SCALABLE GRAPH IN-CONTEXT LEARNING WITH STRONGER PREDICTIVE PERFORMANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Applying graph neural networks (GNNs) typically requires training and hyperparameter tuning for each dataset, incurring repeated costs in time and effort. Graph in-context learning (graph ICL) offers an alternative: a pretrained model predicts missing node labels on unseen graphs through inference alone, using observed labels as context. However, existing graph ICL models rely on dense attention across nodes, limiting inference efficiency and scalability. *We ask whether dense attention is necessary for graph in-context learning.* We introduce Ephris, which realizes graph ICL through sparse message passing, scaling linearly with node-feature entries and graph edges. We pretrain Ephris entirely on synthetic tasks from structural causal models extended with diverse graph structures and relational dynamics, capturing varied dependencies among topology, features, and labels. Across 51 node-classification datasets, we compare Ephris against 15 extensively tuned GNNs and prior graph ICL models under high-label (50/25/25) and low-label (10/10/80) train/validation/test splits. Ephris ranks 1st across all four aggregate metrics, including Elo, improvability, average rank, and accuracy, in both regimes. This strong performance comes at substantially lower adaptation cost, with inference comparable to a single GNN training run. Together, Ephris advances the performance-runtime Pareto frontier, offering a practical alternative to per-dataset training and tuning. Code and model weights are available at link.

1 INTRODUCTION

Node classification arises across diverse graph-structured domains, including social, e-commerce, and road networks (Hamilton et al., 2017; Hu et al., 2020; Liang et al., 2026). Graph neural networks (GNNs) achieve strong predictive performance with message passing between neighboring nodes, allowing node features and graph structure to jointly inform predictions (Hamilton et al., 2017; Kipf & Welling, 2017; Veličković et al., 2018). However, GNNs typically require training and careful hyperparameter (HP) tuning for each dataset, incurring substantial costs in time and effort whenever a new graph is encountered (Luo et al., 2024; Platonov et al., 2026).

In-context learning (ICL) offers an alternative paradigm: a pretrained model predicts missing labels on unseen datasets without parameter updates, using observed labels as context. Prior-data fitted networks (PFNs) (Müller et al., 2022) learn this capability by pretraining Transformers (Vaswani et al., 2017) on synthetic task priors, and tabular foundation models (TFMs) enrich these priors with structural causal models (SCMs) (Pearl et al., 2000) to capture diverse feature-label relationships. Recently, TFMs have surpassed HP-tuned supervised methods on tabular benchmarks (Jäger et al., 2026; Zhang et al., 2026a) and seen broader downstream adoption (Hicham et al., 2026; Wu et al., 2026), challenging the convention of training and tuning a separate model for each dataset.

For graphs, several methods extend this PFN framework to node classification by combining graph-aware architectures with synthetic graph priors. NodePFN (Choi et al., 2026) augments the TabPFN architecture (Müller et al., 2022) with a parallel message-passing branch and pretrains on synthetic graphs with varying homophily. GraphPFN (Eremeev et al., 2026) adds graph-attention adapters to pretrained LimiX (Zhang et al., 2025) and trains them on tasks sampled from an SCM augmented with graph convolution. Their improvements over HP-tuned GNNs on several datasets suggest that graph ICL offers a practical alternative to dataset-specific training, as it has in tabular data.

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.

However, existing methods face two key limitations that challenge the success of graph ICL. First, eliminating parameter updates does not necessarily make adaptation computationally efficient. Existing graph ICL models augment TFM backbones with graph-specific modules (Choi et al., 2026; Eremeev et al., 2026), thereby retaining the backbone's dense self-attention over labeled context nodes. Their computation therefore scales quadratically with context size, making inference increasingly expensive as more labeled nodes are provided and, in some datasets, even slower than training and tuning a GNN.

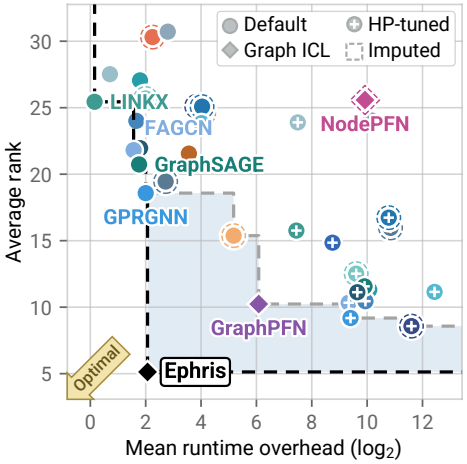


Figure 1: Performance-runtime Pareto plot across 51 datasets under the 50/25/25 split.

Second, outperforming tuned GNNs in their original evaluations does not necessarily establish broad competitiveness. NodePFN (Choi et al., 2026) is evaluated only on graphs with fewer than 50,000 nodes and against 2 GNN baselines, while GraphPFN (Eremeev et al., 2026) includes larger graphs but only datasets with at most 500 features and 4 GNN baselines. In our main experiments (Section 5), we evaluate both methods more broadly across 51 node-classification datasets, spanning up to 568,795 nodes and 8,710 features per node, against 15 GNNs. Their reported gains do not consistently persist under this broader evaluation, with tuned GNNs remaining stronger on many datasets.

To overcome both limitations, we introduce **Ephris**, a new graph in-context learner designed to push the performance-runtime frontier (Figure 1). We first develop a scalable architecture for graph ICL that operates entirely through sparse message passing without dense cross-node attention, scaling linearly with node-feature entries and graph edges (Section 3). We then introduce a synthetic graph prior that extends SCMs with diverse graph structures and relational dynamics, exposing the model to varied dependencies among topology, features, and labels (Section 4).

We evaluate Ephris on 51 node-classification datasets against 15 tuned GNNs and 6 graph foundation models (GFMs), including NodePFN and GraphPFN, under high-label (50/25/25) and low-label (10/10/80) train/validation/test splits. Across this broad evaluation, no prior GFM surpasses tuned GNNs overall; GraphPFN comes closest, ranking 4th and 10th by Elo in the two regimes. In contrast, Ephris ranks 1st in Elo, improvability, average rank, and accuracy (defined in Appendix A) under both regimes. It achieves this performance at the cost of a single GNN training run, while being more than 13× faster than GraphPFN. Together, these results show that dense attention is not necessary for strong graph ICL, advancing the performance-runtime frontier and making graph ICL a practical alternative to per-dataset training and tuning. Code and pretrained weights are available at link.

2 Background and Related Work

Graph in-context learning. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes and $E = |\mathcal{E}|$ edges, with adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N\times N}$, where $A_{ij} = 1$ if $(i,j) \in \mathcal{E}$ and 0 otherwise. Node features are collected in $\boldsymbol{X} \in \mathbb{R}^{N\times F}$, with each row an F-dimensional vector. Let $\boldsymbol{y} \in \{1,\dots,C\}^N$ denote node labels, where y_i is the label of node i and C the number of classes. Nodes are partitioned into disjoint training and test nodes, which we call context nodes $\mathcal{V}_{\text{ctx}}$ and query nodes $\mathcal{V}_{\text{qry}}$, respectively.

Following prior graph ICL models (Choi et al., 2026; Eremeev et al., 2026), we pretrain Ephris on synthetic graph tasks. Each task $\mathcal{D} = (\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{y}, \mathcal{V}_{\text{ctx}})$ is sampled from a prior Π, with $\mathcal{V}_{\text{qry}} = \mathcal{V} \setminus \mathcal{V}_{\text{ctx}}$. Given the graph structure, node features, and context labels, the model predicts unobserved query labels by minimizing their expected average negative log-likelihood:

$$\mathcal{L}_{\text{ICL}}(\boldsymbol{\theta}) = -\mathbb{E}_{\mathcal{D}\sim\Pi}\Big(\frac{1}{|\mathcal{V}_{\text{qry}}|}\sum_{i\in\mathcal{V}_{\text{qry}}} \log p_{\boldsymbol{\theta}}(y_i \mid i, \boldsymbol{A}, \boldsymbol{X}, \boldsymbol{y}_{\mathcal{V}_{\text{ctx}}})\Big). \tag{1}$$

This objective trains a single set of parameters across synthetic tasks, requiring the model to infer each task-specific prediction rule from its observed context. At inference, the pretrained model applies this capability to unseen graphs without updating $\boldsymbol{\theta}$. For Ephris, $p_{\boldsymbol{\theta}}$ is defined by the architecture in Section 3, while Π is defined by the synthetic task prior in Section 4.

Structural causal models. Recent TFMs use structural causal models (SCMs) to generate synthetic tasks with diverse feature-label relationships (Hollmann et al., 2025; QU et al., 2026). An SCM defines a directed acyclic graph (DAG) $\mathcal{G}_{\mathrm{SCM}}$ specifying dependencies over potentially vector-valued intermediate variables. For N samples, $\boldsymbol{U}^{(a)}$ collects the values of variable a, with one row per sample. Let $\mathrm{pa}(a)$ denote its parents. Variables are generated in topological order as

$$\boldsymbol{U}^{(a)} = \mathcal{T}_a(f_a(\{\boldsymbol{U}^{(b)}\}_{b\in\mathrm{pa}(a)})). \tag{2}$$

For root variables, $f_a(\varnothing)$ generates sampled noise. For other variables, f_a is drawn from function families such as multi-layer perceptrons (MLPs), tree ensembles, and linear or quadratic mappings, inducing diverse relationships among variables. The transform $\mathcal{T}_a$ applies operations such as normalization and noise injection. Features and labels are extracted from the variables and transformed as needed, for example by discretizing continuous values into categorical features or class labels.

Graph foundation models. GFMs aim to transfer pretrained models across graph datasets (Liu et al., 2024; Wang et al., 2024). Our work aligns with efforts to enable node classification across datasets with varying graph structures, feature and label spaces. These methods differ in which components they transfer and how they adapt to a new dataset. GraphAny (Zhao et al., 2024) transfers an aggregator over linear least-squares predictors fitted to each dataset. GVT (Lee et al., 2026) and Node4All (Lee & Yoo, 2026) transfer feature encoders while training lightweight downstream predictors. G2T-FM (Eremeev et al., 2025) augments node features with graph-derived features and applies a pretrained TFM for prediction. NodePFN (Choi et al., 2026) and GraphPFN (Eremeev et al., 2026) instead use observed node labels as context for graph ICL, avoiding dataset-specific parameter updates at all. Ephris follows this graph ICL approach, aiming to improve predictive performance and computational scalability. We compare against all six methods in our evaluation.

3 Ephris: A Scalable Graph In-context Learner

Recent in-context learning methods broadly follow two architectural paradigms: *bi-axial* architectures alternate dense attention across samples and features (Hollmann et al., 2025; Zhang et al., 2025; 2026a;b), while *compress-then-ICL* architectures compress each sample into a fixed-dimensional representation before performing ICL through attention across samples (QU et al., 2025; 2026). Ephris redesigns the latter architecture for graph-structured data through five stages:

$$(\underbrace{\boldsymbol{X}}_{N\times F}, \boldsymbol{y}_{\mathcal{V}_{\mathrm{ctx}}}) \xrightarrow{\text{① Token.}} \underbrace{\mathbf{H}^{(0)}}_{N\times F\times d} \xrightarrow{\text{② Ref.}} \underbrace{\mathbf{H}^{(L_{\mathrm{ref}})}}_{N\times F\times d} \xrightarrow{\text{③ Comp.}} \underbrace{\boldsymbol{Z}^{(0)}}_{N\times D} \xrightarrow{\text{④ ICL}} \underbrace{\boldsymbol{Z}^{(L_{\mathrm{ICL}})}}_{N\times D} \xrightarrow{\text{⑤ Head}} \underbrace{\widehat{\boldsymbol{P}}}_{N\times C}.$$

We first outline these stages, which are designed to support diverse graph tasks with shared parameters (§3.1). Then, we introduce the message-passing block $\mathrm{MP}_{\mathrm{ICL}}$, which is the core component of Ephris that is utilized in two different stages of the architecture (§3.2). Finally, we analyze computational complexity and empirically compare scalability with prior graph ICL models (§3.3). An overview is shown in Figure 2, with full implementation details in Appendix B.

3.1 Architecture Overview

① **Tokenization.** We first map each scalar node feature into a common d-dimensional token space. A shared MLP is used for all F features. We also incorporate context labels through learned class embeddings, allowing subsequent layers to use the observed label context; the class embeddings are added to an intermediate representation. The resulting tokens form $\mathbf{H}^{(0)}$.

② **Graph-aware token refinement.** Before compression, we apply L_{ref} blocks, each consisting of per-feature refinement to capture feature distributions across nodes, followed by per-node refinement to incorporate graph structure into each node's feature tokens.

Per-feature refinement. For each feature, we use the induced attention (Lee et al., 2019) to exchange information across nodes: K_F learned inducing tokens attend to the node tokens to *gather* a summary, then node tokens attend to it to receive the *broadcast*. For feature f in block ℓ:

$$\boldsymbol{J}_f^{(\ell)} = \mathrm{Gather}_F^{(\ell)}(\boldsymbol{H}_{:f}^{(\ell-1)}), \qquad \widetilde{\boldsymbol{H}}_{:f}^{(\ell)} = \mathrm{Broadcast}_F^{(\ell)}(\boldsymbol{H}_{:f}^{(\ell-1)}, \boldsymbol{J}_f^{(\ell)}),$$

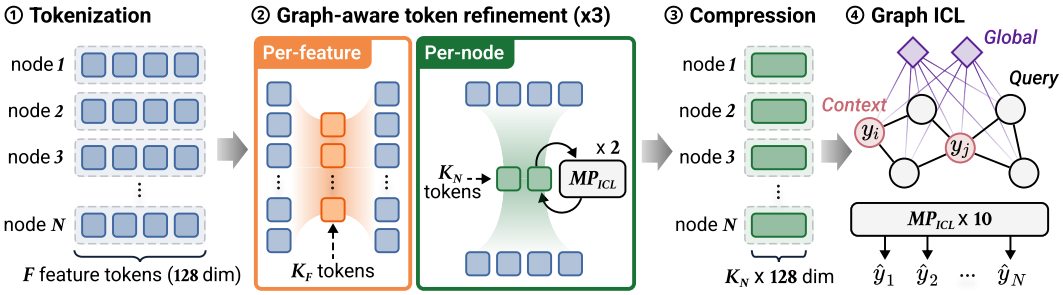


Figure 2: **Architecture overview of Ephris.** Ephris tokenizes each scalar node-feature value into a d-dimensional token, compresses feature tokens into fixed-size node representations while integrating distributional and graph context, and performs ICL through stacked message passing.

where $\boldsymbol{J}_f^{(\ell)}$ contains the K_F summary tokens gathered for feature f. Note that we gather from both context and query nodes, incorporating the full observed distribution, while most TFMs restrict gathering to context samples so that each query prediction is independent of other queries.

Per-node refinement. We then gather each node's feature tokens into a compact summary using K_N learned inducing tokens. Two message-passing blocks exchange information among these summaries, and broadcast returns the updates to the feature tokens:

$$\boldsymbol{I}_i^{(\ell)} = \mathrm{Gather}_N^{(\ell)}(\widetilde{\boldsymbol{H}}_{i:}^{(\ell)}), \quad \widehat{\mathbf{I}}^{(\ell)} = \mathrm{MP}_{\mathrm{ICL}}^{\circ 2}(\mathbf{I}^{(\ell)}, \boldsymbol{A}), \quad \boldsymbol{H}_{i:}^{(\ell)} = \mathrm{Broadcast}_N^{(\ell)}(\widetilde{\boldsymbol{H}}_{i:}^{(\ell)}, \widehat{\boldsymbol{I}}_i^{(\ell)}),$$

where $\boldsymbol{I}_i^{(\ell)}$ contains node i's summary tokens, and $\mathbf{I}^{(\ell)}$ collects them across nodes. The message-passing block includes concatenating each node's summary tokens into a vector before propagation and splitting the updated vector back into tokens before broadcasting (details in §3.2).

Alternation. Alternating per-feature and per-node refinement across L_{ref} blocks allows feature distributions and graph structure to jointly guide which information is retained during compression.

③ Compression. After refinement, an additional gather operation summarizes each node's features using K_N inducing tokens. Concatenating these tokens yields a representation of width $D = K_N d$, independent of feature count. The resulting matrix $\boldsymbol{Z}^{(0)}$ enters the ICL stage.

④ Graph in-context learning. The compressed node representations pass through L_{ICL} successive $\mathrm{MP}_{\mathrm{ICL}}$ blocks, replacing the dense attention across samples used in existing in-context learners:

$$\boldsymbol{Z}^{(L_{\mathrm{ICL}})} = \mathrm{MP}_{\mathrm{ICL}}^{\circ L_{\mathrm{ICL}}}(\boldsymbol{Z}^{(0)}, \boldsymbol{A}). \tag{3}$$

Through successive rounds of message passing, the model infers a task-specific predictive rule relating node features and graph connectivity to the observed context labels, progressively updating query-node representations for label prediction.

⑤ Prediction head. A shared head maps each query node's final representation to class logits. As done in previous work (QU et al., 2026; Hollmann et al., 2025), we use ten learned class embeddings and ten output logits. For $C \leq 10$, softmax is applied to the first C logits. For $C > 10$, error-correcting output codes (ECOC) (Dietterich & Bakiri, 1994) decompose the task into classification problems with at most ten classes. Each original class is scored by averaging the log probabilities assigned to its code entries, and softmax converts these scores into class probabilities.

3.2 Message Passing Block for ICL

The message passing block $\mathrm{MP}_{\mathrm{ICL}}$ is the central building block of Ephris. The block is designed to accommodate diverse graph structures and predictive relationships across tasks with a single set of shared parameters. Specifically, we introduce four complementary design choices below.

Dynamic neighborhood aggregation. Graph structure provides plausible interactions, but the relevance of each connection for prediction can vary across tasks, graphs, and nodes. We therefore use local attention (Shi et al., 2020) to dynamically weight neighboring messages based on their current representations. Let $\boldsymbol{h}_i \in \mathbb{R}^{d_h}$ denote the representation of node i, where d_h is the hidden

dimension, and let $\mathcal{N}(i)$ denote its neighborhood including i itself. We compute

$$\alpha_{ij} = \mathrm{softmax}_{j\in\mathcal{N}(i)}\left(d_h^{-1/2}\boldsymbol{q}_i^\top \boldsymbol{k}_j\right), \qquad \mathcal{M}_i(\boldsymbol{h}) = \textstyle\sum_{j\in\mathcal{N}(i)}\alpha_{ij}\boldsymbol{v}_j, \tag{4}$$

where $\boldsymbol{q}_i = \boldsymbol{W}_Q\boldsymbol{h}_i$, $\boldsymbol{k}_i = \boldsymbol{W}_K\boldsymbol{h}_i$, and $\boldsymbol{v}_i = \boldsymbol{W}_V\boldsymbol{h}_i$ are the query, key, and value projections. This lets the model adapt which neighbors to emphasize as their representations evolve across layers.

Neighborhood-aware attention. Neighborhood sizes also vary widely across nodes and graphs, ranging from only a few to thousands (Hu et al., 2020). With more neighbors competing within the same softmax, attention can become increasingly diffuse, a phenomenon known as attention dilution (Zhang et al., 2024). To address this, we extend TabICLv2's query-aware attention scaling (QU et al., 2026) to jointly account for neighborhood size:

$$\widetilde{\boldsymbol{q}}_i = \boldsymbol{q}_i \odot f_{\boldsymbol{\theta}}(\log|\mathcal{N}(i)|) \odot [1+\tanh g_{\boldsymbol{\theta}}(\boldsymbol{q}_i)], \qquad \alpha_{ij} = \mathrm{softmax}_{j\in\mathcal{N}(i)}\left(d_h^{-1/2}\widetilde{\boldsymbol{q}}_i^\top \boldsymbol{k}_j\right), \tag{5}$$

where $\odot$ denotes element-wise multiplication, and $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\theta}}$ are two-layer MLPs. This lets the model adapt attention sharpness jointly to the current representation and neighborhood size.

Local and global communication. Graph edges can be too sparse, noisy, or weakly aligned with the prediction task. To improve robustness in these graphs, we introduce K_G learned global nodes $\mathcal{V}_g$ that enable interactions beyond the observed edges. Each global node connects bidirectionally to every original node (Cai et al., 2023). For each node i, we augment its neighborhood as

$$\mathcal{N}^+(i) = \mathcal{N}(i)\cup\mathcal{V}_g, \qquad |\mathcal{N}^+(i)| = |\mathcal{N}(i)| + K_G, \tag{6}$$

and apply the same neighborhood-aware attention over $\mathcal{N}^+(i)$. The global nodes allow arbitrary nodes to exchange information within two message-passing layers while adding only $\mathcal{O}(NK_G)$ interactions, which scale linearly in N for fixed K_G.

Deep residual propagation. The interaction range needed to infer a prediction rule can vary across graphs and tasks. We therefore design $\mathrm{MP}_{\mathrm{ICL}}$ for deep stacking using residual connections:

$$\bar{\boldsymbol{h}}_i = \mathrm{LN}\big(\boldsymbol{h}_i + \mathcal{M}_i(\boldsymbol{h})\big), \qquad \mathrm{MP}_{\mathrm{ICL}}(\boldsymbol{h})_i = \mathrm{LN}\big(\bar{\boldsymbol{h}}_i + \mathrm{FFN}(\bar{\boldsymbol{h}}_i)\big), \tag{7}$$

where LN and FFN denote layer normalization and a position-wise feed-forward network, respectively. Stacking these blocks progressively expands each node's receptive field, allowing information to propagate over increasingly long graph distances. Together, these four design choices let $\mathrm{MP}_{\mathrm{ICL}}$ adapt what information to aggregate, where to communicate, and how far to propagate it, while retaining sparse, scalable computation.

3.3 Computational Complexity

Theoretical complexity. We analyze computational complexity under fixed hidden dimensions, inducing-token counts, network depths, and context ratio. Per-feature refinement costs $\mathcal{O}(NF)$, per-node refinement including message passing costs $\mathcal{O}(NF+E)$, and graph ICL costs $\mathcal{O}(N+E)$. Each prediction pass therefore requires $\mathcal{O}(NF+E)$ computation, scaling linearly with the number of node-feature entries and graph edges, rather than quadratically with the number of nodes or features as in prior graph ICL models (Table 1).

Table 1: Theoretical complexity.

Model	Computation
NodePFN	$\mathcal{O}(NF+N^2+E)$
GraphPFN	$\mathcal{O}(N^2F+NF^2+EF)$
Ephris	$\mathcal{O}(NF+E)$

Empirical scaling. We measure inference runtime on synthetic graphs ranging from 50K to 500K nodes, with 32 features, average degree 8, and a 50% context ratio. All methods run without ensembling on a single NVIDIA H200 GPU; runtime includes preprocessing and inference but excludes model loading. We quantify scaling by fitting $T(N) = c + aN^\beta$ over the measured range. As shown in Figure 3, Ephris scales nearly linearly ($\beta = 1.07$), whereas NodePFN ($\beta = 1.98$) and GraphPFN ($\beta = 1.90$) scale nearly quadratically. The resulting runtime gap widens rapidly with graph size.

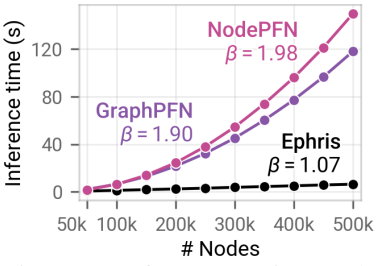


Figure 3: Inference runtime scaling; median over three seeds.

4 PRETRAINING WITH DIVERSE SYNTHETIC GRAPHS

We outline the synthetic graph generation pipeline that defines the pretraining prior Π in Equation (1), followed by the training curriculum and optimization procedure. Detailed descriptions of the prior and training setup are provided in Appendix C and Appendix D, respectively.

4.1 SYNTHETIC GRAPH PRIOR

Our pipeline has three stages: sampling a graph topology, generating features and labels conditioned on the sampled graph, and post-processing the features into diverse form of representations.

Graph sampling. Real-world graphs often exhibit a mixture of connectivity patterns. We capture this diversity by combining three graph-generation rules: *group-based* connectivity favors edges within or across sampled groups, as in stochastic block models (Holland et al., 1983); *degree-heterogeneous* connectivity samples endpoints according to node-wise weights, producing heterogeneous degrees and hubs as in Chung-Lu graphs (Chung & Lu, 2002); and *ordering-based* connectivity links nodes according to a sampled ordering, introducing route-like connection patterns. We sample mixture weights for each graph so that these patterns coexist in varying proportions.

Feature and label sampling. We generate features and labels conditioned on the sampled graph by incorporating graph propagation into the SCM. Inspired by GraphPFN (Eremeev et al., 2026), we extend Equation (2) as

$$\widetilde{\boldsymbol{U}}^{(a)} = f_a(\{\boldsymbol{U}^{(b)}\}_{b\in \mathrm{pa}(a)}), \qquad \boldsymbol{U}^{(a)} = \mathcal{T}_a(\mathrm{Propagate}_a(\widetilde{\boldsymbol{U}}^{(a)}, \boldsymbol{A})). \tag{8}$$

Here, $\widetilde{\boldsymbol{U}}^{(a)}$ collects variable a's values across nodes. Before applying $\mathcal{T}_a$, propagation updates these values by aggregating neighbor information using operators such as mean, max, or min. Graph structure thus shapes the intermediate variables from which features and labels are generated.

We also diversify the *relational dynamics* governing how graph structure influences these variables. For each variable selected for propagation, we sample one of three rules: *diffusion* repeatedly mixes node and neighbor states; *cascade* starts from a small set of active nodes and updates an inactive node only when agreement between its state and the aggregated states of active neighbors reaches a sampled threshold; and *degree-dependent mixing* adjusts the strength of neighbor influence based on node degree and neighbor agreement. These rules capture gradual, conditional, and degree-dependent effects, respectively. We further vary their behavior by sampling rule-specific mixing coefficients, propagation steps, and activation thresholds.

Post-processing. Finally, we diversify how generated features appear to the model. Real-world node features range from meaningful attributes to image or text embeddings whose information is distributed across dimensions (Yan et al., 2023). We therefore sample three representations: *tabular*, which preserves the generated features; *embedding*, which mixes all features through a linear or nonlinear transformation followed by standardization; and *hybrid*, which transforms only a subset.

4.2 TRAINING SCHEDULE AND OPTIMIZATION

Training curriculum. Generalizing across graph datasets requires exposure to diverse graph sizes and feature dimensions, but larger graphs and higher-dimensional features can increase pretraining costs signficantly. We therefore adopt a two-stage curriculum. Stage 1 develops the model's in-context prediction capability on relatively small graphs over 50,000 steps. Stage 2 then adapts the model to a broader range of graph sizes and feature dimensions over 10,000 additional steps. Each step uses 64 freshly generated graphs, yielding 3.84 million graphs in total.

Table 2: Two-stage training setup.

	Stage 1	Stage 2
# Node	512–1,024	128–16,384
# Feature	2–128	2–1,024
# Steps	50,000	10,000
Muon LR	8×10^{-4}	4×10^{-5}
AdamW LR	8×10^{-4}	10^{-5}

Optimization. We optimize eligible matrix-valued parameters with Muon (Jordan et al., 2024) and the remaining parameters with AdamW (Loshchilov & Hutter, 2017). Each stage uses linear warmup followed by cosine decay. In Stage 2, we use lower peak learning rates to preserve learned capabilities while adapting to larger graphs and higher-dimensional features.

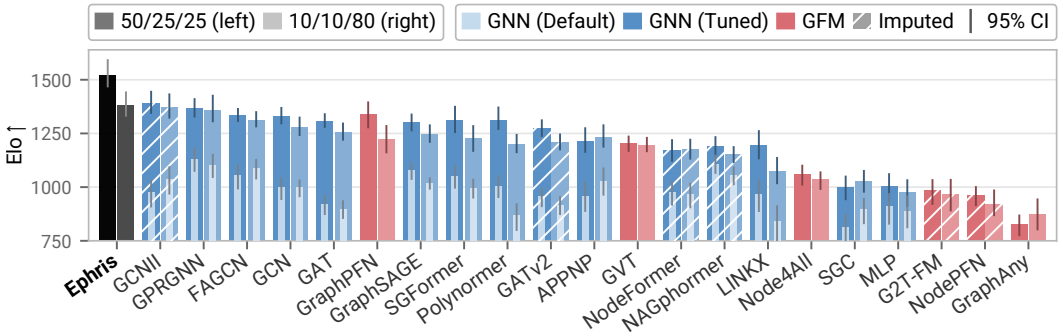


Figure 4: **Elo across label regimes.** Each bar nests a default GNN within its HP-tuned counterpart. Hatching marks OOM results imputed with GCN (Default); error bars show 95% confidence intervals. Methods are ordered by mean Elo across regimes. Ephris achieves the highest Elo in both.

5 Experiments

Model configuration. Our model has 50.16M parameters and uses 128-dimensional feature tokens. Three refinement blocks combine per-feature refinement with 128 inducing tokens and per-node refinement with four inducing tokens. Compression concatenates four tokens into a 512-dimensional node representation, which is then processed by ten $\mathrm{MP}_{\mathrm{ICL}}$ layers for graph ICL. Each $\mathrm{MP}_{\mathrm{ICL}}$ layer uses eight global nodes. Pretraining follows the two-stage curriculum on 3.84M synthetic graphs with no exposure to real-world datasets, requiring 16 GPU-days in our environment. Full model configurations and details of the experimental environment are provided in Appendix D.

Baselines. We compare Ephris against 21 baselines: 15 GNNs and six recent GFMs. The GNN baselines cover neighborhood aggregation (Kipf & Welling, 2017; Hamilton et al., 2017), attention-based aggregation (Veličković et al., 2018; Brody et al., 2021), decoupled feature transformation and propagation (Gasteiger et al., 2018; Wu et al., 2019), and architectures for deep propagation and heterophilous graphs (Chen et al., 2020; Chien et al., 2020; Bo et al., 2021). They also include scalable graph transformers (GTs) that capture graph-wide interactions through multi-hop representations, kernelized attention, or linear attention (Chen et al., 2022; Wu et al., 2022; Deng et al., 2024; Wu et al., 2023). The GFM baselines are GraphAny (Zhao et al., 2024), GVT (Lee et al., 2026) Node4All (Lee & Yoo, 2026), G2T-FM (Eremeev et al., 2025), NodePFN (Choi et al., 2026), and GraphPFN (Eremeev et al., 2026), introduced in Section 2.

Evaluation protocol. We evaluate all methods on 51 node-classification datasets spanning six application domains. The datasets range from 183 to 568,795 nodes, 12 to 8,710 features, and 2 to 70 classes, with adjusted label homophily from -0.30 to 0.94. Each dataset is evaluated under high-label and low-label regimes, using train/validation/test proportions of 50/25/25 and 10/10/80, respectively, with five splits per regime. Following TabArena (Erickson et al., 2026), each GNN is evaluated under two tuning budgets: *default*, using default hyperparameters, and *tuned*, selecting the best of 200 configurations by validation performance. Supervised baselines use validation data for early stopping and hyperparameter selection. Ephris uses only training labels as context and predicts in a single inference pass. Dataset and baseline details, hyperparameter search spaces for each GNN, and adaptation procedures for each GFM are provided in Appendix E.

Metrics. We evaluate predictive performance and adaptation cost. Performance is summarized by Elo, improvability, average rank, and average accuracy. Elo aggregates split-level pairwise wins, draws, and losses, while improvability measures the normalized gap to the best-performing model (Erickson et al., 2026). The first three metrics use AUROC for binary tasks and accuracy for multiclass tasks; average accuracy uses accuracy throughout. Adaptation cost covers a single training run for default GNNs, the full hyperparameter search for tuned GNNs, each GFM's adaptation procedure, and inference for Ephris. We report this cost as runtime overhead, defined as the log-aggregated slowdown relative to the fastest method. Full definitions are in Appendix A.

5.1 Main Results

Due to space constraints, we show selected plots here. Complete results, including tables, bar plots, Pareto plots, subgroup radar charts, and pairwise win matrices, are provided in Appendix H.

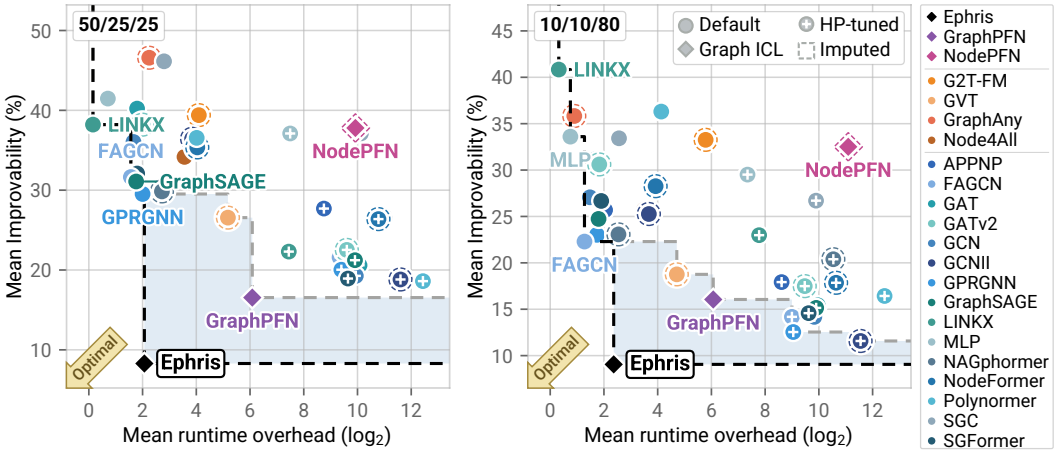


Figure 5: **Performance-runtime trade-offs.** Mean improvability versus runtime overhead under high-label (left) and low-label (right) regimes; lower is better on both axes. Black and gray dashed lines show Pareto frontiers with and without Ephris, with the improvement shaded blue.

Predictive performance. A broader evaluation revealed a substantial gap between existing GFMs and extensively tuned GNNs. In particular, GCNII (Chen et al., 2020) and GPRGNN (Chien et al., 2020) demonstrated leading results under both label regimes. GraphPFN (Eremeev et al., 2026) emerged as the strongest existing GFM, but did not outperform either overall. Therefore, the advantages reported for existing GFMs do not hold for broader datasets and stronger baselines.

Ephris closes this gap, ranking first in Elo, improvability, average rank, and average accuracy in both label regimes. Against tuned GCNII, the strongest baseline, it achieves pairwise win rates of 67% and 54% in the high- and low-label regimes, respectively. These gains are particularly notable because supervised baselines use validation labels for early stopping and hyperparameter selection, whereas Ephris uses none. Against GraphPFN, the strongest prior GFM, it wins 71% of comparisons in both regimes. Ephris achieves these results through sparse message passing, demonstrating that dense all-pairs attention is not required for strong predictive performance in graph ICL.

Performance-runtime trade-off. A central promise of GFMs is to reduce adaptation costs by reusing pretrained knowledge. GVT (Lee et al., 2026) and GraphPFN (Eremeev et al., 2026) partially fulfill this promise: both improve on default GNNs at far lower cost than 200-trial HP tuning, reaching the Pareto frontier in both label regimes (Figure 5). However, their adaptation costs remain substantial in practice. Relative to a single GCN training run, GVT requires approximately 9.4-11.5× the runtime and GraphPFN 21.4-24.1× across the two label regimes.

Ephris also closes this efficiency gap. Across the two label regimes, its inference costs 1.33–1.84× a single GCN training run and is 13.1–16.1× faster than GraphPFN, demonstrating the efficiency of replacing dense attention with message passing for graph ICL. Together with its leading predictive performance, this pushes the performance-runtime Pareto frontier forward, with Ephris remaining the only GFM on the frontier across all metrics (Figure 10).

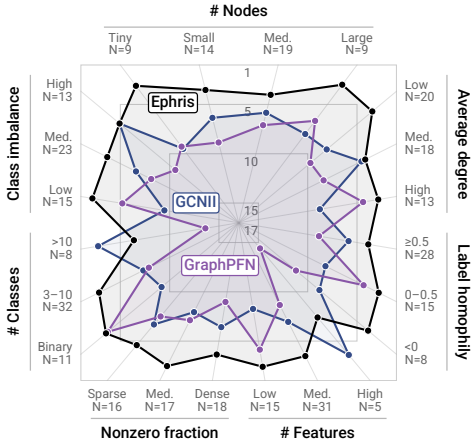


Figure 6: **Subgroup performance.** Average ranks under the high-label regime.

Subgroup Analysis. We assess whether the overall gains of Ephris persist across graph and task characteristics. Figure 6 reports average ranks across subgroups under the high-label regime, with complete results and subgroup definitions in Appendix H. Ephris outperforms GraphPFN in nearly all subgroups under both label regimes, indicating broad improvements of graph ICL rather than gains confined to particular datasets. Against the runner-up, tuned GCNII, it leads in most subgroups but falls behind on datasets with more than 5,000 features or ten classes. Both exceed the ranges seen during pretraining, suggesting that broader pretraining coverage may help close these gaps.

5.2 Beyond Ordinary Graphs and Random Splits

Having established strong performance across diverse graph datasets, we further test whether Ephris generalizes beyond graphs and random train-test splits, considering hypergraph transfer and distribution shift (Appendix F). On ten AllSet hypergraph datasets (Chien et al., 2021), Ephris ranks first or second on eight using simple clique or incidence representations, suggesting that it also provide a promising alternative to per-dataset training and HP tuning of supervised hypergraph models. On the Graph Out-of-Distribution (GOOD) benchmark (Gui et al., 2022), Ephris ranks first or second in 13 of 20 settings against baselines including methods specifically designed for OOD generalization. However, its performance drops substantially in several remaining settings, indicating that robust generalization under distribution shift remains an important direction for future work.

5.3 Ablation Studies

We ablate the key design choices of Ephris to understand their individual contributions. Since full pretraining requires approximately 16 GPU-days, we conduct these analyses at 10% of the Stage 1 scale, using 5,000 updates over 320,000 synthetic graphs and evaluating on the same 51 datasets as in the main experiments. Due to space constraints, we summarize the key findings here; full results, configurations, and detailed analyses are provided in Appendix G.

Disentangling architecture and prior. Ephris introduces improvements along two axes: the architecture and the synthetic graph prior. To disentangle their contributions, we conduct a 3×3 factorial study crossing the architectures and priors of Ephris, GraphPFN (Eremeev et al., 2026) and NodePFN (Choi et al., 2026), training each architecture with every prior. As shown in Figure 7, our architecture achieves the highest mean accuracy under every prior, while our prior performs best across all architectures. These consistent gains show that both contribute independently, rather than only through their specific combination.

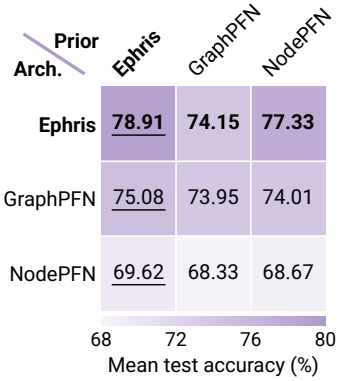


Figure 7: **Architecture–prior factorial study.** Bold: column best, underline: row best.

Component ablations. We ablate key components of the architecture and pretraining prior. For the architecture, restricting per-feature gathering to context nodes, removing token updates after message passing, using static aggregation in $\mathrm{MP}_{\mathrm{ICL}}$, or removing neighborhood-size-aware scaling or global nodes all reduce both mean accuracy and win rate. For the prior, removing propagation from the SCM causes the largest degradation, reducing mean accuracy by roughly 20%, suggesting that jointly modeling topology, features, and labels is critical. Restricting the prior to any single relational dynamic also underperforms their mixture, while removing embedding-like post-processing also reduces performance. Together, these results support the design choices of both the architecture and pretraining prior.

6 Conclusion

We introduced Ephris, a scalable graph in-context learner demonstrating that strong ICL can be achieved through message passing over the observed graph without dense cross-node attention. Pretrained on a synthetic prior spanning diverse graph structures, relational dynamics, and feature representations, Ephris transfers effectively to unseen graphs without per-dataset training or tuning. Across 51 node-classification datasets, Ephris outperforms 15 extensively tuned GNNs and six recent GFMs across all aggregate metrics, including Elo, improvability, average rank, and accuracy, under both high- and low-label regimes. It achieves this performance at roughly the cost of a single GNN training run while running 13.1–16.1$\times$ faster than GraphPFN, pushing the performance–runtime Pareto frontier forward. Overall, the strong performance and fast inference of Ephris across diverse datasets suggest graph ICL as a practical alternative to per-dataset GNN training and tuning.

Limitations and Future Work. Our ablation studies use reduced-budget proxy pretraining, so their conclusions may not fully transfer to full-scale training. As shown in Section 5.2, improving robustness to distribution shifts under non-IID grouped and temporal splits remains important for real-world applicability. Our scope is also limited to node classification; extending scalable graph ICL to link prediction and graph-level tasks is an important step toward general-purpose GFMs.

AI USE STATEMENT

We used generative AI tools during code development and manuscript editing, primarily for code suggestions and improvements to clarity and grammar. The authors reviewed and tested all AI-assisted code and independently reviewed and revised all AI-assisted text. Generative AI was not used to generate experimental results or make research decisions. We take full responsibility for the final content of this work, including all AI-assisted content.

ETHICS STATEMENT

This work does not involve human subjects or the collection of personal data. Our model is pretrained entirely on synthetic graph tasks and evaluated on open node-classification benchmarks. We follow the licenses and intended research use of the datasets and methods used in our experiments, and encourage responsible use of the proposed model in downstream applications.

REPRODUCIBILITY STATEMENT

We provide the model specifications, experimental settings, and evaluation procedures needed to reproduce our results throughout the appendix. Appendix B describes the architecture and its constituent operations, while Appendix C details graph sampling, relational dynamics, and feature post-processing in the synthetic pretraining prior. Appendix D documents the model configuration, software and hardware environment, task-size sampling, two-stage curriculum, optimization settings, and inference procedure.

For evaluation, Appendices A and E specify the metrics, datasets, train/validation/test splits, baseline configurations, hyperparameter search spaces, and adaptation procedures for prior GFMs. Complete aggregate results and supplementary comparisons are provided in Appendix H, and Appendix G reports the configurations and results of our ablation studies.

The pretrained checkpoint and inference code are available in our anonymous repository. These artifacts allow researchers to evaluate Ephris directly without repeating the full pretraining process. Together, the released artifacts and documented procedures support independent verification of our results and comparisons with future methods.

REFERENCES

Gleb Bazhenov, Oleg Platonov, and Liudmila Prokhorenkova. Graphland: Evaluating graph machine learning models on diverse industrial data. *Advances in Neural Information Processing Systems*, 38, 2026.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 3950–3957, 2021.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between mpnn and graph transformer. In *International conference on machine learning*, pp. 3408–3430. PMLR, 2023.

Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. *arXiv preprint arXiv:2206.04910*, 2022.

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

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. *arXiv preprint arXiv:2106.13264*, 2021.

Minyong Cho, Minho Jeong, Dooho Lee, Jinmo Lee, and Jaemin Yoo. Causilo technical report, 2026. URL https://arxiv.org/abs/2609.22866.

Jeongwhan Choi, Jongwoo Kim, Woosung Kang, and Noseong Park. Learning posterior predictive distributions for node classification from synthetic graph priors. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FmxRzlu0rT.

Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002.

Chenhui Deng, Zichao Yue, and Zhiru Zhang. Polynormer: Polynomial-expressive graph transformer in linear time. In *International Conference on Learning Representations*, volume 2024, pp. 18323–18348, 2024.

Thomas G Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1994.

Dmitry Eremeev, Gleb Bazhenov, Oleg Platonov, Artem Babenko, and Liudmila Prokhorenkova. Turning tabular foundation models into graph foundation models. *arXiv preprint arXiv:2508.20906*, 2025.

Dmitry Eremeev, Oleg Platonov, Gleb Bazhenov, Artem Babenko, and Liudmila Prokhorenkova. GraphPFN: A prior-data fitted graph foundation model. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=340Ep8sfJ3.

Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data. *Advances in Neural Information Processing Systems*, 38, 2026.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Mihir Manium, Shi Bin Hoo, Magnus Bühler, Anurag Garg, et al. Tabpfn-3: Technical report. *arXiv preprint arXiv:2605.13986*, 2026.

Shurui Gui, Xiner Li, Limei Wang, and Shuiwang Ji. Good: A graph out-of-distribution benchmark. *Advances in Neural Information Processing Systems*, 35:2059–2073, 2022.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Karim K Ben Hicham, Jan G Rittig, Martin Grohe, and Alexander Mitsos. Tabular foundation models for in-context prediction of molecular properties. *arXiv preprint arXiv:2604.16123*, 2026.

Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

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*, 637(8045):319–326, 2025.

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. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL `https://kellerjordan.github.io/posts/muon/`.

Benjamin Jäger, Nick Erickson, Léo Grinsztajn, Felix Birkel, Klemens Flöge, Oscar Key, Kürşat Kaya, Jonas Kübler, Adèle Frankel, Tobias Schröder, Anurag Garg, Jan Hendrik Metzen, David Salinas, Simon Bing, Kristina Collins, Tuana Çelik, Vahid Balazadeh, Lydia Sidhoum, Tomás Pereda, Brendan Roof, Andrej Tschalzev, Siyuan Guo, Philipp Singer, Lennart Purucker, Jake Robertson, Marie Salmon, Philipp Jund, Jerry Chen, Diana Kriuchkova, Arthur Cahu, Eliott Kalfon, Adrian Hayler, Georg Grab, Vitor Monteiro, Lilly Wehrhahn, Dominik Safaric, Clara Cornu, Alan Arazi, Rylee Grace, Simone Alessi, Mihir Manium, Bernhard Schölkopf, Yann LeCun, Madelon Hulsebos, Sauraj Gambhir, Noah Hollmann, and Frank Hutter. Tabpfn-3.5: Technical report, 2026. URL `https://arxiv.org/abs/2609.17895`.

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

Dooho Lee and Jaemin Yoo. Node4all: Learning node representation beyond datasets. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2461–2472, 2026.

Dooho Lee, Myeong Kong, Minho Jeong, and Jaemin Yoo. View space: Learning representation across arbitrary graphs. In *Forty-third International Conference on Machine Learning*, 2026. URL `https://openreview.net/forum?id=9klc62KROH`.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pp. 3744–3753. PMLR, 2019.

Jundong Li, Xia Hu, Jiliang Tang, and Huan Liu. Unsupervised streaming feature selection in social media. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pp. 1041–1050, 2015.

Huidong Liang, Haitz Sáez de Ocáriz Borde, Baskaran Sripathmanathan, Michael Bronstein, and Xiaowen Dong. Towards quantifying long-range interactions in graph machine learning: a large graph dataset and a measurement. In *International Conference on Learning Representations*, volume 2026, pp. 34655–34688, 2026.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in neural information processing systems*, 34:20887–20902, 2021.

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. In *International conference on learning representations*, volume 2024, pp. 20188–20210, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic gnns are strong baselines: Reassessing gnns for node classification. *Advances in Neural Information Processing Systems*, 37:97650–97669, 2024.

Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=KSugKcbNf9.

Judea Pearl et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19(2):3, 2000.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

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? *arXiv preprint arXiv:2302.11640*, 2023.

Oleg Platonov, Gleb Bazhenov, Dmitry Eremeev, and Liudmila Prokhorenkova. A fair evaluation of graph foundation models for node property prediction. In *Workshop on Graph Foundation Models: A New Era for Graph Machine Learning*, 2026. URL https://openreview.net/forum?id=u6ssKNnrki.

Jingang QU, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=0VvD1PmNzM.

Jingang QU, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICLv2: A better, faster, scalable, and open tabular foundation model. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=SxsyLjIfWB.

Benedek Rozemberczki and Rik Sarkar. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pp. 1325–1334, 2020.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Haoyu Peter Wang, Shikun Liu, Rongzhe Wei, and Pan Li. Generalization principles for inference over text-attributed graphs with large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Zehong Wang, Zheyuan Zhang, Tianyi Ma, Nitesh V Chawla, Chuxu Zhang, and Yanfang Ye. Towards graph foundation models: Learning generalities across graphs via task-trees. *arXiv preprint arXiv:2412.16441*, 2024.

Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

Dan-Ni Wu, Joey Jen, Erickson Fajiculay, Min-Fen Hsu, Ming-Chu Chang, Jen-Chen Yeh, Karen Sargsyan, Juozas Kupcinskas, Jurgita Skieceviciene, Ruta Steponaitiene, et al. Panmetai-a high performance tabular foundation model for accurate pancreatic cancer diagnosis via nmr metabolomics. *Nature Communications*, 17(1):1595, 2026.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. Pmlr, 2019.

Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in neural information processing systems*, 35:27387–27401, 2022.

Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. Sgformer: Simplifying and empowering transformers for large-graph representations. *Advances in neural information processing systems*, 36:64753–64773, 2023.

Hao Yan, Chaozhuo Li, Ruosong Long, Chao Yan, Jianan Zhao, Wenwen Zhuang, Jun Yin, Peiyan Zhang, Weihao Han, Hao Sun, et al. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. *Advances in Neural Information Processing Systems*, 36:17238–17264, 2023.

Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, Sourav S Bhowmick, et al. Scaling attributed network embedding to massive graphs. *Proceedings of the VLDB Endowment*, 14(1):37–49, 2020.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pp. 40–48. PMLR, 2016.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, Jiansheng Li, Jiayun Wu, Lang Mo, Li Mao, Mingchao Hao, et al. Limix: Unleashing structured-data modeling capability for generalist intelligence. *arXiv preprint arXiv:2509.03505*, 2025.

Xingxuan Zhang, Gang Ren, Hao Yuan, Hao Zou, Hongze Tan, Hui Wang, Jianhao Song, Jiansheng Li, Jiayao Zhang, Jinghan Zhang, et al. Limix-2: A contextual mechanism network towards general structured-data intelligence. *arXiv preprint arXiv:2609.17488*, 2026a.

Xiyuan Zhang, Danielle Maddix Robinson, Junming Yin, Nick Erickson, Abdul Fatir Ansari, Boran Han, Shuai Zhang, Leman Akoglu, Christos Faloutsos, Michael Mahoney, et al. Mitra: Mixed synthetic priors for enhancing tabular foundation models. *Advances in neural information processing systems*, 38:15795–15840, 2026b.

Xuechen Zhang, Xiangyu Chang, Mingchen Li, Amit Roy-Chowdhury, Jiasi Chen, and Samet Oymak. Selective attention: Enhancing transformer through principled context control. *Advances in Neural Information Processing Systems*, 37:11061–11086, 2024.

Jianan Zhao, Hesham Mostafa, Michael Galkin, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. Graphany: A foundation model for node classification on any graph. *arXiv preprint arXiv:2405.20445*, 29, 2024.

Jianan Zhao, Zhaocheng Zhu, Mikhail Galkin, Hesham Mostafa, Michael Bronstein, and Jian Tang. Fully-inductive node classification on arbitrary graphs. In *International Conference on Learning Representations*, volume 2025, pp. 23968–23986, 2025.

A Evaluation Metrics

We evaluate the high-label and low-label regimes separately, each using 51 datasets and five splits per dataset. Default and tuned GNNs are treated as separate participants, yielding 37 configurations. Unless otherwise specified, the predictive score $s_{m,d,r}$ for method m on dataset d and split r is AUROC for binary classification and accuracy for multiclass classification. Comparisons are computed within each dataset-split, rather than from scores averaged across splits.

Elo. For every pair of methods on the same dataset-split, we assign an outcome of 1 for a win, 0.5 for an exact tie, and 0 for a loss. We fit a Bradley-Terry model to these outcomes, weighting each split by $1/5$ so that datasets contribute equally. The resulting ratings are expressed on the Elo scale and shifted so that default GCN has a rating of 1000. Higher ratings indicate stronger pairwise performance; the reference rating only fixes the origin of the scale.

Average rank. We rank all configurations within each dataset-split, assigning tied methods the average of their occupied ranks. We then average ranks over the five splits within each dataset and over the 51 datasets. Lower values indicate better average standing. In particular, we do not average predictive scores across splits before computing ranks.

Improvability (Erickson et al., 2026). Let $s_{*,d,r} = \max_j s_{j,d,r}$ denote the best score among all configurations on dataset d and split r. The improvability of method m is

$$I_{m,d,r} = 100\,\frac{s_{*,d,r} - s_{m,d,r}}{1 - s_{m,d,r}}. \tag{9}$$

This measures the fraction of the remaining gap to a perfect score that could be closed by matching the best evaluated configuration. For example, scores of 0.80 and 0.90 for the method and the best configuration yield an improvability of 50%. For AUROC, the denominator is the gap to perfect AUROC, not a misclassification rate. The best configuration has zero improvability; cases with a denominator numerically close to zero are also assigned zero. We average split-level values within each dataset, then average across datasets. Lower mean improvability is better.

Pairwise win rate. For methods a and b, we compute

$$W_{a,b} = \frac{100}{D}\sum_{d=1}^{D}\frac{1}{S}\sum_{r=1}^{S}\left[\mathbf{1}\{s_{a,d,r} > s_{b,d,r}\} + \frac{1}{2}\mathbf{1}\{s_{a,d,r} = s_{b,d,r}\}\right], \tag{10}$$

where $D = 51$ and $S = 5$. Thus, ties count as half a win and each dataset receives equal weight. With five splits for every dataset, this equals the tie-adjusted win rate over all 255 comparisons. The overall leaderboard win rate averages pairwise win rates over all other configurations.

Average accuracy. We additionally report accuracy averaged across splits and datasets, using accuracy for both binary and multiclass tasks. This metric therefore differs from Elo, average rank, and improvability in its treatment of binary classification, for which those metrics use AUROC.

Runtime. Runtime measures the wall-clock cost of adapting a method to a new dataset. It includes training and the full hyperparameter search for tuned supervised methods, or the adaptation procedure for GFMs. Let $t_{m,d}$ denote the runtime of method m on dataset d. We define the relative runtime overhead and its aggregate as

$$\rho_{m,d} = \frac{t_{m,d}}{\min_j t_{j,d}}, \qquad \bar{\rho}_m = \exp\left(\frac{1}{D}\sum_{d=1}^{D}\log\rho_{m,d}\right). \tag{11}$$

The geometric mean weights datasets equally and summarizes multiplicative runtime differences. Pairwise speedups are aggregated in the same way.

B Architecture Details

We describe the five stages of the architecture in Section 3. The model uses feature-token width $d = 128$, $K_F = 128$ per-feature inducing tokens, and $K_N = 4$ per-node inducing tokens, giving a compressed width of $D = K_N d = 512$. Each message-passing stage maintains $K_G = 8$ global nodes, which are updated throughout its successive blocks.

Tokenization. Numeric features are standardized, binary features are encoded as zero or one, and categorical features are ordinally encoded. Each scalar value is then clipped to $[-8, 8]$. For a clipped value $\bar{x}$, we construct an 11-dimensional basis:

$$\boldsymbol{b}(\bar{x}) = [\bar{x},\ \tanh(\bar{x}),\ \operatorname{sign}(\bar{x})\log(1+|\bar{x}|),\ r_1(\bar{x}), \ldots, r_8(\bar{x})], \quad r_q(\bar{x}) = \exp(-(\bar{x}-c_q)^2/(2w^2)).$$

The eight radial basis functions have centers evenly spaced over $[-3, 3]$ and bandwidth w. A shared MLP with gating maps this basis into a token of width d.

To provide information about the other features of the same node, the tokenizer also computes a node summary. It takes the elementwise mean and root-mean-square of the basis vectors separately for numeric, binary, and categorical features, then concatenates these statistics. A learned projection of the summary is added to each scalar representation through a gated residual connection, followed by RMS normalization. This introduces node-level information without assigning positions to features.

Let $\boldsymbol{s}_i$ denote this node summary. The initial tokens are

$$\boldsymbol{h}_{if}^{(0)} = \mathrm{CellEncode}(\bar{x}_{if}, \boldsymbol{s}_i) + \boldsymbol{e}_{\mathrm{type}}(t_f) + \mathbf{1}[i \in \mathcal{V}_{\mathrm{ctx}}]\boldsymbol{e}_{\mathrm{feat}}(y_i).$$

Here, $\mathrm{CellEncode}$ includes the scalar projection and node-summary update, and t_f identifies the feature type. The context-label embedding is shared across all feature tokens of the same node. Query nodes receive a zero label embedding. The resulting tokens form the initial tensor $\mathbf{H}^{(0)}$.

Graph-aware refinement. We apply $L_{\mathrm{ref}} = 3$ refinement blocks. Each block first exchanges distributional information across nodes for each feature, then incorporates structural information into each node's feature tokens.

Per-feature refinement. For each feature, K_F learned inducing tokens attend to its tokens across all nodes to gather a summary. The node tokens then attend to that summary to receive the broadcast:

$$\boldsymbol{J}_f^{(\ell)} = \mathrm{Gather}_F^{(\ell)}(\boldsymbol{H}_{:f}^{(\ell-1)}), \qquad \widetilde{\boldsymbol{H}}_{:f}^{(\ell)} = \mathrm{Broadcast}_F^{(\ell)}(\boldsymbol{H}_{:f}^{(\ell-1)}, \boldsymbol{J}_f^{(\ell)}).$$

Here, $\boldsymbol{J}_f^{(\ell)}$ contains the summary tokens for feature f in block ℓ. Both context and query nodes participate in gathering and broadcasting, while only context labels are available. Features are processed independently with shared module parameters. The updated tokens carry information about each feature's distribution across the graph.

Per-node refinement. We next gather across each node's features using K_N learned inducing tokens. Two successive $\mathrm{MP}_{\mathrm{ICL}}$ blocks exchange information among these node summaries before the updates are broadcast back to the feature tokens:

$$\boldsymbol{I}_i^{(\ell)} = \mathrm{Gather}_N^{(\ell)}(\widetilde{\boldsymbol{H}}_{i:}^{(\ell)}), \quad \widehat{\mathbf{I}}^{(\ell)} = \mathrm{MP}_{\mathrm{ICL},\ell}^{\circ 2}(\mathbf{I}^{(\ell)}, \boldsymbol{A}), \quad \boldsymbol{H}_{i:}^{(\ell)} = \mathrm{Broadcast}_N^{(\ell)}(\widetilde{\boldsymbol{H}}_{i:}^{(\ell)}, \widehat{\boldsymbol{I}}_i^{(\ell)}).$$

Here, $\boldsymbol{I}_i^{(\ell)}$ contains node i's summary tokens, and $\mathbf{I}^{(\ell)}$ collects them across nodes. The message-passing notation includes concatenating each node's tokens before propagation and splitting the updated vector back into tokens afterward. Each two-block stage initializes its own global nodes and updates them alongside the original nodes through both blocks. Alternating per-feature and per-node refinement allows distributional and structural information to guide the representations used for compression.

Compression. A per-node gather produces K_N summary tokens, which are concatenated into a representation of width D, independent of the feature count. Before graph ICL, context-label embeddings are added to these representations, followed by layer normalization:

$$\boldsymbol{z}_i^{(0)} = \mathrm{LN}(\mathrm{Concat}(\boldsymbol{I}_i^{\mathrm{comp}}) + \mathbf{1}[i \in \mathcal{V}_{\mathrm{ctx}}]\boldsymbol{e}_{\mathrm{ICL}}(y_i)).$$

Here, $\boldsymbol{I}_i^{\mathrm{comp}}$ contains the tokens produced by the compression gather. The ICL label embeddings have width D and use a separate table from the tokenizer's width-d embeddings. Both tables are trainable and initialized with orthogonal class embeddings. The resulting matrix $\boldsymbol{Z}^{(0)}$ is the label-conditioned input to graph ICL.

Graph in-context learning. The node representations pass through an independent stage of $L_{\mathrm{ICL}} = 10$ successive message-passing blocks:

$$\boldsymbol{Z}^{(L_{\mathrm{ICL}})} = \mathrm{MP}_{\mathrm{ICL}}^{\circ L_{\mathrm{ICL}}}(\boldsymbol{Z}^{(0)}, \boldsymbol{A}).$$

This stage initializes its own eight global nodes and updates them throughout the ten blocks. Only the representations of the original nodes are retained at the output. Together with the six blocks used during refinement, the model contains 16 message-passing blocks.

Prediction head. A shared head applies layer normalization and a linear projection to produce ten logits per node:

$$\boldsymbol{o}_i = \mathrm{Linear}_{D \to 10}(\mathrm{LN}(\boldsymbol{z}_i^{(L_{\mathrm{ICL}})})).$$

For $C \leq 10$, softmax is applied to the first C logits. For $C > 10$, ECOC decomposes the task into classification problems with at most ten classes, each evaluated using the same pretrained model. Each original class is scored by averaging the log probabilities assigned to its code entries. Softmax then converts these scores into probabilities over the original classes, with the query-node predictions used for evaluation.

C Synthetic Graph Prior Details

This section details the three components of the synthetic prior in Section 4: graph sampling, relational dynamics, and feature post-processing. Together, they vary graph topology, how topology influences features and labels, and how generated features are represented. Task-size sampling and the training setup are described in Appendix D.

C.1 Graph Sampling

Each graph combines group-based, degree-heterogeneous, and ordering-based connectivity. We sample their relative contributions once per graph and select a rule for each proposed edge. Topology is generated before features and labels, so neither determines which nodes are connected.

Node sampling weights. All rules sample source nodes from a shared distribution:

$$p_i = \frac{\exp(\sigma g_i)}{\sum_{j=1}^{N} \exp(\sigma g_j)}, \qquad g_i \sim \mathcal{N}(0,1), \qquad \sigma \sim \mathcal{U}(0.3, 1.6).$$

Larger values of σ concentrate source probability on fewer nodes, increasing out-degree heterogeneity. The rules differ in how they choose a destination.

Group-based connectivity. We sample three group structures. First, nodes are randomly partitioned into 2 to 8 approximately balanced coarse groups. Each coarse group is divided into 1 to 4 approximately balanced fine groups. A separate partition assigns nodes to 2 to 4 additional groups.

The coarse and fine rules sample a destination uniformly from the source node's group. For the additional partition, a fair coin sampled once per graph selects within-group or cross-group connectivity. In the cross-group case, each proposal selects another group uniformly, then samples a destination uniformly within it. These rules produce both within-group concentration and cross-group connections.

Degree-heterogeneous connectivity. The destination is sampled independently from the same distribution as the source. An ordered pair (i, j) is therefore proposed with probability $p_i p_j$. Nodes with larger weights are more likely to appear at either endpoint, producing heterogeneous degrees and hubs without prescribing an exact degree sequence.

Ordering-based connectivity. We sample a random node ordering and connect each sampled source to the node a fixed number of positions ahead, wrapping around at the end. The offset is shared across the graph and sampled uniformly from 1 to $\min(N, 32) - 1$. Writing the ordering as π, the destination of source i is

$$j = \pi((\pi^{-1}(i) + \Delta) \bmod N).$$

Table 3: **Relational-dynamics settings.** Discrete choices and continuous intervals are sampled uniformly. Rule-specific parameters apply only to the indicated operations.

Setting	Choices or range
Rule	Diffusion, cascade, degree-dependent mixing
Aggregation	Mean, max, min, softmax
Direction	Incoming, outgoing, bidirected
Mixing coefficient α	$[0.15, 0.85]$
Steps T (diffusion and cascade)	$\{2, 3, 4, 5, 6\}$
Seed fraction s (cascade)	$[0.08, 0.28]$
Threshold θ (cascade)	$[0.20, 0.55]$
Temperature τ (softmax)	$[0.25, 2.0]$

Here, positions are indexed from zero, and Δ is the sampled offset. This introduces a common transition pattern among the proposed edges. Because only sampled edges are retained and mixed with other rules, the resulting graph need not form a complete cycle.

Mixture weights. The implementation uses six entries: three group-based rules, one ordering-based rule, and two entries with the same degree-heterogeneous rule. For each entry k, we independently sample $u_k \sim \mathcal{U}(0, 1)$ and normalize the clipped weights:

$$w_k = \frac{\max(u_k, 0.05)}{\sum_{r=1}^{6} \max(u_r, 0.05)}.$$

Each edge proposal samples a source from p, an entry from w, and a destination using that entry's rule. The two degree-heterogeneous entries contribute jointly to the same connectivity pattern.

Edge construction. Given target average out-degree d, we draw $M = \min(N(N-1), \mathrm{round}(Nd))$ proposals with replacement, then remove self-edges and duplicates. To vary reciprocity, we sample $\rho \sim \mathcal{U}(0, 1)$ and independently add the reverse of each remaining edge with probability ρ. After deduplication, any excess over M is removed by uniformly selecting M edges. The realized graph can therefore contain fewer than M edges, and isolated nodes are allowed. This directed graph is used to generate features and labels. The model's subsequent symmetrization and self-loop insertion are separate from this sampling procedure.

C.2 Relational Dynamics

Relational dynamics determine how the sampled graph influences the intermediate variables used to generate features and labels. We detail the propagation operator in Equation (8), using the SCM construction described in Equation (2).

Selecting propagation settings. Let L_{SCM} be the number of retained SCM variables needed to generate the outputs. We sample $q_{\mathrm{prop}} \sim \mathcal{U}(0, 1)$ and select $\mathrm{round}(q_{\mathrm{prop}} L_{\mathrm{SCM}})$ variables uniformly without replacement. Each selected variable independently samples a propagation rule, aggregation operator, and direction. Propagation is the identity for unselected variables.

The direction is chosen uniformly from incoming, outgoing, and bidirected propagation. These use the original edges, reversed edges, or their deduplicated union, respectively. The remaining settings are summarized in Table 3. All sampled settings remain fixed throughout propagation for a given variable.

Neighborhood aggregation. For a selected SCM variable a, let $\boldsymbol{u}_i^{(0)}$ denote its value at node i before propagation, corresponding to row i of $\widetilde{\boldsymbol{U}}^{(a)}$. We omit the variable index below and use t to index propagation steps. Let $\mathcal{N}(i)$ contain the neighbors that send information to node i under the sampled direction. The neighborhood aggregate is

$$\boldsymbol{m}_i^{(t)} = \mathrm{Agg}(\{\boldsymbol{u}_j^{(t)} : j \in \mathcal{N}(i)\}).$$

Mean, max, and min operate coordinatewise. Softmax aggregation weights neighbors by cosine similarity to the receiving node:

$$\boldsymbol{m}_i^{(t)} = \sum_{j \in \mathcal{N}(i)} \frac{\exp(\cos(\boldsymbol{u}_i^{(t)}, \boldsymbol{u}_j^{(t)})/\tau)}{\sum_{k \in \mathcal{N}(i)} \exp(\cos(\boldsymbol{u}_i^{(t)}, \boldsymbol{u}_k^{(t)})/\tau)} \boldsymbol{u}_j^{(t)}.$$

For all rules, nodes without eligible sending neighbors remain unchanged.

Diffusion. Diffusion repeatedly mixes each node's current value with its neighborhood aggregate:

$$\boldsymbol{u}_i^{(t+1)} = (1-\alpha)\boldsymbol{u}_i^{(t)} + \alpha \boldsymbol{m}_i^{(t)}, \qquad t = 0, \ldots, T-1.$$

Repeated updates allow neighbor influence to accumulate and propagate beyond immediate neighbors.

Cascade. Cascade starts from a small set of active nodes and updates other nodes only when their agreement with active neighbors reaches a threshold. The initial active set contains $\max(1, \mathrm{round}(sN))$ nodes sampled uniformly without replacement.

At step t, let $\mathcal{A}_t$ be the active set and let $\boldsymbol{m}_{i,\mathrm{act}}^{(t)}$ aggregate only neighbors in $\mathcal{N}(i) \cap \mathcal{A}_t$. The newly activated nodes are

$$\mathcal{B}_t = \{i \notin \mathcal{A}_t : \mathcal{N}(i) \cap \mathcal{A}_t \neq \varnothing,\ (1 + \cos(\boldsymbol{u}_i^{(t)}, \boldsymbol{m}_{i,\mathrm{act}}^{(t)}))/2 \geq \theta\}.$$

Only these nodes are updated:

$$\boldsymbol{u}_i^{(t+1)} = \begin{cases} (1-\alpha)\boldsymbol{u}_i^{(t)} + \alpha \boldsymbol{m}_{i,\mathrm{act}}^{(t)}, & i \in \mathcal{B}_t, \\ \boldsymbol{u}_i^{(t)}, & \text{otherwise}, \end{cases} \qquad \mathcal{A}_{t+1} = \mathcal{A}_t \cup \mathcal{B}_t.$$

The process runs for T steps. Once activated, a node retains its updated value and can influence further activations. Each non-seed node is therefore updated at most once, making propagation conditional rather than continuous.

Degree-dependent mixing. This rule applies a single update whose strength depends on node degree and neighbor agreement. We rank nodes by receiving degree under the sampled direction and assign buckets $b_i \in \{0, 1, 2\}$ to the lower, middle, and upper thirds. Using the aggregate of the initial values, we compute

$$\delta_i = (1 - \cos(\boldsymbol{u}_i^{(0)}, \boldsymbol{m}_i^{(0)}))/2, \qquad \gamma_i = 1 + \tfrac{1}{2} b_i + \delta_i, \qquad \boldsymbol{u}_i^{(1)} = \boldsymbol{u}_i^{(0)} + \alpha\gamma_i(\boldsymbol{m}_i^{(0)} - \boldsymbol{u}_i^{(0)}).$$

Higher degree and greater disagreement both increase the update strength. The effective coefficient $\alpha\gamma_i$ can exceed one, so the update may extrapolate beyond the neighborhood aggregate. This produces node-specific responses to the same propagation mechanism.

C.3 Post-processing

We choose uniformly among tabular, embedding, and hybrid representations. These modes vary how feature information is distributed across dimensions while leaving labels unchanged. The mode is selected before feature generation to determine the required source width. Let F denote the final feature count.

Tabular representation. The generator produces F source columns, which are retained without additional mixing. Numerical and categorical attributes therefore remain individually accessible.

Embedding representation. We transform a compact source matrix into an embedding of width F_{emb}, where $F_{\mathrm{emb}} = F$ in this mode. The source width q is sampled log-uniformly between $\min(4, F_{\mathrm{emb}})$ and $\min(32, F_{\mathrm{emb}})$ and rounded to the nearest integer.

The source columns are first standardized. With probability $1/2$, we apply a random linear map followed by GELU and standardize the result again:

$$\boldsymbol{V} = \mathrm{Std}(\boldsymbol{S}), \qquad \boldsymbol{V} \leftarrow \mathrm{Std}(\mathrm{GELU}(\boldsymbol{V}\boldsymbol{W} + \boldsymbol{1}\boldsymbol{b}^\top)).$$

Table 4: **Embedding variation.** Sampling distributions for projection, noise, and row scaling. Lognormal scaling multiplies row i by $\exp(\sigma_r z_i)$, with $z_i \sim \mathcal{N}(0,1)$.

Parameter	Distribution
Spectral decay η	$\mathcal{U}(0.5, 2.0)$
Nuisance rank ν	Uniform over $\{0, \dots, \min(8, q)\}$
Nuisance amplitude λ	$\mathcal{U}(0, 0.5)$
Noise scale ϵ	$\mathrm{LogUniform}(0.01, 0.35)$
Lognormal scale σ_r	$\mathcal{U}(0.05, 0.5)$

Here, $W_{ab} \sim \mathcal{N}(0, 1/q)$ and $b_a \sim \mathcal{N}(0,1)$. The operator Std centers each column and divides by its population standard deviation, bounded below by 10^{-6}.

A random projection then distributes the source information across the embedding dimensions:

$$\boldsymbol{B}_0 = \mathrm{Std}(\boldsymbol{V}\,\mathrm{diag}(\boldsymbol{s})\boldsymbol{Q}^\top), \qquad s_k = \frac{k^{-\eta}}{\sqrt{q^{-1}\sum_{r=1}^{q} r^{-2\eta}}}.$$

The columns of $\boldsymbol{Q}$ form an orthonormal basis obtained by QR decomposition of a Gaussian matrix. The sampled exponent η controls how strongly the source directions contribute.

We add independent low-rank nuisance variation and Gaussian noise:

$$\boldsymbol{B}_1 = \boldsymbol{B}_0 + \lambda\,\mathrm{Std}(\boldsymbol{R}\boldsymbol{Q}_\nu^\top) + \epsilon\boldsymbol{G}.$$

Here, $\boldsymbol{R}$ has ν independent standard Gaussian columns, $\boldsymbol{Q}_\nu$ is an independent orthonormal projection, and $\boldsymbol{G}$ contains independent standard Gaussian entries. The nuisance term is omitted when $\nu = 0$. Finally, with equal probability, we either normalize each row to norm $\sqrt{F_{\mathrm{emb}}}$ or multiply it by an independent lognormal factor. There is no further column standardization.

Hybrid representation. We combine an embedding with untransformed source columns. The number of retained columns m is sampled uniformly from

$$\{1, \dots, \min(16, F-1, \max(1, \lfloor F/4 \rfloor))\}.$$

We set $F_{\mathrm{emb}} = F - m$ and sample the source width q as above. Of the $q + m$ generated columns, the first q are transformed into an embedding and the remaining m are retained. Concatenating the two parts and randomly permuting their columns produces the final F-dimensional representation.

D Implementation Details

This section describes the experimental environment and implementation details of Ephris, covering its architecture configuration, pretraining setup, and inference protocol.

D.1 Experiment Environment

Experiments were conducted on NVIDIA H200 GPUs. The server was equipped with two Intel Xeon Platinum 8462Y+ CPUs (64 physical cores in total) and approximately 2 TiB of RAM, running Ubuntu 22.04.5 LTS. Pretraining used Python 3.11.9 and PyTorch 2.11.0 with CUDA 13.0.

D.2 Architecture Setup

We instantiate the architecture described in Section 3 using the configuration in Table 5. Each scalar node-feature value is embedded into a 128-dimensional token. Per-feature refinement uses 128 inducing tokens with eight attention heads to model each feature across nodes. Per-node refinement then applies three refinement blocks, each gathering feature tokens into four node-inducing tokens and processing them with two $\mathrm{MP}_{\mathrm{ICL}}$ layers. After the final per-feature refinement and gather operation, the four tokens of each node are concatenated into a 512-dimensional representation. Ten additional $\mathrm{MP}_{\mathrm{ICL}}$ layers perform graph ICL, followed by a prediction head supporting up to ten classes.

Table 5: **Architecture configuration of Ephris.**

Component	Setting	Value
Tokenization	Feature-token dimension	128
Per-feature refinement	Inducing tokens	128
	Attention heads	8
Per-node refinement	Refinement blocks	3
	Node-inducing tokens per node	4
	$\mathrm{MP}_{\mathrm{ICL}}$ layers per block	2
	Concatenated node dimension	512
$\mathrm{MP}_{\mathrm{ICL}}$	Global nodes	8
	Attention heads	8
	Attention-scaling MLP hidden dimension	64
Graph ICL	$\mathrm{MP}_{\mathrm{ICL}}$ layers	10
Prediction	Maximum output classes	10
General	FFN expansion factor	4
	Dropout	0

Table 6: **Two-stage pretraining configuration.** Ranges specify sampling bounds before integer conversion. Stage 2 restricts the feature-count and target-degree supports using memory budgets.

Component	Setting	Stage 1	Stage 2
Task distribution	Updates	50,000	10,000
	Graphs per update	64	64
	Node-count bounds	512 to 1,024	128 to 16,384
	Feature-count bounds	2 to 128	2 to 1,024
	Requested class count	2 to 10	2 to 10
	Context fraction	Uniform on $[0.3, 0.9]$	0.8
Graph sampling	Target average out-degree bounds	1.5 to 500	1.5 to 256
	Node-feature budget B_{NF}	None	2^{21}
	Directed-edge budget B_E	None	2^{19}
Optimization	Muon peak learning rate	8×10^{-4}	4×10^{-5}
	AdamW peak learning rate	8×10^{-4}	10^{-5}
	LR decay endpoint (both groups)	4×10^{-5}	10^{-5}
	Warmup updates	1,000	500

D.3 Pretraining Setup

Two-stage curriculum. We pretrain Ephris on the synthetic node-classification tasks described in Section 4. Stage 1 uses moderately sized graphs with up to 128 features. Stage 2 broadens the distribution to include smaller and larger graphs, with sampling bounds of 128 to 16,384 nodes and 2 to 1,024 features. Both stages share the SCM, relational-dynamics, and feature-observation settings.

Stage 2 starts from the final Stage 1 model weights, with freshly initialized optimizer and scheduler states. Each update uses 64 newly generated tasks. The full curriculum comprises 50,000 Stage 1 updates and 10,000 Stage 2 updates, totaling 3.84M training tasks. Table 6 summarizes the settings.

Task generation. We sample graph sizes, feature counts, class counts, and target average out-degrees within the stage-specific ranges in Table 6. Stage 2 aims to expose the model to both wide feature sets and large graphs despite limited memory: wide features remain available on smaller graphs, while larger graphs are paired with fewer features. We implement this trade-off through a node-feature budget $B_{NF} = 2^{21}$. After sampling the node count N, we sample the feature count up to the smaller of 1,024 and B_{NF}/N, rounded down. Similarly, a directed-edge budget $B_E = 2^{19}$ restricts the target average out-degree to at most the smaller of 256 and B_E/N, allowing denser small graphs while keeping larger graphs sparse.

Training context. For each generated task, we select $\lfloor rN \rfloor$ context nodes uniformly without replacement and reveal their labels to the model. Stage 1 samples the context fraction r uniformly from $[0.3, 0.9]$, while Stage 2 fixes $r = 0.8$ to stabilize training during adaptation to larger tasks. A high context fraction reduces the loss fluctuations associated with sparse supervision, helping preserve the capabilities learned in Stage 1. The model receives all node features and graph edges together with the context labels, and the classification loss is computed on the remaining query nodes. This trains the model to infer query labels from the observed context within each graph.

D.4 Optimization

We optimize eligible matrix parameters with Muon (Jordan et al., 2024) and the remaining parameters with AdamW (Loshchilov & Hutter, 2017). Muon uses momentum 0.95, Nesterov updates, and five Newton-Schulz iterations to approximately orthogonalize each update. For an $a \times b$ parameter matrix, the resulting update is scaled by $0.2\sqrt{\max(a, b)}$ before applying the scheduled learning rate. The peak learning rates reported in Table 6 are specified before this matrix-dependent scaling. AdamW uses $(\beta_1, \beta_2) = (0.9, 0.95)$ and $\epsilon = 10^{-8}$.

Each stage begins with linear warmup from zero to the respective peak learning rates, followed by cosine decay toward the same absolute endpoint for both parameter groups. In Stage 2, the AdamW peak learning rate equals this endpoint, so it remains constant at 10^{-5} after warmup. Over the same period, the Muon learning rate decays from its peak of 4×10^{-5} to 10^{-5}.

D.5 Inference

We use the final Stage 2 checkpoint for all datasets without dataset-specific gradient updates. As in pretraining, the model receives all node features and graph edges, while labels are revealed only for context nodes. Feature preprocessing is fitted on the available node features.

We do not use test-time permutation ensembling, which combines predictions across feature and class permutations in many recent TFMs (QU et al., 2026; Grinsztajn et al., 2026; Cho et al., 2026). Our architecture is invariant to feature permutations, so changing feature order does not affect predictions. Class-label permutations, however, can change predictions even after the outputs are mapped back to the original labels. We use a single deterministic feature and class permutation and leave the potential benefits of class-permutation ensembling to future work.

For datasets with more than ten classes, we apply the ECOC procedure described in Section 3 with fixed redundancy four. A dataset with C classes uses $4\lceil \log_{10} C \rceil$ code columns, each defining a classification problem with at most ten outputs. All code problems are evaluated using the same pretrained model weights and preprocessed features. To recover predictions over the original classes, we average the log probabilities assigned to each class's code symbols across columns, then normalize the resulting scores over the original classes.

E Evaluation Details

This section provides additional details of our evaluation. We describe the datasets and baselines, the training and hyperparameter-tuning protocols for supervised GNNs, and the adaptation procedures of GFMs, including their design and our implementation.

E.1 Datasets

Our 51 datasets are selected to cover a broad range of node-classification settings rather than a narrow family of commonly used benchmarks. They span six application domains and vary substantially in scale, dimensionality, label space, connectivity, and homophily, ranging from 183 to 568,795 nodes, 12 to 8,710 input features, and 2 to 70 classes, with average degrees from 2.30 to 88.28 and adjusted label homophily from -0.30 to 0.94. This diversity allows us to test whether conclusions persist across substantially different graph and task characteristics. The full dataset list, together with domains, key statistics, and sources, is provided in Table 7.

E.2 GNN Training and Hyperparameter Tuning

We extensively tune all 15 supervised GNNs to establish strong per-dataset baselines. When the original work specifies a hyperparameter search space, we follow it; when only selected configurations are reported, we construct a search space that includes the reported choices and varies the corresponding hyperparameters. For conventional GNNs, we additionally tune choices often omitted from default implementations but known to substantially affect node-classification performance, including residual connections, layer normalization, pre-transformation, and dropout where applicable (Luo et al., 2024). Our goal is to avoid comparisons against weak default configurations and provide a stringent test of pretrained methods.

For each method and dataset split, we sample 200 configurations from the resulting search space and select the final configuration solely by validation performance. The complete baseline list, shared training settings, method-specific defaults, and search spaces are reported in Table 8.

E.3 GFM Adaptation Protocols

Applying a pretrained GFM to a new graph does not always mean inference alone. Depending on the method, the original implementation may fit a new predictor, select among pretrained representations, or choose dataset-specific preprocessing and inference settings. In our evaluation, we preserve these target-time procedures while keeping all pretrained backbones fixed. We count any computation and validation required by these procedures as part of adaptation.

GraphAny. GraphAny (Zhao et al., 2025) combines several LinearGNN prediction channels using a pretrained inductive-attention model. On each new graph, its original inference pipeline first recomputes the analytical LinearGNN predictors from the available training labels and then passes their predictions to the fixed pretrained model. We follow the same procedure: the analytical predictors are recomputed for every dataset, while the pretrained model is never updated. Nodes are processed in batches of 100,000. No dataset-specific hyperparameter selection is performed.

GVT. GVT (Lee et al., 2026) transfers a recurrent graph encoder whose depth can be chosen separately for each target dataset. The original implementation freezes the encoder, produces node representations at depths 1–8, trains a lightweight predictor on each representation, and selects the depth using validation performance. We reproduce this procedure directly. For every depth, we train a two-layer prediction head with hidden width 128, ReLU activation, and no dropout using Adam with learning rate 0.005 and zero weight decay. Training runs for up to 2,500 epochs with patience 200, and we restore the checkpoint with the lowest validation CE or BCE, breaking ties by training loss. Thus, both predictor fitting and depth selection contribute to GVT's per-dataset adaptation.

Node4All. Node4All (Lee & Yoo, 2026) transfers a fixed pretrained encoder and uses its node representations as input to a downstream predictor. Following its original evaluation protocol, we keep the encoder frozen and fit a two-layer prediction head of hidden width 512 separately on each target dataset. The full graph is encoded once, with features processed in chunks under a budget of 1,000,000 feature elements when necessary. The head uses Adam with learning rate 0.001 and zero weight decay for up to 2,500 epochs with patience 200, and validation loss selects the checkpoint. Unlike GVT, there is no encoder-depth search; target-specific adaptation consists of fitting and selecting the prediction head.

G2T-FM. G2T-FM (Eremeev et al., 2025) augments node features with graph-derived information and presents the resulting representation to a pretrained tabular foundation model. Its released code supports target-specific optimization, but we use the inference-only configuration in our evaluation so that the pretrained model remains fixed. All training nodes are provided as labeled context, from which the model directly predicts the remaining nodes. We use a single canonical configuration with inference batch size 1, seqlenred=1024, and PEARL batch size 8. Because this configuration is fixed across datasets, no validation-based model or hyperparameter selection is required.

GraphPFN. GraphPFN (Eremeev et al., 2026) supports graph-aware ICL: in its inference mode, training nodes are supplied as labeled context and the pretrained model predicts the remaining nodes without gradient updates. The released interface additionally exposes feature preprocessing, most notably optional PCA for high-dimensional inputs. We therefore keep the model fixed and evaluate the two PCA configurations considered in our evaluation, selecting between them for each target dataset. Each configuration uses the full graph, all training nodes as context, and a requested ensem-

ble size of 10, which the implementation increases automatically when required. Thus, GraphPFN requires no model fitting in our evaluation, but its preprocessing choice is dataset-specific.

NodePFN. NodePFN (Choi et al., 2026) also performs graph ICL without updating its pretrained parameters. Its original evaluation pipeline, however, specifies preprocessing and inference settings separately for each dataset, including dimensionality reduction, the retained dimensionality, and feature smoothing. Applying NodePFN to a new dataset therefore requires choosing these settings even though the model itself is frozen. To capture this adaptation cost, we construct a search space that subsumes the dataset-specific configurations reported by the original implementation and sample up to 200 configurations from it. The search varies ensemble size over $1, 4, 8, 16, 32$, dimensionality reduction between none and truncated SVD with $10, 15, 20, 25, 50$ components, and the exposed smoothing choices. We select the configuration using validation performance, then evaluate with all training nodes as context and the remaining nodes as queries in batches of 32.

Use of validation data. These procedures differ importantly in how much target-specific selection they require. GVT uses validation data to select both predictor checkpoints and recurrent depth; Node4All uses it for predictor checkpoint selection; and GraphPFN and NodePFN use it to resolve dataset-specific preprocessing or inference choices in our evaluation. GraphAny and our inference-only G2T-FM configuration require no such selection. Accordingly, a frozen backbone does not necessarily imply validation-free adaptation: several GFMs still use validation data indirectly to choose how the pretrained model is applied to each new dataset. In contrast, Ephris uses a single fixed inference procedure across all datasets: it receives only the training labels as context, performs no parameter updates or dataset-specific configuration search, and never uses validation labels.

Table 7: **Overview of the 51 node-classification datasets.** We report each dataset's application domain and key characteristics: number of nodes (N), features (F), classes (C), average degree ($\bar{d}$), and adjusted label homophily (h_{adj}). Citations refer to the original dataset or benchmark sources.

Dataset	**Domain**	N	F	C	$\bar{d}$	h_{adj}	**Citation**
actor	Web	7,600	932	5	7.02	0.003	(Pei et al., 2020)
amazon_computer	Commerce	13,752	767	10	35.76	0.682	(Shchur et al., 2018)
amazon_photo	Commerce	7,650	745	8	31.13	0.785	(Shchur et al., 2018)
amazon_ratings	Commerce	24,492	300	5	7.60	0.140	(Platonov et al., 2023)
amherst41	Social	2,235	1,193	2	81.39	0.060	(Lim et al., 2021)
artnet-exp	Social	50,405	75	2	11.12	0.155	(Bazhenov et al., 2026)
blogcatalog	Social	5,196	8,189	6	66.11	0.272	(Li et al., 2015)
chameleon	Web	890	2,325	5	19.90	0.030	(Platonov et al., 2023)
citation_citeseer	Scholarly	4,230	602	6	2.52	0.938	(Bojchevski & Günnemann, 2017)
citeseer	Scholarly	3,327	3,703	6	2.74	0.671	(Yang et al., 2016)
city-reviews	Commerce	148,801	204	2	15.66	0.591	(Bazhenov et al., 2026)
coauthor_cs	Scholarly	18,333	6,805	15	8.93	0.785	(Shchur et al., 2018)
coauthor_physics	Scholarly	34,493	8,415	5	14.38	0.872	(Shchur et al., 2018)
cora	Scholarly	2,708	1,433	7	3.90	0.771	(Yang et al., 2016)
cora_ml	Scholarly	2,995	2,879	7	5.45	0.749	(Bojchevski & Günnemann, 2017)
cornell	Web	183	1,703	5	3.03	-0.220	(Pei et al., 2020)
cornell5	Social	18,660	4,735	2	84.76	0.091	(Lim et al., 2021)
dblp	Scholarly	17,716	1,639	4	5.97	0.679	(Bojchevski & Günnemann, 2017)
deezer	Social	28,281	128	2	6.56	0.030	(Rozemberczki & Sarkar, 2020)
elliptic_bitcoin	Finance	203,769	165	2	2.30	0.516	(Weber et al., 2019)
facebook_large	Social	22,470	4,714	4	15.20	0.821	(Rozemberczki et al., 2021)
flickr	Social	89,250	500	7	10.08	0.094	(Zeng et al., 2019)
full_cora	Scholarly	19,793	8,710	70	6.41	0.556	(Bojchevski & Günnemann, 2017)
genius	Social	421,961	12	2	4.37	-0.053	(Lim et al., 2021)
johnshopkins55	Social	5,180	2,406	2	72.04	0.097	(Lim et al., 2021)
la	Transport	240,587	37	10	2.84	0.618	(Liang et al., 2026)
lastfm_asia	Social	7,624	7,842	18	7.29	0.856	(Rozemberczki & Sarkar, 2020)
london	Transport	568,795	37	10	2.66	0.626	(Liang et al., 2026)
ogbn_arxiv	Scholarly	169,343	128	40	13.67	0.588	(Hu et al., 2020)
paris	Transport	114,127	37	10	3.20	0.570	(Liang et al., 2026)
penn94	Social	41,554	4,814	2	65.56	0.021	(Lim et al., 2021)
pubmed	Scholarly	19,717	500	3	4.50	0.686	(Yang et al., 2016)
reed98	Social	962	745	2	39.11	0.022	(Lim et al., 2021)
shanghai	Transport	183,917	37	10	2.85	0.613	(Liang et al., 2026)
squirrel	Web	2,223	2,089	5	42.28	0.009	(Platonov et al., 2023)
tag_bookchild	Commerce	76,875	3,072	24	30.24	0.265	(Wang et al., 2025)
tag_bookhis	Commerce	41,551	3,072	12	12.11	0.519	(Wang et al., 2025)
tag_citeseer	Scholarly	3,186	3,072	6	2.65	0.729	(Wang et al., 2025)
tag_cora	Scholarly	2,708	3,072	7	3.90	0.771	(Wang et al., 2025)
tag_cornell	Web	191	3,072	5	2.87	-0.225	(Wang et al., 2025)
tag_pubmed	Scholarly	19,717	3,072	3	4.50	0.686	(Wang et al., 2025)
tag_sportsfit	Commerce	173,055	3,072	13	17.45	0.851	(Wang et al., 2025)
tag_texas	Web	187	3,072	4	2.99	-0.294	(Wang et al., 2025)
tag_washington	Web	229	3,072	5	3.19	-0.194	(Wang et al., 2025)
tag_wikics	Web	11,701	3,072	10	36.85	0.579	(Wang et al., 2025)
tag_wisconsin	Web	265	3,072	5	3.46	-0.169	(Wang et al., 2025)
texas	Web	183	1,703	4	3.05	-0.298	(Pei et al., 2020)
tolokers-2	Social	11,758	19	2	88.28	0.093	(Bazhenov et al., 2026)
wiki	Web	2,405	4,973	17	9.64	0.564	(Yang et al., 2020)
wiki_cs	Web	11,701	300	10	36.85	0.579	(Mernyei & Cangea, 2020)
wisconsin	Web	251	1,703	5	3.59	-0.173	(Pei et al., 2020)

Table 8: **Default configurations and hyperparameter search spaces for 15 supervised GNNs.** We sample 200 configurations from the listed candidate values. d denotes hidden dimension and L the number of layers. Unless otherwise specified, methods use the shared training configuration and search space in the first row.

Method	Ref.	Default	Search space
Shared training	–	Adam; lr $= 10^{-2}$; wd $= 5\times10^{-4}$; 2,500 epochs; patience $= 100$	lr $\in \{10^{-3}, 5\times10^{-3}, 10^{-2}\}$; wd $\in \{0, 5\times10^{-5}, 5\times10^{-4}\}$; patience $\in \{20, 100\}$
APPNP	(Gasteiger et al., 2018)	$d = 64$; dropout $= .5$; $K = 10$; $\alpha = .1$; prop. dropout $= 0$	$d \in \{64, 128, 256, 512\}$; dropout $\in \{0, .3, .5, .7\}$; $K \in \{5, 10, 20\}$; $\alpha \in \{.1, .2, .5\}$; prop. dropout $\in \{0, .3, .5\}$
FAGCN	(Bo et al., 2021)	$d = 32$; dropout $= .5$; $L = 2$; $\epsilon = .3$; edge dropout $= .5$	$d \in \{32, 64, 128\}$; feature/edge dropout $\in \{0, .3, .5, .7\}$; $L \in \{1, 2, 3, 4\}$; $\epsilon \in \{.1, .2, .3, .5\}$
GAT	(Veličković et al., 2018)	$d = 64$; heads $= 1$; dropout $= .5$; attn. dropout $= .2$; $L = 2$; norm=None; residual=False; pre-transform=False	$d \in \{64, 128, 256, 512\}$; heads $\in \{1, 4\}$; dropout $\in \{0, .3, .5, .7\}$; attn. dropout $\in \{0, .2\}$; $L \in \{1, 2, 4, 8\}$; norm $\in \{$`none`, `layernorm`$\}$; residual/pre-transform $\in \{\mathrm{F}, \mathrm{T}\}$
GATv2	(Brody et al., 2021)	$d = 64$; heads $= 1$; dropout $= .5$; attn. dropout $= .2$; $L = 2$; norm=None; residual=False; pre-transform=False	$d \in \{64, 128, 256, 512\}$; heads $\in \{1, 4\}$; dropout $\in \{0, .3, .5, .7\}$; attn. dropout $\in \{0, .2\}$; $L \in \{1, 2, 4, 8\}$; norm $\in \{$`none`, `layernorm`$\}$; residual/pre-transform $\in \{\mathrm{F}, \mathrm{T}\}$
GCN	(Kipf & Welling, 2017)	$d = 64$; dropout $= .5$; $L = 2$; norm=None; residual=False; pre-transform=False	$d \in \{64, 128, 256, 512\}$; dropout $\in \{0, .3, .5, .7\}$; $L \in \{1, 2, 4, 8\}$; norm $\in \{$`none`, `layernorm`$\}$; residual/pre-transform $\in \{\mathrm{F}, \mathrm{T}\}$
GraphSAGE	(Hamilton et al., 2017)	$d = 64$; dropout $= .5$; $L = 2$; norm=None; residual=False; pre-transform=False	$d \in \{64, 128, 256, 512\}$; dropout $\in \{0, .3, .5, .7\}$; $L \in \{1, 2, 4, 8\}$; norm $\in \{$`none`, `layernorm`$\}$; residual/pre-transform $\in \{\mathrm{F}, \mathrm{T}\}$
GCNII	(Chen et al., 2020)	$d = 64$; dropout $= .6$; $L = 16$; $\alpha = .1$; $\theta = .5$; shared weights; norm=None; residual=False; pre-transform=False	$d \in \{64, 128, 256, 512\}$; dropout $\in \{0, .3, .5, .7\}$; $L \in \{4, 8, 16, 32\}$; $\alpha \in \{.1, .2, .5\}$; $\theta \in \{.5, 1, 1.5\}$; norm/residual/pre-transform choices
GPRGNN	(Chien et al., 2020)	$d = 64$; feature dropout $= .5$; prop. dropout $= 0$; $K = 10$; $\alpha = .1$; init.=PPR	$d \in \{64, 128, 256, 512\}$; both dropouts $\in \{0, .3, .5, .7\}$; $K \in \{5, 10, 15, 20\}$; $\alpha \in \{.1, .2, .5, .9\}$; init. $\in \{$`ppr`, `sgc`, `random`$\}$
LINKX	(Lim et al., 2021)	$d = 32$; dropout $= .5$; $L = 2$; adjacency layers $= 1$; feature layers $= 1$	$d \in \{32, 64, 128\}$; dropout $\in \{0, .3, .5, .7\}$; $L \in \{1, 2, 3\}$; adjacency/feature layers $\in \{1, 2\}$
MLP	–	$d = 64$; dropout $= .5$; $L = 2$; norm=None; identity skip=False	$d \in \{64, 128, 256, 512\}$; dropout $\in \{0, .3, .5, .7\}$; $L \in \{1, 2, 3, 4\}$; norm $\in \{$`none`, `layernorm`$\}$; identity skip $\in \{\mathrm{F}, \mathrm{T}\}$
NAGphormer	(Chen et al., 2022)	AdamW; $d = 512$; hops $= 7$; $L = 1$; heads $= 8$; dropout $= .1$; PE dim. $= 15$; lr $= 10^{-3}$; wd $= 10^{-5}$	$d \in \{128, 256, 512\}$; hops $\in \{2, \ldots, 20\}$; $L \in \{1, \ldots, 5\}$; dropout $\in \{.1, .3, .5\}$; lr $\in \{10^{-4}, 10^{-3}, 5\times10^{-3}\}$; wd $\in \{10^{-5}, 10^{-4}, 5\times10^{-4}\}$
NodeFormer	(Wu et al., 2022)	$d = 32$; $L = 2$; heads $= 4$; dropout $= 0$; random features $= 30$; Gumbel samples $= 10$; link weight $= 1$; bias order $= 2$; transform=sigmoid; lr $= 10^{-3}$; wd $= 5\times10^{-3}$	$d \in \{32, 64, 128\}$; $L \in \{2, 3\}$; heads $\in \{1, 2, 4\}$; dropout $\in \{0, .3, .5\}$; random features $\in \{30, 50\}$; samples $\in \{5, 10\}$; link weight $\in \{.01, .1, 1\}$; bias order $\in \{1, 2, 3\}$; transform $\in \{$`sigmoid`, `identity`$\}$; activation/JK $\in \{\mathrm{F}, \mathrm{T}\}$; lr $\in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$; wd $\in \{0, 5\times10^{-4}, 5\times10^{-3}, .05\}$
Polynormer	(Deng et al., 2024)	$d = 64$; local/global layers $= 7/2$; heads $= 1$; $\beta = -1$; input/model/global dropout $= .15/.5/.5$; local epochs $= 100$; lr $= 10^{-3}$; wd $= 5\times10^{-4}$	$d \in \{64, 128, 256, 512\}$; local layers $\in \{5, 7, 10\}$; global layers $\in \{1, 2, 3, 4\}$; heads $\in \{1, 2, 4, 8\}$; $\beta \in \{-1, .1, .5, .9\}$; input dropout $\in \{0, .15, .2, .5\}$; model dropout $\in \{.2, .3, .5, .7\}$; global dropout $\in \{.3, .5\}$; pre-LN $\in \{\mathrm{F}, \mathrm{T}\}$; local epochs $\in \{100, 200\}$; lr $\in \{3\times10^{-5}, 5\times10^{-4}, 10^{-3}\}$; wd $\in \{0, 5\times10^{-5}, 5\times10^{-4}\}$
SGC	(Wu et al., 2019)	dropout $= 0$; $K = 2$	dropout $\in \{0, .2, .5\}$; $K \in \{1, \ldots, 8\}$
SGFormer	(Wu et al., 2023)	$d = 64$; graph weight $= .5$; Transformer/GNN layers $= 1/2$; Transformer/GNN dropout $= 0/0$; initial-feature reuse=False	$d \in \{64, 128, 256, 512\}$; graph weight $\in \{.5, .8\}$; Transformer/GNN dropout $\in \{0, .3, .5\}$; GNN layers $\in \{1, 2, 4, 8\}$; initial-feature reuse $\in \{\mathrm{F}, \mathrm{T}\}$

F BEYOND STANDARD NODE CLASSIFICATION

Our main evaluation considers graph node classification under random context-query splits. We further examine how Ephris generalizes along two dimensions: distribution shifts between context and query nodes, and hypergraph-structured data unseen during pretraining. For both, we follow existing benchmarks and their established protocols, using the same pretrained Ephris checkpoint as in the main experiments for single-pass inference.

Robustness under distribution shift. We first ask whether Ephris remains effective when labeled context and query nodes follow different distributions. We evaluate on four real-world node-classification datasets from the Graph Out-of-Distribution (GOOD) benchmark (Gui et al., 2022). Together, they define six dataset-attribute settings: word diversity and degree for Cora, publication time and degree for Arxiv, university for WebKB, and language for Twitch. For each setting, GOOD provides a random *no-shift* split and constructs *covariate shift* by partitioning nodes according to the selected attribute. For *concept shift*, the partition additionally depends on the class label, altering the relationship between the attribute and label across training and test nodes.

We use the official splits provided by GOOD (link) and compare against the nine baselines reported in the original work. These baselines are evaluated under two model-selection protocols: *ID-selected* selects models using validation data from the training distribution, whereas *OOD-selected* uses labeled validation data from the shifted distribution. As in our main experiments, Ephris predicts using only the training-node labels as context, without access to any validation data.

As shown in Tables 9 and 10, Ephris ranks first in all six no-shift settings, consistent with our main experiments. Under distribution shift, its advantage narrows but remains competitive: Ephris achieves the best average rank against ID-selected baselines (4.00) and ranks fourth against OOD-selected baselines (5.08). These results are encouraging given that several baselines are specifically designed for OOD generalization, whereas Ephris is not explicitly designed to handle distribution shift.

The degradation under distribution shift is concentrated in a few specific settings rather than being uniform across tasks. Ephris degrades substantially under degree-based covariate shift on Arxiv, time-based concept shift on Arxiv, university-based covariate shift on WebKB, and language-based concept shift on Twitch, while remaining competitive in the other settings. This overall competitiveness, together with these localized failures, suggests that Ephris can generalize under distribution shift, but its robustness remains case-dependent. A promising direction is to incorporate distribution-shift scenarios into the synthetic graph prior during pretraining.

Hypergraph transfer. We further test whether Ephris, pretrained exclusively on graphs, can transfer to hypergraph node classification. As shown in Table 11, we evaluate on the ten hypergraph datasets from AllSet (Chien et al., 2021) using the same random 50/25/25 splits over 20 runs. Since Ephris operates on graphs, we convert each hypergraph using either clique expansion, which connects nodes sharing a hyperedge, or an incidence representation, which introduces each hyperedge as an unlabeled node. For both representations, we use the same frozen checkpoint as in the main experiments and perform inference from the 50% labeled context, without hypergraph-specific training or validation-based model selection. In contrast, the published baselines are trained separately on each target hypergraph and use the 25% validation split for model selection.

Despite this difference, Ephris achieves or ties the highest reported accuracy on six of the ten datasets and remains competitive on most others. Neither representation consistently dominates, suggesting that the preferred graph representation depends on the target hypergraph. These results show that the learned graph ICL mechanism transfers effectively to hypergraphs through simple graph representations. More broadly, they suggest that the inference-only paradigm of Ephris may extend beyond replacing per-dataset GNN training to reducing the need for specialized training on other graph-structured data such as hypergraphs.

Table 9: **GOOD: no shift and covariate shift.** Results on the official GOOD splits using either ID or OOD validation for baseline model selection. Scores are mean ± SD in percent (ROC-AUC for Twitch; accuracy otherwise). Baseline results are reported by GOOD (Gui et al., 2022), while Ephris uses the same frozen checkpoint in both settings without model selection. Red, blue, and green mark the three best distinct scores in each column. Average ranks are computed separately for ID- and OOD-based selection across the six dataset-attribute settings.

No shift — ID-selected

Method	Cora		Arxiv		WebKB	Twitch	Avg. rank ↓
	Word	Degree	Time	Degree	University	Language	
ERM	69.41 ± 0.30	69.42 ± 0.30	**73.02** ± 0.14	**72.99** ± 0.12	47.85 ± 0.89	68.05 ± 0.52	5.92
IRM	69.42 ± 0.38	69.40 ± 0.38	72.90 ± 0.14	72.92 ± 0.07	47.31 ± 1.21	68.30 ± 0.29	7.25
VREx	69.43 ± 0.29	69.42 ± 0.29	72.84 ± 0.09	72.88 ± 0.09	47.85 ± 0.89	68.07 ± 0.52	7.25
GroupDRO	69.46 ± 0.25	69.40 ± 0.30	72.91 ± 0.12	72.98 ± 0.10	47.85 ± 0.89	**69.19** ± 0.28	5.75
DANN	69.25 ± 0.33	69.24 ± 0.34	73.00 ± 0.12	72.97 ± 0.10	47.85 ± 0.89	68.07 ± 0.52	7.00
Deep Coral	69.46 ± 0.27	69.43 ± 0.30	72.95 ± 0.09	72.91 ± 0.12	48.12 ± 0.89	68.29 ± 0.65	5.42
Mixup	**70.56** ± 0.35	**70.87** ± 0.47	**73.19** ± 0.16	**73.03** ± 0.14	**51.88** ± 1.34	67.09 ± 0.34	3.50
EERM	**70.10** ± 0.22	**70.38** ± 0.24	OOM	OOM	50.54 ± 0.46	**70.80** ± 0.08	5.33
SRGNN	69.05 ± 0.54	69.08 ± 0.53	72.99 ± 0.04	**72.99** ± 0.02	**52.96** ± 1.04	67.69 ± 0.13	6.58
Ephris	**71.59** ± 0.32	**71.57** ± 0.31	**74.98** ± 0.06	**74.97** ± 0.05	**81.45** ± 0.00	**73.06** ± 0.01	**1.00**

Covariate shift — ID-selected

Method	Cora		Arxiv		WebKB	Twitch	Avg. rank ↓
	Word	Degree	Time	Degree	University	Language	
ERM	64.44 ± 0.55	55.76 ± 0.82	70.64 ± 0.47	58.53 ± 0.16	11.64 ± 0.90	47.73 ± 0.72	5.67
IRM	**64.83** ± 0.25	**55.77** ± 0.46	70.55 ± 0.33	**58.70** ± 0.12	11.91 ± 2.62	48.05 ± 0.16	3.83
VREx	64.49 ± 0.55	55.46 ± 0.87	70.54 ± 0.33	**58.59** ± 0.21	10.58 ± 1.02	47.70 ± 0.70	6.83
GroupDRO	64.49 ± 0.66	55.44 ± 0.91	70.67 ± 0.31	58.46 ± 0.21	12.96 ± 1.95	47.23 ± 0.26	6.00
DANN	**64.72** ± 0.22	55.50 ± 0.60	70.57 ± 0.40	58.56 ± 0.16	**15.34** ± 1.02	47.72 ± 0.73	4.67
Deep Coral	64.63 ± 0.38	55.52 ± 0.93	70.59 ± 0.29	**58.63** ± 0.21	**14.29** ± 2.92	46.64 ± 0.70	4.83
Mixup	63.07 ± 1.52	**57.21** ± 1.12	**71.05** ± 0.31	57.43 ± 0.27	10.85 ± 0.66	**51.33** ± 1.50	5.00
EERM	60.80 ± 0.61	55.23 ± 0.40	OOM	OOM	11.90 ± 0.37	**52.48** ± 0.76	7.83
SRGNN	64.49 ± 0.19	54.67 ± 0.36	**70.70** ± 0.42	57.48 ± 0.07	**16.14** ± 3.35	46.17 ± 0.98	6.17
Ephris	**66.41** ± 0.22	**57.20** ± 0.41	**70.91** ± 0.16	56.73 ± 0.46	6.75 ± 0.42	**57.14** ± 0.04	**4.17**

Covariate shift — OOD-selected

Method	Cora		Arxiv		WebKB	Twitch	Avg. rank ↓
	Word	Degree	Time	Degree	University	Language	
ERM	**64.86** ± 0.38	56.30 ± 0.49	71.08 ± 0.23	58.91 ± 0.23	14.29 ± 3.24	48.95 ± 3.19	5.17
IRM	64.77 ± 0.36	56.28 ± 0.63	71.04 ± 0.16	58.98 ± 0.28	13.49 ± 0.75	47.21 ± 0.98	6.92
VREx	64.80 ± 0.28	56.30 ± 0.50	**71.12** ± 0.24	**58.99** ± 0.16	14.29 ± 3.24	48.99 ± 3.20	4.33
GroupDRO	64.72 ± 0.34	56.29 ± 0.43	**71.15** ± 0.20	**59.08** ± 0.16	**17.20** ± 0.76	47.20 ± 0.44	5.08
DANN	64.77 ± 0.42	56.10 ± 0.59	71.05 ± 0.29	**59.00** ± 0.18	15.08 ± 0.37	48.98 ± 3.22	5.42
Deep Coral	64.72 ± 0.36	56.35 ± 0.38	71.07 ± 0.21	58.97 ± 0.20	13.76 ± 1.30	49.64 ± 2.44	5.42
Mixup	**65.23** ± 0.56	**58.20** ± 0.67	**71.34** ± 0.14	57.60 ± 0.31	**17.46** ± 1.94	**52.27** ± 0.78	2.50
EERM	61.98 ± 0.10	**56.88** ± 0.32	OOM	OOM	**24.61** ± 4.86	**51.34** ± 1.41	6.17
SRGNN	64.66 ± 0.21	54.78 ± 0.10	70.83 ± 0.10	57.52 ± 0.10	13.23 ± 2.93	47.30 ± 1.43	8.83
Ephris	**66.41** ± 0.22	**57.20** ± 0.41	70.91 ± 0.16	56.73 ± 0.46	6.75 ± 0.42	**57.14** ± 0.04	**5.17**

Table 10: **GOOD: concept shift.** Results on the official GOOD splits using either ID or OOD validation for baseline model selection. Scores are mean ± SD in percent (ROC-AUC for Twitch; accuracy otherwise). Baseline results are reported by GOOD (Gui et al., 2022), while Ephris uses the same frozen checkpoint in both settings without model selection. Red, blue, and green mark the three best distinct scores in each column. Average ranks are computed separately for ID- and OOD-based selection across the six dataset-attribute settings.

ID-selected

Method	Cora		Arxiv		WebKB	Twitch	Avg. rank ↓
	Word	Degree	Time	Degree	University	Language	
ERM	64.20 ± 0.56	60.38 ± 0.33	**65.70** ± 0.42	**61.77** ± 0.29	24.77 ± 0.43	**48.57** ± 0.17	4.17
IRM	64.16 ± 0.61	**61.00** ± 0.34	**65.69** ± 0.55	61.49 ± 0.36	24.16 ± 0.80	**49.77** ± 0.82	5.50
VREx	64.20 ± 0.54	60.05 ± 0.72	65.40 ± 0.54	61.61 ± 0.32	24.77 ± 0.43	48.56 ± 0.18	6.00
GroupDRO	**64.38** ± 0.34	60.03 ± 0.88	**65.57** ± 0.66	61.59 ± 0.56	24.77 ± 0.43	47.44 ± 1.08	5.67
DANN	64.29 ± 0.33	59.65 ± 0.94	65.42 ± 0.53	61.43 ± 0.40	24.77 ± 0.43	**48.57** ± 0.18	6.17
Deep Coral	**64.38** ± 0.36	60.22 ± 0.55	65.53 ± 0.63	**61.77** ± 0.37	24.77 ± 0.43	47.46 ± 0.32	4.75
Mixup	64.22 ± 0.33	**63.49** ± 0.23	64.01 ± 0.50	60.60 ± 1.01	**27.83** ± 1.53	**51.87** ± 0.37	4.50
EERM	63.35 ± 0.03	57.46 ± 0.87	OOM	OOM	24.77 ± 0.43	44.22 ± 0.81	9.42
SRGNN	**64.90** ± 0.03	59.96 ± 0.89	65.37 ± 0.22	**61.74** ± 0.10	**25.08** ± 1.13	48.27 ± 1.10	5.00
Ephris	**66.30** ± 0.22	**63.22** ± 0.26	63.60 ± 0.20	**62.28** ± 0.41	**48.62** ± 0.00	44.72 ± 0.01	**3.83**

OOD-selected

Method	Cora		Arxiv		WebKB	Twitch	Avg. rank ↓
	Word	Degree	Time	Degree	University	Language	
ERM	**64.60** ± 0.17	60.54 ± 0.44	67.32 ± 0.24	**62.99** ± 0.20	27.83 ± 0.76	57.32 ± 0.18	5.25
IRM	**64.60** ± 0.16	**61.23** ± 0.32	**67.41** ± 0.16	**62.97** ± 0.27	27.52 ± 0.43	**59.17** ± 0.85	4.00
VREx	64.57 ± 0.18	60.58 ± 0.42	67.37 ± 0.27	**63.00** ± 0.33	27.83 ± 0.38	57.37 ± 0.14	4.92
GroupDRO	**64.62** ± 0.17	60.65 ± 0.31	**67.45** ± 0.15	62.88 ± 0.24	28.14 ± 1.12	**60.27** ± 0.62	3.25
DANN	64.51 ± 0.19	60.78 ± 0.38	67.28 ± 0.16	62.91 ± 0.22	26.91 ± 0.63	**57.46** ± 0.14	6.00
Deep Coral	64.58 ± 0.18	60.58 ± 0.40	**67.42** ± 0.22	62.85 ± 0.29	**28.75** ± 1.13	56.97 ± 0.23	5.08
Mixup	64.44 ± 0.10	**63.65** ± 0.39	64.84 ± 0.59	61.28 ± 0.87	**31.19** ± 0.43	55.28 ± 0.12	6.17
EERM	63.09 ± 0.36	58.38 ± 0.04	OOM	OOM	27.83 ± 4.12	51.94 ± 4.52	9.17
SRGNN	**64.62** ± 0.07	61.08 ± 0.09	67.17 ± 0.23	62.09 ± 0.58	27.52 ± 0.43	56.05 ± 0.22	6.17
Ephris	**66.30** ± 0.22	**63.22** ± 0.26	63.60 ± 0.20	62.28 ± 0.41	**48.62** ± 0.00	44.72 ± 0.01	**5.00**

Table 11: **Transfer to hypergraph node classification.** Test accuracy (%, mean ± standard deviation) under random 50/25/25 splits. We evaluate two graph representations: *clique*, which connects nodes that share a hyperedge, and *incidence*, which introduces unlabeled hyperedge nodes with mean member features and a node-kind indicator. Both Ephris variants use the same 20 splits and perform inference only, using the same pretrained checkpoint as in the main experiments without additional training. Published baseline results are taken from AllSet (Chien et al., 2021) (Table 2), which uses the same random 50/25/25 splits over 20 runs. Red, blue indicate the first-, second-highest accuracies, respectively, with ties sharing the same color.

Method	Cora	CiteSeer	PubMed	Cora-CA	DBLP-CA
Published: AllSet (ICLR 2022), Table 2					
AllSetTransformer	78.59±1.47	**73.08**±1.20	88.72±0.37	83.63±1.47	91.53±0.23
AllDeepSets	76.88±1.80	70.83±1.63	**88.75**±0.33	81.97±1.50	91.27±0.27
MLP	75.17±1.21	72.67±1.56	87.47±0.51	74.31±1.89	84.83±0.22
CECGN	76.17±1.39	70.16±1.31	86.45±0.43	77.05±1.26	88.00±0.26
CEGAT	76.41±1.53	70.63±1.30	86.81±0.42	76.16±1.19	88.59±0.29
HNHN	76.36±1.92	72.64±1.57	86.90±0.30	77.19±1.49	86.78±0.29
HGNN	79.39±1.36	72.45±1.16	86.44±0.44	82.64±1.65	91.03±0.20
HCHA	79.14±1.02	72.42±1.42	86.41±0.36	82.55±0.97	90.92±0.22
HyperGCN	78.45±1.26	71.28±0.82	82.84±8.67	79.48±2.08	89.38±0.25
UniGCNII	78.81±1.05	73.05±2.21	88.25±0.40	83.60±1.14	**91.69**±0.19
HAN (full batch)*	**80.18**±1.15	**74.05**±1.43	86.21±0.48	**84.04**±1.02	90.89±0.23
HAN (mini batch)*	**79.70**±1.77	**74.12**±1.52	85.32±2.25	81.71±1.73	90.17±0.65
Frozen Ephris evaluation (20 runs)					
Ephris (clique)	**80.76**±1.17	72.76±1.38	**90.77**±0.35	**84.51**±1.05	**92.20**±0.23
Ephris (incidence)	79.24±1.51	71.63±1.45	**90.72**±0.40	**85.01**±1.02	**92.01**±0.22

Method	Zoo	20News	Mushroom	NTU2012	ModelNet40
Published: AllSet (ICLR 2022), Table 2					
AllSetTransformer	**97.50**±3.59	**81.38**±0.58	**100.00**±0.00	88.69±1.24	**98.20**±0.20
AllDeepSets	**95.39**±4.77	81.06±0.54	**99.99**±0.02	88.09±1.52	96.98±0.26
MLP	87.18±4.44	**81.42**±0.49	**100.00**±0.00	85.52±1.49	96.14±0.36
CECGN	51.54±11.19	OOM	95.27±0.47	81.52±1.43	89.92±0.46
CEGAT	47.88±14.03	OOM	96.60±1.67	82.21±1.23	92.52±0.39
HNHN	93.59±5.88	**81.35**±0.61	**100.00**±0.01	**89.11**±1.44	97.84±0.25
HGNN	92.50±4.58	80.33±0.42	98.73±0.32	87.72±1.35	95.44±0.33
HCHA	93.65±6.15	80.33±0.80	98.70±0.39	87.48±1.87	94.48±0.28
HyperGCN	N/A	81.05±0.59	47.90±1.04	56.36±4.86	75.89±5.26
UniGCNII	93.65±4.37	81.12±0.67	**99.96**±0.05	**89.30**±1.33	98.07±0.23
HAN (full batch)*	85.19±8.18	OOM	90.86±2.40	83.58±1.46	94.04±0.41
HAN (mini batch)*	75.77±7.10	79.72±0.62	93.45±1.31	80.77±2.36	91.52±0.96
Frozen Ephris evaluation (20 runs)					
Ephris (clique)	93.85±4.74	80.86±0.55	**100.00**±0.00	88.23±1.40	**98.26**±0.18
Ephris (incidence)	**96.54**±4.30	81.23±0.48	**100.00**±0.00	**89.17**±1.26	**98.34**±0.12

Note. *HAN uses additional preprocessing in the AllSet source paper. OOM and N/A retain AllSet's reported out-of-memory and numerical-instability entries, respectively.

G Ablation Studies

In this section, we use controlled proxy experiments to isolate the contributions of Ephris's key design choices. Full pretraining requires approximately 16 GPU-days, making repeated full-scale ablations impractical. We therefore train all variants under the same reduced budget, using 10% of Stage 1 training: 5,000 steps over 320,000 synthetic graphs. All other optimization settings follow the Stage 1 configuration in Table 6, including 5% learning-rate warmup followed by cosine decay. We evaluate every variant on the same 51 datasets as in the main experiments.

Because these proxy runs stop well before full pretraining, their absolute performance remains below that of the fully pretrained model, and the relative effects of individual design choices may change with further training. We therefore use them to compare variants under a common early-training budget rather than to estimate full-scale effect sizes. We first disentangle the contributions of the architecture and pretraining prior, and then ablate the individual design choices within each.

Disentangling architecture and prior. Ephris introduces improvements along two axes: the architecture and the synthetic graph prior. To separate their contributions, we conduct a 3×3 factorial study using the architectures and priors of Ephris, GraphPFN (Eremeev et al., 2026), and NodePFN (Choi et al., 2026). We train every architecture with every prior under the same 5,000-step budget, allowing architectures to be compared under a fixed prior and priors under a fixed architecture. GraphPFN supports both training from scratch and adaptation from a pretrained LimiX backbone; we use the latter, consistent with its released checkpoint, and train each prior from the same pretrained backbone. Although this gives GraphPFN an initialization advantage over Ephris and NodePFN, which are trained from scratch, it more closely follows the original GraphPFN pipeline.

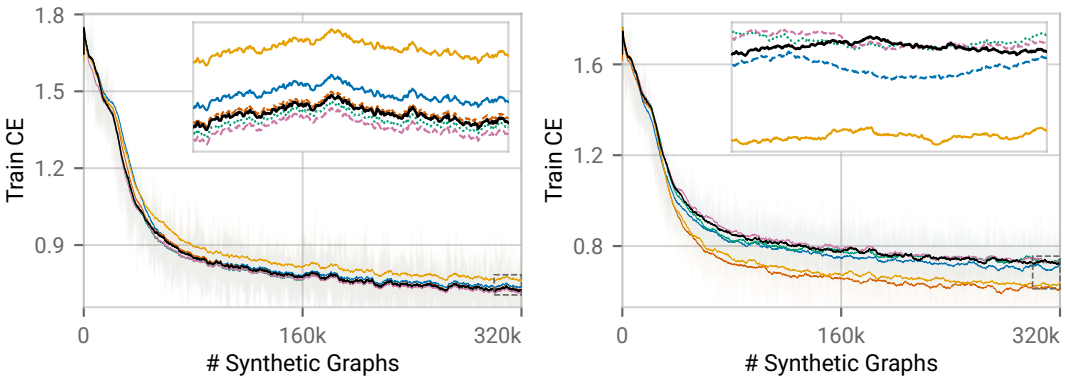
As shown in Figure 7, clear differences emerge even within this reduced training budget. Holding the prior fixed, the Ephris architecture achieves the highest mean accuracy under all three priors. This consistency suggests that its gains are not tied to our synthetic prior. Two explanations may contribute: message passing may provide an inductive bias better suited to graph ICL, or its more constrained interaction structure may learn more efficiently than dense attention under the limited training budget. Our proxy experiment does not distinguish these effects.

Holding the architecture fixed, the Ephris prior likewise performs best across all three architectures. The comparison is informative because the three priors construct graphs differently. NodePFN generates features before constructing graph structure from label-induced communities, whereas GraphPFN and Ephris incorporate sampled graph structure into feature and label generation. The observed ordering does not establish whether feature-first or structure-first generation is inherently preferable, as NodePFN outperforms GraphPFN. However, Ephris consistently improves over GraphPFN despite sharing its structure-first formulation, suggesting that the additional diversity in graph generation and explicitly sampled relational dynamics contributes beyond this high-level choice.

Overall, the factorial study shows that the gains from our architecture and prior are not specific to their pairing: each transfers across substantially different choices of the other. We next examine the individual design choices responsible for these gains.

Architecture Ablations. The left panel of Figure 8 reports training cross-entropy and downstream performance for architectural variants trained on the same prior distribution. For the message-passing block, we replace local attention with uniform aggregation (■), remove global nodes (■), and remove neighborhood-aware scaling (■). We further ablate our graph-aware feature refinement by restricting symmetric gathering to context nodes (■) and removing the *broadcast* after message passing, preventing graph-structural information from updating the feature tokens (■). All five variants degrade downstream performance, but their training losses show a different pattern. Uniform aggregation and removing the broadcast increase training cross-entropy, whereas the other variants achieve comparable or even lower training loss despite worse downstream performance. This contrast suggests that our architectural choices contribute not only to fitting the synthetic pretraining tasks, but also to transferring to real-world graphs.

Prior Ablations. The right panel of Figure 8 ablates the main components of our synthetic graph prior. We remove relational propagation entirely (■), restrict it to diffusion (■), cascade (■), or node-dependent mixing (■), and remove embedding-like post-processing (■). Removing relational propagation causes by far the largest degradation, reducing mean accuracy from 78.91% to 60.61%. This confirms that sampling graph topology alone is insufficient; the topology must also shape



Variant	Acc. ↑	Rank ↓	Win ↑
Full model	**78.91**	**2.84**	–
Uniform aggregation	78.79	3.03	37.25
w/o global nodes	76.60	4.75	19.61
w/o degree scaling	78.48	3.13	39.22
Context-only gather	78.07	2.99	43.14
w/o feature broadcast	77.44	4.26	31.37

Variant	Acc. ↑	Rank ↓	Win ↑
Full model	**78.91**	**2.27**	–
w/o relational updates	60.61	5.61	3.92
Diffusion only	77.64	3.32	35.29
Cascade only	76.89	3.39	21.57
Degree only	78.44	3.34	23.53
No embedding	77.38	3.06	33.33

Figure 8: **Architecture (left) and prior (right) ablations.** Models are pretrained for 5,000 steps. Curves show training cross-entropy smoothed with a trailing 100-step mean, with faint traces indicating raw losses; insets magnify the final 300k–320k training graphs. Tables report mean accuracy (Acc., %), average rank (Rank), and the percentage of strict wins against the full model (Win, %) across 51 datasets under high-label regime. Ranks are computed separately within the architecture and prior ablations, and table swatches indicate the corresponding training curves.

feature and label generation during pretraining. Restricting relational propagation to any single dynamic also underperforms their mixture, while removing embedding-like post-processing further reduces performance. Together, these results support diversity in both relational dynamics and feature representations when constructing the synthetic graph prior.

H Full Experimental Results

This section provides the complete results underlying the main evaluation in Section 5. We report aggregate performance across all 51 datasets, examine its trade-off with adaptation runtime, and further analyze pairwise and subgroup results to characterize where the observed gains arise.

Aggregate performance. Table 13 and Figure 9 report the complete aggregate results under the high-label (50/25/25) and low-label (10/10/80) regimes. Ephris ranks first in all four metrics in both regimes: Elo, improvability, average rank, and average accuracy. In the high-label regime, Ephris achieves an Elo of 1521 compared with 1388 for the runner-up tuned GCNII, while reducing mean improvability from 18.81% to 8.29% and average rank from 8.57 to 5.13. It also achieves the highest average accuracy at 81.27%. The margins narrow under the low-label regime, where Ephris achieves an Elo of 1381 compared with 1371 for the runner-up tuned GCNII, but remains first in improvability, average rank, and average accuracy at 9.06%, 7.55, and 74.23%, respectively.

HP tuning substantially improves many supervised GNNs, with tuned GCNII and GPRGNN emerging as the strongest GNN baselines across the two regimes. Among prior graph ICL methods, GraphPFN is consistently the strongest. We therefore use these methods as the primary reference points in the comparisons below.

Performance and runtime. The predictive gains of Ephris also come with substantially lower adaptation cost. Runtime overhead is aggregated geometrically across datasets, so differences on the $\log_2$ scale in Table 13 correspond directly to multiplicative differences in average runtime. Compared with the strongest tuned GNNs, Ephris is approximately $739\times$ and $584\times$ faster than GCNII in the high-label and low-label regimes, respectively, and $161\times$ and $103\times$ faster than GPRGNN. Compared with GraphPFN, the strongest prior graph ICL baseline, Ephris is approximately $16.1\times$ and $13.1\times$ faster while achieving stronger predictive performance across all four aggregate metrics. Accordingly, Ephris occupies a favorable performance and runtime region across the metrics in Figure 10, improving predictive performance without increasing target-time computation.

Performance across dataset characteristics. Figures 12 and 13 compare performance across subgroups defined by the dataset characteristics in Table 12.

Table 12: **Dataset subgroup definitions.**

Characteristic	Subgroups
Nodes N	Tiny ($\leq$ 2K), Small (2K to 10K), Medium (10K to 100K), Large ($>$ 100K)
Avg. degree $\bar{d}$	Low ($\leq$ 5), Medium (5 to 20), High ($>$ 20)
Adj. homophily h_{adj}	Negative ($<$ 0), Low (0 to 0.5), High ($\geq$ 0.5)
Features F	Low ($\leq$ 500), Medium (500 to 5K), High ($\geq$ 5K)
Nonzero fraction r	Sparse ($\leq 1\%$), Medium (1% to 50%), Dense ($> 50\%$)
Classes C	Binary (2), Medium (3 to 10), Many ($>$ 10)
Class imbalance	Low ($\leq$ 2), Medium (2 to 10), High ($>$ 10)

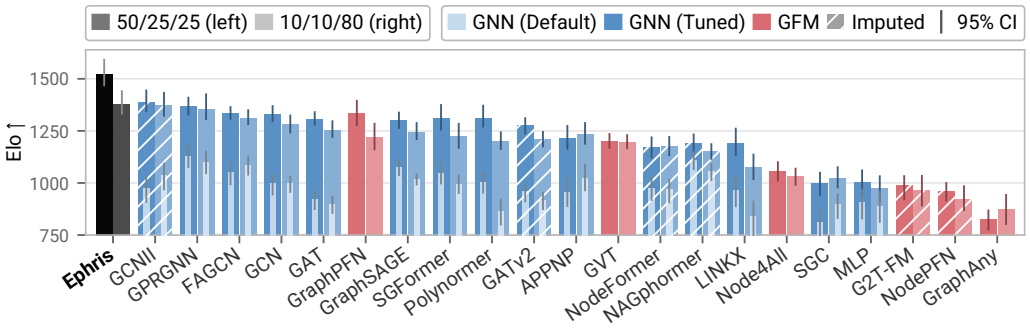
Across these groups, Ephris outperforms GraphPFN in nearly every subgroup under both label regimes. This consistent advantage indicates that the improvement of graph ICL is not tied to a particular graph size, connectivity pattern, homophily level, or task characteristic.

A different pattern emerges against the strongest tuned GNNs. In the high-label regime, Ephris outperforms GCNII and GPRGNN across nearly all subgroups, whereas under low labels the methods become comparable and tuned GNNs lead in several cases. One possible explanation is validation-based adaptation: GCNII and GPRGNN select among 200 configurations for each dataset using validation labels, whereas Ephris uses a single fixed checkpoint and no validation labels. This dataset-specific selection may become particularly valuable when labeled context is low.

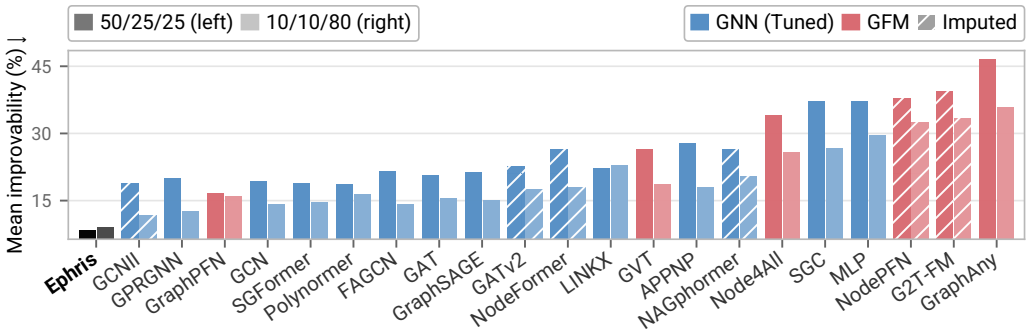
Notably, the two clear exceptions even in the high-label regime are high-dimensional datasets ($F \geq$ 5K) and tasks with more than ten classes ($C > 10$). Both fall outside the distribution directly observed by Ephris during pretraining, which contains at most 1,024 features and ten classes; larger label spaces are instead handled indirectly through ECOC. GraphPFN shows the same failure pattern in these two subgroups. This shared degradation suggests a broader limitation of current graph ICL: generalization remains strongest within the support of the synthetic pretraining distribution and weakens when feature or label dimensionality requires substantial extrapolation.

Table 13: Comparison across 51 datasets under the high-label (50/25/25, left) and low-label (10/10/80, right) train/validation/test regimes. Each side lists all configurations independently in descending Elo order; entries on the same row need not denote the same method. (D)/(T) denote default/HP-tuned configurations. Imp is mean improvability (%), Acc. is mean accuracy (%), and Time is mean fit-time overhead on a $\log_2$ scale. ● Gold, ● silver, and ● bronze mark the top three in each predictive metric separately within each regime. Under high-label, Ephris ranks 1 in Elo, 1 in improvability, 1 in average rank, and 1 in accuracy. Under low-label, Ephris ranks 1 in Elo, 1 in improvability, 1 in average rank, and 1 in accuracy.

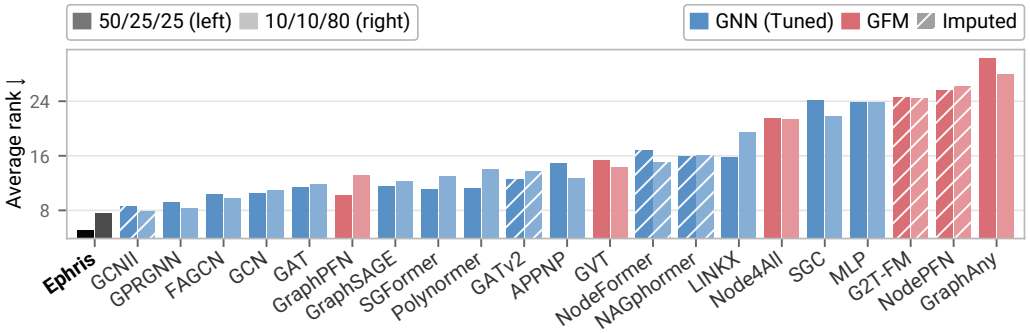
High-label (50/25/25)						Low-label (10/10/80)					
Method	Elo ↑	Imp ↓	Rank ↓	Acc. ↑	Time ↓	Method	Elo ↑	Imp ↓	Rank ↓	Acc. ↑	Time ↓
Ephris	● **1521**	● **8.29**	● **5.13**	● **81.27**	**2.07**	**Ephris**	● **1381**	● **9.06**	● **7.55**	● **74.23**	**2.36**
GCNII (T)	● 1388	18.81	● 8.57	78.02	11.60	GCNII (T)	● 1371	● 11.58	● 7.83	● 73.59	11.55
GPRGNN (T)	● 1369	20.04	● 9.18	77.65	9.40	GPRGNN (T)	● 1356	● 12.54	● 8.28	● 73.42	9.05
GraphPFN	1337	● 16.55	10.24	● 79.76	6.08	FAGCN (T)	1311	14.18	9.78	72.45	9.00
FAGCN (T)	1334	21.59	10.33	76.50	9.33	GCN (T)	1280	14.18	10.88	73.00	9.83
GCN (T)	1331	19.27	10.45	78.49	9.94	GAT (T)	1254	15.43	11.86	72.65	9.96
SGFormer (T)	1312	18.93	11.12	78.54	9.65	GraphSAGE (T)	1246	15.15	12.18	72.91	9.90
Polynormer (T)	1311	● 18.61	11.15	● 79.14	12.44	APPNP (T)	1232	17.92	12.73	71.45	8.61
GAT (T)	1307	20.59	11.31	78.50	10.08	SGFormer (T)	1225	14.58	13.00	73.10	9.62
GraphSAGE (T)	1300	21.23	11.57	78.08	9.91	GraphPFN	1221	16.05	13.18	72.30	6.07
GATv2 (T)	1274	22.52	12.52	77.96	9.60	GATv2 (T)	1210	17.49	13.64	72.02	9.47
APPNP (T)	1215	27.69	14.85	74.87	8.75	Polynormer (T)	1200	16.43	14.05	72.59	12.45
GVT	1202	26.56	15.38	75.43	5.18	GVT	1195	18.76	14.23	71.19	4.71
LINKX (T)	1193	22.30	15.76	78.71	7.44	NodeFormer (T)	1177	17.85	15.01	71.90	10.64
NAGphormer (T)	1188	26.48	15.96	76.60	10.85	NAGphormer (T)	1153	20.39	16.03	70.41	10.54
NodeFormer (T)	1170	26.38	16.73	75.70	10.78	GPRGNN (D)	1100	23.02	18.41	69.15	1.74
GPRGNN (D)	1127	29.52	18.58	74.25	2.00	FAGCN (D)	1086	22.29	19.01	69.48	1.28
NAGphormer (D)	1107	29.84	19.45	74.16	2.72	LINKX (T)	1076	22.98	19.49	71.09	7.78
GraphSAGE (D)	1078	31.09	20.73	73.95	1.76	NAGphormer (D)	1056	23.07	20.36	68.92	2.54
Node4All	1059	34.17	21.55	73.47	3.56	GCNII (D)	1036	25.26	21.25	68.18	3.67
FAGCN (D)	1053	31.65	21.81	73.34	1.56	Node4All	1034	25.72	21.36	69.29	3.49
SGFormer (D)	1049	32.09	21.96	74.12	1.80	APPNP (D)	1025	25.69	21.77	68.15	2.04
Polynormer (D)	1004	36.55	23.84	71.21	4.02	SGC (T)	1025	26.70	21.77	68.51	9.89
MLP (T)	1003	37.10	23.89	72.93	7.49	GraphSAGE (D)	1018	24.73	22.04	69.15	1.80
GCN (D)	1000	36.04	24.01	70.98	1.66	GCN (D)	1000	27.06	22.85	67.93	1.48
SGC (T)	999	37.11	24.06	70.98	10.13	SGFormer (D)	996	26.67	23.02	69.04	1.90
G2T-FM	986	39.40	24.56	72.11	4.09	MLP (T)	977	29.51	23.83	68.31	7.34
NodeFormer (D)	973	35.36	25.08	73.14	4.04	NodeFormer (D)	967	28.26	24.24	68.46	3.92
GCNII (D)	973	36.43	25.11	72.00	3.84	G2T-FM	964	33.26	24.38	65.69	5.79
LINKX (D)	964	38.21	25.45	72.15	0.15	NodePFN	921	32.51	26.13	67.75	11.09
NodePFN	961	37.79	25.58	72.71	9.92	GATv2 (D)	916	30.61	26.32	66.27	1.83
GATv2 (D)	958	38.20	25.69	69.96	2.00	GAT (D)	897	31.17	27.05	66.09	1.69
APPNP (D)	955	37.82	25.80	70.55	2.16	SGC (D)	896	33.42	27.10	66.23	2.56
GAT (D)	922	40.22	27.05	69.47	1.80	MLP (D)	889	33.60	27.34	66.49	0.75
MLP (D)	909	41.47	27.52	71.21	0.71	GraphAny	875	35.83	27.88	65.74	0.90
GraphAny	824	46.59	30.31	67.77	2.25	Polynormer (D)	867	36.30	28.16	63.05	4.13
SGC (D)	810	46.14	30.72	67.86	2.80	LINKX (D)	843	40.84	29.00	62.99	0.32



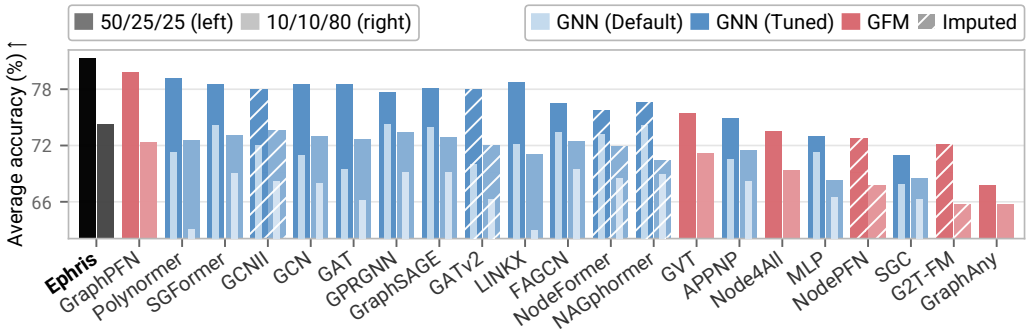
(a) **Elo.**



(b) **Improvability.**

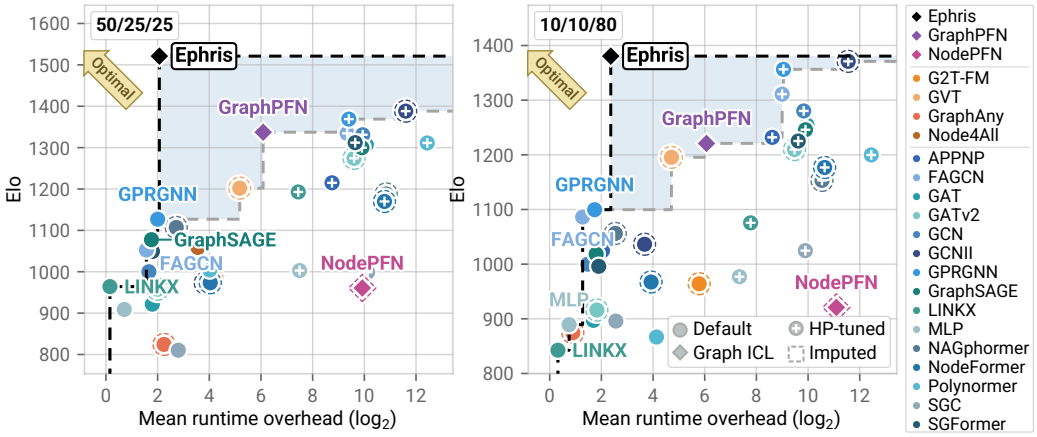


(c) **Average rank.**

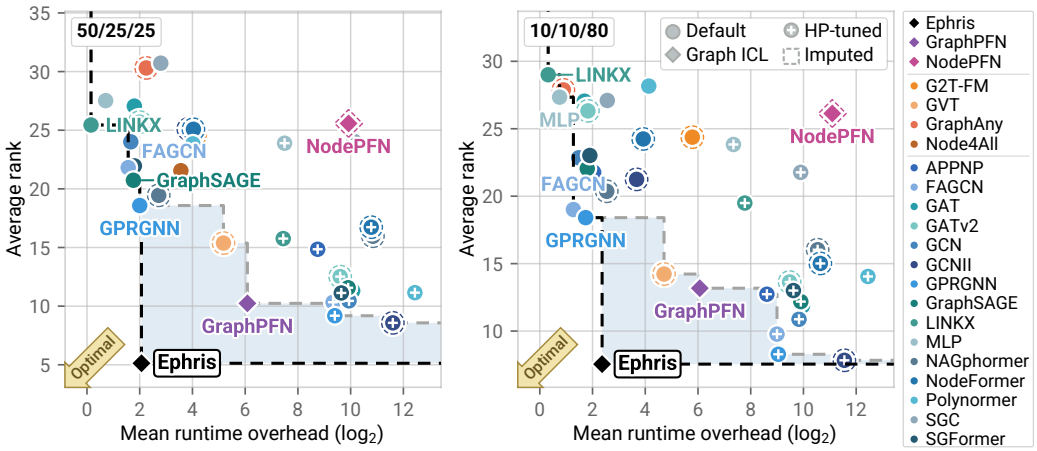


(d) **Average accuracy.**

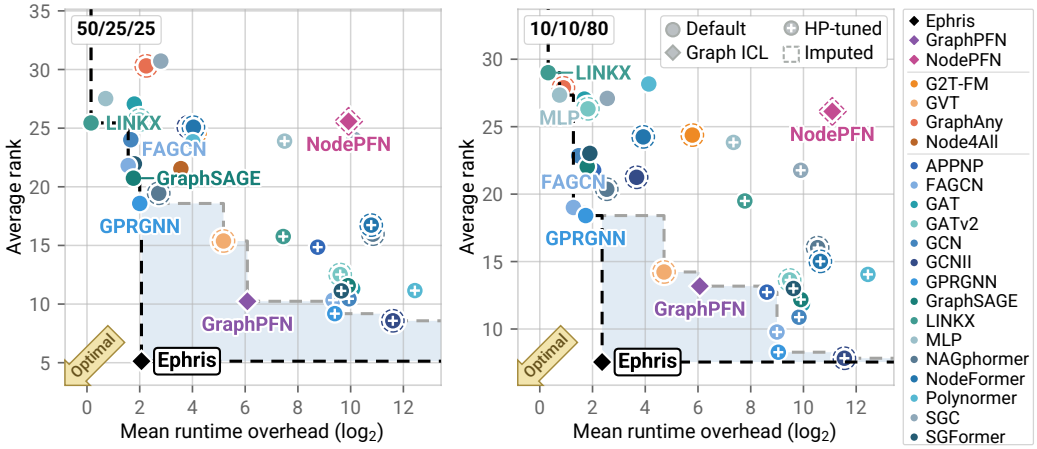
Figure 9: **Aggregate performance across evaluation metrics.** Paired comparisons under the high- and low-label regimes for **(a)** Elo, **(b)** improvability, **(c)** average rank, and **(d)** average accuracy. Each bar shows a default GNN alongside its HP-tuned counterpart. Hatching marks results imputed for OOM runs. Methods are ordered by their mean performance across the two label regimes.



(a) **Elo.**

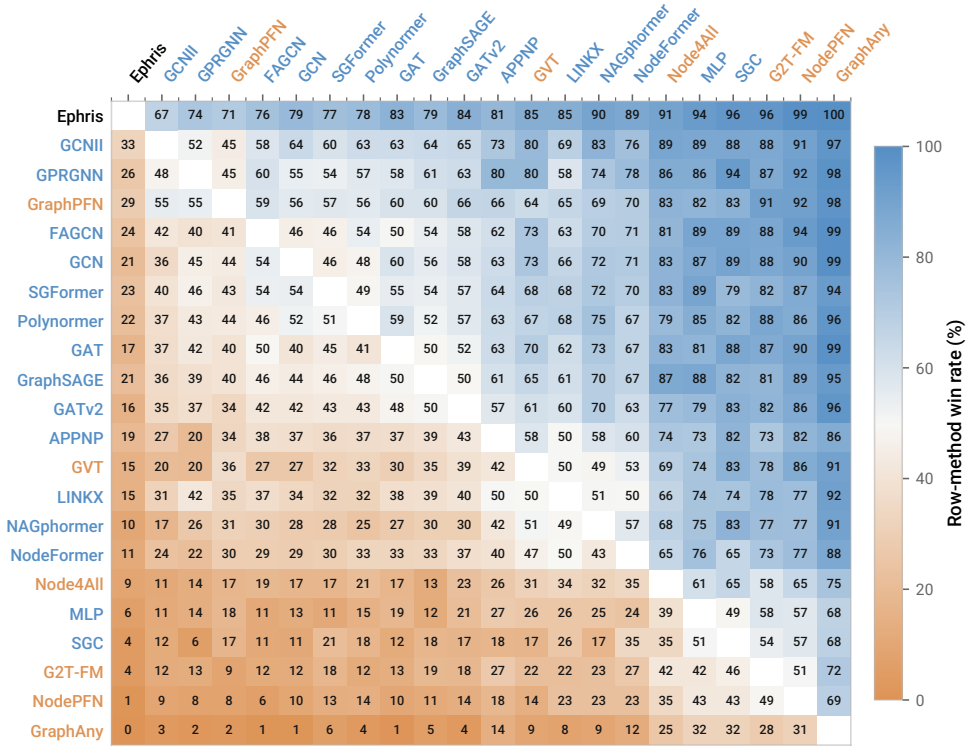


(b) **Average rank.**

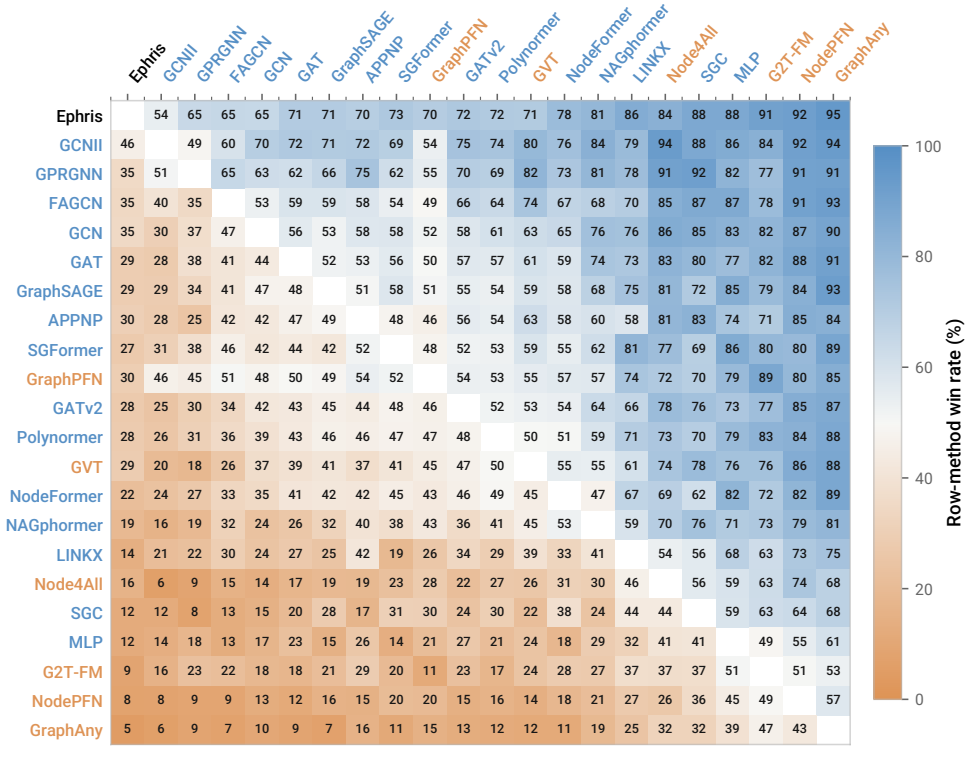


(c) **Average accuracy.**

Figure 10: **Performance-runtime Pareto plots.** Trade-offs between average runtime overhead and **(a)** Elo, **(b)** average rank, and **(c)** average accuracy. Each point represents an evaluated method, and the Pareto frontier identifies methods that are not jointly dominated in performance and runtime.

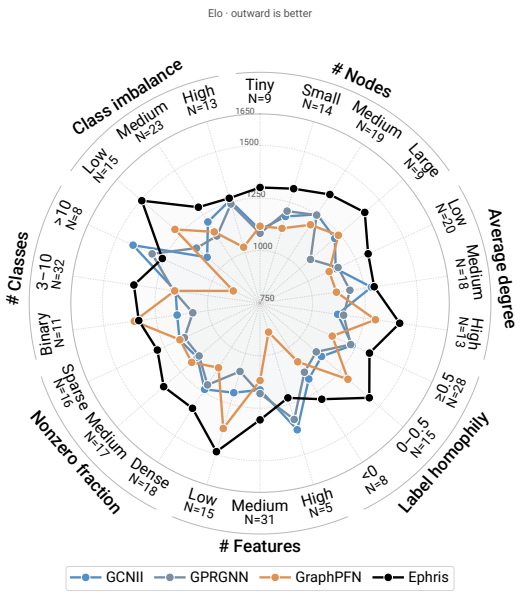


(a) High-label regime (50/25/25)

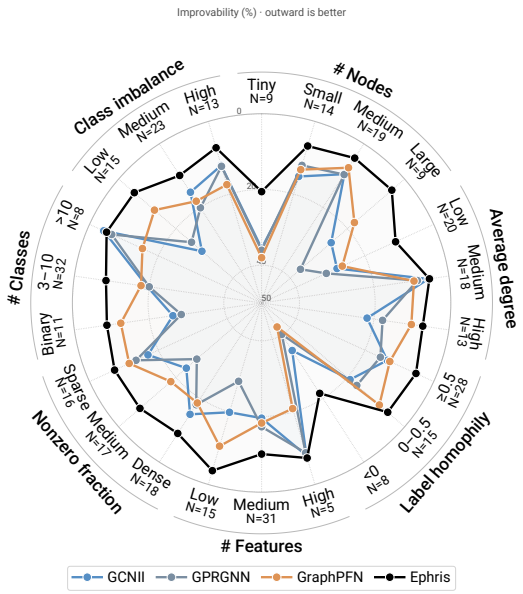


(b) Low-label regime (10/10/80)

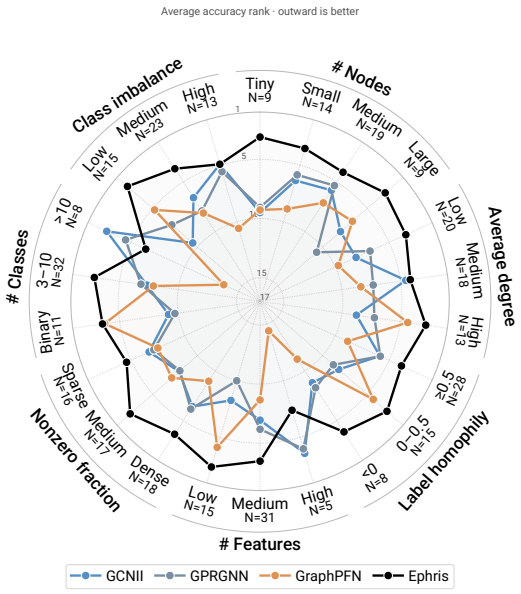
Figure 11: **Pairwise win-rate matrices.** Pairwise comparisons under (a) the high-label and (b) the low-label regime. Each entry reports the fraction of datasets on which the row method achieves higher test accuracy than the column method.



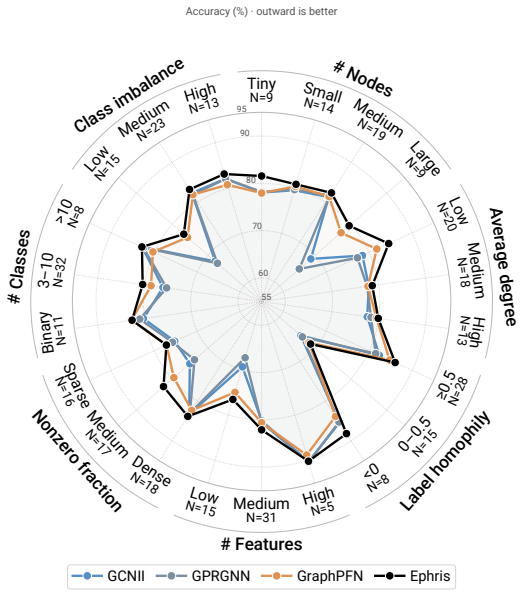
(a) Elo



(b) Improvability

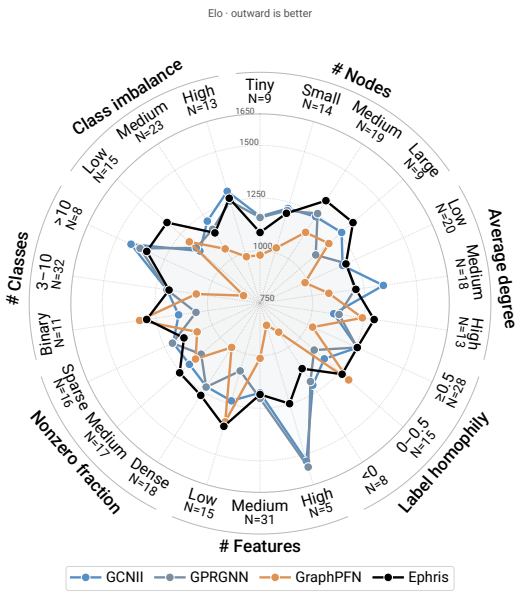


(c) Average rank

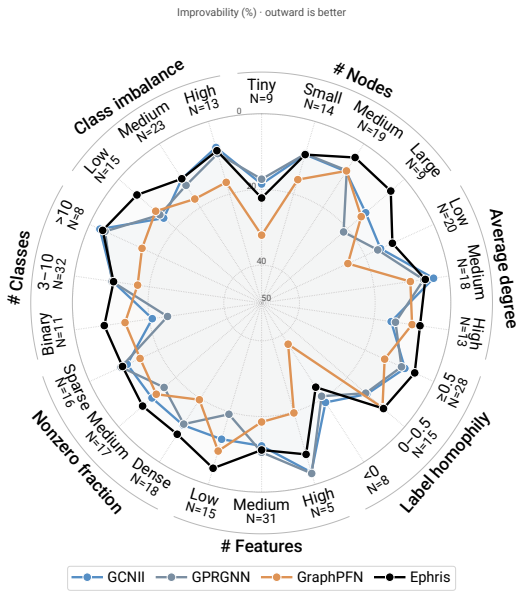


(d) Average accuracy

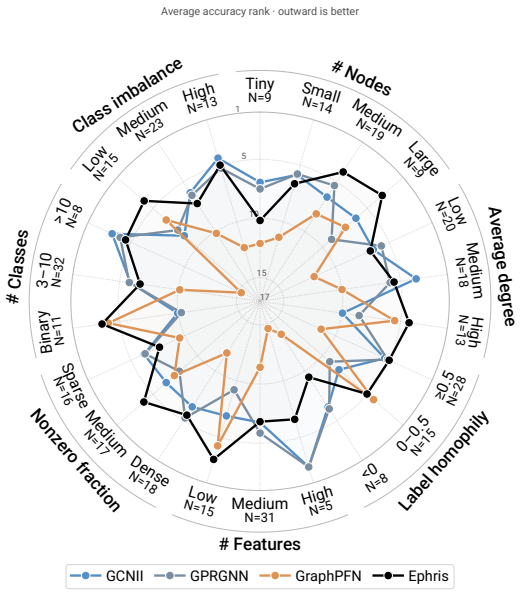
Figure 12: **Performance across dataset subgroups under the high-label regime (50/25/25).** Radar plots show subgroup performance measured by (a) Elo, (b) improvability, (c) average rank, and (d) average accuracy. Each axis corresponds to a dataset subgroup, providing a more detailed view of performance across different dataset characteristics.



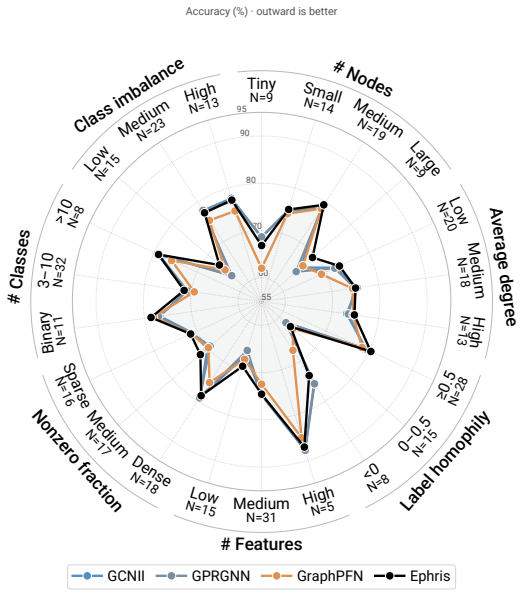
(a) Elo



(b) Improvability



(c) Average rank



(d) Average accuracy

Figure 13: **Performance across dataset subgroups under the low-label regime (10/10/80).** Radar plots show subgroup performance measured by (a) Elo, (b) improvability, (c) average rank, and (d) average accuracy. Each axis corresponds to a dataset subgroup, providing a more detailed view of performance across different dataset characteristics.