LOCAL PREDICTED POSITIONAL ENCODING FOR ZERO-SHOT LINK PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Zero-shot link prediction requires a model to recognize transferable graph structure without fitting to a new graph, yet positional and structural encodings (PSEs) can be costly to compute on large graphs. We introduce Local Predicted Positional Encoding (LPPE), a two-stage, structure-only approach that learns node representations from bounded neighborhoods and uses them in a scalable graph transformer for link prediction. By computing structural supervision on local subgraphs during training, LPPE avoids whole-graph PSE calculations; at inference, the frozen model produces an encoding from a new graph without directly computing PSEs or requiring node features. With neighborhood size, PPR sampling, and encoder configuration fixed, producing LPPE encoding for all nodes at inference scales linearly with the numbers of nodes and edges, compared with the cubic scaling of dense full-graph spectral metric calculations. LPPE exceeds the performance of explicitly computed local structural encodings. On 18 held-out datasets, jointly training on 1 up to 15 training datasets increases zero-shot average precision from 0.676 to 0.758, closing 45.3% of the gap to a separately trained in-domain reference. These results suggest that local structural learning can make structural pretraining more scalable, while joint training on more datasets improves zero-shot link prediction on unseen graphs.

1 INTRODUCTION

Zero-shot link prediction requires a model to recognize useful structural patterns in graphs. We are interested in efficient zero-shot graph link prediction: predicting links in previously unseen graphs without graph-specific fine-tuning, while keeping training and inference computationally tractable. This is difficult when graphs differ in size, density, and connectivity across datasets. Achieving this requires graph foundation models (GFMs) to learn transferable structural patterns across graphs with diverse densities, diameters, and local connectivity.

Graph transformers are used to build GFMs Wang et al. (2025). To make graph topology accessible to the model, transformers often rely on positional and structural encodings (PSEs) that provide structural information beyond node features. However, computing PSEs can be expensive, especially when they depend on global structural properties. The challenge is to learn structural representations that transfer across graphs while keeping their calculations on new graphs tractable. Conventional training approaches still face two bottlenecks:

1. **Preprocessing Overhead:** Computing structural metrics to obtain positional encodings introduces substantial overhead during both training and inference. These metrics must be calculated for the training graphs and for each unseen graph at inference time.
2. **Scalability on Large Graphs:** Computing global structural metrics becomes increasingly expensive as graphs grow. For example, full eigendecomposition of a dense graph matrix can require $\mathcal{O}(N^3)$ time, where N is the number of nodes. These costs can prevent massive real-world graphs from being used for GFM training or inference.

One approach is to train an encoder to predict structural quantities and reuse its learned representations. Graph Positional and Structural Encoder (GPSE), for example, learns from multiple positional and structural targets computed on pretraining graphs (Cantürk et al., 2024). This reduces the need

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

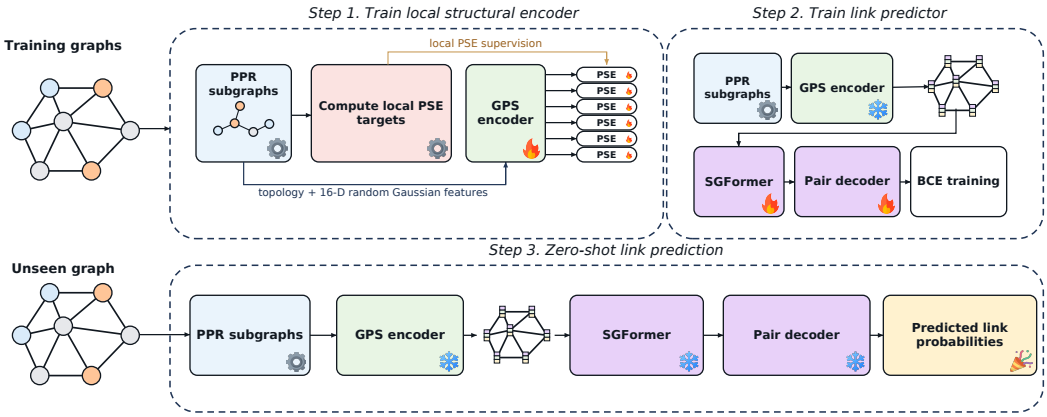


Figure 1: Overview of Local Predicted Positional Encoding (LPPE). First, root-centered personalized PageRank (PPR) subgraphs from training graphs provide local positional and structural encoding (PSE) targets for pretraining a GPS encoder with shared-output multitask supervision. The representations from the pretrained encoder then serve as node features for an SGFormer link predictor trained with a pairwise decoder. On an unseen graph, both models remain frozen: PPR subgraphs are constructed from observed edges, and the models produce link probabilities without computing PSE targets.

to compute structural encodings at inference, but the need to calculate the PSEs for the pretraining graphs can still limit the size of graphs used for training. Local subgraphs offer another source of structural information: prior link prediction work has shown that neighborhoods around candidate links can be informative (Zhang & Chen, 2018; Chamberlain et al., 2023). We investigate whether structural metrics computed on local neighborhoods can help train a node encoder that transfers to unseen graphs.

These bottlenecks motivate LPPE: an approach to predict positional encodings that supports local training and inference. LPPE eliminates whole-graph calculations and reduces the computation required to generate positional encodings during both training and inference, while retaining the structural information needed for zero-shot link prediction. This enables GFMs to train on large graphs and transfer efficiently to unseen graphs.

We introduce *Local Predicted Positional Encoding* (LPPE), a method for learning node encodings from locally computed positional and structural targets. We evaluate it in a two-stage, structure-only link prediction pipeline, illustrated in Figure 1. For each selected training node, LPPE constructs a root-centered subgraph from a bounded personalized PageRank (PPR) neighborhood and computes positional and structural metrics within that subgraph. A graph Transformer learns to predict these targets from the graph topology; its root representation becomes the node's encoding. A downstream link predictor is then fitted on training graphs using these encodings. On an unseen test graph, both models remain frozen, as shown in the final stage of Figure 1: LPPE builds neighborhoods from observed edges and produces node encodings without computing structural metrics directly or using the possible node features, and the link predictor scores candidate edges without graph-specific optimization or calibration.

Localizing target generation changes where structural computation is required. LPPE needs explicit targets only for sampled training neighborhoods, while its pretrained encoder can produce representations for all nodes of a new graph. With bounded neighborhood size, a fixed number of bounded-length walks per root, and a fixed encoder, generating positional encodings for all nodes has $\mathcal{O}(N + M)$ time complexity for a graph with N nodes and M observed edges.

We evaluate transfer using 15 training datasets and 18 held-out test datasets from the AnyGraph collection (Xia & Huang, 2026). LPPE exceeds the zero-shot performance of explicitly computed local structural encodings in the same transfer setting. When we jointly train on 15 datasets instead

of one, mean average precision on held-out test datasets increases from 0.676 ± 0.059 to 0.758 ± 0.022. The latter result reduces the gap to a separately trained, within-dataset reference of 0.857 by 45.3%.

Together, these results present LPPE as a way to learn transferable node representations from local structural supervision. The method confines structural metrics computation to bounded local neighborhoods, supports encoding generation on unseen large graphs under a fixed computational budget, and shows improved average zero-shot link prediction as training expands across more datasets.

2 LPPE: Local Predicted Positional Encoding

Zero-shot link prediction requires structural representations that transfer to unseen graphs without graph-specific fine-tuning. Positional and structural encodings can provide this information, but computing them on entire graphs is costly, especially as graphs grow; even methods that predict these encodings may still require full-graph targets during training. The challenge is to retain useful structural information while making both training and inference practical on large graphs.

We propose a two-stage structural pipeline for scalable zero-shot link prediction, shown in Figure 1 along with its application to unseen graphs. The first stage learns *Local Predicted Positional Encodings* (LPPEs) from small root-centered subgraphs, while the second stage uses the resulting frozen node representations as positional features for a scalable graph-transformer link predictor. The pipeline is structural-only: it does not require dataset-specific input node attributes.

Given an observed graph $\mathcal{G}$, we construct a personalized PageRank (PPR) neighborhood for each node v. Each neighborhood contains v and the highest-scoring PPR nodes, up to 50 nodes in total, and we retain the subgraph induced by these nodes. Structural targets are computed within each induced local subgraph rather than on the full graph. The LPPE trains to predict the same target metrics as in Cantürk et al. (2024). These metrics include node-level electrostatic, Laplacian, random-walk, and heat-kernel encodings, as well as graph-level Laplacian-eigenvalue and cycle-count descriptors.

LPPE is trained on PPR neighborhoods rooted at nodes from $\mathcal{G}_{\text{train}}$. To make training scalable for many datasets, LPPE uses a subset of the available roots. Following GPSE Cantürk et al. (2024), each subgraph node receives random Gaussian input features, and task-specific heads train an encoder to predict multiple positional and structural targets. LPPE applies this approach with a GPS-based encoder (Rampášek et al., 2022) and targets computed within each PPR subgraph, as depicted in the first training stage of Figure 1. The encoder's latent representation at the root node, before the prediction heads, serves as that node's LPPE encoding.

In the second stage, the LPPE encoder is frozen and applied to the PPR neighborhood of each node in the graph. This produces one encoding vector per node, which serves as a learned positional encoding for the downstream task. Crucially, inference requires no explicit computation of structural targets on new graphs: the pretrained LPPE predicts topology-aware representations directly from PPR-induced subgraphs and random node inputs. PPR neighborhoods are still constructed at inference time, but expensive target generation is avoided.

The frozen LPPE encodings are then used as node features for a scalable full-graph transformer (Wu et al., 2023) followed by a pairwise decoder that predicts whether two nodes are linked, as shown in the second training stage of Figure 1. Unlike the LPPE encoder, which operates on small local subgraphs, the downstream transformer processes the full observed graph efficiently and combines local structural representations with broader graph context.

For strict zero-shot transfer, both the LPPE model and the link-prediction model are fitted on training graphs and remain frozen for the test graphs, as shown in the final stage of Figure 1. On a new test graph, LPPE encodings are generated using only the target's observed structure, and the frozen link predictor produces link scores without using the features of the test dataset or needing fine-tuning to it. This separation enables the model to transfer learned structural regularities across graphs while avoiding the need to recompute handcrafted positional encodings for each test dataset.

3 Computational Efficiency and Graph-Size Scalability

Structural encodings introduce distinct computational overheads during both pretraining and deployment, depending on whether targets are evaluated globally or locally, and whether they are explicitly computed or predicted.

Table 1 summarizes trade-offs across global and local, explicit and predicted structural encoding strategies. The following sections isolate the two main sources of efficiency in LPPE: localizing structural target generation and replacing repeated explicit computation with a learned predictor. The four strategies differ in what becomes a node feature: global explicit methods compute full-graph metrics directly, global predicted methods such as GPSE Cantürk et al. (2024) learn to predict those metrics, local explicit methods calculate metrics on each node's subgraph, and LPPE predicts locally defined metrics from bounded neighborhoods. Table 1 rates their pretraining cost, required metric coverage, upfront and inference time, scalability, and cost of using multiple structural targets. The ratings refer to obtaining encodings for all nodes of an unseen graph; predicted methods first require training an encoder, whereas explicit methods do not.

Table 1: Qualitative comparison of structural encoding strategies.

Feature	**Global explicit**	**Global predicted**	**Local explicit**	**Local predicted** (LPPE)
Pretraining cost	None	High	None	Moderate
Training metric nodes	All	Subset	All	Subset
Upfront time	Very high	High	High	Moderate
Inference time	Very slow	Faster	Slow	Faster
Scalability to large graphs	Poor	Limited	Good	Best
Multi-target inference cost	Linear	Fixed	Linear	Fixed

3.1 Predicted versus Explicit Structural Encodings

Beyond the reduction obtained by localizing structural computations, predicting positional encodings provides an additional advantage over using explicitly calculated structural metrics as downstream node features, as demonstrated by GPSE Cantürk et al. (2024). The key difference is that explicit encodings require the desired structural quantities to be recomputed whenever a new graph is encountered, whereas a pretrained encoder reduces this cost: after pretraining, structural information is obtained through a forward pass of the LPPE model rather than by repeatedly executing the original metric-specific algorithms.

This distinction is particularly important in the zero-shot setting. For an unseen target graph, explicitly computed encodings require running the full set of structural procedures needed by the downstream model before link prediction can begin. As with GPSE Cantürk et al. (2024), LPPE keeps structural target computations confined to pretraining; LPPE additionally computes those targets on bounded local subgraphs. At inference time, the pretrained and frozen encoder maps each PPR-induced neighborhood directly to a topology-aware encoding, without accessing or recomputing the structural targets themselves. Consequently, the target graph only requires neighborhood construction and model evaluation, rather than a new round of eigendecompositions, heat-kernel calculations, or cycle counting.

The inference-time ratings in Table 1 reflect this difference. Global explicit metrics can be very slow to recompute on each new graph. Local explicit metrics avoid full-graph calculations but still require extracting and processing a subgraph for each node, so their repeated inference cost remains substantial. Both global and local predicted encoders instead evaluate the trained model for each requested root.

Prediction also makes multi-target structural supervision substantially more efficient at inference, as demonstrated by GPSE Cantürk et al. (2024). When several explicit structural encodings are used together, each additional metric introduces another computation that must be performed on every required node or subgraph. Like GPSE, LPPE learns a shared latent representation from multiple structural objectives. Once the encoder has been trained, the same forward pass produces an embedding that can jointly capture information associated with all supervised targets. With the encoder

architecture fixed, inference cost does not grow linearly with the number of structural metrics used during pretraining.

A further benefit is that prediction decouples the amount of supervision required during training from the number of nodes that must receive encodings at inference. Explicit local encodings are exhaustive: if every node needs a structural feature, the corresponding local metrics must be calculated for every node. In LPPE, ground-truth structural targets are required only for the sampled roots used to train the encoder. The learned mapping can then be applied to all nodes, including nodes in previously unseen graphs. This converts structural metric computation from a repeated preprocessing requirement into a one-time source of supervision for learning a reusable encoder.

The metric-coverage ratings in Table 1 extend to global methods: direct global and local features require explicit metric values for every node that receives them, whereas both global and local predictors need ground-truth targets only for their training subset. With multiple targets, explicit methods also add a separate metric computation for each target at inference, whether on the full graph or on each local subgraph. Predicted methods share a fixed encoder pass across targets, so the cost of producing the embedding does not increase with the number of supervised metrics when the architecture is fixed.

Therefore, the benefit of predicted encodings is not merely that the underlying targets can be defined locally. Prediction changes when and how structural computation is paid for. Expensive metric generation is shifted to a sampled pretraining stage, while deployment uses a fixed neural encoder whose cost is independent of the number of structural targets being approximated. This property is especially attractive for graph foundation models, where the same representation mechanism must be applied repeatedly across many large and previously unseen graphs.

3.2 Complexity Analysis

Consider an observed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, let $N = |\mathcal{V}|$ and $M = |\mathcal{E}|$. LPPE constructs a PPR-induced neighborhood for each requested root, retaining at most 50 nodes. The active PPR method uses a fixed number of bounded-length random walks per root. Building the observed graph representation costs $\mathcal{O}(N + M)$, after which constructing each neighborhood takes constant work. LPPE applies a GPS encoder for the PPR subgraphs. Its depth, feature dimensions, and neighborhood size are fixed, so an encoder forward pass takes constant work per root.

Computing full-graph spectral PSEs can take $\mathcal{O}(N^3)$ time. For example, obtaining Laplacian eigenvectors or heat-kernel encodings through a dense eigendecomposition of an $N \times N$ graph Laplacian has cubic cost.

This explains the upfront-time and graph-size ratings in Table 1. Global explicit encodings can have very high upfront cost and poor large-graph scalability because full-graph spectral algorithms scale poorly. Global prediction moves these calculations to pretraining, but generating global targets and training the encoder still impose high upfront and pretraining costs, limiting scalability to large training graphs. Local explicit encodings avoid global calculations and their computations can be parallelized, which improves scalability; however, processing a subgraph for every node makes their aggregate upfront cost high. LPPE instead generates targets on sampled, bounded subgraphs and trains the encoder, giving it moderate pretraining and upfront costs in the table and avoiding the global target-generation bottleneck.

At inference, PPR neighborhoods and GPS encodings are computed for all N nodes. Structural targets are not computed. The total inference complexity is therefore

$$\mathcal{O}(N + M)\,. \tag{1}$$

Training additionally requires local structural targets. Let R_{train} be the number of selected training roots, and let T be the number of training epochs. Constructing their PPR neighborhoods costs $\mathcal{O}(N + M + R_{\text{train}})$. The most expensive targets have cubic cost in neighborhood size, so generating the targets once costs

$$\mathcal{O}\left(\sum_{v \in \mathcal{V}_{\text{train}}} |\mathcal{V}_v|^3\right) = \mathcal{O}\big(R_{\text{train}} \cdot 50^3\big) = \mathcal{O}(R_{\text{train}})\,. \tag{2}$$

GPS forward and backward passes over these roots cost $\mathcal{O}(TR_{\text{train}})$ across T epochs. Combining PPR construction, one-time local target generation, and GPS training gives

$$\mathcal{O}(N + M + R_{\text{train}} + TR_{\text{train}}) = \mathcal{O}(N + M + TR_{\text{train}}), \tag{3}$$

where the final equality uses $T \geq 1$. Thus, the simplified training bound includes local target computation.

By replacing full-graph spectral calculations with predictions from bounded PPR neighborhoods and a fixed-size GPS encoder, LPPE reduces the inference cost of producing these encodings from $\mathcal{O}(N^3)$ to $\mathcal{O}(N + M)$.

4 Empirical evaluation

We evaluate LPPE using 33 link-prediction datasets, examining structural-encoding baselines, within-dataset performance, single-dataset transfer, and the effect of training on more datasets. On the 18 held-out Link2 datasets, LPPE achieves a mean zero-shot average precision (AP) of 0.758 when trained on all 15 Link1 datasets, compared with 0.705 for explicitly computed local encodings. A model trained on just one dataset also transfers, though with lower mean AP (0.650 on Link2); increasing the number of training datasets improves zero-shot AP from 0.676 to 0.758, still below the 0.857 within-dataset non zero-shot reference.

4.1 Shared Experimental Setting

We evaluate LPPE using the 33 link-prediction datasets from AnyGraph (Xia & Huang, 2026): 15 *Link1* datasets used for taining, and 18 *Link2* datasets used as held-out test datasets. The dataset graphs are canonicalized as undirected, unweighted simple homogeneous graphs. User-item graphs are transformed into undirected bipartite graphs with explicit user and item partitions, which are retained for type-correct negative sampling.

For a dataset with positive splits, $\mathcal{G}_{\text{train}}$ contains only the training positives and $\mathcal{G}_{\text{train+val}}$ additionally contains the validation positives. LPPE supervision, LPPE checkpoint selection, and link-predictor training use $\mathcal{G}_{\text{train}}$ only. After validation selection, embeddings for final evaluation are produced from $\mathcal{G}_{\text{train+val}}$; test positives are never included in message passing, PPR construction, LPPE supervision, or embedding generation. For transfer, the target graph is handled in the same way, but the source LPPE and link predictor remain frozen and no target-side fitting, calibration, validation, or positional-encoding supervision is used. All completed experiments use fixed, filtered 1:1 binary negative edges and report average precision (AP), binary cross-entropy (BCE; lower is better), and ROC-AUC. We aggregate by an unweighted macro over datasets whenever a cohort contains multiple datasets. We use AP as the primary main-paper metric because it directly summarizes the ranking of positive edges; appendix C reports other metrics as well.

4.2 Comparison with Structural Encoding Baselines

Table 2: Zero-shot AP on the 18 held-out Link2 datasets after training on all 15 Link1 datasets. Values are mean ± sample standard deviation across three training seeds. Global methods did not complete because encoding computation timed out on large graphs.

Model	Structural encoding strategy	AP
Random	None	0.6672 ± 0.0149
Global PSE	Global explicit	—
GPSE	Global predicted	—
Local PSE	Local explicit	0.7045 ± 0.0047
LPPE	Local predicted	$\mathbf{0.7582 \pm 0.0224}$

We compare LPPE with alternative structural encoding strategies under the $K = 15$ transfer setting: models are trained on the 15 Link1 datasets and evaluated zero-shot on the 18 held-out Link2 datasets. The random-feature SGFormer uses no structural encoding, Local PSE computes local

encodings explicitly, and LPPE predicts them. Table 2 reports Link2 macro AP, averaged across the three training seeds.

These results support predicted local encodings as an effective source of structural information for zero-shot link prediction. Because the global encoding runs timed out, this experiment does not establish a performance comparison with Global PSE or GPSE.

4.3 Within-Dataset Evaluation on Link1 and Link2 Datasets

We evaluate the two-stage pipeline when both stages are trained and evaluated on the same dataset (trained on the training set and evaluated on the test set of the respective dataset). We run the model independently on every Link1 and Link2 dataset; Link2 is held out only for the transfer studies; its within-dataset values are treated as a reference for comparison with the later zero-shot result.

The model reaches macro AP of 0.841 on Link1 and 0.857 on Link2. Its corresponding ROC-AUC values are 0.846 and 0.863, respectively. These within-dataset Link2 results provide the reference to Figure 3: they should not be interpreted as a test performance goal, because unlike a transferred model, they train on the test dataset.

4.4 Single-Dataset Transfer

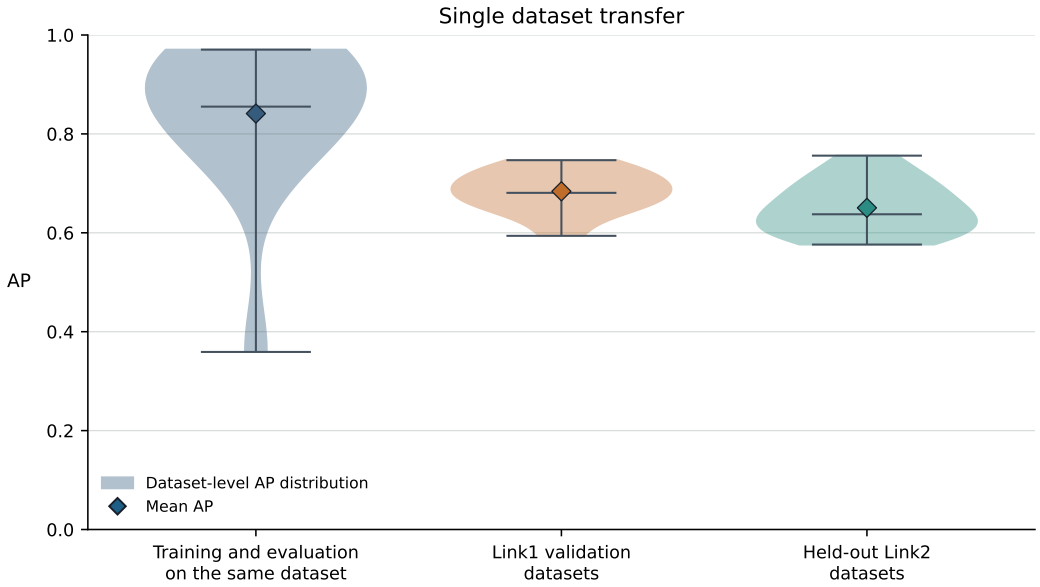


Figure 2: Transfer measured by AP. Training and evaluation on the same dataset is the within-dataset evaluation; Link1 validation datasets and held-out Link2 datasets are transfer zero-shot evaluations. Each violin summarizes the 15 trained models.

We ask whether a model trained on one Link1 dataset transfers without adaptation. We use each Link1 checkpoint from the within-dataset evaluation and evaluate it on all 33 datasets.

Figure 2 summarizes AP. Mean AP is 0.841 on the training dataset, 0.684 over the other 14 Link1 targets, and 0.650 over Link2. Hence, useful structural signal survives strict zero-shot transfer, but a sizeable gap to within-dataset performance remains.

4.5 Dataset Scalability

The dataset-scaling evaluation tests whether joint training on more datasets improves zero-shot transfer. It trains a new LPPE and link predictor for each nested Link1 prefix of size $K \in \{1, 2, 3, 4, 6, 8, 10, 12, 14, 15\}$, with equal dataset weighting. We use three prespecified dataset orderings and associated training seeds. Each task is evaluated on all of the Link2 datasets. A point

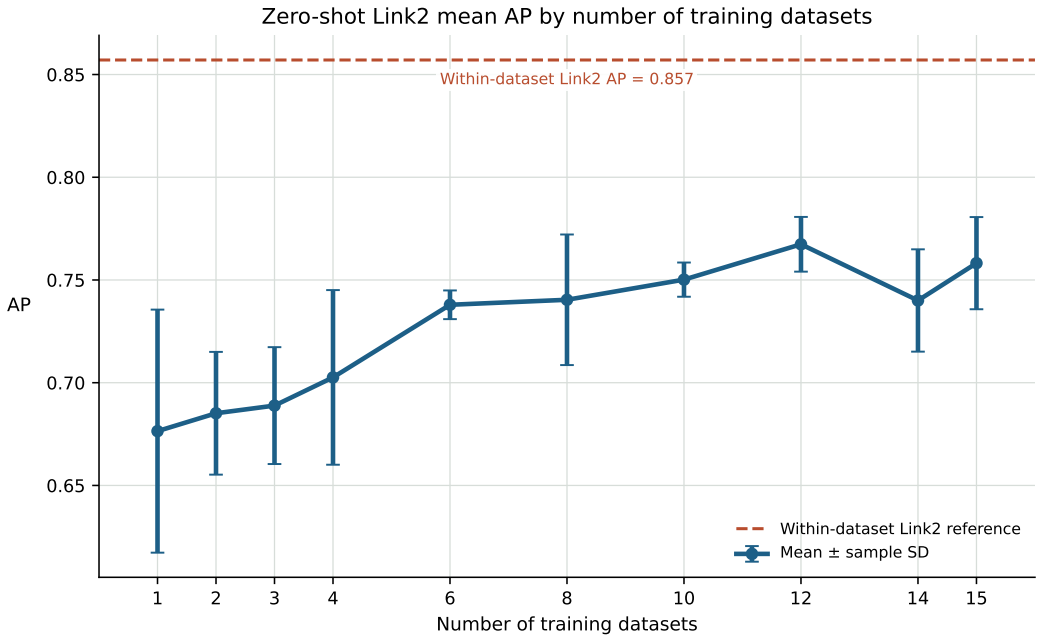


Figure 3: Datasets scaling measured by mean AP on the 18 held-out Link2 targets. Points are means across three ordering/training-seed tasks. The dashed line is the within-dataset Link2 performance.

on the scaling curve is the Link2 macro averaged over the three ordering-level macros; its error bar is their standard deviation.

Average precision increases from 0.676 ± 0.059 at $K = 1$ to 0.758 ± 0.022 at $K = 15$; ROC-AUC similarly rises from 0.676 ± 0.049 to 0.783 ± 0.022. Relative to the matched Link2 within-dataset reference, the AP gap decreases from 0.181 at $K = 1$ to 0.099 at $K = 15$ (a 45.3% reduction). The trend, together with the additional evidence in Appendix C.6, shows that training on multiple datasets improves average zero-shot performance on unseen graphs.

Figure 3 shows that zero-shot mean AP on the 18 held-out Link2 datasets generally improves as more Link1 datasets are used for training, rising from 0.676 at $K = 1$ to 0.758 at $K = 15$. The error bars show variation across the three dataset orderings and training seeds. Even at $K = 15$, performance remains below the 0.857 within-dataset reference, indicating that zero-shot performance does not match the performance of a model that was trained directly on test datasets.

5 Related Work

Learned positional and structural encodings. Graph Transformers commonly rely on positional or structural encodings to capture graph topology and distinguish structurally different nodes. Graph Positional and Structural Encoder (GPSE) is the closest methodological predecessor to our approach. GPSE trains an encoder from random node inputs to predict multiple hand-designed positional and structural targets, then reuses the learned hidden representations in downstream models (Cantürk et al., 2024). Our approach, LPPE, builds on this principle while aiming to improve upon GPSE and address additional fundamental challenges in learned positional encodings, particularly the scalability and locality of structural supervision. Rather than computing pretraining targets on each full pretraining graph, as GPSE does, LPPE computes its targets within bounded, root-centered PPR-induced subgraphs and exports the root representation as a reusable node encoding. In this way, LPPE computes its structural supervision on bounded subgraphs; GPSE learns to approximate PSEs computed on its pretraining graphs and reports efficient inference after pretraining.

Several recent methods provide direct alternatives to this design. PEARL generates positional encodings by processing random node inputs or standard basis vectors with a GNN and pooling the

outputs, avoiding a fixed dependence on explicit spectral decompositions (Kanatsoulis et al., 2025). CondPSE is particularly close to GPSE and LPPE; it applies learnable polynomial graph filters to Gaussian probes and uses conditional modulation to reconstruct structural targets (Lee & Park, 2026). GFSE learns transferable structural representations by combining shortest-path distance regression and motif counting with local community detection and graph-level contrastive learning; GPSE instead trains on a specified collection of positional and structural encoding targets (Chen et al., 2026; Cantürk et al., 2024).

Zero-shot graph foundation models and universal link prediction. A separate line of work studies transfer across graph domains. AnyGraph is the closest benchmark-level comparison for our setting because it directly targets graph foundation modeling and zero-shot link prediction across the dataset collection used in our experiments (Xia & Huang, 2026). Its transfer mechanism differs from LPPE, using a mixture of graph experts and selecting an expert based on scores computed for the input graph rather than a locally pretrained structural encoder. UniLP frames universal link prediction as an in-context learning problem, using examples of existing and non-existing links from the target graph to adapt its predictions to that graph's structural patterns (Dong et al., 2025). OpenGraph develops unified graph tokenization, scalable Transformer learning, and synthetic graph generation for cross-domain graph learning (Xia et al., 2024). TFMLinker casts cross-dataset link prediction as in-context inference with a tabular foundation model over topology-aware link representations (Liao et al., 2026). These approaches demonstrate variety of strategies for cross-dataset link predictions. In contrast, LPPE trains its structural encoder and link predictor on training graphs, freezes both at evaluation, and uses the permitted observed edges of an unseen target graph without using node features, updating model parameters, or needing any dataset-specific calibration.

Local structure for link prediction. The link-prediction literature provides evidence that local topology is informative. SEAL predicts candidate edges from their enclosing subgraphs, establishing local subgraphs as powerful representations for link prediction (Zhang & Chen, 2018). ELPH and BUDDY make local structural learning more scalable by replacing repeated subgraph materialization with structural sketches and precomputation (Chamberlain et al., 2023). This evidence motivates LPPE's emphasis on local structural encodings. A root-centered LPPE encoding captures the structural role of a node within its surrounding neighborhood and can be computed independently of any downstream prediction.

6 Conclusions

We introduced Local Predicted Positional Encoding (LPPE), a structure-only approach for zero-shot link prediction that learns from positional and structural targets computed on bounded, root-centered neighborhoods. Once trained, the frozen encoder produces node representations on unseen graphs without recomputing those targets or using node attributes. Under fixed neighborhood sampling and encoder settings, generating encodings for all nodes has $\mathcal{O}(N + M)$ complexity in the number of nodes and observed edges, which improves on the cubic scaling $\mathcal{O}(N^3)$ of full-graph metric calculations.

On 18 held-out test datasets, LPPE achieved a mean zero-shot average precision (AP) of 0.758, compared with 0.705 for explicitly computed local structural encodings. Increasing the number of training datasets from 1 to 15 raised mean AP from 0.676 to 0.758, closing 45.3% of the gap to the 0.857 within-dataset reference. These findings suggest that local structural supervision can support scalable, transferable graph representations, and that training across more graphs improves their transfer. The remaining gap motivates experiments with a larger and more diverse collection of training datasets to test whether zero-shot performance continues to scale.

AI USE STATEMENT

The authors developed the LPPE pipeline, including the use of local subgraphs, and determined the research questions, experimental design, and conclusions. We used generative AI to help formulate and write the paper's complexity claims based on our analysis; to assist with experimental code and dataset preprocessing; and to generate figures specified by the authors. We also used it to identify related literature and improve the phrasing and readability of the manuscript. We did not use generative AI to generate synthetic data, interpret experimental results, or draw conclusions. All numerical results reported in the tables came from the experiment scripts and were not directly processed by an LLM. The authors vetted the full manuscript, including AI-assisted text and figures, and checked literature suggested by AI. We take responsibility for the final content of this work, including all text, claims, code, and figures produced with AI assistance.

REPRODUCIBILITY STATEMENT

The main text describes the LPPE pipeline and evaluation setting. Implementation details are provided in Appendix B and Appendix C.1, and the appendix also documents the datasets, splits, training protocol, and evaluation procedure. Code for reproducing the experiments is available in the anonymized GitHub repository: `https://anonymous.4open.science/r/anonymous-submission-5687/README.md`

ACKNOWLEDGMENTS

We gratefully acknowledge the icon creators (meaicon, Magnific, and Mihimihi) for providing graphic resources via Flaticon, which were used in Figure 1.

REFERENCES

Semih Cantürk, Renming Liu, Olivier Lapointe-Gagné, Vincent Létourneau, Guy Wolf, Dominique Beaini, and Ladislav Rampášek. Graph positional and structural encoder. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5533–5566. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/canturk24a.html.

Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Yannick Hammerla, Michael M. Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=m1oqEOAozQU.

Jialin Chen, Haolan Zuo, Haoyu Wang, Siqi Miao, Pan Li, and Rex Ying. Towards a universal graph structural encoder. In *Proceedings of the ACM Web Conference 2026*, WWW '26, pp. 1457–1468, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723070. doi: 10.1145/3774904.3792656. URL https://doi.org/10.1145/3774904.3792656.

Kaiwen Dong, Haitao Mao, Zhichun Guo, and Nitesh V Chawla. Universal link predictor by in-context learning on graphs. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=EYpqmoejB8.

Charilaos Kanatsoulis, Evelyn Choi, Stefanie Jegelka, Jure Leskovec, and Alejandro Ribeiro. Learning efficient positional encodings with graph neural networks. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 36125–36152, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/59883a1524bf58cf2dd86d0fc8d499e2-Paper-Conference.pdf.

Woohyun Lee and Hogun Park. CondPSE: A polynomial-filtered structural encoder with conditional modulation for graphs. In *The 2nd Frontiers in Graph Machine Learning for the Large Model Era*, 2026. URL https://openreview.net/forum?id=Nj5RtcLguI.

Tianyin Liao, Chunyu Hu, Yicheng Sui, Xingxuan Zhang, Peng Cui, Jianxin Li, and Ziwei Zhang. TFMLinker: Universal link predictor by graph in-context learning with tabular foundation models. In *Workshop on Graph Foundation Models: A New Era for Graph Machine Learning*, 2026. URL https://openreview.net/forum?id=wukGf8u9Um.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 14501–14515. Curran Associates, Inc., 2022. doi: 10.52202/068431-1054. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/5d4834a159f1547b267a05a4e2b7cf5e-Paper-Conference.pdf.

Zehong Wang, Zheyuan Liu, Tianyi Ma, Jiazheng Li, Zheyuan Zhang, Xingbo Fu, Yiyang Li, Zhengqing Yuan, Wei Song, Yijun Ma, et al. Graph foundation models: A comprehensive survey. *arXiv preprint arXiv:2505.15116*, 2025.

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. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 64753–64773. Curran Associates, Inc., 2023. doi: 10.52202/075280-2826. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/cc57fac10eacadb3b72a907ac48f9a98-Paper-Conference.pdf.

Lianghao Xia and Chao Huang. AnyGraph: Graph foundation model in the wild. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 882–896, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.44. URL https://aclanthology.org/2026.findings-acl.44/.

Lianghao Xia, Ben Kao, and Chao Huang. OpenGraph: Towards open graph foundation models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2365–2379, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.132. URL https://aclanthology.org/2024.findings-emnlp.132/.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/53f0d7c537d99b3824f0f99d62ea2428-Paper.pdf.

A Additional Complexity Analysis

A.1 Complexity Derivation

This appendix derives the inference and training complexity bounds in Section 3.2 and compares full-graph and local target-generation costs. Consider an observed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes and $M = |\mathcal{E}|$ edges. Let R be the number of roots processed and K the maximum number of nodes retained in each PPR-induced neighborhood. In our configuration, we set $K = 50$ to limit the cost of local target generation and encoding. GPSE's results on small graphs motivate this design choice (Cantürk et al., 2024).

PPR neighborhood construction. The active method builds a graph representation in $\mathcal{O}(N + M)$ time and estimates PPR endpoint scores from W random walks per root. The bounds below assume that, after preprocessing, adjacency membership can be checked in $\mathcal{O}(1)$ time. Hash-based adjacency sets provide this lookup bound in expectation. Each walk has restart probability α and at most L transitions. Roots are processed in chunks of at most B. Across R roots, sampling requires

$\mathcal{O}(RW(1+L))$ work when each walk takes at most L transitions. The expected number of transitions per walk is at most $(1-\alpha)/\alpha$, giving expected sampling work of $\mathcal{O}(RW[1+(1-\alpha)/\alpha])$, including the cost of initiating and recording each walk.

Aggregating and ranking sampled endpoints has a sorting-based bound of $\mathcal{O}(RW\log(BW))$. Retaining at most K nodes per root and storing their induced neighborhood records contributes $\mathcal{O}\big(RK^2\big)$ under the stated adjacency-lookup assumption. Hence, a work bound for constructing the requested neighborhoods, with at most L transitions per walk, is

$$C_{\mathrm{PPR}}(R) = \mathcal{O}\big(N + M + RW\left[1 + L + \log(BW)\right] + RK^2\big)\,. \tag{4}$$

For expected sampling work, L in this expression is replaced by $(1-\alpha)/\alpha$. The active configuration fixes $W = 100{,}000$, $L = 86$, $\alpha = 0.15$, $B = 250$, and $K = 50$. Treating these configured quantities as constants gives $C_{\mathrm{PPR}}(R) = \mathcal{O}(N + M + R)$.

GPS encoder. LPPE uses a D-layered GPS encoder with input dimension f, hidden dimension h, and output dimension o. Consider a batch of b PPR neighborhoods with $\mathcal{V}_b$ real nodes and E_b directed local edges in total. Let S be the largest neighborhood size in the batch, so $S \le K$. In one GPS layer, the local GCN branch costs $\mathcal{O}\big(\mathcal{V}_b h^2 + E_b h\big)$. The attention projections cost $\mathcal{O}\big(bSh^2\big)$, and attention within each neighborhood costs $\mathcal{O}\big(bS^2h\big)$. Packed neighborhoods attend independently; attention does not operate over all $\mathcal{V}_b$ nodes as one graph.

Including the input and output projections, one encoder forward pass on this batch costs

$$\mathcal{O}\big(\mathcal{V}_b fh + D\left[\mathcal{V}_b h^2 + E_b h + bSh^2 + bS^2 h\right] + \mathcal{V}_b ho\big)\,. \tag{5}$$

Each neighborhood has at most K nodes and $\mathcal{O}\big(K^2\big)$ directed edges. Summing over R roots therefore yields

$$C_{\mathrm{GPS}}(R) = \mathcal{O}\big(R\left[Kfh + D\left(Kh^2 + K^2h\right) + Kho\right]\big)\,. \tag{6}$$

For a selected model, K, f, h, o, and D are fixed, so the encoder forward pass costs $\mathcal{O}(R)$. The backward pass has the same asymptotic work.

Inference complexity. At inference, LPPE constructs a PPR neighborhood and computes a GPS embedding for every node, so $R = N$. Structural targets and backward passes are not required. Combining Equations 4 and 6, with the configured PPR and model parameters held fixed, gives

$$C_{\mathrm{infer}} = \mathcal{O}(N + M + N) = \mathcal{O}(N + M)\,. \tag{7}$$

Training complexity. Let R_{train} denote the selected roots used for LPPE supervision, including the fixed-fraction validation split, and let T denote the number of training epochs. Their PPR neighborhoods are constructed once, at cost $\mathcal{O}(N + M + R_{\mathrm{train}})$ with the configured PPR parameters.

Local structural targets are also generated once. Let V_v denote the PPR-induced neighborhood rooted at v. The most expensive targets have cubic cost in the neighborhood size, so generating targets for the selected roots costs

$$C_{\mathrm{targets}}(R_{\mathrm{train}}) = \mathcal{O}\left(\sum_{v\in\mathcal{V}_{\mathrm{train}}} |\mathcal{V}_v|^3\right) = \mathcal{O}\big(R_{\mathrm{train}}K^3\big) = \mathcal{O}(R_{\mathrm{train}})\,, \tag{8}$$

where the last equality holds because $K = 50$ is fixed. GPS forward and backward passes over the selected roots cost $\mathcal{O}(TR_{\mathrm{train}})$ across T epochs. Combining the one-time PPR construction, the one-time local target computation, and repeated GPS training gives

$$\begin{aligned} C_{\mathrm{train}} &= \mathcal{O}(N + M + R_{\mathrm{train}} + R_{\mathrm{train}} + TR_{\mathrm{train}}) \\ &= \mathcal{O}(N + M + TR_{\mathrm{train}})\,, \qquad T \ge 1. \end{aligned} \tag{9}$$

Thus, the simplified training expression includes local target generation; its one-time linear term is absorbed by the GPS training term.

Full-graph metrics and local target generation. The most expensive structural targets include spectral quantities for which dense eigendecomposition has cubic complexity in the matrix dimension. Using dense eigendecomposition to calculate such a target on the full graph costs

$$\mathcal{O}\big(N^3\big)\,. \tag{10}$$

During LPPE training, targets are instead calculated on selected PPR-induced neighborhoods. If targets were calculated for every root, their total cost would be

$$\mathcal{O}\left(\sum_{v\in\mathcal{V}} |\mathcal{V}_v|^3\right). \tag{11}$$

Let $s = \max_{v\in\mathcal{V}} |\mathcal{V}_v|$. Since

$$\sum_{v\in\mathcal{V}} |\mathcal{V}_v|^3 \le Ns^3, \tag{12}$$

the all-root local target-generation cost is bounded by

$$\mathcal{O}\big(Ns^3\big)\,. \tag{13}$$

Here $s \le 50$, so this bound is $\mathcal{O}\big(N \cdot 50^3\big)$. Actual LPPE training may use fewer than N roots. At inference, LPPE does not calculate these structural targets. As derived in Equation 7, constructing PPR neighborhoods and producing GPS embeddings for all nodes costs $\mathcal{O}(N + M)$ with the configured sampling and encoder parameters held fixed.

Alternative PPR construction by local push. The sampling method above uses W random walks per root to estimate PPR scores, with $W = 100{,}000$. Approximate local push offers an alternative that avoids this high fixed walk count. Compared with Equation 4, this replaces the per-root sampling and ranking term $W[1 + L + \log(BW)]$ with a tolerance-dependent term. Both methods yield $\mathcal{O}(N + M)$ all-root inference complexity when their respective parameters and the encoder dimensions are fixed.

B Method details

B.1 Local Predicted Positional Encoding

Local Predicted Positional Encoding (LPPE) learns a structural representation from small, root-centred graph views, without using the input node attributes. For each LPPE-supervision root r (and every root at inference) in an observed graph $\mathcal{G}$, we compute a personalized PageRank (PPR) score vector with restart probability $\alpha = 0.15$. We retain at most 50 vertices, always including r, ranked by descending PPR score with node ID as the deterministic tie-breaker. The LPPE input graph $\mathcal{G}_r^{\mathrm{PPR}}$ is the subgraph induced by those retained vertices. PPR scores are estimated by GPU Monte-Carlo random walks (with $100{,}000$ walks per root and maximum walk length 86). The root's index in the PPR-ranked local vertex list is recorded explicitly; it is not assumed to be zero.

LPPE receives no node IDs, original features, or externally supplied positional encodings. Instead, each vertex of each local graph receives an independent Gaussian feature vector $x_v \sim \mathcal{N}(0, I_d)$. These features are resampled during training.

The encoder first projects the random features to a hidden representation and then applies a GPS-style stack Rampášek et al. (2022). Each layer sums a local GCN convolution and the projected multi-head global self-attention output, applies dropout, adds this result to the layer input, and applies LayerNorm. Global attention is masked to the individual PPR graph. The final normalized node states are linearly projected to the LPPE embedding dimension. The encoder uses three layers, hidden and output widths of 64, and four attention heads. The backbone uses GELU activations and 0.1 dropout (including attention dropout); the compact model uses the same normalization and residual arrangement.

For a local graph, the root representation is the projected node state at the stored root index, and the graph representation is the mean of all local projected node states. Each PSE is optimized with its masked regression objective; target losses are averaged over enabled targets and combine equally weighted MAE and cosine components, as specified in Appendix C.1.

LPPE root splits are made separately for every dataset. Roots are stratified by deciles of $\log(1 + \deg_{\mathcal{G}}(r))$, shuffled with the experiment seed within strata, and split into 90% training and 10% validation roots, with safeguards to leave a validation root whenever possible. Target normalisation statistics (mean and standard deviation) are estimated only from the LPPE training roots and then applied unchanged to validation roots. The PPR subgraphs and positional targets used for LPPE supervision are both computed on $\mathcal{G}_{\text{train}}$, preventing validation or test edges from entering the pretraining task.

Checkpoint selection. Validation is run after every LPPE epoch, and the sole retained checkpoint (`best_lppe.pt`) is replaced only when the total validation loss is strictly lower than the previous best. For joint multi-dataset training, the loss is first averaged over the validation roots of each dataset and then averaged uniformly across datasets, rather than pooled over all roots. Training stops after the configured number of consecutive non-improving validation epochs; downstream embeddings always use the retained best checkpoint, not necessarily the final training epoch.

Large walks number. Although $W = 100{,}000$ walks per root is a large amount, the walks are independent and can be processed in GPU batches rather than sequentially. The implementation keeps multiple walks on the GPU and advances active them through vectorized sampling and masking operations; endpoint counts are also computed on the GPU before CPU extraction of the top-50 subgraphs. Meaning that GPU batching makes it more manageable in practice without changing the $O(N + M)$ all-root bound when W and the other sampling parameters are fixed.

B.1.1 Structural and Positional Targets

All six targets are computed anew on $\mathcal{G}_r^{\text{PPR}}$, not on the original full graph. Let A be its undirected adjacency matrix, D its degree matrix, $L = I - D^{-1/2}AD^{-1/2}$ its symmetric normalized Laplacian, and let (λ_j, u_j) denote the nontrivial eigenpairs of L. Eigenvalues whose absolute value is at most 10^{-8} are removed. This both excludes the trivial zero modes and supplies a defined convention for disconnected local graphs.

- **ElstaticPE** is a seven-dimensional node-level target. From the off-diagonal normalized-Laplacian pseudoinverse $Q = L^{\dagger} - \text{diag}(L^{\dagger})$, it records at the root the minimum, mean, and standard deviation of its potential column; the minimum and standard deviation of the corresponding row; and the mean potential over its neighbours in each direction. Because the local graphs are undirected, Q is symmetric: the row and column minima coincide, as do their standard deviations and the two neighbour means. A root with no local neighbours uses zero for both neighbour aggregates.
- **LapPE** is a four-dimensional node-level target comprising the absolute values of the first four retained eigenvectors at the root. Taking absolute values removes the arbitrary eigenvector sign, and each eigenvector column is ℓ_2-normalized across local vertices before the root value is read. Missing eigenvectors are padded with zeros.
- **RWSE** is a 20-dimensional node-level target. With $P = D^{-1}A$ and no added self-loops, it contains the root return probabilities $(P^k)_{\text{root,root}}$ for $k = 2, \ldots, 21$, where root denotes the stored local root index.
- **HKdiagSE** is a 20-dimensional node-level target. At times $t = 1, \ldots, 20$, its entries are the root diagonal heat-kernel values $\sum_j u_j(\text{root})^2 \exp(-t\lambda_j)$. All retained eigenpairs are used; no spectral approximation is applied.
- **EigValSE** is a four-dimensional graph-level target: the first four retained eigenvalues of L. It is zero-padded when a local graph has too few nontrivial eigenvalues.
- **CycleSE** is a three-dimensional graph-level target containing the GPSE/GraphGPS cycle features at lengths 2, 3, and 4, where the length-2 feature equals the undirected-edge count.

Node-level targets (ElstaticPE, LapPE, RWSE, and HKdiagSE) are predicted from the root representation; graph-level targets (EigValSE and CycleSE) are predicted from the mean-pooled local representation.

B.2 Frozen Embeddings and Link Predictor

After LPPE checkpoint selection, the selected checkpoint is put in evaluation mode and every parameter is frozen. We construct a root-local PPR graph for every node in each downstream observed graph and run the frozen encoder on every root-local graph, packed as disconnected batches for inference. The resulting root embeddings are stored in original node-ID order, giving exactly one LPPE vector per node. Positional targets are not used during this stage. Link-predictor training and validation use embeddings generated from $\mathcal{G}_{\text{train}}$ and message passing on $\mathcal{G}_{\text{train}}$; final test uses embeddings and message passing on $\mathcal{G}_{\text{train+val}}$. Test edges are never present in the observed graph.

The downstream encoder is PyG's SGFormer Wu et al. (2023). It maps the frozen LPPE inputs to node states through its scalable transformer approximation and a GNN branch; the two branches are combined with configured graph weight 0.5 and additive aggregation. It uses 64-dimensional inputs, hidden states, and outputs, four transformer heads, and two GNN layers. Training uses `LinkNeighborLoader` with directed two-hop neighbourhood sampling and fanouts $[20, 10]$, so the method remains practical on graphs for which full-graph transformer attention is not. This downstream architecture differs from the bounded GPS LPPE encoder: LPPE operates on at most 50 nodes per root, whereas link prediction propagates over an observed graph that may be much larger.

For a candidate pair (u, v) with SGFormer states h_u, h_v, the decoder forms the feature vector

$$[h_u \,\|\, h_v \,\|\, |h_u - h_v| \,\|\, h_u \odot h_v]$$

and maps it to one logit with an MLP. Its hidden widths are 128 and 32, with GELU and 0.1 dropout. To make the endpoint concatenation order-invariant, endpoints are first sorted by their original node IDs. Thus exchanging u and v leaves the link logit unchanged.

Link training is binary classification with `BCEWithLogitsLoss` over positive split edges and one fixed, filtered negative per positive. Negatives are excluded if they occur in the union of train, validation, or test edges; homogeneous graphs use unordered non-self pairs, while bipartite graphs use only valid user–item pairs. The checkpoint is selected by validation BCE; binary metrics are evaluated exhaustively on the selected test protocol.

C Additional Experimental Results

C.1 Reproducibility Protocol

C.1.1 Datasets, Graph Construction, and Splits

All datasets are taken from the AnyGraph collection (Xia & Huang, 2026). The official training group is Link1: `amazon-book`, `amazon_textfeat`, `citation-2019`, `citation-classic`, `citeseer`, `email-Enron`, `p2p-Gnutella06`, `ppa`, `products_home`, `products_tech`, `pubmed`, `soc-Epinions1`, `steam_textfeat`, `yelp2018`, and `yelp_textfeat`. The held-out target group is Link2: `CS`, `Fitness`, `Goodreads`, `Photo`, `arxiv`, `arxiv-ta`, `collab`, `cora`, `ddi`, `gowalla`, `ml10m`, `ml1m`, `proteins_spec0`, `proteins_spec1`, `proteins_spec2`, `proteins_spec3`, `roadNet-PA`, and `web-Stanford`. The collection has 33 datasets: 15 Link1 datasets and 18 Link2 datasets.

For each dataset, the supplied positive matrices `trn_mat.pkl`, `val_mat.pkl`, and `tst_mat.pkl` define $\mathcal{E}_{\text{train}}$, $\mathcal{E}_{\text{val}}$, and $\mathcal{E}_{\text{test}}$. We construct $\mathcal{G}_{\text{train}} = (\mathcal{V}, \mathcal{E}_{\text{train}})$ and $\mathcal{G}_{\text{train+val}} = (\mathcal{V}, \mathcal{E}_{\text{train}} \cup \mathcal{E}_{\text{val}})$. LPPE training and validation, link-model training, and link-model validation use message passing on $\mathcal{G}_{\text{train}}$ only. The final test is run once with embeddings inferred from $\mathcal{G}_{\text{train+val}}$; no test edge is visible to message passing, PPR construction, positional-encoding supervision, checkpoint selection, or model fitting. For transfer and dataset-scaling targets, this final-graph construction is retained, but both stages trained on the Link1 dataset are frozen and no target label is used for adaptation. LPPE validation uses held-out PPR roots from $\mathcal{G}_{\text{train}}$; link-model training uses $\mathcal{E}_{\text{train}}$ candidates and checkpoint selection uses $\mathcal{E}_{\text{val}}$ candidates, both with $\mathcal{G}_{\text{train}}$ embeddings. Only the final test scores $\mathcal{E}_{\text{test}}$ candidates with $\mathcal{G}_{\text{train+val}}$ embeddings.

Table 3: Model configurations. LPPE tuples are (number of layers, hidden width, output width, heads); SGFormer tuples are (hidden width, number of layers, heads; GNN layers). Both models use four attention heads where shown and the stated pair-decoder hidden widths.

Configuration	Random features	LPPE	LPPE embedding width	SGFormer	Pair decoder
Compact-capacity	16	$(2, 28, 32, 4)$	32	$(32, 2, 4; 2)$	$[64, 16]$
Standard-capacity	16	$(3, 64, 64, 4)$	64	$(64, 2, 4; 2)$	$[128, 32]$

C.1.2 Candidates, Metrics, and Aggregation

For every split, seeded sampling supplies one binary negative per positive. Candidates are sampled only after filtering against the union $\mathcal{E}_{\text{train}} \cup \mathcal{E}_{\text{val}} \cup \mathcal{E}_{\text{test}}$. Homogeneous negatives are unordered non-self pairs; bipartite negatives are restricted to user–item pairs. The resulting 1:1 candidate sets are fixed for training, validation, and test, and validation/test scoring is exhaustive over the respective candidate set.

The evaluation metrics are calculated from logits: BCE is the mean logistic loss, and ROC-AUC uses average ranks for ties. AP sums precision weighted by recall increments at distinct score thresholds, with ties grouped, without interpolation. A dataset cohort is summarized by an unweighted arithmetic macro. In single-dataset transfer, we first average over targets for each training dataset and then average those dataset means. In dataset scaling, we first average the 18 Link2 targets for each task and then report the mean and sample standard deviation over the three ordering-level task macros.

C.1.3 Two-Stage Models and Training

LPPE is a GPS backbone with GCN local message passing, Transformer global attention, LayerNorm, residual connections, GELU activations, and dropout 0.1. In each layer, the GCN output and projected global-attention output are added, dropout is applied, and this result is added to the layer input before LayerNorm. The compact-capacity model uses the same arrangement. It receives 16-dimensional standard-normal random node features; features are resampled during LPPE training but deterministic for validation and embedding inference. The downstream link predictor is the scalable SGFormer with a two-layer GNN component and a pair decoder that concatenates ordered endpoint representations, their absolute difference, and their product. We call the configuration used in the main paper the *standard-capacity* model; the *compact-capacity* model is the reduced-width/depth ablation.

For LPPE, approximate GPU-Monte-Carlo PPR produces induced, top-50-node subgraphs with restart probability of 0.15. At most 100,000 seeded G_{train} roots per dataset are used for supervision, split per dataset into 90% training and 10% validation roots using ten log-degree strata. LPPE predicts normalized structural targets from six types taken from Cantürk et al. (2024) (with slight modifications): Elstatic (7 dimensions), LapPE (4), RWSE at steps 2–21 (20), HKdiag at times 1–20 (20), eigenvalues (4), and graph-level CycleSE at lengths 2–4 (3). Within each target type MAE and cosine-similarity loss also receive equal weight. All-root PPR inference is then used to produce frozen node embeddings for the link prediction stage.

For a target type f and root b, valid training-target entries are standardized with a single mean and standard deviation for that type, $z_{bfd} = (y_{bfd} - \mu_f)/\max(\sigma_f, 10^{-8})$; the statistics pool all valid coordinates and training roots of that type, rather than being estimated separately per coordinate. With width D_f and mask $m_{bfd} \in \{0, 1\}$, the per-root loss for that type is

$$\ell_{bf} = \frac{\sum_d m_{bfd} |\hat{z}_{bfd} - z_{bfd}|}{D_f} + \mathbf{1}\left[\max_d m_{bfd} = 1\right] \left(1 - \frac{\hat{z}_b \cdot z_b}{\max(\|\hat{z}_b\|_2, 10^{-8}) \max(\|z_b\|_2, 10^{-8})}\right),$$

where the cosine term is evaluated over the complete vector for that type (the mask only gates the term); the LPPE loss is the arithmetic mean of ℓ_{bf} over the six enabled types and then over roots in a packed batch. For the PSE targets, every target mask is all-true, including zero-padded LapPE and EigVal coordinates: zero padding is therefore supervised as a zero target and contributes to both MAE and the full-vector cosine loss. More generally, a type with no valid entries contributes zero. A zero vector has cosine similarity zero under the 10^{-8} safeguard and hence cosine loss one when the target type is valid. The same training-only normalization is fixed for validation.

Table 4: Within-dataset results. Values are unweighted macros over datasets in the stated group; BCE is lower-is-better.

Group	Configuration	AP	ROC-AUC	BCE
Link1 ($n = 15$)	Compact-capacity	0.801	0.808	0.520
Link1 ($n = 15$)	Standard-capacity	0.841	0.846	0.525
Link2 ($n = 18$)	Compact-capacity	0.837	0.842	0.457
Link2 ($n = 18$)	Standard-capacity	0.857	0.863	0.469

Both stages use AdamW with learning rate 10^{-3}, weight decay 10^{-4}, and gradient-norm clipping at 1. The within-dataset runs use packed LPPE batches of 512; link training uses two-hop neighbor fanouts $[20, 10]$, FP32, at most 40 epochs, patience 8, and at most 2,048 deterministic without-replacement link mini-batches per epoch. The physical link batch size is 32,768 for the compact-capacity model and 24,576 for the standard-capacity model; validation and final test remain exhaustive. LPPE uses at most 100 epochs and patience 20 in both within-dataset and joint training. The single-dataset transfer experiment performs no optimization: it reuses the corresponding within-dataset LPPE/link-model pair exactly.

The dataset scaling experiment trains new standard-capacity models for every task. Each joint stage uses an epoch-local shuffled round-robin schedule with 1,000 steps per epoch: it repeatedly shuffles all selected dataset names using the training seed and epoch, truncates the resulting sequence to 1,000 entries, reinitializes every loader at the epoch boundary, and restarts a loader when it is exhausted. Thus each selected Link1 dataset contributes either $\lfloor 1000/K \rfloor$ or $\lceil 1000/K \rceil$ batches. For bounded link training, the seeded shuffled sampler creates a fresh deterministic permutation of training labels for each epoch; the first at most 2,048 physical mini-batches are used and the sampler generator state advances between epochs. The LPPE checkpoint minimizes the unweighted dataset-macro validation total loss; after freezing it, the link checkpoint minimizes the unweighted dataset-macro validation BCE. Best checkpoints are retained by atomic replacement. The joint link stage uses the same 40-epoch/patience-8 limit as the within-dataset runs.

C.1.4 Seeds, Determinism, and Execution Optimizations

The within-dataset and single-dataset transfer evaluations use seed 42 for every dataset–configuration pair. Dataset scaling fixes ordering/training-seed pairs $(101, 42)$, $(202, 43)$, and $(304, 44)$, and its nested dataset counts are $\{1, 2, 3, 4, 6, 8, 10, 12, 14, 15\}$. The corresponding ordered Link1 lists are: 101: `products_tech`, `yelp2018`, `citation-classic`, `products_home`, `email-Enron`, `ppa`, `amazon-book`, `steam_textfeat`, `citeseer`, `amazon_textfeat`, `citation-2019`, `yelp_textfeat`, `p2p-Gnutella06`, `soc-Epinions1`, `pubmed`; 202: `steam_textfeat`, `p2p-Gnutella06`, `pubmed`, `ppa`, `yelp2018`, `citation-classic`, `products_tech`, `yelp_textfeat`, `amazon-book`, `products_home`, `email-Enron`, `citeseer`, `soc-Epinions1`, `amazon_textfeat`, `citation-2019`; and 304: `citation-2019`, `citation-classic`, `p2p-Gnutella06`, `products_home`, `ppa`, `amazon-book`, `yelp2018`, `citeseer`, `amazon_textfeat`, `pubmed`, `steam_textfeat`, `products_tech`, `soc-Epinions1`, `email-Enron`, `yelp_textfeat`. Each nested task uses the first K entries of its list. Python, NumPy, and PyTorch are seeded; deterministic PyTorch algorithms are enforced; cuDNN benchmarking is disabled; and `CUBLAS_WORKSPACE_CONFIG=:4096:8` is set. Negative sampling uses a fixed seed offset of 10,000 and bounded link-training sampling an offset of 30,000.

C.2 Within-Dataset Binary Metrics

Table 4 gives the unweighted dataset macros for the within-dataset evaluation.

C.3 Per-Dataset Transfer

Figure 4 shows that zero-shot evaluation on other Link1 datasets and held-out Link2 datasets generally yields lower AP, and ROC-AUC, and higher BCE, than training and evaluation on the same dataset.

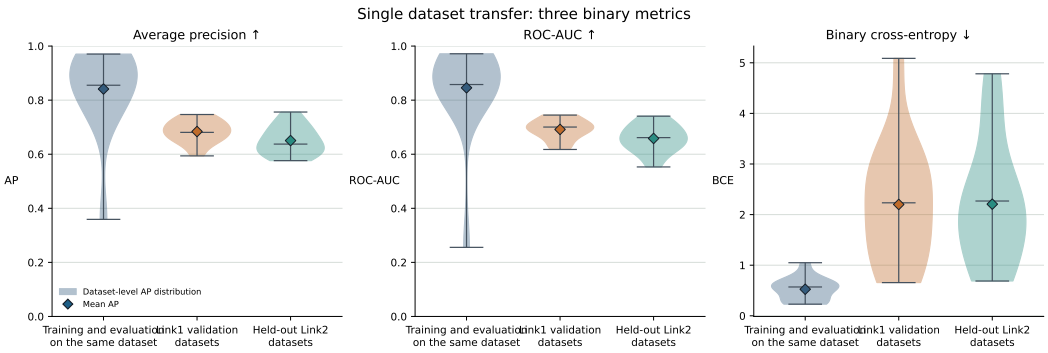


Figure 4: Standard-capacity-model transfer across dataset groups for each evaluation metric.

C.4 Model-Capacity Ablation

In the paired Link2 comparison in Figure 5, the training dataset, evaluation dataset, and candidate edges are held fixed; only model capacity (size) changes. The average changes (standard-capacity model performance minus compact-capacity model performance) are $+0.023$ AP, $+0.025$ ROC-AUC, and $+0.405$ BCE. Unlike the other 2 metrics, the BCE difference does not show an improvement in probability calibration.

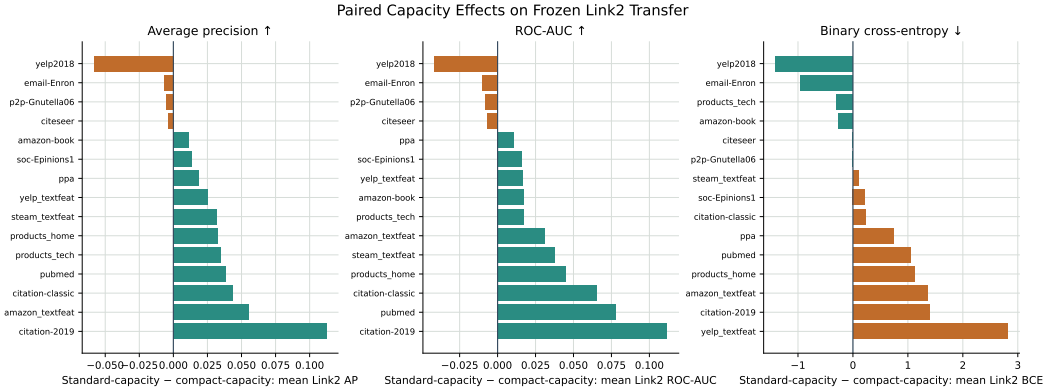


Figure 5: Paired standard-capacity-minus-compact-capacity effects on Link2 transfer, averaged across the 18 targets separately for each of the 15 datasets. Positive values favor the standard-capacity model for AP, and ROC-AUC; negative values favor it for BCE.

Table 5: Zero-shot performance on the 18 held-out Link2 datasets after training on all 15 Link1 datasets. Values are mean ± sample standard deviation across three training seeds. Global PSE and GPSE did not complete because encoding computation timed out on large graphs. Higher AP and ROC-AUC and lower BCE are better.

Model	Structural encoding strategy	AP	ROC-AUC	BCE
Random-feature SGFormer	None	0.6672 ± 0.0149	0.6895 ± 0.0136	$\mathbf{0.9688 \pm 0.1278}$
Global PSE	Global explicit	—	—	—
GPSE	Global predicted	—	—	—
Local PSE	Local explicit	0.7045 ± 0.0047	0.7127 ± 0.0074	5.2724 ± 1.3240
LPPE	Local predicted	$\mathbf{0.7582 \pm 0.0224}$	$\mathbf{0.7825 \pm 0.0224}$	1.4936 ± 0.2279

C.5 Full Dataset-Scaling Curve

Table 6 reports the dataset-scaling Link2 macro at every dataset count. Each entry is the mean ± sample standard deviation across the three ordering/training-seed tasks, after taking an unweighted

Table 6: Standard-capacity-model Link2 scaling. Values are mean ± sample SD across three ordering/training-seed tasks. BCE is lower-is-better.

K	AP	ROC-AUC	BCE
1	0.676 ± 0.059	0.676 ± 0.049	3.441 ± 1.789
2	0.685 ± 0.030	0.696 ± 0.041	2.041 ± 0.700
3	0.689 ± 0.028	0.707 ± 0.005	1.690 ± 0.095
4	0.703 ± 0.042	0.722 ± 0.057	1.732 ± 0.478
6	0.738 ± 0.007	0.748 ± 0.024	1.684 ± 0.459
8	0.740 ± 0.032	0.755 ± 0.033	1.369 ± 0.594
10	0.750 ± 0.008	0.771 ± 0.016	1.678 ± 0.263
12	0.767 ± 0.013	0.791 ± 0.010	1.665 ± 0.446
14	0.740 ± 0.025	0.760 ± 0.022	1.934 ± 1.166
15	0.758 ± 0.022	0.783 ± 0.022	1.494 ± 0.228

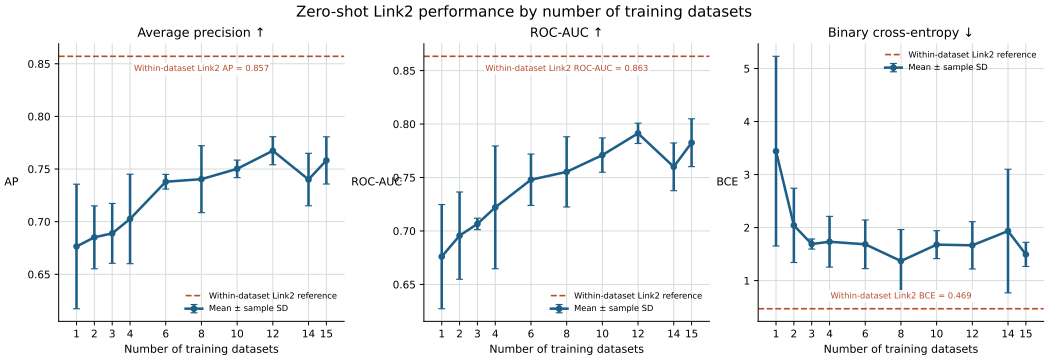


Figure 6: Dataset-scaling trajectories for all evaluation metrics.

macro over the same 18 Link2 targets in each task. Figure 6 shows the Link2 macro AP, ROC-AUC, and BCE at each training-dataset count, averaged across three dataset-ordering and training-seed runs; error bars indicate the sample standard deviation across those runs.

C.6 Dataset heterogeneity effect

In this experiment we test whether the gains observed when training on more datasets can be explained by having more structural training examples alone, rather than by source-graph diversity. We selected citation-classic, the largest Link1 dataset, with 1,016,241 nodes, and formed ten disjoint random 100,000-root supervision units from its G_{train} graph. For each training size $K \in \{1, 2, 3, 4, 6, 8, 10\}$ not up to 15 because we are limited by the dataset size, we trained a LPPE model using K units and three fixed ordering/seed replicates. The link-prediction stage always used the same single citation-classic G_{train} loader, isolating the amount of LPPE structural supervision as the varying factor. Each trained model was evaluated zero-shot on the 18 Link2 evaluation datasets used in the other experiments.

Figure 7 compares training on multiple datasets with training on an equivalent amount of data drawn from the single citation-classic graph. Both curves show mean Average Precision across the same 18 Link2 evaluation datasets; error bars indicate the sample standard deviation across three orderings. As more citation-classic supervision units are added, performance stays approximately constant near 0.71. By contrast, performance generally improves as the number of distinct training datasets increases. The additional examples from one graph therefore do not reproduce the gains from training across multiple graphs, indicating that diverse source datasets are needed to improve zero-shot performance in this setting.

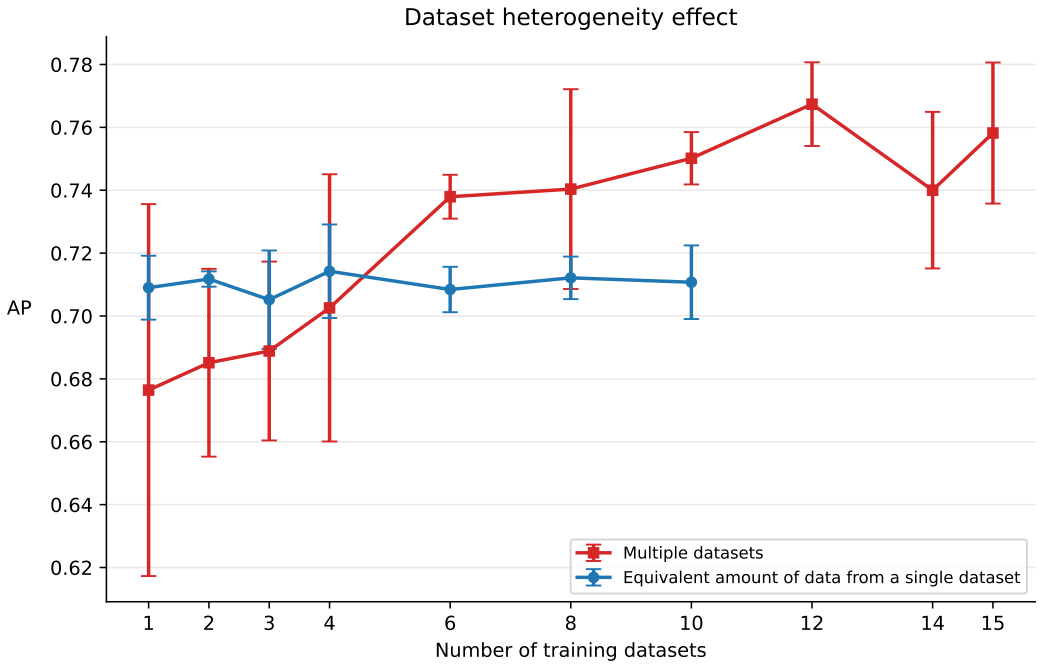


Figure 7: Effect of training-source diversity on zero-shot link prediction. Mean Average Precision is measured across 18 fixed Link2 evaluation datasets. Red squares show training with an increasing number of distinct datasets; blue circles show training with an equivalent amount of structural supervision drawn from the single citation-classic graph. Error bars indicate the sample standard deviation across three orderings. Performance remains approximately constant when more data come from one graph, while it generally increases with the number of distinct training datasets.