THE UNIVERSAL CLASSIFIER FOR GRAPH LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

While foundation models have revolutionized natural language processing and computer vision by leveraging universal vocabularies, Graph Machine Learning (GML) remains fractured due to the absence of a unified feature and structural representation across diverse domains. Existing works claiming to be Graph Foundation Models (GFMs) are typically restricted to node-level predictions or require fixed feature dimensions, failing to provide a truly task-agnostic backbone for the full spectrum of graph learning applications. In this paper, we introduce the **UNIVERSAL CLASSIFIER (UC)**, which supports arbitrary feature and class cardinalities, unifying node-, edge-, and graph-level objectives under a single similarity-based classification objective. The UC reformulates all node-, edge-, and graph-level prediction tasks as maximizing similarity in the latent space: by lifting heterogeneous features and labels into 3D latent tensors, the model learns transferable features independent of specific input schemas. This architecture allows a single pre-trained model to generalize to node classification, node regression, and link prediction across unseen graphs with varying feature semantics. Experiments show strong zero-shot transfer performance across node-, link-, and graph-level tasks.

1 INTRODUCTION

The emergence of foundation models has fundamentally redefined the landscape of machine learning, demonstrating an unprecedented capacity for zero-shot generalization and cross-domain transfer in natural language processing (Raffel et al., 2020; Touvron et al., 2023) and computer vision (Radford et al., 2021; Dosovitskiy et al., 2021). However, the transition of this "foundational" paradigm to Graph Machine Learning (GML) has proven uniquely difficult due to two fundamental barriers.

First, unlike language or vision models that rely on a shared vocabulary of tokens or pixels, graphs exhibit extreme attribute heterogeneity; a node feature might represent a chemical property in a molecular graph or a textual embedding in a social network. This diversity necessitates schema invariant architectures capable of operating across arbitrary feature and label spaces (Zhao et al., 2025; Finkelshtein et al., 2025). Second, whereas traditional foundation models leverage a single training objective like next-token prediction or masked patch reconstruction, the GML landscape spreads across a wide array of node-, edge-, and graph-level objectives that usually require bespoke architectures. While the GML community has progressed in building models that address these challenges independently, a true Graph Foundation Model must encapsulate both properties.

> **Graph Foundation Model (GFM).** A single pre-trained model capable of performing inference across arbitrary feature schemas while simultaneously addressing node-, edge-, and graph-level tasks across unseen graphs, without task-specific architectural modifications.

We introduce this terminology to bring clarity to a field where "GFM" is frequently used as a misnomer. Current literature offers a fractured landscape where models often claim the GFM title while satisfying only a subset of its essential properties. For instance, recent advancements in schema-invariant models (Zhao et al., 2024; 2025; Finkelshtein et al., 2025) are strictly confined to node classification. Similarly, generalist models for link prediction (Lee et al., 2023; Galkin et al., 2024a; Huang et al., 2026; Kim et al., 2026) can hardly generalize to node- and graph-level tasks.

In this paper, we unify discriminative prediction tasks on graphs under a single framework. To this end, we introduce the UNIVERSAL CLASSIFIER (UC), a *schema-invariant* model, which generalizes

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.

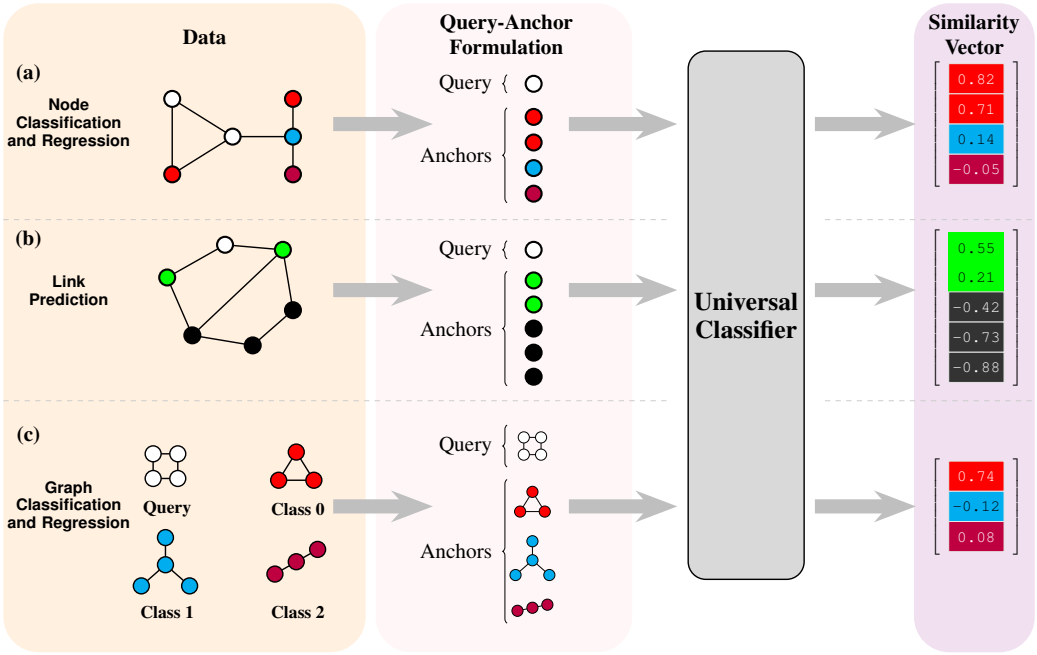

Figure 1: Overview of the UNIVERSAL CLASSIFIER generalizing across tasks. **(a) Node Classification and Regression:** a query node is compared to a support set of anchors, with similarities aggregated by class. In regression, continuous targets are discretized into bins, treated as distinct classes, and recovered via similarity-weighted interpolation. **(b) Link Prediction:** a source node acts as a query and is evaluated against target nodes. Nodes connected to the query serve as anchors of a *positive* class, while all other nodes act as anchors of a *negative* class. **(c) Graph Classification and Regression:** a source graph is represented by pooling its respective features into a single global query, which is then compared against anchor graphs representing specific labels, similar to node-level tasks.

to arbitrary feature and label spaces and performs node-, edge-, and graph-level tasks without task-specific treatment. UC blends representation learning with in-context learning and reformulates all prediction tasks as similarity in a joint latent space. Specifically, UC treats a set of labeled items (that could be nodes, edges, or graphs) as *anchors*, and probes them against unlabeled items, *queries*. By lifting anchors and queries to the shared latent space through expressive encoders (that mix features, labels, and graph topology), UC estimates how similar a query is to each of the anchors. This formulation allows UC to unify all graph learning tasks to be trained with a single cross-entropy loss, lending it the name the *universal classifier*.

Unifying graph learning tasks. UC encodes each feature independently, representing each node's embedding as a lifted matrix in $\mathbb{R}^{F \times D}$. Depending on the specific task, UC dynamically casts these representations as either *query* or *anchor* embeddings. This formulation unifies all tasks by identifying semantic relationships through pairwise similarities in a shared latent space. See Figure 1 for details.

Our query-anchor paradigm naturally extends to multi-relational domains like knowledge graphs and relational databases, as well as to temporal and heterogeneous graphs, which we leave for future work. Our core contributions are:

- **We frame all graph learning tasks as similarity estimation** via the schema-invariant encoder, applying cardinality independence jointly to features and labels so that a single parameter set generalizes to unseen feature schemas and arbitrary feature and class cardinalities.
- To our knowledge, UC is the first model to perform node-, edge-, and graph-level tasks without architectural modifications, task-specific heads, or maximum feature and class dimensions.
- UC demonstrates strong generalization performance in practical graph learning benchmarks.

2 THE UNIVERSAL CLASSIFIER

2.1 NOTATION

We denote $[i] = \{1, \dots, i\}$ and the cardinality of a finite set $\mathcal{I}$ by $|\mathcal{I}|$. Vectors, matrices, and tensors are denoted in bold. For any matrix or tensor $\mathbf{H} \in \mathbb{R}^{N \times M \times K}$, we denote its i-th row (or slice) by $\mathbf{H}_i$. For a subset of indices $\mathcal{I}$, the corresponding submatrix or subtensor is denoted as $\mathbf{H}_{\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}| \times M \times K}$.

We consider an undirected, unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X}, \boldsymbol{Y})$, where $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ is a matrix of node features, and $\boldsymbol{Y}$ denotes the labels. The structure of $\boldsymbol{Y}$ is task-dependent: for node classification, $\boldsymbol{Y} \in \{0, 1\}^{N \times C}$ has one-hot rows $\boldsymbol{Y}_v$ over C classes, while for node regression, $\boldsymbol{Y} \in \mathbb{R}^{N \times 1}$ and each node label $\boldsymbol{Y}_v$ is a continuous scalar. The graph size is $|\mathcal{G}| = |\mathcal{V}|$. We denote the k-hop subgraph around a node v by $\mathcal{G}^k(v)$. We also denote N as the number of nodes, F as the number of input features, D as the hidden dimension, and C as the number of output classes or regression targets.

The query-anchor pair $(\mathcal{Q}, \mathcal{A})$ unifies different graph learning objectives: $\mathcal{Q}$ represents a batch of query entities, and $\mathcal{A}$ represents a support set of labeled anchors. These entities are structurally flexible, capable of representing either individual nodes or entire graph topologies depending on the task.

2.2 DATA SAMPLING AND SCHEMA INVARIANCE

Sampling and Label Masking. To prevent label leakage during training, we sample disjoint query ($\mathcal{Q}$) and anchor ($\mathcal{A}$) sets ($\mathcal{Q} \cap \mathcal{A} = \emptyset$). In the label matrix $\boldsymbol{Y}$, query labels are masked as zero-vectors, while unmasked labels are one-hot encoded. At inference, target nodes act as queries against the training set (or a subset), which serves as the anchor context. See Appendix D for more details.

Feature Normalization. We first apply feature-wise z-normalization to raw node features $\boldsymbol{X}$ to canonicalize the numerical range and provide a stable input distribution. Next, a shared MLP $\phi_{\text{feat}} : \mathbb{R} \to \mathbb{R}^D$ projects each normalized scalar feature independently, resulting in the 3D tensor:

$$\boldsymbol{X}_{\text{init}})_{n,f,:} = \phi_{\text{feat}}(\boldsymbol{X}_{n,f}), \quad \forall n \in [N],\ f \in [F], \qquad \text{so that} \qquad \boldsymbol{X}_{\text{init}} \in \mathbb{R}^{N \times F \times D}.$$

Following (Finkelshtein et al., 2025; Bevilacqua et al., 2025), treating the varying feature dimension F as an unordered set enables schema invariance. All further UC components operate along the fixed latent D axis, decoupling model parameters from F at both training and inference.

Label Lifting for Node-level Tasks. Anchor labels can also serve as supplementary 3D node features. An additional learnable MLP $\phi_{\text{label}} : \mathbb{R} \to \mathbb{R}^D$ independently processes each entry of a node's label vector (whether a one-hot anchor or a zero-masked query). Similar to feature lifting, this maps the 2D label matrix $\boldsymbol{Y} \in \mathbb{R}^{N \times C}$ to a 3D tensor $\boldsymbol{Y}_{\text{init}} \in \mathbb{R}^{N \times C \times D}$ where, similarly to feature lifting, operating exclusively along the shared D axis allows UC to accommodate any number of classes C.

2.3 MODEL ARCHITECTURE

The UNIVERSAL CLASSIFIER includes four modules (Figure 2): **(1) a GNN Encoder** that independently processes the localized topology of each query and anchor node; **(2) a Query-Anchor Transformer** that performs cross-feature and cross-anchor attention to contextualize the query within the anchor set; **(3) a Similarity Estimator** that aggregates these refined representations to produce the final query-anchor similarity scores; and **(4) an Anchor-Class Aggregator** that pools these pairwise scores by class to produce probabilistic class predictions.

(1) Subgraph Sampling and GNN Encoder (Figure 2a). To ensure that query and anchor representations capture their local structural contexts, the forward pass begins by extracting an induced k-hop subgraph $\mathcal{G}^k(r)$ around each root node $r \in \mathcal{Q} \cup \mathcal{A}$, creating a unified graph $\mathcal{G}_{\text{uni}} = \bigcup_{r \in \mathcal{Q} \cup \mathcal{A}} \mathcal{G}^k(r)$. The shared D-dimensional GNN Encoder then performs message passing over the unified graph, operating on the input 3D tensors (separately for feature $\boldsymbol{X}$ and label $\boldsymbol{Y}$ representations):

$$\boldsymbol{X}^{l+1} = \text{GNN}\Big(\boldsymbol{X}^l, \mathcal{G}_{\text{uni}}\Big) \in \mathbb{R}^{|\mathcal{G}_{\text{uni}}| \times F \times D}, \quad \boldsymbol{Y}^{l+1} = \text{GNN}\Big(\boldsymbol{Y}^l, \mathcal{G}_{\text{uni}}\Big) \in \mathbb{R}^{|\mathcal{G}_{\text{uni}}| \times C \times D}$$

Following the final GNN layer, we extract the representations of the root nodes $\{\boldsymbol{X}_r, \boldsymbol{Y}_r\}_{r \in \mathcal{Q} \cup \mathcal{A}}$ to serve as the final structure-induced feature and label representations for the queries and anchors.

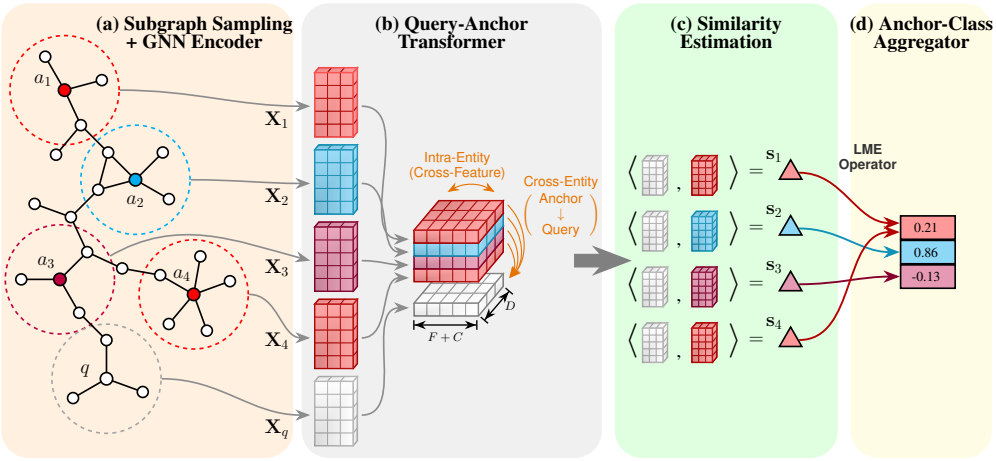

Figure 2: The four primary modules of the UNIVERSAL CLASSIFIER. **(a) GNN Encoder** that processes the localized topology (obtained via subgraph sampling) of each query and anchor node. **(b) Query-Anchor Transformer** that performs intra-entity and cross-entity attention to contextualize the query within the anchor set. **(c) Similarity Estimator** that aggregates the representations of queries and anchors to produce query-anchor similarity scores. **(d) Anchor-Class Aggregator** that pools those pairwise similarity scores to produce probabilistic class predictions.

(2) Query-Anchor Transformer (Figure 2b). After extracting structural representations for each query and anchor, we seek to refine these embeddings by capturing their pairwise relational affinities. As we now operate on unordered sets of query and anchor representations, Transformers are natively suited for this task. Self-attention without positional encodings is permutation-equivariant over a channel set whose size varies across datasets, whereas any mixer indexing channels by position would reintroduce the dependence on F and C that the lifting removes. We do not claim this design is optimal, but it is this property that makes it compatible with schema invariance. Each l-th block performs two interleaved and orthogonal attention operations: *Intra-Entity Attention* and *Cross-Entity Attention*.

Intra-Entity Attention: This mechanism operates across the $(F + C)$ channel axis to capture internal correlations between the features and labels of the *same node*. Concatenating the two matrices allows a query's value on one attribute to be read together with the labels of the anchors sharing that pattern; carried separately, no relationship between a feature and a label could be represented. For a specific query or anchor $r \in \mathcal{Q} \cup \mathcal{A}$, self-attention is applied directly to the concatenated matrices in $\mathbb{R}^{(F+C)\times D}$:

$$[\bar{\boldsymbol{X}}_r^l, \bar{\boldsymbol{Y}}_r^l] = \texttt{TransformerBlock}\left([\boldsymbol{X}_r^l \parallel \boldsymbol{Y}_r^l], [\boldsymbol{X}_r^l \parallel \boldsymbol{Y}_r^l]\right) \in \mathbb{R}^{(F+C)\times D}$$

where $\parallel$ denotes concatenation along the feature-label axis, and the *query-key* notation highlights the self-attention nature of the function. We split $\bar{\boldsymbol{X}}_r^l$ into query representations $\bar{\boldsymbol{X}}_{\mathcal{Q}}^l \in \mathbb{R}^{|\mathcal{Q}|\times F\times D}$ and $\bar{\boldsymbol{Y}}_{\mathcal{Q}}^l \in \mathbb{R}^{|\mathcal{Q}|\times C\times D}$ and anchor representations $\bar{\boldsymbol{X}}_{\mathcal{A}}^l \in \mathbb{R}^{|\mathcal{A}|\times F\times D}$ and $\bar{\boldsymbol{Y}}_{\mathcal{A}}^l \in \mathbb{R}^{|\mathcal{A}|\times C\times D}$, respectively.

Cross-Entity Attention: This mechanism operates across the query-anchor dimension, allowing queries to gather relevant context by directly comparing their own features and labels to the anchor set, that is, the query set $\mathcal{Q}$ attends directly to the corresponding channel across the anchor set $\mathcal{A}$:

$$\boldsymbol{X}_{\mathcal{Q}}^{l+1} = \texttt{TransformerBlock}\left(\bar{\boldsymbol{X}}_{\mathcal{Q}}^l, \bar{\boldsymbol{X}}_{\mathcal{A}}^l\right), \quad \boldsymbol{Y}_{\mathcal{Q}}^{l+1} = \texttt{TransformerBlock}\left(\bar{\boldsymbol{Y}}_{\mathcal{Q}}^l, \bar{\boldsymbol{Y}}_{\mathcal{A}}^l\right)$$

Here, queries are query representations, while keys and values are anchor representations. Only queries are updated by this step; the anchors carry their intra-entity representations, as is standard in tabular foundation models (Hollmann et al., 2025; Qu et al., 2025). See Appendix D for details.

(3) Similarity Estimation (Figure 2c). Having mapped the features and labels of each query and anchor into a shared latent space, we quantify the relational affinity between each query $q \in \mathcal{Q}$ and anchor $a \in \mathcal{A}$ by computing a scalar similarity score $s(q, a) \in \mathbb{R}$ using both feature and label representations. The scoring function can be parameterized by another neural net or can be non-learnable, such as a scaled dot product. We employ a scaled Frobenius inner product similarity, calculated as

$$s(q,a) = \frac{1}{2D}\left(\frac{1}{F}\langle \boldsymbol{X}_q, \boldsymbol{X}_a\rangle_{\text{Frob}} + \frac{1}{C}\langle \boldsymbol{Y}_q, \boldsymbol{Y}_a\rangle_{\text{Frob}}\right)$$

where $\langle\cdot,\cdot\rangle_{\text{Frob}}$ denotes the Frobenius inner product (a dot product over flattened matrices). By normalizing over the hidden dimensionality D and averaging over the channel sets, we ensure the resulting similarity metric remains numerically stable and invariant to the input data cardinality (F and C).

(4) Anchor-Class Aggregator (Figure 2d). To derive class-level predictions, we aggregate the pairwise similarity scores across anchors by their labels. The aggregation should be robust to class imbalance and produce logits for a standard cross-entropy loss. An operation that achieves both is the **LogMeanExp** (LME) operator. Given a single query with a vector of similarity scores $\boldsymbol{s} \in \mathbb{R}^{|\mathcal{A}|}$, where each score corresponds to the similarity with one of the anchors, and a binary mask matrix $\boldsymbol{M} \in \{0,1\}^{C\times|\mathcal{A}|}$ indicating the class each anchor is affiliated with, the class-level logit $\boldsymbol{z}_c$ is defined as:

$$\boldsymbol{z}_c = \text{LME}(\boldsymbol{s},\boldsymbol{M})_c = \begin{cases} \log\left(\frac{1}{\|\boldsymbol{M}_c\|_1}\boldsymbol{M}_c \exp(\boldsymbol{s})\right), & \text{if } \|\boldsymbol{M}_c\|_1 > 0, \\ -\infty, & \text{otherwise,} \end{cases} \quad \forall c \in [C]$$

where $\|\boldsymbol{M}_c\|_1 = \sum_{a=1}^{|\mathcal{A}|} |\boldsymbol{M}_{c,a}|$ denotes the total number of anchors belonging to class c. By explicitly dividing by the class support size, this operator ensures that the contextual contribution of any given class is determined by its average anchor likelihood, preventing imbalanced classes from dominating the summation. The predicted class is then the index of the maximum logit, $y_{\text{pred}} = \arg\max_c \boldsymbol{z}_c$.

2.4 Node Regression

We follow a common protocol to convert regression into classification (Stewart et al., 2026) via two deterministic transformations of the target space. **Input Discretization** first transforms continuous values into soft bins, while **Output Interpolation** maps the model's aggregated classification logits from bin centroids back into scalars. Neither introduces learned parameters or a separate regression head.

Input Discretization. Given the continuous target matrix $\boldsymbol{Y} \in \mathbb{R}^{N\times 1}$, we partition its value range into C discrete bins[1] defined by their edges $\boldsymbol{e} = (e_0,\ldots,e_C) \in \mathbb{R}^{C+1}$. Rather than assigning each node $n \in [N]$ to a single hard bin, we transform its scalar target $\boldsymbol{Y}_{n,0} \in \mathbb{R}$ into a soft probability distribution $\boldsymbol{Y}'_{n,:} \in [0,1]^C$. Since a hard assignment would treat the bins as unordered, independent categories and erase the numerical distance between them, our soft approach explicitly preserves the ordinal relationships. This ensures the model understands that adjacent bins are numerically closer to one another than distant bins. This is achieved via an exponential softening function:

$$\boldsymbol{Y}'_{n,c} = \frac{\exp(-\tau|\mu_c - \boldsymbol{Y}_{n,0}|)}{\sum_{j=1}^{C}\exp(-\tau|\mu_j - \boldsymbol{Y}_{n,0}|)}, \quad \forall n \in [N], c \in [C]$$

where $\mu_c = \frac{e_{c-1}+e_c}{2}$ represents the centroid of bin c, and τ is a temperature controlling the concentration of the distribution. These soft labels are then processed as in classification, lifting each scalar into the initial 3D label tensor via the shared MLP $(\boldsymbol{Y}_{\text{init}})_{n,c,:} = \phi_{\text{label}}(\boldsymbol{Y}'_{n,c})$ for all $n \in [N], c \in [C]$ — the same ϕ_{label} as in Section 2.2, whose parameters are independent of the bin count C.

Output Interpolation. The main classification pipeline outputs a vector of class-level logits $\boldsymbol{z} \in \mathbb{R}^C$ for a given query. At the output stage, we recover the final continuous prediction y_{pred} by computing a similarity-weighted average over all bin centroids $y_{\text{pred}} = \sum_{c=1}^{C} \text{softmax}(\boldsymbol{z})_c \mu_c$. By allowing the query prediction to be interpolated across a range of relevant bin centroids, this expectation accurately captures the underlying continuous distribution. Furthermore, this architectural unification allows the model to be optimized using Kullback-Leibler (KL) divergence between the soft target distributions $\boldsymbol{Y}'$ generated during the discretization phase and the predicted output probabilities $\text{softmax}(\boldsymbol{z})$. This seamlessly aligns with our classification framework: minimizing KL divergence is fundamentally equivalent (the gradients are identical) to minimizing cross-entropy for soft labels, and strictly reduces to the standard cross-entropy objective when targets are one-hot encoded.

[1]The number of bins and binning strategies are not fixed and can vary for each dataset.

2.5 EDGE- AND GRAPH-LEVEL TASKS

Link Prediction. In this setting, the objective is to score edges rather than assign discrete categories. We adapt the core pipeline for this task with several modifications: *(1)* we bypass the 3D label branch since there are no node-level classes to embed; *(2)* we treat all nodes as candidate anchors, allowing the Similarity Estimator to directly output a pairwise similarity vector $\boldsymbol{s} \in \mathbb{R}^N$ for a given query, representing edge likelihoods (see Figure 1b); *(3)* we remove the target edges from message passing during both training and inference to prevent the GNN encoder from trivially memorizing the training topology; and *(4)* we frame training as a binary classification task over the edge scores.

Given a batch of observed edges, we designate the source nodes as queries $\mathcal{Q}$, the target nodes as positive anchors $\mathcal{A}_{\text{pos}}$, and augment this support set with negative anchors $\mathcal{A}_{\text{neg}}$ where 256 negative anchors are sampled uniformly without replacement from non-neighbors in the training graph per positive edge. We then use LME to independently aggregate the similarities of the positive and negative anchor sets for each query. Finally, we apply cross-entropy over the two class logits.

Graph-Level Tasks. For graph classification and regression, the unit of representation is not a root node but the entire graph (see Figure 1c). The core pipeline is adapted through: *(1)* computing a single graph representation after the GNN encoder by mean pooling; *(2)* injecting the 3D label tensor into the pooled graph-level representations before the Query-Anchor Transformer. Once these global embeddings are contextualized against the anchor graphs, similarity estimation and prediction proceed as in the node-level pipeline, with graph regression supported as in Section 2.4.

3 EXPERIMENTS

Datasets. Following recent works on the quality of graph learning benchmarks (Platonov et al., 2023; Bechler-Speicher et al., 2025), we evaluate the UC across a diverse and representative corpus of real-world datasets spanning various input feature distributions, graph properties (e.g., sparse, dense, homophilic, heterophilic), and labels. We partition our suite across four tasks: **(1) Node Classification**: *arxiv* (Hu et al., 2020), *roman-empire*, *amazon-ratings* (Platonov et al., 2023), *full-dblp*, *full-cora* (Bojchevski & Günnemann, 2018), *hm-categories*, *tolokers-2*, *city-reviews*, and *artnet-exp* (Bazhenov et al., 2025); **(2) Node Regression**: Random Low (RL) and High (RH) splits of *city-roads-M*, *city-roads-L*, *twitch-views*, and *artnet-views* (Bazhenov et al., 2025); **(3) Link Prediction**: *roman-empire* (Platonov et al., 2023), *wiki-cs* (Mernyei & Cangea, 2022), *city-roads-M* and *tolokers-2* (Bazhenov et al., 2025); and **(5) Graph Regression**: electronic-circuit datasets *EC5*, *EC7* and *EC10* with efficiency (EFF) and output-voltage (VOUT) targets (Stoll et al., 2025). We make all directed graphs undirected by adding inverse edges. Dataset statistics are presented in Appendix E.

Pre-training. We train UC on a mixture of openly available and synthetic datasets comprising node- and edge-level tasks. While organizational policy prevents us from releasing the mixture, our training code, or the parameter count, we state the property on which the validity of our results depends: no evaluation dataset appears in the pre-training mixture — no evaluation graph, no variant of one, and no split of one is seen under any task, either as a graph, in its features, or in its labels.

We evaluate a single model's out-of-the-box generalization via zero-shot (ZS) transfer, while assessing headroom performance via per-dataset fine-tuning (FT). Experiments were run on TPUv7. Re-training a model several times solely to estimate initialization variance is prohibitively expensive, so we evaluate a single fixed checkpoint and report UC results without error bars. Fine-tuning seed randomness stays below 0.1% in most cases.

3.1 NODE CLASSIFICATION

Baselines. We compare our framework against state-of-the-art inductive node classifiers GraphAny (Zhao et al., 2025) and TS-Mean (Finkelshtein et al., 2025), along with recent methods inspired by tabular foundation models: NodePFN (Choi et al., 2026) and GraphPFN (Eremeev et al., 2026). We also compare against best-reported supervised GNN models trained end-to-end on each dataset in the full-batch mode to measure a possible performance headroom. A fair comparison lies with methods that can be applied zero-shot to an unseen graph of arbitrary feature and label cardinality without fitting a dataset-specific output layer. Baseline results are sourced from the respective papers.

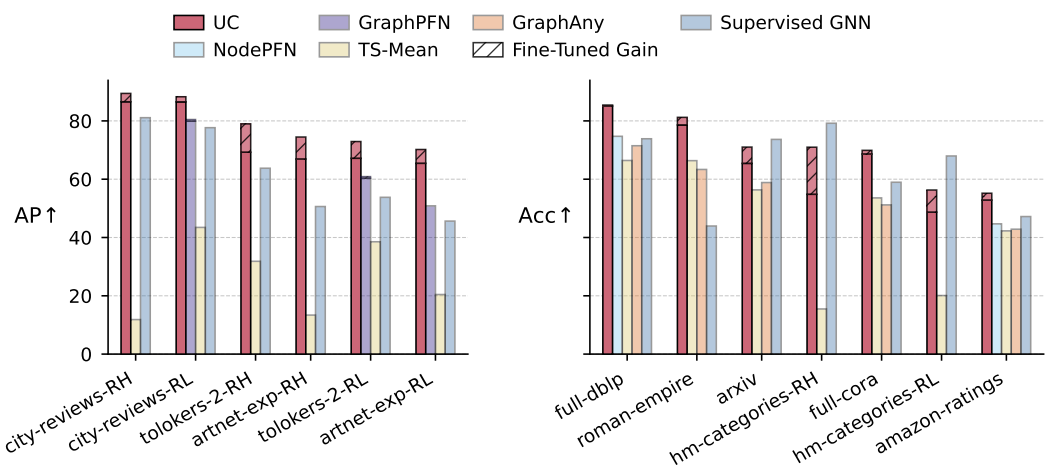

Figure 3: The UC performance on binary (left, Average Precision) and multi-class node classification (right, Accuracy) datasets in the zero-shot transfer and fine-tuning scenarios compared to inductive classifiers, tabular FM baselines, and GNNs trained end-to-end on each dataset.

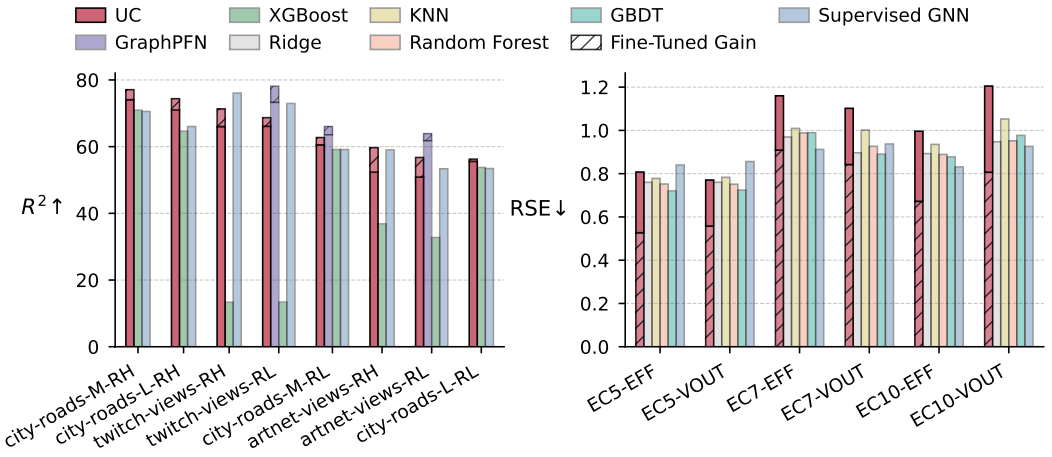

Figure 4: The UC performance on node (left, R^2, higher is better) and graph (right, Relative Squared Error, lower is better) regression datasets in the zero-shot transfer and fine-tuning scenarios compared to tabular FMs and end-to-end trained baselines.

Results and Observations. The results in Figure 3 demonstrate the UC consistently outperforms both inductive classifiers and tabular FM-derived baselines on graphs with different structural, feature, and label properties. Further short dataset-specific fine-tuning enables even higher classification performance. Notably, in most cases, the UC even surpasses supervised GNNs trained separately on each graph end-to-end. The single outlier on *hm-categories* can be attributed to the dataset's larger average degree (230+) to which full-batch trained GNNs are more amenable.

3.2 Generalizing To Node Regression

Baselines. We evaluate UC against end-to-end supervised GNN and XGBoost (Chen & Guestrin, 2016) baselines. While included to establish a performance ceiling, comparing our zero-shot framework to models optimized on the target datasets is inherently disadvantaged. A fair comparison is GraphPFN (Eremeev et al., 2026), currently the only other zero-shot regression method.

Results and Observations. As shown in Figure 4, framing regression as soft-binned classification via KL divergence is highly effective. The UC remains highly competitive; matching or exceeding the transfer performance of specialized tabular foundation models. Notably, a key advantage of

tabular-inspired baselines stems from their extensive pre-training on large mixtures of synthetic datasets. Upon fine-tuning, the UC improves markedly across all datasets, narrowing the gap.

3.3 Generalizing to Graph-Level Tasks

Baselines. To our knowledge, no generalist model performs graph regression on unseen graphs, so we compare the UC against models trained per dataset: a supervised GNN and four classical regressors, Ridge, k-nearest neighbors (KNN), Random Forest, and gradient-boosted trees (GBDT). We report the relative squared error (RSE), where 1.0 equals predicting the training mean.

Results and Observations. Figure 4 shows that the fine-tuned UC attains the lowest error on all six targets, without adding a graph-level head or any parameters. Remarkably, although the checkpoint was never pre-trained on a graph-level objective, the zero-shot UC already matches the supervised GNN on EC-5. We found discrepancies in the original EC dataset splits from Stoll et al. (2025) and re-ran the baselines, more details in Appendix E.

3.4 Link Prediction as Classification

Baselines. As no existing multi-task model performs link prediction without end-to-end training, we evaluate the UC against end-to-end supervised link prediction models SEAL (Zhang & Chen, 2018), NBFNet (Zhu et al., 2021), and BUDDY (Chamberlain et al., 2023) which establish a performance ceiling; comparing against them is inherently disadvantageous, but it contextualizes the out-of-the-box generalization capabilities of the UC on unseen graphs.

Evaluation Protocol. We discard the labels and construct a link prediction task on each graph. Validation/test hold out 200/250 edges with distinct and disjoint source nodes. Message passing uses training edges only at every stage, so no held-out edge is visible. Each positive is ranked against 200 uniformly drawn candidate targets, with the source's true neighbors masked and ties broken pessimistically. All baselines are trained on the same subset of 10,000 edges (further details in Appendix E).

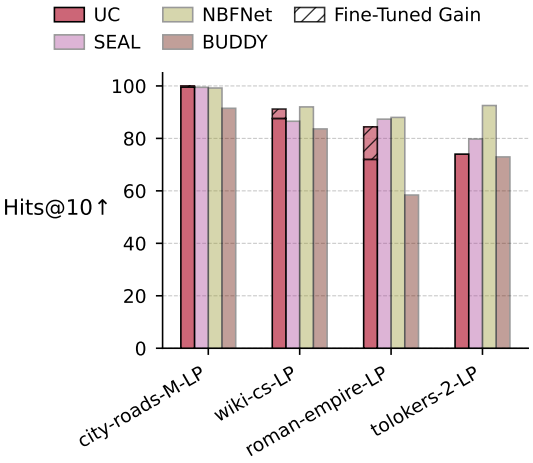

Figure 5: The UC performance on link prediction, reporting Hits@10 in the zero-shot transfer and fine-tuning scenarios compared to end-to-end trained baselines.

Results and Observations. Figure 5 shows that the UC generalizes to link prediction on unseen graphs: zero-shot, it outperforms BUDDY on every dataset and all baselines on *city-roads-M*. Fine-tuning further narrows the gap, attaining the best result on *city-roads-M* and approaching NBFNet on *wiki-cs*; only on the dense *tolokers-2* do path-based models retain an edge.

3.5 Ablations and Qualitative Analysis

Ablations. We probe the importance of certain components of the UC architecture and re-train the entire model from scratch with ablated modules and report performance on *roman-empire* (classification) and *city-roads-M-RH* (regression). Table 1 summarizes the results indicating that a shallow 2-layer Query-Anchor Transformer is detrimental, and 3-hop subgraph sampling offers no benefit over 2 hops. Removing or replacing whole modules is even more revealing, as no such variant approaches the full model. The GNN encoder contributes most: without it, performance on *city-roads-M-RH* falls below the

Table 1: UC architectural ablations.

Parameter	*roman-empire* (Acc)	*city-roads-M-RH* (R^2)
Base UC	78.61	74.08
- 2 Trf layers	75.63	70.90
- 3 hops	77.25	67.94
- GNN encoder	64.35	53.30
- Tabular encoder	74.27	73.11
- Mean score agg	72.83	71.79

XGBoost and supervised GNN baselines (Table 6), indicating that the structure does substantial work rather than decorating a tabular pipeline. Replacing the LME aggregator with simple averaging and removing the tabular encoder both degrade performance, the former slightly more on both tasks.

Inference-time scaling. The UC is also an in-context learning method and we posit that more context anchors improve performance at the cost of inference-time compute. To measure the impact of the context size, we probe downstream classification and regression performance by progressively reducing the support set of anchors. We employ two strategies for anchor sampling: *balanced* with the same number of anchors per class; and *random* where class sizes may differ. Results indicate (Figure 6) that the UC benefits from inference-time compute on both classification and regression but is still able to recover $80 + \%$ of the final performance in extremely compute-limited cases where only $1 - 5\%$ of anchors are present, thus saving $20 - 100\times$ flops of compute. Notably, the *random* anchor sampling strategy is overall more effective and, e.g., recovers 90+% of the final performance with $< 1\%$ of anchors on *arxiv* and *artnet-views-RH*.

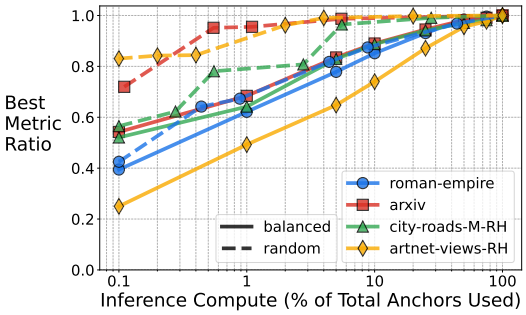

Figure 6: Inference-time performance (accuracy for classification and R^2 for regression) scaling with larger context sizes. 5% of anchors is often enough to recover 80+% of the final performance.

Peeking into representations. UC anchor embeddings form clear label clusters (Appendix A).

4 Related Work

Inductive Node Classifiers. A handful of GNNs (Zhao et al., 2024; 2025; Finkelshtein et al., 2025) have been proposed for performing node classification on unseen graphs. Typically, such methods rely on classification-specific feature and class preprocessing, do not leverage representation learning, do not show benefits of scaling data and compute, and cannot generalize to node regression nor link-, and graph-level tasks. While some recent models (Bechler-Speicher et al., 2026) extend to both node classification and link prediction, they remain bound to one graph and fixed input feature dimensions.

Tabular and Relational FMs. The success of tabular FMs (Hollmann et al., 2023; 2025; Qu et al., 2025) and their temporal extensions (Das et al., 2024; Hoo et al., 2025) inspired graph variants that encode the topology as additional Laplacian- or random walk-derived feature columns with non-parametric message passing (Eremeev et al., 2025; Hayler et al., 2025), sometimes trained on synthetic graphs (Eremeev et al., 2026; Choi et al., 2026). Beyond single tables, Relational Deep Learning processes multi-table databases as temporal, heterogeneous graphs linked by primary-foreign keys (Wang et al., 2024a; Dwivedi et al., 2025), and generalist models (Hudovernik et al., 2026; Xu et al., 2026) either rely on the graph structure induced by the schema or adapt single-table FMs. Albeit effective, such methods are confined to node classification and regression tasks and are generally agnostic to the input graph structure and, therefore, cannot perform link- or graph-level tasks out of the box. We posit that tabular FMs are a strict subset of GFMs as they treat row-wise data as a set of disconnected nodes. We discuss more related work in Appendix B.

5 Conclusion and Future Work

We introduced the Universal Classifier (UC) that frames common graph learning tasks under the same classification objective. The UC blends representation learning with the in-context learning mechanism built around *queries* and *anchors* that allows a single model to effectively generalize to node-, link-, and graph-level tasks. Extensive experiments demonstrate that a single pre-trained UC model can achieve strong zero-shot generalization across diverse, unseen graphs with different structural properties, feature and label distributions, and frequently outperforms dataset-optimized, task-specific architectures. More discussion on limitations and future work is in Appendix G.

AI USE STATEMENT

In this work, we used generative AI tools to draft and edit parts of the paper for readability and length, format tables and references, identify relevant literature, provide feedback on the manuscript, and edit software code. Additionally, we have not used generative AI tools to develop the method, propose hypotheses, design experiments, generate synthetic datasets, or implement methods, and the remaining required-disclosure tasks (formulating or proving mathematical claims, translation, and qualitative data analysis) are not applicable to this work. We have reviewed all AI-assisted work: every AI-suggested edit was reviewed and, where needed, rewritten by the authors, all reported numbers were checked against our experiment logs, and all suggested citations were verified against the original sources. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the details needed to reimplement and evaluate the UC. While organizational policy prevents us from releasing the pre-training mixture and training code, the reported results do not hinge on its exact composition: we have pre-trained the UC on many different mixtures of node- and edge-level datasets and consistently obtained performance similar to that reported here, so our results can be reproduced by pre-training on any comparable mixture of openly available graphs that excludes the evaluation datasets. The architecture and objectives are specified in Sections 2.2 to 2.5, the sampling procedure and a pseudocode training step in Section D (Algorithm 1), and all architectural and training hyperparameters in Section F. All evaluation datasets are publicly available; their statistics, splits, and the full link-prediction protocol are given in Section E, and complete per-dataset results in Section H.

REFERENCES

Gleb Bazhenov, Oleg Platonov, and Liudmila Prokhorenkova. Graphland: Evaluating graph machine learning models on diverse industrial data. In *NeurIPS*, 2025.

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. In *ICML*, 2025.

Maya Bechler-Speicher, Yoel Gottlieb, Andrey Isakov, David Abensur, Ami Tavory, Daniel Haimovich, Ido Guy, and Udi Weinsberg. Billion-scale graph foundation models, 2026. arXiv.

Beatrice Bevilacqua, Joshua Robinson, Jure Leskovec, and Bruno Ribeiro. Holographic node representations: Pre-training task-agnostic node embeddings. In *ICLR*, 2025.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *ICLR*, 2018.

Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M. Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. In *ICLR*, 2023.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *KDD*, 2016.

Jeongwhan Choi, Jongwoo Kim, Woosung Kang, and Noseong Park. Learning posterior predictive distributions for node classification from synthetic graph priors. In *ICLR*, 2026.

Yuanning Cui, Zequn Sun, and Wei Hu. A prompt-based knowledge graph foundation model for universal in-context reasoning. In *NeurIPS*, 2024.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *ICML*, 2024.

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 *ICLR*, 2021.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *JMLR*, 2023.

Vijay Prakash Dwivedi, Charilaos Kanatsoulis, Shenyang Huang, and Jure Leskovec. Relational deep learning: Challenges, foundations and next-generation architectures. In *KDD*, 2025.

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

Dmitry Eremeev, Oleg Platonov, Gleb Bazhenov, Artem Babenko, and Liudmila Prokhorenkova. GraphPFN: A prior-data fitted graph foundation model. In *ICML Workshop on Graph Foundation Models: A New Era for Graph Machine Learning*, 2026.

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

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

Mikhail Galkin, Jincheng Zhou, Bruno Ribeiro, Jian Tang, and Zhaocheng Zhu. A foundation model for zero-shot logical query reasoning. In *NeurIPS*, 2024b.

Adrian Hayler, Xingyue Huang, İsmail İlkan Ceylan, Michael Bronstein, and Ben Finkelshtein. Bringing graphs to the table: Zero-shot node classification via tabular foundation models, 2025. arXiv.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *ICLR*, 2023.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 2025.

Shi Bin Hoo, Samuel Müller, David Salinas, and Frank Hutter. From tables to time: How TabPFN-v2 outperforms specialized time series forecasting models, 2025. arXiv.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

Xingyue Huang, Pablo Barceló, Michael M. Bronstein, İsmail İlkan Ceylan, Mikhail Galkin, Juan L. Reutter, and Miguel Romero Orth. How expressive are knowledge graph foundation models? In *ICML*, 2025.

Xingyue Huang, Mikhail Galkin, Michael M. Bronstein, and İsmail İlkan Ceylan. HYPER: A foundation model for inductive link prediction with knowledge hypergraphs. In *ICLR*, 2026.

Valter Hudovernik, Federico López, Vid Kocijan, Akihiro Nitta, Jan Eric Lenssen, Jure Leskovec, and Matthias Fey. KumoRFM-2: Scaling foundation models for relational learning, 2026. arXiv.

Jinwoo Kim, Xingyue Huang, Krzysztof Olejniczak, Kyungbin Min, Michael M. Bronstein, Seunghoon Hong, and İsmail İlkan Ceylan. Flock: A knowledge graph foundation model via learning on random walks. In *ICLR*, 2026.

Jaejun Lee, Chanyoung Chung, and Joyce Jiyoung Whang. InGram: Inductive knowledge graph embedding via relation graphs. In *ICML*, 2023.

Péter Mernyei and Cătălina Cangea. Wiki-CS: A wikipedia-based benchmark for graph neural networks, 2022. arXiv.

Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In *ICLR*, 2023.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *ICML*, 2025.

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. In *ICML*, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

Lawrence Stewart, Francis Bach, and Quentin Berthet. Building bridges between regression, clustering, and classification. In *AISTATS*, 2026.

Timo Stoll, Chendi Qian, Ben Finkelshtein, Ali Parviz, Darius Weber, Fabrizio Frasca, Hadar Shavit, Antoine Siraudin, Arman Mielke, Marie Anastacio, Erik Müller, Maya Bechler-Speicher, Michael Bronstein, Mikhail Galkin, Holger Hoos, Mathias Niepert, Bryan Perozzi, Jan Tönshoff, and Christopher Morris. GraphBench: Next-generation graph learning benchmarking. *arXiv*, 2025.

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? reassessing the long-range graph benchmark. *TMLR*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models, 2023. arXiv.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9(11), 2008.

Minjie Wang, Quan Gan, David Wipf, Zhenkun Cai, Ning Li, Jianheng Tang, Yanlin Zhang, Zizhao Zhang, Zunyao Mao, Yakun Song, Yanbo Wang, Jiahang Li, Han Zhang, Guang Yang, Xiao Qin, Chuan Lei, Muhan Zhang, Weinan Zhang, Christos Faloutsos, and Zheng Zhang. 4DBInfer: A 4D benchmarking toolbox for graph-centric predictive modeling on relational DBs. In *NeurIPS*, 2024a.

Zehong Wang, Zheyuan Zhang, Nitesh V. Chawla, Chuxu Zhang, and Yanfang Ye. GFT: Graph foundation model with transferable tree vocabulary. In *NeurIPS*, 2024b.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Linjie Xu, Yanlin Zhang, Quan Gan, Minjie Wang, and David Wipf. No need to train your RDB foundation model. In *ICML*, 2026.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NeurIPS*, 2018.

Yucheng Zhang, Beatrice Bevilacqua, Mikhail Galkin, and Bruno Ribeiro. TRIX: A more expressive model for zero-shot domain transfer in knowledge graphs. In *LoG*, 2024.

Haihong Zhao, Aochuan Chen, Xiangguo Sun, Hong Cheng, and Jia Li. All in one and one for all: A simple yet effective method towards cross-domain graph pretraining. In *KDD*, 2024.

Jianan Zhao, Zhaocheng Zhu, Mikhail Galkin, Hesham Mostafa, Michael Bronstein, and Jian Tang. Fully-inductive node classification on arbitrary graphs. In *ICLR*, 2025.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: a general graph neural network framework for link prediction. In *NeurIPS*, 2021.

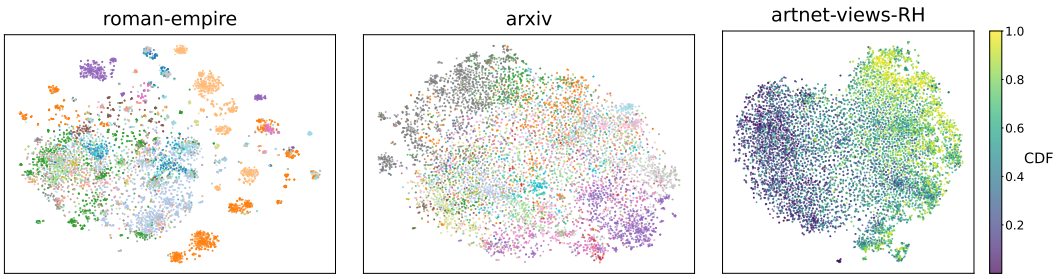

Figure 7: t-SNE projections of UC anchor representations for node classification (*roman-empire, arxiv*) and regression (*artnet-views-RH*) datasets cluster around the labels. In regression datasets, labels are ordinal bins constructed around centroids, and nearby bins cluster together.

A Visualizing Learned Representations

The UC is a representation learning method and we examine the quality of anchor representations (coming from the training set) $\boldsymbol{X}_{\mathcal{A}}$ of different classification and regression datasets. When passed through the GNN and Query-Anchor Transformer encoders and projected with t-SNE (Van der Maaten & Hinton, 2008), we observe (Figure 7) the representations to form clusters indicative of the node labels both for heterophilic *roman-empire* and homophilic *arxiv*. When applied to regression (*artnet-views-RH)*, continuous labels are discretized into bins that have an inherent ordinal structure, e.g., a centroid of i-th bin μ_i is numerically closer to μ_{i-1} than to μ_{i-5}. Figure 7 demonstrates that the representations successfully capture this ordinality: bins corresponding to lower target values (the bottom 40%) and higher target values (the top 40%) naturally organize into distinct clusters.

B Additional Related Work

We complement Section 4 with additional lines of work that relate to UC.

Multi-Task Models on Text-Attributed Graphs. Another line of work merges task levels rather than schemas. GFT (Wang et al., 2024b) is representative: node-, link- and graph-level prediction are all cast as classification over a shared vocabulary of computation trees, so one backbone serves all three. However, it is not schema invariant. Node and edge text is mapped to a fixed 768-dimensional space by a sentence transformer, and the decoder is a classifier sized to the target dataset's class count, so a graph with non-textual attributes, or with a different number of classes, calls for a new model with new weights rather than a new context.

Knowledge Graph FMs. KGs are graphs coming (in most cases) without node features and only having distinct edge types. The meaningful tasks on such graphs revolve around single- and multi-hop KG completion (link prediction) and such a constrained setup was amenable for the first attempts at creating generalist, inductive link predictors (Lee et al., 2023; Galkin et al., 2024a;b; Zhang et al., 2024; Cui et al., 2024; Huang et al., 2025; Kim et al., 2026), which kickstarted the community interest in GFMs. Despite the fast progress and extensions to novel structures like relational hypergraphs (Huang et al., 2026), such models can hardly generalize to node- and graph-level tasks, cannot deal with arbitrary features, and do not seem to benefit from scaling (as leveraging pure structural patterns is often enough for high link prediction performance).

Scalar Lifting and Label Handling. Lifting a scalar into a shared latent axis is not specific to graphs: tabular foundation models (TFMs) encode individual table cells similarly (Hollmann et al., 2025; Qu et al., 2025), and schema-invariant graph models (Bevilacqua et al., 2025; Finkelshtein et al., 2025) adopt the construction on the feature axis. The UC departs from them in two respects. First, it lifts labels alongside features, a trait shared only by Finkelshtein et al. (2025). Second, they differ in what follows the lifting. Prior graph constructions and TFMs let the label space set the output dimension, either through a prediction head whose width is the class count or by reading predictions directly off the lifted label tensor. The UC does neither: an anchor's label is input information only, not a channel to project onto, and a prediction is a comparison, with each query scored against every anchor and those scores pooled over the anchors sharing a class (see Section 2.3).

C Time Complexity and Runtime

We derive the asymptotic cost of a forward pass of the UC, compare it to a standard message-passing network, and report its runtime in practice.

Time complexity. We analyze the cost in three parts: (1) the GNN Encoder, (2) intra-entity attention in the Query-Anchor Transformer, and (3) cross-entity attention: (1) A standard message-passing layer costs $O(|\mathcal{E}| \cdot D + N \cdot D^2)$, comprising edge-wise aggregation and the node-wise update. The UC carries the lifted axis as a batch dimension, so both terms are multiplied by the channel count, F for the feature branch and C for the label branch, with nothing else changed. (2) The Query-Anchor Transformer operates on the query and anchor sets rather than on the graph: intra-entity attention is self-attention over the $(F + C)$ channels of each entity and costs $O\big((|\mathcal{Q}| + |\mathcal{A}|)(F + C)^2 D\big)$ per block. (3) Cross-entity attention lets each query attend to the anchors within each channel at $O\big(|\mathcal{Q}||\mathcal{A}|(F + C)D\big)$ per block. Letting L_{GNN} denote the number of message-passing layers and L_{T} the number of Transformer blocks, a forward pass costs

$$O\big(L_{\text{GNN}}(|\mathcal{E}| \cdot D + N \cdot D^2)(F + C) \;+\; L_{\text{T}}\big[(|\mathcal{Q}| + |\mathcal{A}|)(F + C)^2 D \;+\; |\mathcal{Q}||\mathcal{A}|(F + C)D\big]\big).$$

Since $|\mathcal{Q}|$ and $|\mathcal{A}|$ are fixed at 256 and 1024, they are constants and can be absorbed, leaving

$$O\big(L_{\text{GNN}}(|\mathcal{E}| \cdot D + N \cdot D^2)(F + C) + L_{\text{T}}(F + C)^2 D\big).$$

Against $O\big(L_{\text{GNN}}(|\mathcal{E}| \cdot D + N \cdot D^2)\big)$ for a message-passing network of the same depth, the UC is roughly a factor $(F + C)$ more expensive, and scales with graph size in exactly the same way.

This is also a pessimistic estimate, since the UC never performs a full-graph forward pass: each step samples anchors and queries and extracts k-hop subgraphs around them, so $|\mathcal{E}|$ and N above are those of the sampled subgraphs rather than of the dataset.

Runtime. With a single fixed checkpoint on a TPUv7, a complete pass over the test set takes under five minutes on *roman-empire*, *amazon-ratings* and *arxiv*. A cost of that order per pass keeps training and repeated evaluation over many epochs tractable, in spite of the $(F + C)$ factor in the time complexity.

D Sampling, Label Masking, and Training Step

This section gives the sampling procedure of Section 2.2 in full: the distribution the query and anchor sets are drawn from, the masking applied to query labels before they enter the model, and the resulting training step written out as pseudocode.

Let $\mathcal{V}_{tr}$ denote the set of elements in the training split, which are nodes for node- and edge-level tasks and graphs for graph-level tasks. At each training step we draw

$$\mathcal{A} \sim \text{Unif}\big(\{S \subseteq \mathcal{V}_{tr} : |S| = n_A\}\big), \qquad \mathcal{Q} \mid \mathcal{A} \sim \text{Unif}\big(\{S \subseteq \mathcal{V}_{tr} \setminus \mathcal{A} : |S| = n_Q\}\big)$$

with $n_A = 1024$ and $n_Q = 256$, giving the joint law

$$P(\mathcal{A}, \mathcal{Q}) = \binom{|\mathcal{V}_{tr}|}{n_A}^{-1} \binom{|\mathcal{V}_{tr}| - n_A}{n_Q}^{-1}.$$

Drawing $\mathcal{Q}$ from $\mathcal{V}_{tr} \setminus \mathcal{A}$ is what enforces $\mathcal{Q} \cap \mathcal{A} = \emptyset$. The distribution is exchangeable over elements: there is no degree-, class- or centrality-based weighting, so every element is marginally equally likely to serve as an anchor or as a query. Label masking is then the deterministic map

$$\widetilde{\boldsymbol{Y}}_v = \boldsymbol{Y}_v \;\text{ if } v \in \mathcal{A}, \qquad \widetilde{\boldsymbol{Y}}_v = \boldsymbol{0} \;\text{ if } v \in \mathcal{Q},$$

applied before ϕ_{label}, so that no query label can enter message passing or attention. For link prediction, given an observed training edge (u, v), the source u is the query and v a positive anchor, and ten negative anchors are drawn uniformly without replacement from the non-neighbors of u in the training graph. At inference, $\mathcal{A}$ is the training split, or a fixed subsample of it where the split exceeds the anchor budget, and $\mathcal{Q}$ is the set of target elements.

Training step. The training step follows the description in Sections 2.2 and 2.3, and Algorithm 1 states the resulting procedure for node classification. Node regression replaces the one-hot labels

Algorithm 1 One UC training step for node classification.

Require: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{X}, \boldsymbol{Y})$; training split $\mathcal{V}_{tr}$; budgets n_A, n_Q; hops k; layers $L_{\text{GNN}}, L_{\text{T}}$
Ensure: cross-entropy loss over the query set

$\mathcal{A} \sim \text{Unif}\big(\{S \subseteq \mathcal{V}_{tr} : |S| = n_A\}\big)$ ▷ anchors
$\mathcal{Q} \sim \text{Unif}\big(\{S \subseteq \mathcal{V}_{tr} \setminus \mathcal{A} : |S| = n_Q\}\big)$ ▷ queries; enforces $\mathcal{Q} \cap \mathcal{A} = \emptyset$
$\widetilde{\boldsymbol{Y}}_v \leftarrow \boldsymbol{Y}_v$ for $v \in \mathcal{A}$; $\widetilde{\boldsymbol{Y}}_v \leftarrow \boldsymbol{0}$ for $v \in \mathcal{Q}$ ▷ label masking
$\mathcal{G}_{\text{uni}} \leftarrow \bigcup_{r \in \mathcal{Q} \cup \mathcal{A}} \mathcal{G}^k(r)$ ▷ subgraph sampling
$\boldsymbol{X} \leftarrow \text{ZNORMALIZE}(\boldsymbol{X})$ ▷ feature-wise
$(\boldsymbol{X}^0)_{n,f,:} \leftarrow \phi_{\text{feat}}(\boldsymbol{X}_{n,f}) \ \ \forall n, f$ ▷ feature lifting
$(\boldsymbol{Y}^0)_{n,c,:} \leftarrow \phi_{\text{label}}(\widetilde{\boldsymbol{Y}}_{n,c}) \ \ \forall n, c$ ▷ label lifting
for $l = 0$ **to** $L_{\text{GNN}} - 1$ **do** ▷ GNN Encoder
    $\boldsymbol{X}^{l+1} \leftarrow \texttt{GNN}(\boldsymbol{X}^l, \mathcal{G}_{\text{uni}})$; $\boldsymbol{Y}^{l+1} \leftarrow \texttt{GNN}(\boldsymbol{Y}^l, \mathcal{G}_{\text{uni}})$
end for
retain root representations $\{\boldsymbol{X}_r, \boldsymbol{Y}_r\}_{r \in \mathcal{Q} \cup \mathcal{A}}$
for $l = 1$ **to** L_{T} **do** ▷ Query-Anchor Transformer
    for all $r \in \mathcal{Q} \cup \mathcal{A}$ **do** ▷ intra-entity attention
        $[\bar{\boldsymbol{X}}_r, \bar{\boldsymbol{Y}}_r] \leftarrow \texttt{TransformerBlock}([\boldsymbol{X}_r \parallel \boldsymbol{Y}_r], [\boldsymbol{X}_r \parallel \boldsymbol{Y}_r])$
    end for
    $\boldsymbol{X}_{\mathcal{Q}} \leftarrow \texttt{TransformerBlock}(\bar{\boldsymbol{X}}_{\mathcal{Q}}, \bar{\boldsymbol{X}}_{\mathcal{A}})$ ▷ cross-entity attention
    $\boldsymbol{Y}_{\mathcal{Q}} \leftarrow \texttt{TransformerBlock}(\bar{\boldsymbol{Y}}_{\mathcal{Q}}, \bar{\boldsymbol{Y}}_{\mathcal{A}})$
    $\boldsymbol{X}_{\mathcal{A}} \leftarrow \bar{\boldsymbol{X}}_{\mathcal{A}}$; $\boldsymbol{Y}_{\mathcal{A}} \leftarrow \bar{\boldsymbol{Y}}_{\mathcal{A}}$ ▷ anchors unchanged by queries
end for
for all $q \in \mathcal{Q},\ a \in \mathcal{A}$ **do** ▷ Similarity Estimator
    $\boldsymbol{s}_{q,a} \leftarrow \frac{1}{2D}\big(\frac{1}{F}\langle \boldsymbol{X}_q, \boldsymbol{X}_a \rangle_{\text{Frob}} + \frac{1}{C}\langle \boldsymbol{Y}_q, \boldsymbol{Y}_a \rangle_{\text{Frob}}\big)$
end for
$\boldsymbol{M}_{c,a} \leftarrow \mathbb{1}[\text{anchor } a \text{ belongs to class } c]$ ▷ anchor-class mask
$\boldsymbol{z}_c \leftarrow \text{LME}(\boldsymbol{s}, \boldsymbol{M})_c \ \ \forall c \in [C]$ ▷ Anchor-Class Aggregator
return $\text{CrossEntropy}\big(\text{softmax}(\boldsymbol{z}),\ \boldsymbol{Y}_{\mathcal{Q}}\big)$ ▷ loss over queries only

with the soft bins of Section 2.4; edge- and graph-level tasks change only which entities serve as queries and anchors, as described in Section 2.5. Note that cross-entity attention updates the query representations only; anchors carry their intra-entity form forward unchanged, as recorded on the last line of the loop. The asymmetry is deliberate and follows the same convention as in-context learning over tables (Hollmann et al., 2025; Qu et al., 2025). The anchor set is the context against which a prediction is made, and a context that absorbed information from the query would no longer be the same context for every query: an anchor's representation would depend on which query happened to be scored alongside it, so the prediction for one target would shift with the other targets in the batch. Keeping the anchors fixed removes that coupling, makes each query's prediction a function of the context alone, and allows the anchor representations to be computed once and reused across the whole query set.

The loop in Algorithm 1 over $r \in \mathcal{Q} \cup \mathcal{A}$ and the one over $q \in \mathcal{Q},\ a \in \mathcal{A}$ are written elementwise for clarity only. Both are implemented as batched tensor operations, with the entity and channel axes carried as batch dimensions, so neither is materialized as an explicit loop at runtime.

E DATASETS

Our dataset selection is motivated by recent works calling for a paradigm shift in graph learning benchmarks. First, responding to the position paper (Bechler-Speicher et al., 2025), we move beyond narrow academic domains to target impactful applications from e-commerce, crowdsourcing, and circuit design modeled in GraphLand and GraphBench. This prevents evaluation overfitting and stagnation by exposing the model to complex, cross-domain abstractions. Second, following the empirical analysis of Platonov et al. (2023), we discard popular but flawed benchmarks (like *Squirrel* and *Chameleon*) that suffer from data leakage caused by node duplication. Instead, we prioritize more reliable, diverse datasets that properly evaluate model behavior on both homophilic and heterophilic structures. Third, graph-level benchmarks built to probe a specific property have not always turned out to require it: the Long Range Graph Benchmark was shown to close most of its reported gap once baselines are tuned and normalization is corrected, leaving little evidence that long-range interaction is what it measures (Tönshoff et al., 2024) and the widely used *zinc* regression target is similarly narrow, being largely recoverable from cycle counts rather than from the molecular structure that quantum-chemical properties depend on (Dwivedi et al., 2023).

GraphBench Electronic Circuits. We therefore use the Electronic Circuits graph regression datasets of GraphBench (Stoll et al., 2025) for our graph-level evaluation. We found, however, that the new splits in the repository are substantially different in size from those originally reported in GraphBench which makes reported baseline GNN performance incomparable. We implemented new baselines - classical ML baselines coming from the `sklearn` package which use *49-dimensional* topological features: component counts (9 dims), number of nodes, edges, density, connected components, mean/max/std degree, top 15 eigenvalues of undirected simple graph, mode feature mean (9 dims), node feature std (9 dims).

The *Supervised GNN* is GIN with best hyperparameters found from running sweeps on EC datasets. The resulting architecture includes 4 layers with hidden dimension of 128 and was trained with Adam with learing rate 0.001.

Table 2 provides summary statistics for the datasets. For the GraphLand datasets, we use both the Random Low (*RL*) splits, which divide train, validation, and test nodes into $10\%/10\%/80\%$, and the Random High (*RH*) splits, which divide them into $50\%/25\%/25\%$.

The link prediction datasets were selected to cover sparse, dense, homophilic, heterophilic graphs. The datasets are listed apart from the rest because the protocol of Section 3 rebuilds each graph before use, so their counts differ from the node-level figures above; *city-roads-M* differs most, as its upstream edges are directed. Table 3 reports them.

In-depth link prediction protocol. Every graph is made undirected, without self-loops; the node-level label is discarded and an edge-level link prediction task is constructed on the same graph. We hold out $1{,}000$ validation and $1{,}000$ test edges at random, using each source node at most once; the two sets share no edges or source nodes. These $2{,}000$ edges are removed from the graph, so message passing uses only training edges and never sees a held-out edge. We evaluate on a fixed subset of 250 validation and 250 test edges. Each positive is ranked against 200 random candidate targets, one

Table 2: Summary statistics of the datasets used in our experiments. Node and edge counts are averaged per graph, and edges are counted after symmetrization, matching the graph representation used at runtime. The link-prediction datasets are reported separately in Table 3. EC graphs carry no node features, so we turn node types into one-hot encoded features. For the EC datasets, the number of graphs is given per train/validation/test split, and the EFF and VOUT targets share the same graphs.

Task	Dataset	Graphs	Nodes	Edges	Features	Classes
Node Classification	amazon-ratings	1	24,492	93,050	300	5
	artnet-exp	1	50,405	280,348	75	2
	arxiv	1	169,343	1,157,799	128	40
	city-reviews	1	148,801	1,165,415	37	2
	full-cora	1	19,793	63,421	8,710	70
	full-dblp	1	17,716	52,867	1,639	4
	hm-categories	1	46,563	10,730,995	35	21
	roman-empire	1	22,662	32,927	300	18
	tolokers-2	1	11,758	519,000	16	2
Node Regression	artnet-views	1	50,405	280,348	50	–
	city-roads-L	1	142,257	231,550	26	–
	city-roads-M	1	57,073	107,104	26	–
	twitch-views	1	168,114	6,797,557	4	–
Graph Regression	EC5	234,084/33,442/66,884	10.0	20.0	9	–
	EC7	907/129/259	12.0	28.0	9	–
	EC10	1,389/463/2,778	16.0	40.0	9	–

shared list per split. We mask candidates that are the source or one of its true neighbors, and count ties as losses.

Table 3: Statistics of the link prediction datasets after the preprocessing described above. *Message-passing edges* excludes the 2,000 held-out validation and test edges. Feature counts are post-transform (one-hot encoding and imputation), so *city-roads-M* and *tolokers-2* exceed the raw GraphLand counts.

Dataset	Nodes	Undirected edges	Message-passing edges	Features	Mean degree
roman-empire	22,662	32,927	30,927	300	2.9
wiki-cs	11,701	215,603	213,603	300	36.9
city-roads-M	57,073	107,104	105,104	68	3.8
tolokers-2	11,758	519,000	517,000	19	88.3

F Implementation Details and Hyperparameters

All results in Section 3 are obtained with a single pre-trained checkpoint, whose training and architectural hyperparameters are listed below.

Training. We pre-train the model for 20,000 steps on TPUv7 accelerators using AdamW with a learning rate of 0.001, applying a dropout probability of 0.2. During training, the context anchor node set is sampled with a batch size of 1024, while the queried target nodes are sampled with a batch size of 256.

Architecture. Initial node embeddings are processed via a 2-layer MLP with a hidden dimension of 16. Our GNN Encoder is a 2-layer Graph Isomorphism Network (GIN) (Xu et al., 2019) with a hidden dimension of 16, a self-loop weight initialization of $\epsilon = 0.1$, a maximum of 20 neighbors per hop, and interleaved Layer Normalization. Finally, the query-anchor module consists of 2 Transformer layers with a hidden dimension of 32. A logit soft-capping scalar of 50 is applied to stabilize gradient flows and prevent extreme logit magnitudes. We create 32 bins and centers for regression values. Then, each regression value has a probabilistic assignment to bins, derived using exponential softening.

G Limitations and Future Work

While the Universal Classifier (UC) offers a unified foundation for graph learning, lifting features into 3D tensors for multiple queries and anchors introduces memory and inference overhead compared to standard task-specific GNNs. However, as demonstrated in our scaling ablations (Figure 6), the UC is highly sample-efficient. Utilizing only a small fraction ($1 - 5\%$) of available anchors recovers the vast majority of peak performance, effectively mitigating this computational bottleneck in practice.

A second limitation concerns task coverage. The evaluated checkpoint was pre-trained on node- and edge-level objectives only; although it already transfers zero-shot to some graph-level targets,

Looking forward, we intend to add graph-level datasets to the pre-training mixture, so that zero-shot transfer can be evaluated at that level as well, and to extend the Universal Classifier to multi-relational domains; treating missing knowledge graph triplets or unlinked relational database records as queries evaluated against known relations could enable complex, multi-table inferences without requiring specialized heads.

H Detailed Experimental Results

In this section, we present the full per-dataset results. Specifically, the results displayed in Figure 3 can be found in Table 4, and in Table 5 for the GraphLand datasets. The results displayed in Figure 4 can be found in Table 6 for node regression and Table 8 for graph regression dataset. The link prediction results displayed in Figure 5 can be found in Table 7.

Table 4: Node classification accuracies.

Model	**amazon-ratings**	**full-cora**	**full-dblp**	**roman-empire**	**arxiv**
Supervised GNN	47.18 ± 0.42	58.95 ± 0.55	73.87 ± 1.35	43.93 ± 0.45	73.65 ± 0.11
Zero-Shot Transfer					
GraphAny	42.80 ± 0.09	51.18 ± 0.78	71.48 ± 1.44	63.34 ± 0.58	58.85 ± 0.03
TS-Mean	42.27 ± 1.40	53.58 ± 0.73	66.42 ± 3.65	66.36 ± 1.02	56.33 ± 2.58
NodePFN	44.68 ± 0.48	-	74.71 ± 0.39	-	-
UC	52.83	68.64	85.10	78.61	65.43
Fine-Tuned					
UC	55.19	69.90	85.44	81.22	71.02

Table 5: Node classification accuracies on GraphLand datasets.

Model	**hm-categories** (Acc)		**tolokers-2** (AP)		**city-reviews** (AP)		**artnet-exp** (AP)	
	RL	RH	RL	RH	RL	RH	RL	RH
Supervised GNN	67.96 ± 0.33	79.19 ± 0.21	53.78 ± 1.34	63.76 ± 1.30	77.67 ± 0.13	81.10 ± 0.11	46.62 ± 0.32	50.62 ± 0.35
Zero-Shot Transfer								
TS-Mean	20.09 ± 1.29	15.48 ± 1.93	38.54 ± 0.94	31.83 ± 2.55	43.46 ± 5.17	-	20.44 ± 1.05	13.38 ± 2.59
GraphPFN	-	-	60.33 ± 0.60	-	79.89 ± 0.10	-	50.85 ± 0.31	-
UC	48.69	54.83	67.18	69.26	86.53	86.57	65.46	66.92
Fine-Tuned								
GraphPFN	-	-	60.90 ± 0.98	-	80.49 ± 0.16	-	50.64 ± 1.27	-
UC	56.29	70.98	72.92	78.97	88.28	89.43	70.19	74.47

Table 6: Node regression R^2 on GraphLand datasets.

Model	city-roads-M		city-roads-L		twitch-views		artnet-views	
	RL	RH	RL	RH	RL	RH	RL	RH
End-to-end								
XGBoost	59.14 ± 0.11	70.93 ± 0.05	53.75 ± 0.07	64.62 ± 0.07	13.39 ± 0.00	13.34 ± 0.02	32.74 ± 0.04	36.83 ± 0.06
Supervised GNN	59.11 ± 0.20	70.53 ± 0.40	53.43 ± 0.20	66.03 ± 0.24	72.93 ± 0.17	76.06 ± 0.30	53.36 ± 0.78	59.01 ± 0.52
Zero-Shot Transfer								
GraphPFN	63.55 ± 0.29	-	-	-	73.27 ± 0.13	-	61.77 ± 0.23	-
UC	60.54	74.08	55.58	70.98	66.09	66.02	50.85	52.34
Fine-Tuned								
GraphPFN	66.04 ± 0.41	-	-	-	78.17 ± 0.13	-	63.90 ± 0.12	-
UC	62.73	77.08	56.27	74.38	68.69	71.29	56.76	59.69

Table 7: Link prediction Hits@10 on the four link prediction datasets.

Model	roman-empire	wiki-cs	city-roads-M	tolokers-2
End-to-end				
SEAL	87.33 ± 1.15	86.53 ± 1.97	99.47 ± 0.61	79.73 ± 0.61
NBFNet	88.00 ± 0.01	92.00 ± 0.40	99.20 ± 0.01	92.53 ± 0.92
BUDDY	58.40 ± 11.09	83.60 ± 4.51	91.47 ± 5.43	72.93 ± 1.22
Zero-Shot Transfer				
UC	72.00	87.60	99.60	74.00
Fine-Tuned				
UC	84.40	91.20	100.00	73.60

Table 8: Graph Regression RSE on Electronic Circuits datasets.

	EC-5		EC-7		EC-10	
Method	EFF	VOUT	EFF	VOUT	EFF	VOUT
End-to-end						
Ridge (α=1.0)	0.760	0.760	0.970	0.897	0.893	0.948
KNN (k=10)	0.778	0.783	1.009	1.001	0.935	1.053
Random Forest (200, d=5)	0.752	0.751	0.988	0.926	0.889	0.952
GBDT (50, d=3)	0.720	0.724	0.989	0.890	0.877	0.977
Supervised GNN	0.840	0.856	0.912	0.937	0.831	0.926
Zero-Shot Transfer						
UC	0.808	0.771	1.160	1.102	0.996	1.205
Fine-tuned						
UC	0.527	0.558	0.909	0.842	0.672	0.806