Billion-Scale Graph Foundation Models

Maya Bechler-Speicher[1], **Yoel Gottlieb**[2], **Andrey Isakov**[1], **David Abensur**[1], **Ami Tavory**[1], **Daniel Haimovich**[1], **Ido Guy**[1], **Udi Weinsberg**[1]

[1] Meta, [2]Work done at Meta

Graph-structured data underpins many critical applications. While foundation models have transformed language and vision via large-scale pretraining and lightweight adaptation, extending this paradigm to general, real-world graphs is challenging. In this work, we present Graph Billion-Foundation-Fusion (GraphBFF): the first end-to-end recipe for building billion-parameter Graph Foundation Models (GFMs) for arbitrary heterogeneous, billion-scale graphs. Central to the recipe is the GraphBFF Transformer, a flexible and scalable architecture designed for practical billion-scale GFMs. Using the GraphBFF, we present the first neural scaling laws for general graphs and show that loss decreases predictably as either model capacity or training data scales, depending on which factor is the bottleneck. The GraphBFF framework provides concrete methodologies for data batching, pretraining, and fine-tuning for building GFMs at scale. We demonstrate the effectiveness of the framework with an evaluation of a 1.4 billion-parameter GraphBFF Transformer pretrained on one billion samples. Across ten diverse, real-world downstream tasks on graphs unseen during training, spanning node- and link-level classification and regression, GraphBFF achieves remarkable zero-shot and probing performance, including in few-shot settings, with large margins of up to 31 PRAUC points. Finally, we discuss key challenges and open opportunities for making GFMs a practical and principled foundation for graph learning at industrial scale.

Correspondence: Maya Bechler-Speicher mayabs@meta.com

Meta

1 Introduction

Graph-structured data is ubiquitous across domains such as security, social networks, recommender systems, and many others. While foundation models have revolutionized natural language processing and computer vision through large-scale pretraining and lightweight adaptation to downstream tasks (Bommasani et al., 2021), extending these advances to graphs in the form of Graph Foundation Models (GFMs) is fundamentally challenging. First, graph-data distributions differ substantially across domains, for example, molecular graphs and social networks vary in node-feature distributions, topological structure, and scale. This raises a fundamental question about the validity of pretraining, as there might be little transferable structure across graphs. Furthermore, the scarcity of public, high-quality, large-scale graph datasets (Bechler-Speicher et al., 2025) limits the ability to rigorously study GFMs at billion-node data scales and billion-parameter model sizes. Finally, graph learning problems span multiple levels of granularity, including node-, edge-, and graph-level tasks.

While a truly generic GFM is challenging, many data modalities, including text and images, can be viewed as instances of graph-structured data with characteristic graph distributions (Veličković, 2023; Bronstein et al., 2021). Under this perspective, Large Language Models (LLMs) (Vaswani et al., 2017; Brown et al., 2020; Devlin et al., 2019) and Vision Transformers (ViTs) (Dosovitskiy et al., 2021) can be interpreted as billion-scale GFMs. LLMs assume a sequential token order and exploit it through positional encodings. ViTs assume images lie on a fixed two-dimensional grid, leveraging this structure via patchification and parameter sharing. Figure 1 illustrates FMs as GFMs, using the minimal graph structure they operate on. However, these architectures embed strong inductive biases tailored to their underlying data distributions, which can hinder performance when applied to general graphs drawn from different distributions. For example, reducing graphs to text sequences as input to LLMs may yield poor performance and unstable predictions (Fatemi et al., 2024). In addition, this approach may require LLMs to have orders of magnitude more parameters to outperform models designed for graphs (Ranjan et al., 2025), such as Graph Neural Networks (GNNs)

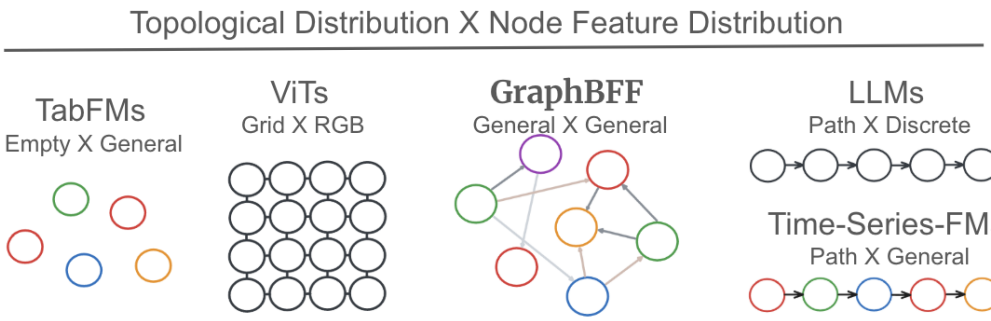


Figure 1 An illustration of FMs as GFMs over specific topological and feature distributions, using the minimal graph structure they operate on. Node colors correspond to token types, with each type undergoing distinct transformations within the model. The GraphBFF is designed for general graphs and feature distributions, supporting any number of token types.

(Gilmer et al., 2017; Kipf and Welling, 2017; Hamilton et al., 2017) and Graph Transformers (Dwivedi and Bresson, 2021; Rampášek et al., 2022). We expand on the viewpoint of FMs as GFMs in Section 3.

In this work:

1. We present the Graph Billion-Foundation-Fusion (GraphBFF) - the first end-to-end framework for building billion-scale parameters GFMs on arbitrary heterogeneous graphs, at real industrial scale and settings.
2. We introduce the GraphBFF Transformer, a flexible, scalable and effective architecture for building billion-scale GFMs. By leveraging two heterogeneous attention components and incorporating a sparse softmax, our transformer efficiently supports real-world large-scale heterogeneous graphs. We formally show that the two attention components of GraphBFF Transformer are necessary to increase its expressiveness.
3. Using GraphBFF Transformer, we present the first neural scaling laws for arbitrary graphs in terms of data and model size. These laws show strict model and data bottlenecks for GFMs, suggesting that model and data must grow together, as previously observed in LLMs.
4. We show a wide range of practical and effective design choices, including batching, architectural choices and pre-training, to support billion-scale models and graphs, including the introduction of *KL-Batching* and *Round-Robin Batching*, new storage-level and gpu-level batching strategy for effective pre-training of GFMs.
5. We perform an extensive evaluation of a 1.4 billion-parameter GraphBFF Transformer pretrained on one billion samples from real-world graph data, using the GraphBFF framework. We examine ten diverse, real-world industrial downstream tasks over graphs that were unseen during training, spanning node- and link-level classification and regression, and report strong zero-shot and probing performance, including in a few-shot setting. To the best of our knowledge, these are the first publicly available insights from billion-parameter GFMs pre-trained on billion-scale arbitrary heterogeneous graphs.
6. We evaluate the GraphBFF Transformer as a task-specific model, decoupled from GFMs, and show that it consistently outperforms existing task-specific heterogeneous graph transformers.

While absent from the public domain, many organizations maintain billion-scale graphs and many industrial applications rely on large-scale graph data. Although we cannot release the data used in this research, our goal is to provide a concrete, reproducible blueprint, backed by strong empirical evidence, for building effective GFMs in practice. We discuss numerous new theoretical and methodological questions arising from our work, and outlining key open challenges and promising directions for deploying effective GFMs.

2 Related Work

Graph Foundation Models The public graph-data landscape still lacks billion-scale, diverse, high-quality data. This scarcity constrains GFMs research and may partly explain why progress on training billion-parameter GFMs lags behind other domains with abundant public data, such as text and vision (Bechler-Speicher et al., 2025). Another core challenge in building GFMs, emphasized by recent GFM surveys Wang et al. (2025b), is heterogeneity along three axes: (i) heterogeneity, (ii) structural heterogeneity, and (iii) task heterogeneity (e.g., node-, edge-, and graph-level tasks). A prominent line of works focuses on designing GFMs for specific types of graphs with their specific tasks of interest, such as molecular graphs (Shoghi et al., 2024) or knowledge graphs (Galkin et al., 2024). Another line of work focuses on feature heterogeneity, a challenge also central to TabFMs (Gorishniy et al., 2021; Somepalli et al., 2022; Hollmann et al., 2022; Shaw et al., 2018; Hu et al., 2020). Feature heterogeneity asks whether a pre-trained model can be applied to samples with previously unseen feature sets. In practice, this often reduces to a technical mismatch between model parameters and the input space, motivating approaches that restrict inputs to a predefined vocabulary (Mao et al., 2024; Wang et al., 2025a). A different line of work extends TabFMs ideas to graphs, by partitioning features into predefined groups (e.g., numerical, categorical, text) and enforcing shared transformations within each group (Eremeev et al., 2025; Finkelshtein et al., 2025; Zhao et al., 2025; Ranjan et al., 2025; Liu et al., 2024). While these strategies ensure dimensional compatibility for unseen node types, they can limit expressivity by forcing semantically distinct features through the same transformation. This exposes an inherent trade-off: grouping is a practitioner-driven design choice, and how to select it remains an open question. Importantly, our framework is compatible with any of the above methods, and focuses at the billion-scale model parameters and data.

LLMs for Graphs Recent work has explored whether *frozen* LLMs can be turned into graph reasoners by serializing graphs into text and relying on in-context prompting. A systematic study by Fatemi et al. (2024) shows that performance on basic graph tasks can be surprisingly brittle: it depends strongly on (i) the chosen textual graph encoding, (ii) the prompt template, and (iii) even the graph structure itself and highlighting that naive text encodings can make even classical graph primitives unreliable. In response, Perozzi et al. (2024) propose GraphToken, which replaces hand-crafted text serialization with a small trainable graph encoder that outputs soft prompt tokens prepended to a frozen LLM, yielding large gains on GraphQA while updating only the encoder. Finally, CLRS-Text (Markeeva et al., 2024) converts algorithm execution traces, including graph algorithms, into textual supervision to evaluate and train LLMs as generalist algorithm executors, emphasizing out-of-distribution evaluation via procedural resampling.

Despite these advances, these approaches are not sufficient to constitute a GFMs. Text-serialization methods remain constrained by context length and quadratic prompt growth for dense structure, and their robustness hinges on non-canonical design choices in the serialization and prompting. GraphToken improves structured injection, but still requires task-specific supervised training signals and access to model internals, to align encoder outputs with the LLM embedding space, and its evidence is primarily on synthetic graph-reasoning suites rather than broad, heterogeneous real-world graph distributions. CLRS-Text, while valuable for benchmarking algorithmic reasoning, targets procedurally generated traces and shows limited extrapolative generalization in autoregressive LLMs, i.e., it does not directly address the core GFM desiderata. Collectively, these works suggest that "LLMs-for-graphs" can be powerful components , but do not remove the need for GFMs trained to handle graph heterogeneity, scale, and transfer across domains.

Transformers on Graphs Applying Transformers to graphs requires choosing how to represent a graph as tokens, for example by converting it into node and edge tokens arranged as a set or sequence (Dwivedi and Bresson, 2021; Kreuzer et al., 2021; Zhang et al., 2020; Ying et al., 2021). Recent works show that this design choice strongly influences both expressivity and scalability (Yehudai et al., 2025; Sanford et al., 2023, 2024). To better capture topology, another dominant approach involves constraining or biasing the attention mechanism. By employing graph-informed attention masks or structural biases, models can explicitly inject graph-structure priors into the self-attention process (Ying et al., 2021; Veličković et al., 2018; Zhang et al., 2020; Dwivedi and Bresson, 2021; Kreuzer et al., 2021; Rampášek et al., 2022). Heterogeneous graph Transformers target graphs with multiple node and edge types, requiring relation-aware attention rather than treating all edges uniformly. HGT conditions self-attention on node/edge types via type-specific projections and relation-dependent parameters (Hu et al., 2020). Related heterogeneous attention models leverage schema

structure through meta-path–guided aggregation, as in HAN and MAGNN (Wang et al., 2019; Fu et al., 2020). See Shehzad et al. (2024) for a recent Survey on Graph Transformers.

3 Preliminaries

As in most foundation model settings, we assume a data universe from which both pre-training data and downstream task data are drawn. In the general case, this universe can be represented as a heterogeneous graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \phi)$$

where $\mathcal{V}$ is a set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, $\tau : \mathcal{V} \to \mathcal{T}_V$ assigns each node a type from a finite set of node types $\mathcal{T}_V$, and $\phi : \mathcal{E} \to \mathcal{T}_E$ assigns each edge a relation type from a finite set of edge types $\mathcal{T}_E$. A *heterogeneous graph* is a graph with $|\mathcal{T}_E| > 1$ or $|\mathcal{T}_V| > 1$.
Each node $v \in \mathcal{V}$ is associated with an input feature vector $\mathbf{x}_v \in \mathcal{X}_{\tau(v)} \subseteq \mathbb{R}^{d_{\tau(v)}}$, where $\mathcal{X}_{\tau(v)}$ denotes the feature support of node type $\tau(v)$. We assume that node features are drawn from a type-specific marginal distribution $\mathbf{x}_v \sim P_{\tau(v)}$ with $\text{supp}(P_{\tau(v)}) = \mathcal{X}_{\tau(v)}$, while the joint distribution over all node features may exhibit arbitrary dependencies induced by the graph structure. Each edge $e = (u, v) \in \mathcal{E}$, with relation type $r = \phi(e)$, encodes a typed interaction between nodes u and v.

This general formulation recovers standard foundation model modalities as special cases. For LLMs, the graph reduces to a directed, homogeneous path graph with $|\mathcal{T}_V| = |\mathcal{T}_E| = 1$, where nodes correspond to token positions and node features take values in a finite discrete support corresponding to a vocabulary; these discrete tokens are mapped to continuous embeddings $\mathbf{x}_v \in \mathbb{R}^d$, typically augmented with positional encodings derived from the underlying path structure. For example, in ViTs, the graph is a fixed, homogeneous two-dimensional grid graph with $|\mathcal{T}_V| = |\mathcal{T}_E| = 1$, where edges connect spatially adjacent pixels, and each node corresponds to a pixel carrying a three-dimensional RGB feature vector $\mathbf{x}_v \in \mathcal{X}_{\text{RGB}} := \{0, 1, \ldots, 255\}^3$. We provide more examples and formulations in the Appendix.

In this work, we consider universes of arbitrary $\mathcal{G}$s. We discuss in the appendix the trade-offs in designing $\mathcal{G}$, and in Section 4.2 how extend it to accommodate new feature, node or edge types outside of it. $\mathcal{G}$ may be a single connected graph or consist of multiple connected components, for example as a disjoint union of graphs. Since this is a property of the data representation rather than a conceptual distinction relevant to our framework, we treat $\mathcal{G}$ as a single graph, possibly with multiple connected components, and do not distinguish between these cases in the remainder of the work.

We consider the node features $\mathbf{x}_v$ to be the token inputs to the model, regardless of whether they are obtained from raw data or through a learned tokenization, which is the standard for LLMs and recently suggested for graphs as well (Wang et al., 2025a). For the rest of this paper, we refer to tokens simply as nodes. The initial representation of a node is its initial features $\mathbf{h}_v^{(0)} \in \mathbb{R}^{d_{\tau(v)}}$. The representation of a node $v \in V$ in layer $\ell \in \{1, \ldots, L\}$ is denoted by $\mathbf{h}_v^{(\ell)} \in \mathbb{R}^{d_\ell}$, where d_ℓ is the hidden dimension at layer ℓ. We denote by $\mathcal{N}_v^a \subseteq V$ a *neighborhood* (or context) of node v, defined as a set of nodes that are allowed to exchange information with v according to some function $a : V \to 2^V$. Neighborhoods are not required to coincide with direct connectivity between nodes. All attention mechanisms use scaled dot-product attention with head dimension $d_h = H/d_\ell$ where H is the number of attention heads. Bold lowercase letters denote vectors and bold uppercase letters denote matrices.

4 The GraphBFF Transformer

In this section, we introduce GraphBFF Transformer used in our framework. GraphBFF Transformer is designed for billion-parameter GFMs, training on billion-scale heterogeneous graphs, with feasible resources. This design reflects three empirical properties of real-world heterogeneous graphs. First, node and edge type distributions may be highly imbalanced. Second, rare relation types may carry disproportionately strong signal, and third, nodes may have million-scale degrees.

The GraphBFF Transformer updates node representations in each layer, based on their neighborhoods in the graph. Each GraphBFF Transformer layer follows the standard transformer encoder block structure (Vaswani et al., 2017), with residual connections, layer normalization, and a feed-forward network (FFNs), while replacing the all-pairs self-attention sub-block with two heterogeneous graph-aware masked attention modules.

The GraphBFF introduces the *Type-Conditioned Attention* (TCA) which also builds on type-specific attention transformations as previously suggested in Heterogeneous-Graph Transformer (HGT) (Hu et al., 2020), with a small yet crucial difference: a sparse softmax is applied to each type of neighbors separately, rather than to all neighbors at once. As softmax computation is often a bottleneck in self-attention due to the need to materialize and normalize the full attention matrix, leading to high memory bandwidth and I//O costs when applied to large neighborhoods.

To enable cross-type attention, GraphBFF Transformer also introduces a *Type-Agnostic Attention* (TAA) component, which applies a shared attention among neighbors, with a sparse, fixed-degree neighbor sampling. This guarantees that efficiency is preserved, while also reducing the risk of overfitting to the graph structure in cases where the node degrees follow highly non-regular distributions Bechler-Speicher et al. (2024). Importantly, this component shares its attention matrices across all edge types, therefore introducing a relatively small number of additional parameters compared with the TCA component, yet it strictly increases the model expressivity as we prove in Theorem 4.1. The final node representation in heterogeneous attention block is then a learned combination of the TAA and TCA. See Figure 2 for an overview of the GraphBFF Transformer block.

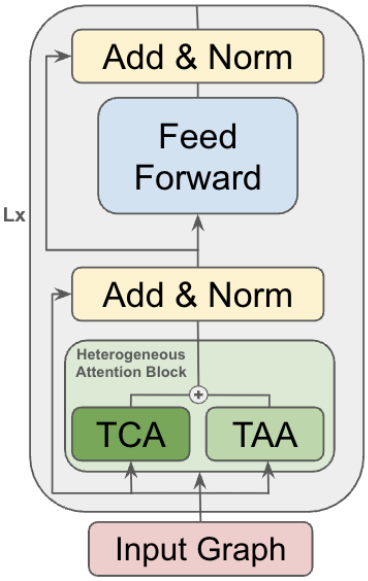


Figure 2 The GraphBFF Transformer block.

Type-Conditioned Attention (TCA) The TCA component performs masked self-attention operations within neighborhoods $\mathcal{N}_v^{\text{tca}}$, limited to specific subsets of edge types. For a set $S \subseteq \mathcal{T}_E$, the node is allowed to attend only to nodes within $\mathcal{N}_v^{\text{tca}}$ that are connected to $\mathcal{N}_v^{\text{tca}}$ through edge types in S. Multiple sets $\mathcal{S}$ can be defined, and TCA applies an independent self-attention with respect to each set S, and then aggregates the resulting representations into a single one. For example, TCA can simply be edge-type specific attention if $\mathcal{N}_v^{\text{tca}}$ is set to the direct neighbors of v, and each set corresponds to one edge type.

For every node $u \in \mathcal{N}_v^S$:

$$\mathbf{q}_v^{(S,\ell)} = \mathbf{W}_Q^{(S,\ell)}\,\mathbf{h}_v^{(\ell)} \in \mathbb{R}^{d_h}, \quad \mathbf{k}_u^{(S,\ell)} = \mathbf{W}_K^{(S,\ell)}\,\mathbf{h}_u^{(\ell)} \in \mathbb{R}^{d_h}, \quad \mathbf{v}_u^{(S,\ell)} = \mathbf{W}_V^{(S,\ell)}\,\mathbf{h}_u^{(\ell)} \in \mathbb{R}^{d_h}$$

where

$$\mathbf{W}_Q^{(S,\ell)}, \mathbf{W}_K^{(S,\ell)}, \mathbf{W}_V^{(S,\ell)} \in \mathbb{R}^{d_{\ell-1} \times d_h}$$

are key, query, and value projection matrices for layer ℓ for a subset of edge types $S \subseteq \mathcal{T}_E$. The attention weights for set S are:

$$\alpha_{uv}^{(S,\ell)} = \frac{\exp\left(\frac{(\mathbf{q}_v^{(S,\ell)})^\top \mathbf{k}_u^{(S,\ell)}}{\sqrt{d_h}}\right)}{\sum_{u' \in \mathcal{N}_v^S} \exp\left(\frac{(\mathbf{q}_v^{(S,\ell)})^\top \mathbf{k}_{u'}^{(S,\ell)}}{\sqrt{d_h}}\right)}$$

The set-specific representation of node v for set S is:

$$\mathbf{h}_v^{(\ell,\text{S})} = \sum_{u \in \mathcal{N}_v^S} \alpha_{uv}^{(S,\ell)}\,\mathbf{v}_u^{(S,\ell)} \in \mathbb{R}^{d_h}$$

The overall TCA representation of node v in layer ℓ is obtained by aggregating its set-specific representations across all defined sets in $\mathcal{S}$:

$$\mathbf{h}_v^{(\ell,\text{tca})} = \sum_{S \in \mathcal{S},\ \mathcal{N}_v^S \neq \emptyset} \mathbf{h}_v^{(\ell,\text{S})}$$

Type-Agnostic Attention (TAA) The TAA component performs self-attention between a node and all other nodes in its TAA neighborhood $\mathcal{N}_v^{\text{taa}}$. It first projects all nodes to the same embedding dimension. This component is parameter-efficient as the attention matrices are shared across all node types, yet it can be expensive when applied to large neighborhoods. Therefore, we further apply a fixed sampling function q over the neighborhood $\mathcal{N}_v^{\text{taa}}$, $q : \mathcal{N}_v^{\text{taa}} \to 2^{\mathcal{N}_v^{\text{taa}}}$.

Formally, node representations are first mapped into a shared latent space. Each node type $\tau \in \mathcal{T}_V$ is associated with a projection matrix

$$\mathbf{W}_\tau^{(\ell)} \in \mathbb{R}^{d_\ell \times d_\tau}$$

, yielding

$$\widehat{\mathbf{h}}_v^{(\ell)} = \mathbf{W}_{\tau(v)}^{(\ell)} \mathbf{h}_v^{(\ell)} \in \mathbb{R}^{d_\ell}.$$

We then use *shared* attention matrices across all nodes and edge types:

$$\mathbf{W}_Q^{(\ell)}, \mathbf{W}_K^{(\ell)}, \mathbf{W}_V^{(\ell)} \in \mathbb{R}^{d_h \times d_\ell}.$$

For any neighbor $u \in \mathcal{N}_v$, we compute

$$\mathbf{q}_v^{(\ell)} = \mathbf{W}_Q^{(\ell)} \widehat{\mathbf{h}}_v^{(\ell)} \in \mathbb{R}^{d_h}, \qquad \mathbf{k}_u^{(\ell)} = \mathbf{W}_K^{(\ell)} \widehat{\mathbf{h}}_u^{(\ell)} \in \mathbb{R}^{d_h}, \qquad \mathbf{v}_u^{(\ell)} = \mathbf{W}_V^{(\ell)} \widehat{\mathbf{h}}_u^{(\ell)} \in \mathbb{R}^{d_h}.$$

The type-agnostic attention weights are

$$\beta_{uv}^{(\ell)} = \frac{\exp\left(\frac{(\mathbf{q}_v^{(\ell)})^\top \mathbf{k}_u^{(\ell)}}{\sqrt{d_h}}\right)}{\sum_{u' \in \mathcal{N}_v^{\text{taa}}} \exp\left(\frac{(\mathbf{q}_v^{(\ell)})^\top \mathbf{k}_{u'}^{(\ell)}}{\sqrt{d_h}}\right)},$$

and the resulting TAA embedding is

$$\mathbf{h}_v^{(\ell,\text{taa})} = \sum_{u \in \mathcal{N}_v^{\text{taa}}} \beta_{uv}^{(\ell)} \mathbf{v}_u^{(\ell)} \in \mathbb{R}^{d_h}.$$

Finally, the TCA and TAA embeddings are combined into a single output for node v, using an FFN $\Phi^{(\ell)}$:

$$\widetilde{\mathbf{h}}_v^{(\ell)} = \Phi^{(\ell)}\left(\mathbf{h}_v^{(\ell,\text{tca})}, \mathbf{h}_v^{(\ell,\text{taa})}\right) \in \mathbb{R}^{d_\ell},$$

Following the heterogeneous attention sub-block, $\widetilde{\mathbf{h}}_v^{(\ell)}$ and $\mathbf{z}_v^{(\ell)}$ are passed through the remaining components of the transformer encoder block:

$$\mathbf{z}_v^{(\ell)} = \text{LayerNorm}\left(\mathbf{h}_v^{(\ell)} + \widetilde{\mathbf{h}}_v^{(\ell)}\right), \quad \mathbf{h}_v^{(\ell+1)} = \text{LayerNorm}\left(\mathbf{z}_v^{(\ell)} + \text{FFN}\left(\mathbf{z}_v^{(\ell)}\right)\right).$$

The next theorem shows that a GraphBFF Transformer with both TCA and TAA is strictly more expressive than one lacking either component.

Theorem 4.1. *Consider a GraphBFF Transformer layer with hidden dimension $d_\ell = d$ and number of heads H, and heterogeneous attention sub-block TCA and TAA. Let $\mathcal{F}_{\text{GraphBFF}}, \mathcal{F}_{\text{TAA}}, \mathcal{F}_{\text{TCA}}$ be the sets of realizable functions by the GraphBFF Transformer with both TCA and TAA, with just TAA and with just TCA, respectively. There exists a function f such that: $f \in \mathcal{F}_{\text{GraphBFF}}$ but $f \notin \mathcal{F}_{\text{TAA}}$ and $f \notin \mathcal{F}_{\text{TCA}}$.*

We prove the theorem in the Appendix. To prove it, we construct a function that TCA fails to realize due to softmax normalization within edge-type subsets effectively masking the relative cardinality of neighbor sets, and TAA fails to realize due to shared parameters rendering it blind to specific edge-type distinctions, yet TAA and TCA together can realize it. In Section 5 we show that using both TCA and TAA is also preferable empirically, leading to better generalization.

4.1 Pre-Training

In this section, we describe the pre-training methodology of GraphBFF, including self-supervision and batching. Many self-supervised objectives have been proposed for graphs (Veličković et al., 2019; Hou et al., 2023; You et al., 2020; Hu* et al., 2020), typically to inject stronger, task- or structure-specific inductive biases. Inspired by recent results showing that scale can compensate for inductive bias (Brehmer et al., 2025; Tay et al., 2022; Bahri et al., 2024) we conjecture that a simple masked link prediction (Kipf and Welling, 2016; Hou et al., 2022) applied at billion-scale is sufficient for obtaining effective GFMs.

Batching graph data is commonly used to enable machines with constrained memory to load and process large-scale graphs. Since we address heterogeneous and potentially type-skewed graphs, we want to ensure that training is done over batches that represent the graph well, specifically, distribution of nodes and edges. Furthermore, the type-skewness of the graphs might result in biased training due to highly common edge types, and noisy gradients for rare types.

To address these issues, we deploy a two stage batching process. First, we pre-process the graph into batches of edges using a novel *KL-Batching* algorithm. This is achieved by first partitioning large connected-components through clustering and then greedily aggregating clusters that are the most similar to the overall graph distribution, using KL-divergence as a measure of distribution similarity, until reaching desired batch size. This way, we have batches, each having a similar distribution to the large graph, and at the required size to fit the machine memory.

The second batching is *Round-Robin Batching which* happens during pre-training. GPUs can handle much smaller batches, and in order to make sure common edge-types doesn't overwhelm the training dynamics, we group edges by type, and iterate over groups from different types in a fixed cyclic order.

Storage-level batching As transferring data from storage into memory is costly, we aim to maximize batch size while respecting a memory budget M. Standard practice is to partition the graph into clusters (Chiang et al., 2019), e.g., via Leiden (Traag et al., 2019), however, such methods result in varying cluster size and node/edge-type distribution. This may result in non-representative clusters, and in case of large effective batch sizes in multi-machine pre-training, under-fill the batch capacity and create early distributional bias that destabilizes optimization. To address this, we propose *KL-Batching*, which operates as follows (1) use a graph clustering algorithm to partition the graph into small, disjoint clusters, (2) compute, for each cluster an empirical distribution p_k over a chosen categorical attribute (e.g., node or edge type) and measures its representativeness via $\mathrm{KL}(p_k \| p_G)$ against the global reference distribution p_G (optionally combining multiple attributes with weighted KL terms), (3) constructs batches by greedily aggregating whole clusters into batches, by increasing KL value, until the estimated batch cost reaches the memory limit M. This yields memory-efficient batches that better match global type distributions and utilize memory more effectively.

GPU-level batching KL-batching yields batches that can be loaded into memory, but must be further subdivided into mini-batches that fit GPU memory. Randomly splitting the KL-batches for highly skewed graph may produce mini-batches overwhelmingly composed of common edge types, yielding a biased training signal and noisy gradients for rare relations. To avoid this, we propose *Round-Robin Batching (RRB)*. In RRB, we group supervision edges by their type, and iterate over types in a fixed cyclic order. At each step, we form a mini-batch by sampling supervision edges (and their negative counterparts) of the current type, and materialize their neighborhoods from the KL-batch, excluding the supervised edges themselves. This simple scheduling leads to more stable training and better coverage of the heterogeneous edge-type space.

4.2 Fine-Tuning and Extending $\mathcal{G}$

Fine-tuning can be performed in both supervised and unsupervised settings, and serves several distinct use cases. The most straightforward scenarios involve adapting a pretrained GraphBFF to new data drawn from the same underlying universe, or multi-task supervised training to improve performance on one or more downstream tasks. In all of these cases, standard parameter-efficient fine-tuning methods such as LoRA (Hu et al., 2022) can be applied to GraphBFF with the additional flexibility of constraining updates to specific type-indexed matrices for node and edge types, and to selected sub-modules such as the TAA or TCA components.

	Model -Context	Task 1 PRAUC	Task 2 PRAUC	Task 3 PRAUC	Task 4 PRAUC	Task 5 PRAUC	Task 6 PRAUC	Task 7 MAE	Task 8 PRAUC	Task 9 PRAUC	Task 10 PRAUC
	NN	$65.61_{\pm 1.01}$	$72.06_{\pm 0.01}$	$51.43_{\pm 0.01}$	$63.05_{\pm 3.80}$	$91.20_{\pm 4.00}$	$58.51_{\pm 0.35}$	$0.032_{\pm 0.01}$	$85.76_{\pm 1.21}$	$63.66_{\pm 3.26}$	$55.48_{\pm 0.79}$
Task-Specific	HGT-1	$61.23_{\pm 1.95}$	$66.85_{\pm 2.27}$	$71.83_{\pm 0.20}$	$90.43_{\pm 0.18}$	$60.99_{\pm 12.3}$	$71.06_{\pm 2.02}$	$0.236_{\pm 0.02}$	$69.54_{\pm 1.21}$	$30.08_{\pm 5.99}$	$60.78_{\pm 0.49}$
	HAN-1	$61.73_{\pm 1.85}$	$62.83_{\pm 1.50}$	$64.12_{\pm 0.88}$	$94.70_{\pm 0.30}$	$76.34_{\pm 5.81}$	$70.10_{\pm 0.94}$	$0.083_{\pm 0.01}$	$67.64_{\pm 2.35}$	$36.87_{\pm 2.31}$	$60.59_{\pm 3.28}$
	GraphBFF-1	$64.77_{\pm 1.70}$	$67.94_{\pm 0.24}$	$73.86_{\pm 0.71}$	$95.34_{\pm 0.51}$	$75.70_{\pm 4.12}$	$75.47_{\pm 0.69}$	$0.073_{\pm 0.01}$	$72.47_{\pm 4.75}$	$42.62_{\pm 4.80}$	$62.92_{\pm 3.22}$
	HGT-2	$64.73_{\pm 0.61}$	$72.34_{\pm 0.86}$	$71.94_{\pm 3.31}$	$95.02_{\pm 1.41}$	$63.26_{\pm 18.8}$	$72.86_{\pm 2.18}$	$0.186_{\pm 0.02}$	$70.67_{\pm 0.87}$	$30.71_{\pm 7.18}$	$58.63_{\pm 3.55}$
	HAN-2	$61.65_{\pm 1.88}$	$76.26_{\pm 0.79}$	$67.14_{\pm 0.96}$	$94.70_{\pm 5.87}$	$43.26_{\pm 5.97}$	$69.74_{\pm 0.95}$	$0.161_{\pm 0.01}$	$59.38_{\pm 2.04}$	$31.92_{\pm 13.9}$	$63.23_{\pm 2.03}$
	GraphBFF-2	$69.81_{\pm 1.51}$	$81.48_{\pm 0.66}$	$79.45_{\pm 1.82}$	$94.41_{\pm 2.61}$	$63.15_{\pm 3.97}$	$72.29_{\pm 4.21}$	$0.156_{\pm 0.02}$	$71.13_{\pm 3.78}$	$37.41_{\pm 4.75}$	$63.76_{\pm 3.03}$
	HGT-3	$76.88_{\pm 2.14}$	$77.66_{\pm 0.16}$	$67.67_{\pm 0.95}$	$91.75_{\pm 0.15}$	$45.59_{\pm 4.55}$	$69.51_{\pm 2.56}$	$0.135_{\pm 0.01}$	$71.18_{\pm 0.78}$	$33.88_{\pm 6.76}$	$57.85_{\pm 3.83}$
	HAN-3	$74.32_{\pm 0.96}$	$74.75_{\pm 3.21}$	$72.47_{\pm 1.18}$	$94.83_{\pm 0.69}$	$53.98_{\pm 17.5}$	$69.46_{\pm 1.74}$	$0.138_{\pm 0.02}$	$70.23_{\pm 1.08}$	$30.71_{\pm 9.95}$	$57.58_{\pm 2.21}$
	GraphBFF-3	$74.42_{\pm 1.74}$	$79.34_{\pm 2.17}$	$80.15_{\pm 1.75}$	$93.56_{\pm 0.49}$	$54.16_{\pm 0.01}$	$69.65_{\pm 1.19}$	$0.146_{\pm 0.03}$	$78.83_{\pm 2.80}$	$42.54_{\pm 3.40}$	$63.24_{\pm 2.39}$
1 Shot	GFM-1	$50.07_{\pm 0.03}$	$47.99_{\pm 0.72}$	$59.34_{\pm 6.40}$	$93.96_{\pm 1.26}$	$90.10_{\pm 5.21}$	$73.79_{\pm 0.68}$	$0.059_{\pm 0.09}$	$72.19_{\pm 6.71}$	$63.48_{\pm 11.43}$	$77.22_{\pm 1.37}$
	GFM-2	$51.11_{\pm 2.03}$	$60.52_{\pm 1.87}$	$48.48_{\pm 2.25}$	$91.28_{\pm 0.10}$	$97.61_{\pm 1.02}$	$74.29_{\pm 1.48}$	$0.072_{\pm 0.21}$	$78.44_{\pm 1.12}$	$59.22_{\pm 10.84}$	$76.77_{\pm 1.30}$
	GFM-3	$62.18_{\pm 1.20}$	$53.98_{\pm 4.55}$	$36.38_{\pm 5.10}$	$88.88_{\pm 4.09}$	$34.26_{\pm 0.63}$	$70.41_{\pm 0.49}$	$0.039_{\pm 0.02}$	$48.56_{\pm 2.22}$	$49.52_{\pm 11.21}$	$54.36_{\pm 2.83}$
2 Shot	GFM-1	$60.69_{\pm 1.65}$	$60.97_{\pm 6.12}$	$40.93_{\pm 1.70}$	$94.11_{\pm 2.52}$	$93.25_{\pm 1.10}$	$48.79_{\pm 2.10}$	$0.052_{\pm 0.04}$	$82.37_{\pm 1.64}$	$76.32_{\pm 3.74}$	$74.30_{\pm 1.02}$
	GFM-2	$66.24_{\pm 1.96}$	$62.09_{\pm 2.79}$	$58.84_{\pm 3.58}$	$90.28_{\pm 0.95}$	$78.21_{\pm 0.28}$	$55.52_{\pm 2.40}$	$0.071_{\pm 0.24}$	$77.85_{\pm 0.46}$	$81.55_{\pm 0.15}$	$62.66_{\pm 1.51}$
	GFM-3	$43.51_{\pm 1.37}$	$58.51_{\pm 0.71}$	$67.95_{\pm 4.00}$	$87.88_{\pm 2.81}$	$82.63_{\pm 3.92}$	$71.86_{\pm 0.91}$	$0.038_{\pm 0.03}$	$82.81_{\pm 0.03}$	$66.04_{\pm 0.70}$	$62.24_{\pm 1.01}$
5 Shot	GFM-1	$59.97_{\pm 6.31}$	$59.74_{\pm 2.87}$	$41.53_{\pm 3.23}$	$85.05_{\pm 1.32}$	$85.70_{\pm 9.48}$	$64.82_{\pm 0.65}$	$0.037_{\pm 0.02}$	$82.81_{\pm 0.03}$	$88.84_{\pm 0.38}$	$76.90_{\pm 0.71}$
	GFM-2	$60.03_{\pm 2.69}$	$55.79_{\pm 1.65}$	$64.41_{\pm 1.35}$	$84.96_{\pm 1.33}$	$79.94_{\pm 0.01}$	$44.71_{\pm 1.35}$	$0.025_{\pm 0.02}$	$74.01_{\pm 1.09}$	$79.98_{\pm 0.46}$	$62.01_{\pm 8.98}$
	GFM-3	$68.45_{\pm 0.71}$	$51.89_{\pm 1.52}$	$63.43_{\pm 1.26}$	$96.10_{\pm 0.31}$	$54.84_{\pm 3.49}$	$67.95_{\pm 0.34}$	$0.028_{\pm 0.01}$	$79.41_{\pm 0.17}$	$70.02_{\pm 0.41}$	$57.80_{\pm 0.85}$
10 Shot	GFM-1	$65.41_{\pm 0.28}$	$57.25_{\pm 0.56}$	$45.77_{\pm 1.80}$	$98.75_{\pm 0.06}$	$79.33_{\pm 2.31}$	$62.55_{\pm 1.85}$	$0.036_{\pm 0.01}$	$83.86_{\pm 0.17}$	$87.36_{\pm 0.08}$	$81.55_{\pm 2.92}$
	GFM-2	$58.45_{\pm 0.46}$	$53.06_{\pm 1.36}$	$63.33_{\pm 1.91}$	$96.49_{\pm 0.12}$	$81.57_{\pm 6.15}$	$74.08_{\pm 0.35}$	$0.023_{\pm 0.01}$	$76.61_{\pm 3.05}$	$78.85_{\pm 0.33}$	$42.10_{\pm 0.01}$
	GFM-3	$55.54_{\pm 1.27}$	$83.86_{\pm 0.40}$	$58.16_{\pm 2.40}$	$96.47_{\pm 046}$	$90.07_{\pm 1.12}$	$67.97_{\pm 1.33}$	$0.027_{\pm 0.01}$	$93.92_{\pm 0.08}$	$76.25_{\pm 0.73}$	$75.25_{\pm 1.32}$
Probing	GFM-1	$84.07_{\pm 0.37}$	$82.58_{\pm 0.01}$	$83.03_{\pm 0.07}$	$98.31_{\pm 0.21}$	$60.70_{\pm 0.39}$	$75.85_{\pm 1.90}$	$0.021_{\pm 0.01}$	$91.48_{\pm 0.92}$	$\mathbf{95.19}_{\pm 0.20}$	$\mathbf{79.65}_{\pm 1.46}$
	GFM-2	$83.50_{\pm 0.13}$	$\mathbf{85.73}_{\pm 0.05}$	$84.36_{\pm 0.04}$	$98.36_{\pm 0.25}$	$83.93_{\pm 0.69}$	$\mathbf{82.05}_{\pm 0.04}$	$\mathbf{0.018}_{\pm 0.01}$	$92.78_{\pm 0.20}$	$89.03_{\pm 0.58}$	$60.77_{\pm 4.64}$
	GFM-3	$\mathbf{88.13}_{\pm 0.15}$	$83.00_{\pm 0.01}$	$\mathbf{86.14}_{\pm 0.11}$	$\mathbf{99.48}_{\pm 0.19}$	$\mathbf{95.60}_{\pm 0.32}$	$75.54_{\pm 1.04}$	$0.021_{\pm 0.01}$	$\mathbf{95.39}_{\pm 0.23}$	$85.70_{\pm 0.56}$	$78.48_{\pm 0.01}$

Table 1 A comparison between the GFM with up to 2 layers NN prediction head (Probing) and task-specific heterogeneous transformers (GraphBFF, HGT, HAN). For each model, we compare 3 context sizes corresponding to ego-graphs of varying radius centered on the target node or edge. The best performance overall is highlighted in bold. The best performance for each context size for the task-specific baselines is highlighted in blue, green and red.

Downstream tasks may introduce features, node types, or edge types that were not included in G as they did not exist during pre-training, or as a modeling choice, for example if these types very sparse over the graph, or relevant only to a small amount of downstream tasks. An approach to avoid the need to extend $\mathcal{G}$, is to map new features or node types into existing ones. These approaches were extensively researched in the TabFMs domain, with recent works applying them to graphs as well. We discuss these approaches in Section 2. In an extension of $\mathcal{G}$, we extend it to a richer universe G' with new features or types, by introducing new learned weights to the GraphBFF Transformer. We then train these weights through fine-tuning, possibly with the rest of the model frozen (Tai et al., 2020). This approach may be preferable when some feature or node types are relevant for only a small fraction of downstream tasks, or when they occur in only a small fraction of the data.

5 Probing, Few-Shot and Zero-Shot

In this section, we present an extensive evaluation of a 1.4 billion-parameters GraphBFF Transformer pre-trained on one billion samples. We evaluate it over diverse downstream tasks, spanning node and edge level classification and regression. We consider probing, few-shot probing, and zero-shot settings. We also perform an ablation study, evaluating a GraphBFF Transformer with only the TCA or the TAA components.

5.1 Setting and data

We pre-trained a 1.4 billion-parameters GraphBFF Transformer using the complete GraphBFF recipe, on one billion edges sampled from the same graph used in Section 6. This graph has ∼50 billion nodes and edges, 12 node types and 20 relation types, including financial, social, business, infra, and more. Each node type is associated with its own features. As the neighborhoods for the GraphBFFTransformer, we use direct neighbors for $\mathcal{N}^{\text{tca}}$, i.e., nodes connected by a single edge. For $\mathcal{N}^{\text{taa}}$, we use nodes within two hops of the target node. Within the TAA component, sampling function q selects up to 10 random neighbors per hop. This accommodates graph scale and mitigates structural overfitting, a common issue for sparse graph

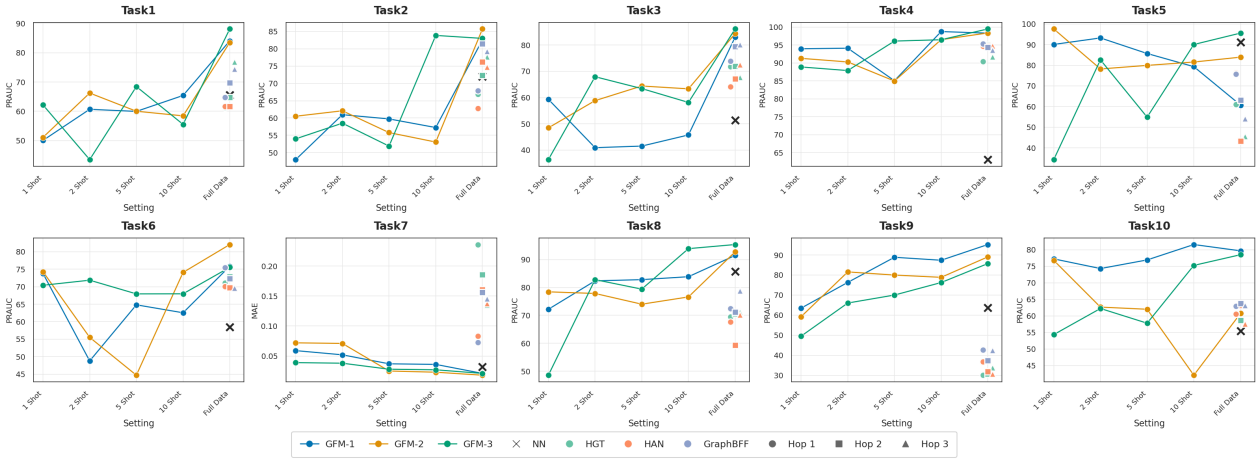


Figure 3 Performance across Tasks 1–10 under different data settings: 1/2/5/10-shot and full-data. Lines show GFM-1/2/3. Markers at the full-data point indicate task-specific baselines with context size encoded by marker shape.

transformers operating on large-variance node-degree distributions Bechler-Speicher et al. (2024). The TCA component is the primary driver of model capacity, comprising about 85/Pre-training used eight Nvidia B200 machines and was completed in just 12 hours with 64 GPUs.

We use 10 diverse real-world industrial tasks including node and edge classification and regression, denoted as Task1 to Task10, where tasks $1-7$ are node-level tasks, and tasks $8-10$ are edge-level tasks. The tasks span finance, security, social and Attribution tasks, and defined over different node and edge types. As evaluation metrics, we use PRAUC for classification and MAE for regression. All the tasks' graphs are not part of the pre-training data, nor their nodes are connected to nodes in the pre-training data. All tasks are out-of-distribution with respect to the pre-training data. The amount of labels for the full data setting ranges from a minimum of 98 samples in Task 4 to a maximum of $50k$ labels in Task 5. Across the evaluation, we used as context for the GFM the ego-graph (Stanley Wasserman, 1994) centered around target nodes or edges. We evaluate ego-graph denoted as *k-hops* with radius $k\in\{1,2,3\}$ to also examine how the context size affects performance. Each task has its own train (80%) validation (10%) and test (10%) splits. We perform a grid search with 52 hyper-param configurations for each setting, select the best configuration over the validation set, and report the mean and standard-deviation (std) of the measured metric over the test set with 3 random seeds using the selected configuration. All task evaluations uses a grid search with layers in $\{1,2\}$, hidden dimensions in $\{64,128,256\}$, dropout ratio in $\{0,0.2,0.6\}$, learning rate in $\{0.001,0.0001,0.0005\}$. We use 1000 epochs with early stopping on the validation loss, with patience of 200 steps.

Full-Data probing setup We evaluate all ten tasks using (i) one-layer linear probes and (ii) up to two-layer non-linear probes. The GFM weights remain frozen and only the probe is trained. In particular, we train a vanilla feed-forward NN probe on top of the frozen GFM and compare it against other models of the same size trained on the same data. We compare against a feed-forward neural network (NN) probe, HGT (Hu et al., 2020), HAN (Wang et al., 2019), and a GraphBFF Transformer. Note that these graph-transformer baselines are trained on the input graph information, namely, the task-specific graph structure. In contrast, in the GFM probing setup these are only forwarded through the frozen GFM, and the NN probe is trained solely on the resulting GFM output embeddings.

Few-Shot probing setup For few-shot, we examine the performance of the NN probe over the GFM with 1,2, 5 and 10 training samples from class, randomly sampled from the training set. For regression, we use 1,2, 5 and 10 samples.

Zero-Shot setup For zero-shot, we evaluate the link-prediction tasks using the frozen link-prediction head of the pre-trained GFM. Node-level zero-shot is not well-defined for nodes with a feature dimension that is different than the task prediction dimension. The same issue arises in vision tasks, as the ViT output is in high dimension, and some projection to the label space must be learned. As recently noted by Eremeev et al. (2025), existing "zero-shot" node level evaluation uses labeled data from the pre-training graph, similarly to transduction settings, which essentially violates the definition of zero-shot prediction. Moreover, this

Model -Context	Task 1 PRAUC	Task 2 PRAUC	Task 3 PRAUC	Task 4 PRAUC	Task 5 PRAUC	Task 6 PRAUC	Task 7 MAE	Task 8 PRAUC	Task 9 PRAUC	Task 10 PRAUC
TCA-only-1	$80.92_{\pm 0.48}$	$79.86_{\pm 0.22}$	$80.41_{\pm 0.31}$	$97.62_{\pm 0.28}$	$56.44_{\pm 0.55}$	$71.03_{\pm 1.62}$	$0.027_{\pm 0.01}$	$88.73_{\pm 0.61}$	$90.84_{\pm 0.41}$	$73.28_{\pm 1.81}$
TCA-only-2	$79.71_{\pm 0.29}$	$82.63_{\pm 0.18}$	$81.92_{\pm 0.20}$	$97.71_{\pm 0.33}$	$79.54_{\pm 0.74}$	$77.14_{\pm 0.34}$	$0.024_{\pm 0.01}$	$89.86_{\pm 0.33}$	$84.62_{\pm 0.77}$	$55.33_{\pm 3.90}$
TCA-only-3	$84.65_{\pm 0.25}$	$80.34_{\pm 0.12}$	$83.02_{\pm 0.19}$	$98.84_{\pm 0.23}$	$91.02_{\pm 0.46}$	$70.92_{\pm 1.21}$	$0.026_{\pm 0.01}$	$92.18_{\pm 0.28}$	$80.61_{\pm 0.68}$	$72.05_{\pm 0.82}$
Δ_{best}	-3.48↓	-3.10↓	-3.12↓	-0.64↓	-4.58↓	-4.91↓	+0.006↑	-3.21↓	-4.35↓	-6.37↓
TAA-only-1	$61.44_{\pm 0.72}$	$58.92_{\pm 0.41}$	$60.23_{\pm 0.55}$	$90.35_{\pm 0.66}$	$41.08_{\pm 0.83}$	$52.77_{\pm 2.05}$	$0.044_{\pm 0.02}$	$72.61_{\pm 1.10}$	$77.43_{\pm 0.95}$	$49.92_{\pm 2.40}$
TAA-only-2	$59.80_{\pm 0.54}$	$62.15_{\pm 0.49}$	$61.07_{\pm 0.43}$	$90.61_{\pm 0.58}$	$58.02_{\pm 1.10}$	$56.48_{\pm 0.88}$	$0.041_{\pm 0.02}$	$74.38_{\pm 0.84}$	$70.05_{\pm 1.22}$	$38.40_{\pm 4.95}$
TAA-only-3	$64.92_{\pm 0.49}$	$59.61_{\pm 0.37}$	$62.84_{\pm 0.46}$	$92.18_{\pm 0.44}$	$71.35_{\pm 0.92}$	$52.10_{\pm 1.67}$	$0.043_{\pm 0.02}$	$78.92_{\pm 0.63}$	$66.21_{\pm 1.31}$	$47.38_{\pm 1.73}$
Δ_{best}	-23.21↓	-23.58↓	-23.30↓	-7.30↓	-24.25↓	-25.57↓	+0.023↑	-16.47↓	-17.76↓	-29.73↓

Table 2 Ablation study of the GFM components evaluation the full-data probing performance of using only TCA or TAA. Δ_{best} denotes the performance margin between the best-performing context configuration in the ablation and the overall best performance achieved by the full GFM architecture as reported in Table 1.

Model - Context	Task 8 PRAUC	Task 9 PRAUC	Task 10 PRAUC
GFM-1	54.23	77.13	73.05
GFM-2	55.35	39.48	53.21
GFM-3	49.89	41.16	73.84

Table 3 Zero-shot link prediction using the GFM's pre-trained link predictor on three link-classification tasks. We evaluate three context sizes, corresponding to ego-graphs of varying radius centered on the target edge.

evaluation is not possible if inference tasks data have no intersection with the pre-training data at all, as in our case. Therefore, for the node-level tasks we perform zero-shot separation analysis instead.

5.2 Results

The probing and few-shot results, detailed in Table 1 and Figure 3, demonstrate the significant performance gains enabled by the GFM. Most notably, we find that a simple NN probing head applied to frozen GraphBFF representations consistently outperforms all task-specific models trained from scratch. Specifically, GFM Probing achieves the best overall performance in 10 out of 10 tasks, frequently surpassing the highest task-specific baseline by margins exceeding 10% in metrics like PRAUC. For instance, in Task 1, the best task-specific model (GraphBFF-3) reaches 74.42, while the GFM Probing achieves 88.13. This consistent dominance suggests that the GFM effectively captures a generalized structural and semantic language of the graph that supervised models struggle to learn in isolation. A particularly striking example is found in Task 9, where the best task-specific model (GraphBFF-1) reaches only 42.62 PRAUC, while the GFM Probing achieves 95.19—representing a gap of over 52 points. The necessity of the full GraphBFF Transformer architecture is supported by Table 2, where isolating the TCA or TAA components leads to a consistent performance drop across all 10 tasks. Nonetheless, even when isolating the TCA component, the model outperforms all task-specific baselines across all tasks.

Interestingly, the results indicate that increasing the context size does not lead to monotonic improvements. In Task 2, for example, a 2-hop context (GraphBFF-2 at 81.48) provides stronger generalization than 3 hops (GraphBFF-3 at 79.34), and this pattern is even more pronounced in the GFM Probing where GFM-2 achieves 85.73 compared to 83.00 for GFM-3. This may occur if larger neighborhoods introduce structural noise.

Within the category of task-specific baselines, GraphBFF consistently emerges as the strongest performer compared to HGT and HAN. GraphBFF outperforms both HGT and HAN in 8 out of 10 tasks when comparing their best respective context sizes, proving to be a robust backbone not only for foundation modeling but also for direct supervised applications.

Our few-shot analysis reveals that GFM representations are highly expressive even with minimal supervision. While performance generally improves with more data, we observe that the increase from 1-shot to 10-shot is not always monotonic, echoing findings in other domains (Luo et al., 2023; Radford et al., 2021) where low-shot regimes can be sensitive to the specific samples selected. Despite this, the 10-shot GFM performance

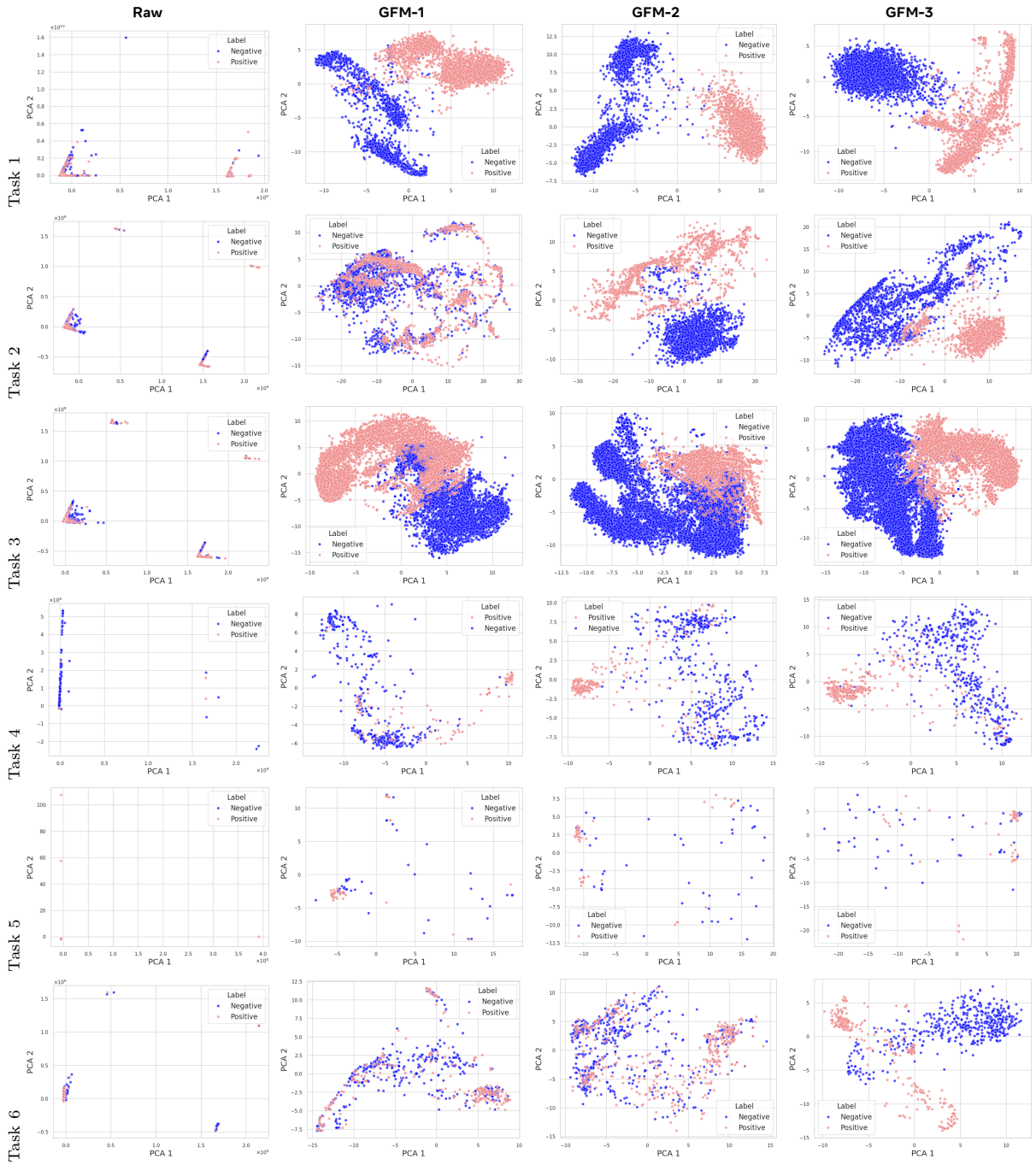
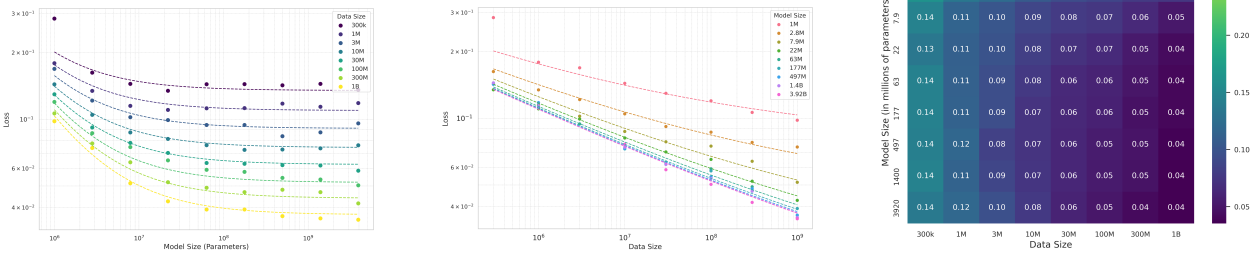


Figure 4 Zero-shot separation for the datasets 1-6. The raw features and GFM embeddings for context sizes 1 2 and 3 are projected to 2D space using PCA, and the points are colored according to their label.

frequently rivals or exceeds the full-dataset performance of standard HGT and HAN models. A striking example is seen in Task 9, where the 10-shot GFM-1 (87.36) outperforms all versions of HGT and HAN trained on the full dataset, which fail to break the 40.00 PRAUC barrier, underscoring the efficiency and superior feature alignment of the learned foundation representations.

Figure 4 presents the zero-shot separation results for Task 4. The separation for the rest of the tasks show similar trends and are deferred to the Appendix due to space limitations. With just the raw features, we do not observe meaningful separation between the positive and negative classes. In contrast, the GFM yields strong zero-shot separation. The zero-shot link-prediction performance are presented in Table 3. For Task 10,



(a) Validation loss as a function of the model size, for a fixed data size.

(b) Validation loss as a function of the data size, for a fixed data model.

(c) Validation loss as a function of the model size and data size.

Figure 5 The early-stop validation loss as a function of model size and data size. The dashed lines are obtained by fitting the exponents in Equation (1). Loss decreases predictably depending on the model and data size, across 5 orders of magnitudes.

GFM in the zero-shot setting performs strictly better than all GNN baselines, even though those baselines are trained on the full task dataset, with an absolute improvement of 10.08 points.

6 Neural Scaling Laws

In this section, we present the first neural scaling laws for GFMs on general heterogeneous graphs. Prior works on LLMs have shown that test loss follows predictable power-law relationships in both model size and data size, with clear transitions between data-limited and model-limited regimes as one of these resources is held fixed and the other is scaled (Kaplan et al., 2020). Here we show that GFMs follow the same trends, and show that gains are smooth when scaling model and data together, but saturate when one is fixed. Test loss continues to decrease in a predictable manner when both model capacity and training data are increased jointly; by contrast, holding either capacity or data fixed while scaling the other leads to a clear diminishing-returns regime.

6.1 Setup

We begin with necessary notation, aligned with the notation in Kaplan et al. (2020). Let N denote the number of model parameters and let D denote the number of distinct supervised training edges, see the Appendix for discussion on the definition of D. For each pair (N, D), we define $L(N, D)$ to be the best validation loss achieved when training on exactly D samples with a model of size N. We fit the joint scaling-law form from Kaplan et al. (2020):

$$L(N, D) \;=\; L_\infty \;+\; \left(\frac{N_c}{N}\right)^{\alpha_N} \;+\; \left(\frac{D_c}{D}\right)^{\alpha_D}, \tag{1}$$

where L_∞ is the irreducible loss floor. Here, N_c and D_c are normalization constants that characterize the model and data scales at which the corresponding terms begin to dominate the loss, and are estimated by fitting the scaling law to empirical performance curves. Overall, we fit five parameters: the irreducible loss floor L_∞, the normalization constants N_c and D_c, and the scaling exponents α_N and α_D. Therefore, we demonstrate that with only 5 parameters, the scaling law effectively models the validation loss across multiple orders of magnitude in both model size and dataset size.

We fit scaling behavior using logarithmically spaced data sizes with $\Delta \log_{10} D \approx 0.5$, and model sizes with $\Delta \log_{10} N \approx 0.45$, providing sufficient resolution to reliably estimate scaling exponents across regimes. We consider nine model sizes spanning four orders of magnitude:

$$N \in \{1.0 \times 10^6,\ 2.8 \times 10^6,\ 7.9 \times 10^6,\ 2.2 \times 10^7,\ 6.3 \times 10^7,\ 1.77 \times 10^8,\ 4.97 \times 10^8,\ 1.4 \times 10^9,\ 3.92 \times 10^9\}$$

, and eight data sizes spanning five orders of magnitude:

$$D \in \{3 \times 10^5,\ 10^6,\ 3 \times 10^6,\ 10^7,\ 3 \times 10^7,\ 10^8,\ 3 \times 10^8,\ 10^9\}.$$

The data is sampled from a graph with ∼50 billion nodes and edges, with 12 node types and 20 relation types, including financial, social, business, infra, and more. Each node type is associated with its own features. For each (N, D) configuration, we train a GraphBFF Transformer using the complete GraphBFF recipe. We use four learning rates optimize with a sufficient number of epochs as done in Kaplan et al. (2020). We evaluate the loss on a fixed holdout set of approximately 6 million edges, drawn from a connected component that is not part of the training data. Each training step processes a batch of 1024 supervised edges. For a given data size D, we define one epoch as a single pass over the D distinct training edges, corresponding to $\lceil D/1024 \rceil$ optimization steps. Learning-rate schedules are parameterized by the total number of training steps for each run, with a warmup phase (1–3% of steps) as in Hoffmann et al. (2022). The model and neighborhoods definitions are the same as in Section 5.

6.2 Results

Figure 5 shows that the loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (1). This figure reveals both data- and model-bottlenecks: beyond a certain point, increasing D at fixed N yields negligible improvements in loss, and likewise increasing N at fixed D eventually provides little benefit. These results show that continued loss reductions require scaling data and model capacity together. The test loss $L(\cdot)$ of a GraphBFF Transformer can be predicted using a power-law when performance is limited by N or D. For models with a limited number of parameters, trained to convergence on sufficiently large datasets:

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \qquad \alpha_N \approx 0.703, \quad N_c \approx 2.1 \times 10^4$$

For large enough models trained with a limited dataset size:

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \qquad \alpha_D \approx 0.188, \quad D_c \approx 4.7$$

We also observe that larger GFMs attain a given test loss using fewer training examples than smaller models.

7 Discussion and Future Work

Many new questions arise from our work.

Defining the pre-training universe $\mathcal{G}$ While it is always possible to merge all available graphs into a single pre-training corpus, it is unclear whether this is optimal, and whether it would yield the best generalization across downstream tasks. This is not only a question of transferability between graphs, but also of how pre-training dynamics are shaped by long-tail distributions over nodes and edges. In particular, rare types in the tail may matter only for a small subset of downstream tasks, it remains unclear whether they should be prioritized during pre-training or reserved for task-specific fine-tuning.

A related challenge concerns the definition of node type and feature groups. We discussed grouping features to ensure applicability to previously unseen features, as done in Tabular FMs. In graphs, the same approach can be also done on the node-type level, on on combination of the two. However, this may come at the cost of expressivity. It would therefore be very interesting to derive concrete rules, potentially as a function of feature, node-type distributions and sparsity with respect to target downstream tasks, for selecting an appropriate level of granularity.

Compute-Optimal Allocation In ViTs and LLMs scaling-laws, the context size is fixed and therefore compute is well-approximated by $O(N \cdot D)$ where N represents the number of parameters and D represents the number of training tokens (Hoffmann et al., 2022). This abstraction allows for the derivation of precise compute-optimal frontiers, as the cost per "unit of data" remains constant. In graph models with neighborhood-based context, the computational cost per supervised edge depends on the size of the sampled subgraph, which in turn depends on graph structure. As a result, total pre-training compute cannot be expressed as a universal function of the labeled data and model size alone. Therefore, any compute-optimal conclusions must be interpreted as conditional on the underlying graph and sampling distribution, rather than as universal prescriptions. It would be very valuable to design compute-allocation measures for GFMs, which are transferable across different graphs.

8 Conclusion

This work advances the Graph Foundation Models (GFMs) paradigm by presenting GraphBFF - the first end-to-end recipe for training billion-parameter GFMs over billion-scale heterogeneous graphs. We introduced the GraphBFF Transformer, a scalable architecture tailored to the practical constraints and statistical properties of real-world graphs, and we established the first empirical scaling laws for general graphs, highlighting coupled data–model bottlenecks and predictable improvements when both are scaled jointly. We demonstrated the effectiveness of GraphBFF across extensive experiments. We showed that a 1.4 billion-parameter GraphBFF Transformer pretrained on over one billion samples achieved consistent and often substantial gains across diverse downstream tasks, including strong probing, few-shot, and zero-shot performance on graphs unseen during training.

Acknowledgments

We thank Kfir Amitai for the consistent support and enablement of this research. We thank Petar Veličković, Amir Bar, Randall Balestriero for the insightful discussions on LLMs and ViTs as GFMs. We thank Taco Cohen, Arjun Subramonian, Kaveh Hassani and Saurabh Verma for the insightful discussions and constructive feedback on early versions of this work.

References

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27):e2311878121, 2024. doi: 10.1073/pnas.2311878121.

Maya Bechler-Speicher, Ido Amos, Ran Gilad-Bachrach, and Amir Globerson. Graph neural networks use graphs when they shouldn't, 2024.

Maya Bechler-Speicher, Ben Finkelshtein, Fabrizio Frasca, Luis Müller, Jan Tönshoff, Antoine Siraudin, Viktor Zaverkin, Michael M. Bronstein, Mathias Niepert, Bryan Perozzi, Mikhail Galkin, and Christopher Morris. Position: Graph learning will lose relevance due to poor benchmarks, 2025.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman,

Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021.

Johann Brehmer, Sönke Behrends, Pim de Haan, and Taco Cohen. Does equivariance matter at scale?, 2025.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021. https://arxiv.org/abs/2104.13478.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 257–266, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330925.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021.

Dmitry Eremeev, Gleb Bazhenov, Oleg Platonov, Artem Babenko, and Liudmila Prokhorenkova. Turning tabular foundation models into graph foundation models, 2025.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Ben Finkelshtein, İsmail İlkan Ceylan, Michael Bronstein, and Ron Levie. Equivariance everywhere all at once: A recipe for graph foundation models, 2025.

Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, WWW '20, page 2331–2341, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370233. doi: 10.1145/3366423.3380297.

Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning, 2024.

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

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *NeurIPS 2022 First Table Representation Workshop*, 2022.

Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 594–604, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539321.

Zhenyu Hou, Yufei He, Yukuo Cen, Xiao Liu, Yuxiao Dong, Evgeny Kharlamov, and Jie Tang. Graphmae2: A decoding-enhanced masked self-supervised graph learner. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 737–746, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394161. doi: 10.1145/3543507.3583379.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Weihua Hu*, Bowen Liu*, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, WWW '20, page 2704–2710, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370233. doi: 10.1145/3366423.3380027.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017.

Thomas Kipf and Max Welling. Variational graph auto-encoders. *ArXiv*, abs/1611.07308, 2016.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One for all: Towards training one graph model for all classification tasks, 2024.

Xu Luo, Hao Wu, Ji Zhang, Lianli Gao, Jing Xu, and Jingkuan Song. A closer look at few-shot classification again. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: graph foundation models are already here. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Larisa Markeeva, Sean McLeish, Borja Ibarz, Wilfried Bounsi, Olga Kozlova, Alex Vitvitskyi, Charles Blundell, Tom Goldstein, Avi Schwarzschild, and Petar Veličković. The clrs-text algorithmic reasoning language benchmark, 2024.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data, 2025.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers, 2023.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2074.

Ahsan Shehzad, Feng Xia, Shagufta Abid, Ciyuan Peng, Shuo Yu, Dongyu Zhang, and Karin M. Verspoor. Graph transformers: A survey. *IEEE transactions on neural networks and learning systems*, PP, 2024.

Nima Shoghi, Adeesh Kolluru, John R. Kitchin, Zachary W. Ulissi, C. Lawrence Zitnick, and Brandon M. Wood. From molecules to materials: Pre-training large generalizable models for atomic property prediction, 2024.

Gowthami Somepalli, Avi Schwarzschild, Micah Goldblum, C. Bayan Bruss, and Tom Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. In *NeurIPS 2022 First Table Representation Workshop*, 2022.

Katherine Faust Stanley Wasserman. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, UK, 1994. ISBN 9780521387071.

Wen Tai, H. T. Kung, Xin Dong, Marcus Comiter, and Chang-Fu Kuo. exBERT: Extending pre-trained models with domain-specific vocabulary under constrained training resources. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1433–1439, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.129.

Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q. Tran, Dani Yogatama, and Donald Metzler. Scaling laws vs model architectures: How does inductive bias influence scaling?, 2022.

Vincent A Traag, Ludo Waltman, and Nees Jan van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019. doi: 10.1038/s41598-019-41695-z.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, 2023. ISSN 0959-440X. doi: https://doi.org/10.1016/j.sbi.2023.102538.

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

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

Limei Wang, Kaveh Hassani, Si Zhang, Dongqi Fu, Baichuan Yuan, Weilin Cong, Zhigang Hua, Hao Wu, Ning Yao, and Bo Long. Learning graph quantized tokenizers, 2025a.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, WWW '19, page 2022–2032, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313562.

Zehong Wang, Zheyuan Liu, Tianyi Ma, Jiazheng Li, Zheyuan Zhang, Xingbo Fu, Yiyang Li, Zhengqing Yuan, Wei Song, Yijun Ma, Qingkai Zeng, Xiusi Chen, Jianan Zhao, Jundong Li, Meng Jiang, Pietro Lio, Nitesh Chawla, Chuxu Zhang, and Yanfang Ye. Graph foundation models: A comprehensive survey, 2025b.

Gilad Yehudai, Clayton Sanford, Maya Bechler-Speicher, Orr Fischer, Ran Gilad-Bachrach, and Amir Globerson. Depth-width tradeoffs in algorithmic reasoning of graph tasks with transformers, 2025.

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? In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method, 2020.

Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations, 2020.

Jianan Zhao, Zhaocheng Zhu, Mikhail Galkin, Hesham Mostafa, Michael M. Bronstein, and Jian Tang. Fully-inductive node classification on arbitrary graphs. In *The Thirteenth International Conference on Learning Representations*, 2025.

A Proof of Theorem 4.1

Here we prove Theorem 4.1.

Proof of Theorem 4.1. Fix a node $v \in \mathcal{V}$ and let $\mathcal{N}_v^{\text{taa}}$ be its TAA neighborhood. For any set of edge types $S \subseteq \mathcal{T}_E$, let $\mathcal{N}_v^S \subseteq \mathcal{N}_v^{\text{tca}}$ denote the S-masked neighborhood. In particular, with $\mathcal{N}_v^{\text{tca}} = \mathcal{N}_v^{\text{taa}}$ and $\mathcal{S} = \big\{\{r^\star\}, \{r'\}\big\}$ we have two disjoint masked neighborhoods $\mathcal{N}_v^{\{r^\star\}}$ and $\mathcal{N}_v^{\{r'\}}$.

Consider a family of graphs where all node types are identical (so the type projection $\mathbf{W}_{\tau(v)}^{(\ell)}$ can be taken as the identity without loss of generality), and where each neighbor $u \in \mathcal{N}_v^{\text{taa}}$ has a one-dimensional feature $x_u \in \{0, 1\}$, i.e., $\mathbf{h}_u^{(\ell)} = x_u$.

Define the two scalars

$$a(v) \;:=\; \frac{1}{|\mathcal{N}_v^{\{r^\star\}}|} \sum_{u \in \mathcal{N}_v^{\{r^\star\}}} x_u, \qquad b(v) \;:=\; \frac{1}{|\mathcal{N}_v^{\text{taa}}|} \sum_{u \in \mathcal{N}_v^{\text{taa}}} x_u.$$

Fix any $\varepsilon \in (0, \frac{1}{4})$ and define the continuous "soft-threshold" function

$$s_\varepsilon(t) \;:=\; \text{clip}\Big(\frac{t - (\frac{1}{2} - \varepsilon)}{2\varepsilon},\, 0,\, 1\Big), \qquad \text{clip}(z, 0, 1) := \min(1, \max(0, z)).$$

Define the target

$$\tilde{f}(v) \;:=\; s_\varepsilon(a(v)) \;+\; s_\varepsilon(b(v)).$$

.

$\tilde{f} \in \mathcal{F}_{\text{GraphBFF}}$

(i) TAA computes $b(v)$. Choose the (shared) TAA attention parameters so that all attention logits are equal. It suffices to set $\mathbf{W}_Q^{(\ell)} = \mathbf{0}$ and $\mathbf{W}_K^{(\ell)} = \mathbf{0}$, so that for every $u \in \mathcal{N}_v^{\text{taa}}$ the score $\frac{(\mathbf{q}_v^{(\ell)})^\top \mathbf{k}_u^{(\ell)}}{\sqrt{d_h}}$ is constant and thus

$$\beta_{uv}^{(\ell)} = \frac{1}{|\mathcal{N}_v^{\text{taa}}|}.$$

Set $\mathbf{W}_V^{(\ell)}$ so that the first coordinate of $\mathbf{v}_u^{(\ell)}$ equals x_u. Then the first coordinate of $\mathbf{h}_v^{(\ell,\text{taa})}$ is exactly

$$\sum_{u \in \mathcal{N}_v^{\text{taa}}} \beta_{uv}^{(\ell)}\, x_u \;=\; b(v).$$

(ii) TCA computes $a(v)$, and uses $\sum$ over sets. For the set $S = \{r^\star\}$, set $\mathbf{W}_Q^{(\{r^\star\},\ell)} = \mathbf{0}$ and $\mathbf{W}_K^{(\{r^\star\},\ell)} = \mathbf{0}$ so the softmax is uniform on $\mathcal{N}_v^{\{r^\star\}}$:

$$\alpha_{uv}^{(\{r^\star\},\ell)} = \frac{1}{|\mathcal{N}_v^{\{r^\star\}}|}.$$

Choose $\mathbf{W}_V^{(\{r^\star\},\ell)}$ so that the first coordinate of $\mathbf{v}_u^{(\{r^\star\},\ell)}$ equals x_u. Then the first coordinate of $\mathbf{h}_v^{(\ell,\{r^\star\})}$ equals $a(v)$.

For $S = \{r'\}$, set $\mathbf{W}_V^{(\{r'\},\ell)} = \mathbf{0}$ so that $\mathbf{h}_v^{(\ell,\{r'\})} = \mathbf{0}$. Since in this variant the set-aggregation is *sum*,

$$\mathbf{h}_v^{(\ell,\text{tca})} = \sum_{S \in \mathcal{S}} \mathbf{h}_v^{(\ell,S)} = \mathbf{h}_v^{(\ell,\{r^\star\})} + \mathbf{h}_v^{(\ell,\{r'\})} = \mathbf{h}_v^{(\ell,\{r^\star\})},$$

so $\mathbf{h}_v^{(\ell,\text{tca})}$ contains $a(v)$ in a fixed coordinate.

(iii) A ReLU-MLP readout computes $\tilde{f}(v)$ exactly from $(a(v), b(v))$. Consider the 2-layer ReLU network (one hidden layer) taking input $x = \begin{bmatrix} a \\ b \end{bmatrix}$ and producing output $\tilde{f}$:

$$h = \mathrm{ReLU}(W_1 x + b_1), \qquad \tilde{f} = W_2 h + b_2,$$

with

$$W_1 = \begin{bmatrix} \frac{1}{2\varepsilon} & 0 \\ \frac{1}{2\varepsilon} & 0 \\ 0 & \frac{1}{2\varepsilon} \\ 0 & \frac{1}{2\varepsilon} \end{bmatrix}, \quad b_1 = \begin{bmatrix} \frac{1}{2} - \frac{1}{4\varepsilon} \\ -\frac{1}{2} - \frac{1}{4\varepsilon} \\ \frac{1}{2} - \frac{1}{4\varepsilon} \\ -\frac{1}{2} - \frac{1}{4\varepsilon} \end{bmatrix}, \quad W_2 = \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}, \quad b_2 = 0.$$

This implements

$$\tilde{f} \;=\; \big(\mathrm{ReLU}(z(a)) - \mathrm{ReLU}(z(a) - 1)\big) \;+\; \big(\mathrm{ReLU}(z(b)) - \mathrm{ReLU}(z(b) - 1)\big) \;=\; s_\varepsilon(a) + s_\varepsilon(b),$$

where $z(t) = \frac{t - (\frac{1}{2} - \varepsilon)}{2\varepsilon}$ and we use the identity $\mathrm{clip}(z, 0, 1) = \mathrm{ReLU}(z) - \mathrm{ReLU}(z - 1)$. Thus, choosing $\Phi^{(\ell)}$ to contain the above ReLU-MLP yields $\tilde{f}(v)$. Hence $\tilde{f} \in \mathcal{F}_{\mathrm{GraphBFF}}$.

$\tilde{f} \notin \mathcal{F}_{\mathrm{TAA}}$

We build two graphs that any TAA-only model must map to the same output at v, yet $\tilde{f}(v)$ differs. Let $\mathcal{N}_v^{\mathrm{taa}} = \{u_1, u_2\}$ and set $(x_{u_1}, x_{u_2}) = (1, 0)$. Define two graphs $\mathcal{G}_A, \mathcal{G}_B$ that are identical except for swapping edge-type labels:

$$\phi((u_1, v)) = r^\star,\ \phi((u_2, v)) = r' \ \text{ in } \mathcal{G}_A, \qquad \phi((u_1, v)) = r',\ \phi((u_2, v)) = r^\star \ \text{ in } \mathcal{G}_B.$$

Then $b(v) = \frac{1}{2}$ in both graphs, while $a(v) = 1$ in $\mathcal{G}_A$ and $a(v) = 0$ in $\mathcal{G}_B$. Since $\varepsilon < \frac{1}{2}$, we have $s_\varepsilon(1) = 1$ and $s_\varepsilon(0) = 0$, and also $s_\varepsilon(\frac{1}{2}) = \frac{1}{2}$. Therefore

$$\tilde{f}_{\mathcal{G}_A}(v) = 1 + \tfrac{1}{2} = \tfrac{3}{2}, \qquad \tilde{f}_{\mathcal{G}_B}(v) = 0 + \tfrac{1}{2} = \tfrac{1}{2},$$

so $\tilde{f}_{\mathcal{G}_A}(v) \neq \tilde{f}_{\mathcal{G}_B}(v)$.

However, under the stated TAA parameter sharing $\mathbf{W}_{Q,r}^{(\ell)} = \mathbf{W}_Q^{(\ell)}$, $\mathbf{W}_{K,r}^{(\ell)} = \mathbf{W}_K^{(\ell)}$, $\mathbf{W}_{V,r}^{(\ell)} = \mathbf{W}_V^{(\ell)}$ for all r, the TAA computation at v depends only on the multiset of neighbor representations $\{\widehat{\mathbf{h}}_u^{(\ell)} : u \in \mathcal{N}_v^{\mathrm{taa}}\}$ and is invariant to swapping edge-type labels. Since this multiset is identical in $\mathcal{G}_A$ and $\mathcal{G}_B$, the TAA-only embedding at v is identical, and any (deterministic) MLP readout must output the same value on both graphs, a contradiction. Therefore $\tilde{f} \notin \mathcal{F}_{\mathrm{TAA}}$.

$\tilde{f} \notin \mathcal{F}_{\mathrm{TCA}}$

We build two graphs that any TCA-only model must map to the same output at v, yet $\tilde{f}(v)$ differs, using only that each set-specific TCA vector $\mathbf{h}_v^{(\ell,S)}$ is a softmax-weighted average over $\{\mathbf{h}_u^{(\ell)} : u \in \mathcal{N}_v^S\}$.

Construct two graphs $\mathcal{G}_C, \mathcal{G}_D$ such that

$$x_u = 1 \ \ \forall u \in \mathcal{N}_v^{\{r^\star\}}, \qquad x_u = 0 \ \ \forall u \in \mathcal{N}_v^{\{r'\}},$$

and the only difference is the neighborhood cardinalities:

$$|\mathcal{N}_v^{\{r^\star\}}| = 1,\ |\mathcal{N}_v^{\{r'\}}| = 3 \ \text{ in } \mathcal{G}_C, \qquad |\mathcal{N}_v^{\{r^\star\}}| = 3,\ |\mathcal{N}_v^{\{r'\}}| = 1 \ \text{ in } \mathcal{G}_D.$$

Then $a(v) = 1$ in both graphs, but

$$b_{\mathcal{G}_C}(v) = \tfrac{1}{4}, \qquad b_{\mathcal{G}_D}(v) = \tfrac{3}{4}.$$

With $\varepsilon \in (0, \frac{1}{4})$, we have $s_\varepsilon(\frac{1}{4}) = 0$ and $s_\varepsilon(\frac{3}{4}) = 1$, and $s_\varepsilon(1) = 1$. Thus

$$\tilde{f}_{\mathcal{G}_C}(v) = s_\varepsilon(1) + s_\varepsilon(\tfrac{1}{4}) = 1 + 0 = 1, \qquad \tilde{f}_{\mathcal{G}_D}(v) = s_\varepsilon(1) + s_\varepsilon(\tfrac{3}{4}) = 1 + 1 = 2,$$

so $\tilde{f}_{\mathcal{G}_C}(v) \neq \tilde{f}_{\mathcal{G}_D}(v)$.

Now consider any TCA-only instantiation. For $S = \{r^\star\}$, all nodes in $\mathcal{N}_v^{\{r^\star\}}$ have the same input representation $\mathbf{h}_u^{(\ell)} = 1$, hence the projected values $\mathbf{v}_u^{(S,\ell)} = \mathbf{W}_V^{(S,\ell)}\mathbf{h}_u^{(\ell)}$ are identical across $u \in \mathcal{N}_v^{\{r^\star\}}$, and similarly for $S = \{r'\}$ (with $\mathbf{h}_u^{(\ell)} = 0$). Therefore, in each set S, the attention output

$$\mathbf{h}_v^{(\ell,S)} = \sum_{u \in \mathcal{N}_v^S} \alpha_{uv}^{(S,\ell)}\, \mathbf{v}_u^{(S,\ell)}$$

is the same in $\mathcal{G}_C$ and $\mathcal{G}_D$, because it is a convex combination of identical vectors (and thus does not depend on $|\mathcal{N}_v^S|$). Consequently, for every $S \in \mathcal{S}$, the vectors $\mathbf{h}_v^{(\ell,S)}$ match between $\mathcal{G}_C$ and $\mathcal{G}_D$, and so does their *sum*

$$\mathbf{h}_v^{(\ell,\mathrm{tca})} = \sum_{S \in \mathcal{S}} \mathbf{h}_v^{(\ell,S)}.$$

Hence any TCA-only model (followed by any MLP readout) must output the same value at v on $\mathcal{G}_C$ and $\mathcal{G}_D$, contradicting $\tilde{f}_{\mathcal{G}_C}(v) \neq \tilde{f}_{\mathcal{G}_D}(v)$. Therefore $\tilde{f} \notin \mathcal{F}_{\mathrm{TCA}}$. □

B KL-Batching Additional Formulation

Here we formulate and describe in details *KL-Batching*. As loading data from storage into memory is relatively slow, we aim to minimize the number of such transfers. We therefore maximize the mini-batch size subject to the available memory budget, denoted by M.

A common strategy for forming mini-batches from large graphs is to first partition the graph into clusters Chiang et al. (2019), e.g., using the Leiden algorithm (Traag et al., 2019), which scales to billion-edge graphs and optimizes modularity efficiently. However, cluster sizes can vary substantially, and the node/edge type composition within a cluster can deviate markedly from the global data distribution. In pre-training, we typically optimize on only a subset of the full graph (e.g., 1B nodes sampled from a 100B-node graph). In this regime, randomly selecting clusters to load into memory can (i) underutilize the desired batch size B due to cluster-size variance, induce distributional bias when early updates are dominated by a few atypical clusters and lead to unstable training (Zeng et al., 2020; Bengio et al., 2009). This effect is amplified in multi-machine training, where the effective batch size per optimization step can reach tens to hundreds of millions of samples Keskar et al. (2017). Consequently, biased node/edge-type exposure in the first steps can steer optimization toward suboptimal solutions. To mitigate these issues, we propose *KL-Batching*, a simple and efficient procedure that assembles memory-efficient mini-batches whose type distributions are close to the global distribution.

KL-Batching first partitions the graph into K disjoint clusters, $\{C_1, \ldots, C_K\}$, $C_k \subseteq V$, which are never splitted across batches to avoid introducing additional cross-batch edge cuts. For each cluster C_k, we compute an empirical distribution over a chosen discrete attribute of interest, denoted by $a(\cdot)$. This attribute can correspond to node types, edge types, or any other crucial categorical property used to control representativeness. Let T denote the support of this attribute and let $p_k(t)$ be the empirical distribution induced by C_k over $t \in T$. The full graph (or the pre-training population) similarly induces a global reference distribution $p_G(t)$.

For each cluster, we then compute the Kullback–Leibler (KL) divergence to the global distribution,

$$\mathrm{KL}(p_k \,\|\, p_G) = \sum_{t \in T} p_k(t)\, \log \frac{p_k(t)}{p_G(t)},$$

which quantifies how representative C_k is with respect to the selected attribute distribution. When multiple attributes are important, one can compute multiple KL terms and combine them via, e.g., a weighted sum.

Given a target batch capacity B, measured as an upper bound on the total storage load associated with the batch, we construct batches by joining entire clusters. Let $\mathrm{size}(C_k) > 0$ be a cost estimate for cluster C_k (e.g., accounting for its number of nodes and edges, the mix of node/edge types, and type-specific feature dimensionalities). KL-Batching proceeds by (1) sorting all clusters in ascending order of $\mathrm{KL}(p_k \,\|\, p_G)$, and

then (2) traversing this list and sequentially aggregating clusters into batches as long as the cumulative batch cost does not exceed the memory budget M.

This yields a collection of batches $\mathcal{B} = \{B_1, \ldots, B_S\}$, where each batch B_ℓ is the disjoint union of whole clusters,

$$B_\ell = \bigcup_{k \in I_\ell} C_k, \qquad \sum_{k \in I_\ell} \text{size}(C_k) \leq M,$$

and size$(\cdot)$ denotes the estimated memory cost. Because batches are assembled from low-KL clusters first and filled as close as possible to the capacity constraint, we (i) obtain batches that are representative with respect to the chosen attribute(s) and (ii) improve memory utilization at each training worker. The batch construction stage can be viewed as a constrained combinatorial optimization problem; due to space limitations, we provide a formal formulation in the Appendix.

Suppose we have clusters indexed by $k = 1, \ldots, K$, each with a size (cost) $s_k = \text{size}(C_k) > 0$ and a KL value

$$\kappa_k = \text{KL}(p_k \,\|\, p_G).$$

Fix a batch capacity B. For any batch, we seek a subset of indices $I \subseteq \{1, \ldots, K\}$ such that

$$\sum_{k \in I} s_k \leq B, \tag{2}$$

$$\sum_{k \in I} s_k \text{ is maximized}, \tag{3}$$

under the preference that clusters with lower κ_k are chosen first (i.e., we want the batch to be composed of clusters whose node-type distribution is close to the global one).

If we fix a set of candidate clusters that all have the same KL value, $\kappa_k = \kappa^\star$ for all k in some index set $\mathcal{K}^\star$, then selecting a subset $I \subseteq \mathcal{K}^\star$ maximizing (3) subject to (2) is exactly the classical 0–1 knapsack problem. This problem is known to be NP-hard in general, and solving it exactly for every batch is computationally infeasible at the scales we consider.

In our setting, exact ties in κ_k are rare in practice, because KL values are continuous and clusters exhibit diverse type distributions. Consequently, the number of genuine "knapsack" situations, where we must choose among many clusters with effectively identical KL values-, is small relative to the total number of batches. We therefore adopt a simple greedy heuristic:

- We traverse clusters in ascending order of κ_k.
- For each batch, we add clusters sequentially as long as the capacity constraint $\sum s_k \leq B$ is satisfied.
- When multiple candidate clusters share the same κ_k, we add them in arbitrary (or size-sorted) order, without attempting to solve the knapsack problem exactly.

This heuristic does not guarantee globally optimal packing with respect to B, but it works well empirically: capacity utilization is typically high (batches are close to full), and pre-training remains stable. Given the rarity of large same-KL groups and the overwhelming scale of the graph, the suboptimality introduced by this greedy step is negligible in practice. Due to space limitation, the formal algorithm description is provided in the Appendix.