_k(1)$ witness records, yielding a scratchpad size of $\mathcal{O}(n)$. We now scan the scratchpad from left to right. Every witness record $s = ((u, w), v, \text{wit})$ maintains three constant-size fields: c_s, p_s, b_s. At scan step r, the current scratch record broadcasts its key (a, b). Every witness record compares (a, b) with its own key (u, w). If the keys match and the current record is a witness record, c_s is incremented, saturated at $k+1$. If the keys match and the current record is an original-edge record, then p_s is set to 1. The flag b_s records whether an earlier witness with the same key has already appeared; this is used to choose one canonical witness record for each key. After the scan, each witness record knows whether its pair is an edge of the improved graph: $p_{imp} = p_s \vee \mathbf{1}[c_s \geq k+1]$. Moreover, if s is the first witness record for its key, $c_s \geq k+1$, and $p_s = 0$, then the corresponding spare edge token is activated as the new improved-graph edge (u, w). Original edges are already marked as improved-graph edges. Finally, each low-degree vertex v checks all constant-many witness records created from pairs of its neighbors. It sets $p_{simp,v} = p_{low,v} \cdot (1 - p_{friendly,v}) \cdot \bigwedge_{\{u,w\} \subseteq N(v)} p_{imp_{(u,w),v}}$. For degree 0 or 1, the conjunction is empty and therefore true. Hence v is marked precisely when it is low degree, not friendly, and its neighborhood forms a clique in the improved graph. The construction uses $\mathcal{O}(n)$ scratch tokens and $\mathcal{O}(n)$ loop iterations. This subroutine costs $\mathcal{O}(n^2)$ work, since each iteration updates $\mathcal{O}(n)$ tokens.

Adapting the tree decomposition based on the operations. One of the the more intricate parts of simulating the recursive algorithm is that the transformer needs to be able to reconstruct the tree

decomposition of graph G from the tree decomposition of G'; in addition, we need to be able to reconstruct the reduced graph for each recursion step, since we need it as input to the subroutine for obtaining an optimal width tree decomposition after the contraction step. We do this by having submodules that, given the operations done to G to obtain G', adapt the tree decomposition (T', B') of G' to obtain the tree decomposition (T, B) of G. These operations can be either (i) contracting edges via maximal matchings, or (ii) removing a set of I-simplicial vertices. We use a scratchpad of size $\mathcal{O}(n)$ that stores the type of operation and which nodes/edges where affected in each iteration. We next describe the matching contraction and removing simplicial vertices in detail.

Contracting maximal matchings. We explain how the encoding is adapted when we contract the graph given a maximal matching. We explain it for a single edge; the operations can be conducted in parallel for all edges. Without loss of generality, assume we contract the edge $v_i v_j$ with $i < j$.

- We set $x_{v_j} = (j^*, 0, 1, 0, 0, i^*)$, that is, we set node v_j to inactive and note down that it collapsed into node v_i.
- We check whether collapsing node v_j into node v_i led to any of the existing edges being self-loops or duplicates; if yes, we set their respective flag to 1.
- In the scratchpad, we add an entry with a designated start token and a designated end token, and write down $(t^*, 1, 0, i^*, j^*)$ for each edge in the matching that gets contracted; t denotes the recursion depth.

All these updates are implemented using the primitives for selecting edges by endpoints and selecting nodes by incident edges. The matched-edge flag selects the edges to be contracted. Endpoint attention writes the smaller endpoint as the new representative, and gated MLP updates set the larger endpoint inactive and update the pointer fields. Self-loops and duplicates are detected by comparing the representatives of edge endpoints. This can be done iteratively, and we keep track of contracted edges/collapsed nodes via the p_i pointers. When we unfold the recursion, we can achieve the pre-contraction graph by consulting all scratchpad entries of depth $t - 1$.

Next, assume we have (T', B') given, and constructed G from G'. We adapt X_{TD} by first identifying the bags that contain the edges contracted going from G to G'. That is, if we contracted edge $v_i v_j$ with $i < j$ then we will have at least one bag that contains v_i. Each bag that contains v_i, we also add v_j. Since the contracted edges form a matching, every current vertex in a bag expands to at most two original vertices when undoing the contraction. Thus a bag of size at most $k + 1$ becomes a bag of size at most $2(k + 1)$, i.e. width at most $2k + 1$. This remains a valid tree decomposition of G' (Bodlaender, 1993, Lemma 2.8). We obtain width k by using our subroutine in Theorem A.12 giving G and (T, B) as input. Finally, we undo the contraction step do obtain the graph of the previous recursion.

Removing I-simplicial vertices. We assume we already have the I-simplicial vertices identified. We do the following steps when we remove vertex v_i; again, this can be done in parallel for multiple nodes.

- We set $x_{v_i} = (i^*, 0, 1, 0, 0, i^*, 0)$, that is, we set node v_i to inactive and set its representative $r_i = 0$.
- We set all edges with $r_i = v_i$ as endpoint to inactive, i.e. $a_{i\cdot} = a_{\cdot i} = 0$.
- In the scratchpad, we add an entry with a designated start token and a designated end token, and write down $(t^*, 0, 1, i^*)$ for each deleted node; t denotes the recursion depth.

Next, assume we have (T', B') given. We next need to find, for each deleted vertex v_i, the bag b_i in (T', B') that contains all neighbors of v_i. Then, we add a leaf bag that contains v_i and all neighbors of v_i and connect it to b_i. Such a bag exists because v_i is I-simplicial: its active neighborhood has size at most k and is made into a clique in the improved graph. In every tree decomposition, every clique is contained in some bag. Therefore some bag of (T', B') contains $N(v_i)$. Correctness follows by induction over the recursion depth. The scratchpad records which reduction was applied at each level, while inactive tokens and pointer fields preserve enough information to reconstruct the previous

graph. During unfolding, the contraction and deletion reconstruction steps described above transform a tree decomposition of the reduced graph into a valid tree decomposition of the previous graph, increasing the width only by a function of k. Since Bodlaender's recursion has $\mathcal{O}(\log n)$ steps and each step performs $\mathcal{O}(n)$ loop iterations over the current graph tokens, the total number of iterations is $\mathcal{O}(n \log n)$. The scratchpad size is $\mathcal{O}(n)$, as the graph is reduced by a constant fraction in each iteration. □

Remark A.16. The proposed encoding is not permutation-invariant, i.e., the same graphs with different permutation of vertex labelings might result in different tree decompositions. However, in the context of accepting/rejecting graphs based on the MSO-expressible property they satisfy, this does not matter, as the automaton would accept or reject both vertex labelings in the same way.

A.6 Graph transformers express monadic second-order logic

In this section, we provide the proofs for our main results.

Theorem A.17. *Fix $k \in \mathbb{N}$. For every $n \in \mathbb{N}$ and every MSO-definable property P, there exists a graph transformer $GT : \mathcal{G}^{n,k} \to \mathbb{R}^d$ that expresses P in FPT $\mathcal{O}(n^3 \log n)$ time for any graph $G \in \mathcal{G}^{n,k}$. Given a nice and balanced tree decomposition (T, B) of width k, we only require FPT $\mathcal{O}(n \log n)$ time.*

Proof. Let φ be the MSO sentence that expresses property P, and let φ^* be such that $G \models \varphi$ iff $(T, B) \models \varphi^*$ (Kreutzer, 2011). If (T, B) is not given, we can construct an optimal-width (T, B) using graph transformer GT_{TD} as shown in Theorem A.18. Let t be the Σ-tree t that encodes (T, B) and G, which we can compute by first converting it into a nice tree decomposition of width k and $\mathcal{O}(nk)$ bags using transformer GT_{nice} (Theorem A.11), and then balancing the tree decomposition to $\mathcal{O}(\log n)$ depth using transformer GT_{bal} (Theorem A.13). Since (T, B) is balanced it has depth $\mathcal{O}(\log n)$, and we can compute its Σ-tree t using graph transformer GT_{Σ} in $\mathcal{O}(n \log n)$ time. There exists a DTA $\mathcal{A}_{\varphi *} = (\Sigma, Q, q_0, F, \delta)$ that outputs an accepting state $q_F \in F$ iff $t \models \varphi^*$ (Libkin, 2004, Thm. 7.30). Finally, the root token contains the one-hot encoding $\mathbf{e}_q$ of the DTA state $q = f_{\mathcal{A}}(t)$. Since $\mathcal{A}$ accepts exactly when $q \in F$, we append a final readout MLP ρ defined on state encodings by $\rho(\mathbf{e}_q) = \mathbf{e}_{\text{acc}}$ if $q \in F$, and $\rho(\mathbf{e}_q) = \mathbf{e}_{\text{rej}}$ if $q \notin F$; this finite lookup table on one-hot encodings can be implemented by a constant-depth ReLU MLP. As we have shown in Theorem A.4, we can build a graph transformer $GT_{\mathcal{A}_{\varphi^*}}$ that simulates $\mathcal{A}_{\varphi^*}$ in FPT $\mathcal{O}(n \log n)$ time. □

Theorem A.18. *Fix $k \in \mathbb{N}$. For every $n \in \mathbb{N}$, there exists a graph transformer $GT : \mathcal{G}^{n,k} \to \mathcal{G}^{n,k}$ with $\mathcal{O}(n \log n)$ layers, embedding dimension $\mathcal{O}(\log n)$, MLP width $\mathcal{O}(1)$ and $\mathcal{O}(1)$ attention heads that computes a nice and balanced tree decomposition of width $\mathcal{O}(k)$ in FPT $\mathcal{O}(n^3 \log n)$ time.*

Proof. We prove Theorem A.18 by a concatenation of four transformers that each simulate different graph algorithms. First, we use a transformer GT_{TD} that takes G, k as input, and outputs its tree decomposition (T, B) of width k in $\mathcal{O}(n^3 \log n)$ time Given G and (T, B), we next make (T, B) *nice* and *balanced* transformers GT_{nice} (Theorem A.11) and GT_{bal} (Theorem A.13) in $\mathcal{O}(n^2)$. The overall time complexity of the construction $GT = GT_{TD} \circ GT_{\text{nice}} \circ GT_{\text{bal}}$ is dominated by $\mathcal{O}(n^3 \log n)$ time.

□

We next want to bound the size of a transformer required to express MSO for bounded-treewidth graphs.

Theorem A.19. *Let $G \in \mathcal{G}^{n,k}$, (T, B) be its tree decomposition of width k, fix MSO-definable property P and let $s = g(k, P)$ for some computable function g. Then, for every MSO-definable property P, there exists a graph transformer $GT : \mathcal{G}^{n,k} \to \mathbb{R}^d$ with $\mathcal{O}(\log n)$ layers, embedding dimension $\mathcal{O}(s \log n)$, MLP width $\mathcal{O}(s^3)$ and $\mathcal{O}(1)$ attention heads that expresses P.*

Proof. We prove Theorem A.19 analagously to Theorem A.17. Let φ be the MSO sentence that expresses property P, and let φ^* be such that $G \models \varphi$ iff $(T, B) \models \varphi^*$ (Kreutzer, 2011). If (T, B) is not given, we can construct an optimal-width (T, B) using graph transformer GT_{TD} as shown in Theorem A.18. Let t be the Σ-tree t that encodes (T, B) and G, which we can compute with GT_{Σ} (Theorem A.14) by first converting it into a nice tree decomposition of width k and $\mathcal{O}(nk)$ bags using

transformer GT_{nice} (Theorem A.11), and then balancing the tree decomposition using transformer GT_{bal} (Theorem A.13). We remark that the size of t is $\mathcal{O}(n)$ and the depth of t is $\mathcal{O}(\log n)$. There exists a DTA $\mathcal{A}_{\varphi *} = (\Sigma, Q, q_0, F, \delta)$ that outputs an accepting state $q_F \in F$ iff $t \models \varphi^*$ (Libkin, 2004, Thm. 7.30). The number of states $|Q|$ and the alphabet size $|\Sigma|$ are only dependent on k and φ, and independent of n, that is, $|\Sigma| + |Q| \leq g(k, \varphi)$ for some computable function g (Flum & Grohe, 2006). As we have shown in Theorem A.4, we can build a graph transformer $GT_{\mathcal{A}_{\varphi^*}}$ that simulates $\mathcal{A}_{\varphi^*}$ on tree t of depth $\mathcal{O}(\log n)$ with $\mathcal{O}(\log n)$ layers, embedding dimension $\mathcal{O}(|Q| + |\Sigma|)$, MLP width $\mathcal{O}(|Q|^2|\Sigma|)$ and $\mathcal{O}(1)$ attention heads.

□