GLEX: EXECUTABLE PROGRAM VOCABULARIES FOR TEXT-FREE GRAPH FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Few-shot transfer across heterogeneous graphs is challenging in the text-free setting, where source and target domains may differ in feature semantics, structural statistics, and label spaces. Existing transfer interfaces are often built on representations, aligned coordinates, or structural patterns whose usefulness can depend on the graph on which they are formed. We introduce GLEX (**G**raph **L**exicon of **Ex**ecutable Programs), which instead builds a transferable vocabulary of reusable graph computations. The same computation can be instantiated differently on each graph, providing a common interface while naturally adapting to heterogeneous features and structures. On an unseen target, the pretrained encoder remains fixed, and a few labeled examples determine which computations are most useful for the target task and refine the resulting predictions. Extensive experiments on seven heterogeneous target graphs cover both node and graph classification from one-shot to ten-shot settings. GLEX improves average one-shot node-classification accuracy from 46.52% to 52.95% over the strongest text-free multi-domain baseline BRIDGE, while also improving average graph-classification accuracy from 51.97% to 53.54%. Ablation studies identify the executable vocabulary and support-conditioned adaptation as the main contributors, while efficiency, sensitivity, and program-use analyses further characterize how GLEX behaves across target graphs.

1 INTRODUCTION

Graph foundation models aim to transfer reusable knowledge across graphs with minimal task supervision. This goal becomes particularly challenging in the text-free setting, where source and target graphs may differ substantially in feature coordinates, structural statistics, and label semantics. Without a shared semantic space, cross-domain transfer requires an interface that remains meaningful across heterogeneous graphs while still allowing task-specific specialization from only a few labels on an unseen target.

Existing approaches largely differ in how they construct such an interface. Self-supervised graph pretraining (Veličković et al., 2019; You et al., 2020; Sun et al., 2020) and graph prompting (Sun et al., 2022; Liu et al., 2023; Fang et al., 2023) transfer through reusable representations or task-specific prompts, while recent multi-domain graph foundation models use structural tokens (Yu et al., 2025), feature alignment (Yuan et al., 2025a), or natural-language semantics (Liu et al., 2024; He et al., 2025). Vocabulary-based methods further make transferable units explicit through recurring graph patterns (Yuan et al., 2025b). Despite these designs, the transferred units are still instantiated objects, such as representations, aligned coordinates, tokens, prompts, or realized patterns, whose usefulness can depend on the feature space or graph realization in which they are formed. This motivates a more fundamental question: **can transferable knowledge be defined at the level of reusable graph computations, rather than at the level of graph-specific objects?**

We answer this question by defining a graph vocabulary at the level of **executable computations**. Each entry is a lightweight graph program that can be executed on any attributed graph. For example, "propagate twice" has the same syntax across domains, while its execution produces a graph-specific view according to the local structure and attributes. This separates transferable knowledge from its graph-specific realization: the computation itself can be shared across domains, while its output

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.

naturally adapts to each graph. Thus, GLEX transfers reusable graph computations rather than fixed graph patterns or aligned representations.

Based on this idea, GLEX organizes cross-domain transfer into three stages. First, shared rules generate candidate programs and select a compact graph-specific vocabulary from their unlabeled executions. Second, a shared encoder is pretrained on source-graph program views and then frozen. Third, target support labels reweight the resulting program predictors, while a regularized residual correction captures the remaining target-specific error. Thus, vocabulary induction determines the available computations, while support-conditioned adaptation determines their task-specific contributions without updating the encoder.

Empirically, we evaluate GLEX on node and graph classification across seven target graphs spanning citation, co-purchase, and social-network domains. On one-shot node classification, GLEX achieves the best average accuracy of 52.95% and the highest mean on six targets, compared with 46.52% for BRIDGE. It also achieves the best average one-shot graph-classification accuracy of 53.54%, compared with 51.97% for BRIDGE. Multi-shot evaluations further show that the benefits extend to broader support budgets, although the relative advantage remains target- and task-dependent. Ablations identify the multi-program vocabulary and support-conditioned adaptation as the main contributors, while program-use, efficiency, and sensitivity analyses further characterize the behavior of GLEX.

Our contributions are:

- We introduce an **executable graph vocabulary** that transfers reusable graph computations rather than fixed representations or structural patterns, with shared rules inducing compact graph-specific vocabularies from unlabeled executions.
- We develop a **support-conditioned transfer mechanism** that keeps the shared source-trained encoder frozen while using target labels to infer the task-specific contributions of program predictors and fit a regularized residual correction.
- We demonstrate strong few-shot performance across seven heterogeneous targets for both node and graph classification from one to ten shots, including a 6.43-point average gain over the strongest baseline in one-shot node classification, together with controlled ablations and diagnostics of vocabulary selection, target adaptation, and efficiency.

2 PROBLEM SETUP

Let $G = (V, E)$ denote a graph with node set V and edge set E. Given a collection of source graphs $\mathcal{G}_s = \{G_1, \ldots, G_S\}$ and an unseen target graph G_t, we consider few-shot transfer from multiple source graphs to G_t. For each target class, m labeled examples form the support set $\mathcal{S}_t$, and a separate held-out subset forms the query set $\mathcal{Q}_t$.

We consider both node and graph classification. For graph classification, each example is a node-centered ego-graph that inherits its center node's label. The target graph is available without query labels, and no node text or shared class semantics across graphs are assumed.

3 THE GLEX METHOD

3.1 OVERVIEW

GLEX comprises three stages: executable vocabulary induction, shared source pretraining, and support-conditioned target adaptation. Each graph induces a compact vocabulary from shared program rules; a shared encoder is pretrained on source program views and frozen at transfer; and target support labels adapt the resulting predictors. Figure 1 illustrates the workflow, and Appendix D gives the complete algorithm.

3.2 EXECUTABLE PROGRAM VOCABULARY INDUCTION

GLEX constructs a graph-specific vocabulary by generating candidate programs with shared rules and selecting a compact subset based on their unlabeled executions.

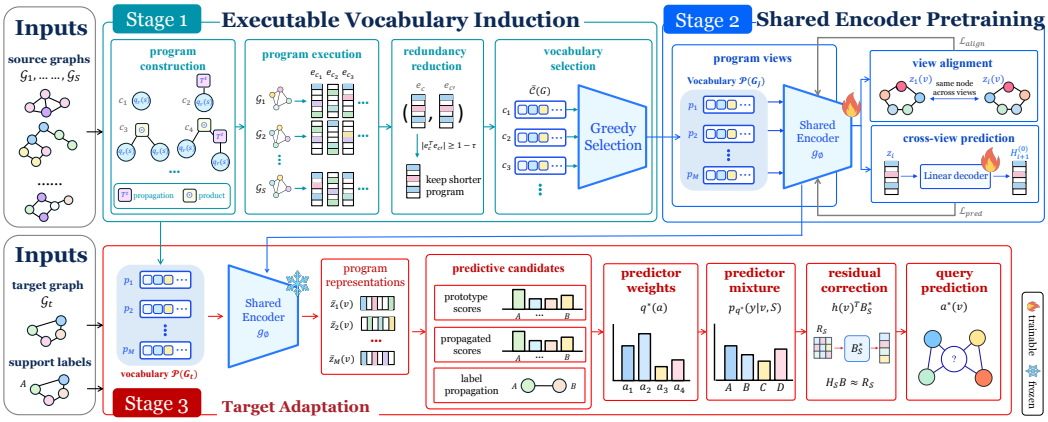


Figure 1: GLEX workflow: executable vocabulary induction, shared encoder pretraining, and support-conditioned target adaptation.

3.2.1 SHARED PROGRAM CONSTRUCTION.

We construct candidate programs using feature propagation, differences across propagation depths, and elementwise interactions. Let $\tilde{A}$ be the row-normalized adjacency with self-loops, and define $\mathsf{T}(h) = \tilde{A}h$ for a node-feature matrix h. Starting from $h_0 = X$, let $h_k = \mathsf{T}^k(X)$ and define the seed set $\mathcal{S}_G$ as the propagated features and their pairwise differences:

$$\mathcal{S}_G = \{h_k : 0 \le k \le K\} \cup \{h_j - h_i : 0 \le i < j \le K\}. \tag{1}$$

To reduce sensitivity to feature scales, map each seed $s \in \mathcal{S}_G$ coordinate-wise to $u_s = 2\widehat{F}_s(s) - 1$, where $\widehat{F}_s$ is the empirical CDF across nodes. This rank transform preserves the relative ordering of node values while reducing the influence of raw scales and extreme values. Define $q_0(s) = \mathrm{std}(s)$ and $q_r(s) = P_r(u_s)$ for $1 \le r \le R$, where std standardizes each coordinate across nodes and P_r is the degree-r Legendre polynomial. For each $q_r(s)$, we form three program variants: the atom itself, its t-step propagation, and their elementwise product. The candidate set is

$$\mathcal{C}(G) = \left\{q_r(s),\ \mathsf{T}^t q_r(s),\ q_r(s) \odot \mathsf{T}^t q_r(s) : s \in \mathcal{S}_G,\ 0 \le r \le R,\ 1 \le t \le K\right\}. \tag{2}$$

Appendix B specifies the complete construction rules.

3.2.2 GRAPH-SPECIFIC VOCABULARY SELECTION.

We obtain a compact graph-specific vocabulary by merging programs with similar executions and selecting a subset that covers the remaining execution space.

Program redundancy reduction. For each candidate c, let v_c be the vectorized output of c on graph G, and let e_c be the centered and ℓ_2-normalized version of v_c. We treat c and c' as equivalent when

$$\left|e_c^\top e_{c'}\right| \ge 1 - \tau, \tag{3}$$

where $\tau \in (0, 1)$ is the matching tolerance. A smaller τ requires more similar executions. Within each group, we retain the program with the smallest description length $\ell(c)$, measured by the number of operations used to construct c. This yields the reduced candidate set $\overline{\mathcal{C}}(G)$.

Vocabulary reconstruction error. For a candidate vocabulary $\mathcal{P} \subseteq \overline{\mathcal{C}}(G)$, let $B_\mathcal{P}$ stack its normalized outputs and define the reconstruction residual of candidate c as

$$r(c \mid \mathcal{P}) = \min_{\beta \in \mathbb{R}^{|\mathcal{P}|}} \|e_c - B_\mathcal{P}\beta\|_2^2, \tag{4}$$

where β gives the linear combination of the selected program views that best approximates e_c. A smaller $r(c \mid \mathcal{P})$ means that vocabulary $\mathcal{P}$ represents c more accurately. We weight candidates by

$\mu_G(c) = 2^{-\ell(c)} / \sum_{c' \in \overline{\mathcal{C}}(G)} 2^{-\ell(c')}$, favoring shorter programs. Given a vocabulary budget M, the weighted reconstruction error and the optimal vocabulary are

$$D_G(\mathcal{P}) = \sum_{c \in \overline{\mathcal{C}}(G)} \mu_G(c) r(c \mid \mathcal{P}), \qquad \mathcal{P}^\star(G) \in \arg \min_{\mathcal{P} \subseteq \overline{\mathcal{C}}(G),\ |\mathcal{P}| \le M} D_G(\mathcal{P}). \tag{5}$$

Greedy vocabulary construction. Since exact subset selection is combinatorial, we use a greedy approximation. Starting from $\mathcal{P}_1 = \{q_0(h_0)\}$, each step selects the program that gives the largest reduction in D_G per unit description length:

$$p_t = \arg \max_{c \in \overline{\mathcal{C}}(G) \setminus \mathcal{P}_{t-1}} \frac{\Delta_G(c \mid \mathcal{P}_{t-1})}{\ell(c)}, \qquad \mathcal{P}_t = \mathcal{P}_{t-1} \cup \{p_t\}, \tag{6}$$

for $t = 2, \ldots, M$, where $\Delta_G(c \mid \mathcal{P}) = D_G(\mathcal{P}) - D_G(\mathcal{P} \cup \{c\})$. The resulting vocabulary $\mathcal{P}(G) = \mathcal{P}_M$ approximates $\mathcal{P}^\star(G)$ without enumerating all feasible subsets.

Proposition 1 (Strict reconstruction-error decrease)**.** *Let $\mathcal{P}$ be the current vocabulary and let c be an admitted candidate with $r(c \mid \mathcal{P}) > \epsilon_{\text{dep}}$, where $\epsilon_{\text{dep}} > 0$ is the dependence threshold. Then*

$$D_G(\mathcal{P}) - D_G(\mathcal{P} \cup \{c\}) \ge \mu_G(c) r(c \mid \mathcal{P}) > \mu_G(c) \epsilon_{\text{dep}} > 0. \tag{7}$$

Thus every admitted program strictly decreases D_G, although the greedy solution need not be globally optimal. Section 3.5 later relates this objective to downstream score approximation.

3.3 Multi-Domain Pretraining with a Shared Encoder

Shared encoder architecture. For each selected program p_i, its execution $H_i^{(0)} = p_i(A, X)$defines a program view. A shared L_{enc}-layer encoder maps these views to program representations by propagating over $\widetilde{A}$:

$$m_v^{(l)} = \sum_{u \in \mathcal{N}^+(v)} \alpha_{vu}^{(l)} V_l h_u^{(l-1)}, \qquad \alpha_{vu}^{(l)} = \frac{a_{vu}^{(l)}}{\sum_{w \in \mathcal{N}^+(v)} a_{vw}^{(l)}}, \tag{8}$$

where

$$a_{vu}^{(l)} = \widetilde{A}_{vu}\, \sigma\left(\frac{(Q_l h_v^{(l-1)})^\top (K_l h_u^{(l-1)})}{\sqrt{d_g}} \right). \tag{9}$$

Here, $\mathcal{N}^+(v)$ includes v and its neighbors, Q_l, K_l project to gate vectors of dimension d_g, and S_l is the self-transformation. The hidden representation at each intermediate encoder layer is updated as

$$h_v^{(l)} = \text{Dropout}\Big(\text{ReLU}\Big(S_l h_v^{(l-1)} + m_v^{(l)} \Big) \Big), \qquad l = 1, \ldots, L_{\text{enc}} - 1. \tag{10}$$

The final encoder output is

$$\widetilde{z}_i(v) = S_{L_{\text{enc}}} h_v^{(L_{\text{enc}}-1)} + m_v^{(L_{\text{enc}})}, \qquad z_i(v) = \frac{\widetilde{z}_i(v)}{\|\widetilde{z}_i(v)\|_2}. \tag{11}$$

We use $L_{\text{enc}} = 2$ with parameters shared across all programs and source graphs.

Pretraining objectives. We train the encoder to preserve information shared across program views using view alignment and cross-view prediction. The alignment term matches each program representation z_i to the original-feature representation z_1 with symmetric InfoNCE:

$$\mathcal{L}_{\text{align}} = \frac{1}{M-1} \sum_{i=2}^{M} \mathcal{L}_{\text{NCE}}(z_1, z_i). \tag{12}$$

For cross-view prediction, a shared linear decoder d_ψ predicts the next program view $H_{i+1}^{(0)}$ from representation z_i, encouraging information that remains useful across program transformations. We use cyclic pairing, $H_{M+1}^{(0,j)} = H_1^{(0,j)}$, and define for a node batch $\mathcal{B}_j$:

$$\mathcal{L}_{\text{pred}}^{(j)} = \frac{1}{M|\mathcal{B}_j| d_x} \sum_{i=1}^{M} \sum_{v \in \mathcal{B}_j} \Big\| d_\psi\Big(z_i^{(j)}(v) \Big) - H_{i+1}^{(0,j)}(v) \Big\|_2^2,$$
$$\mathcal{L}_j = \mathcal{L}_{\text{align}}^{(j)} + \lambda_{\text{pred}} \mathcal{L}_{\text{pred}}^{(j)}. \tag{13}$$

where d_x is the program-output dimension.

3.4 SUPPORT-CONDITIONED PREDICTION ADAPTATION

Target adaptation reweights program predictors derived from the encoded vocabulary entries and then corrects their residual errors. Let $\mathcal{S} = \mathcal{S}_t$ for the target support set; the encoder remains frozen. The equations below describe node classification; Appendix F specifies the ego-graph readout and candidate set for graph classification.

Predictor reweighting. Each vocabulary entry p_i uses the frozen encoder to produce a normalized program representation $\bar{z}_i(v) = z_i(v)$, already unit-normalized by the encoder. For class y, its support prototype and the resulting class score are

$$c_{i,y} = \frac{\sum_{u \in \mathcal{S}_y} \bar{z}_i(u)}{\left\| \sum_{u \in \mathcal{S}_y} \bar{z}_i(u) \right\|_2}, \qquad a_{i,0,y}(v) = \bar{z}_i(v)^\top c_{i,y}. \tag{14}$$

To also use neighborhood information, we define $\mathsf{T}_G(B) = \widetilde{A}B$, form the propagated scores $a_{i,1} = \mathsf{T}_G(a_{i,0})$, and include the label-propagation candidate $a_{\mathrm{lab}} = \mathsf{T}_G(\widetilde{Y}_{\mathcal{S}})$, where $\widetilde{Y}_{\mathcal{S}} \in \mathbb{R}^{|V| \times C}$ contains one-hot support labels and zeros elsewhere. These predictors form $\mathcal{A} = \{a_{i,0}, a_{i,1} : i = 1, \ldots, M\} \cup \{a_{\mathrm{lab}}\}$.

For each candidate $a \in \mathcal{A}$, define its support loss as

$$\mathcal{E}(a; \mathcal{S}) = -\frac{1}{|\mathcal{S}|} \sum_{(v, y_v) \in \mathcal{S}} \log \operatorname{softmax}(a(v))_{y_v}, \tag{15}$$

and its complexity prior as $\pi_\eta(a) = \exp[-\eta \widetilde{\ell}(a)] / \sum_{b \in \mathcal{A}} \exp[-\eta \widetilde{\ell}(b)]$, where $\eta > 0$ controls the preference for simpler candidates. We infer a probability distribution q over the candidate predictors by minimizing

$$\mathcal{J}_{\mathcal{S}}(q) = \omega \sum_{a \in \mathcal{A}} q(a) \mathcal{E}(a; \mathcal{S}) + T(\mathcal{S}) \operatorname{KL}(q \| \pi_\eta). \tag{16}$$

Here, $\operatorname{KL}(q \| \pi_\eta) = \sum_{a \in \mathcal{A}} q(a) \log[q(a)/\pi_\eta(a)]$. The first term fits support evidence, while the KL term regularizes toward the complexity prior; $T(\mathcal{S}) > 0$ controls this trade-off and is defined in Appendix C.3.

Because the objective is strictly convex in q, its unique minimizer is

$$q^\star(a) = \frac{\pi_\eta(a) \exp[-\omega \mathcal{E}(a; \mathcal{S})/T(\mathcal{S})]}{\sum_{b \in \mathcal{A}} \pi_\eta(b) \exp[-\omega \mathcal{E}(b; \mathcal{S})/T(\mathcal{S})]}. \tag{17}$$

The resulting predictive distribution is

$$p_{q^\star}(y \mid v, \mathcal{S}) = \sum_{a \in \mathcal{A}} q^\star(a) \operatorname{softmax}(a(v))_y. \tag{18}$$

Residual correction. The weighted predictor mixture captures the part of the target task explained by reusable program predictors but can still leave target-specific errors. We therefore concatenate the normalized representations from all programs,

$$h(v) = M^{-1/2} \left[\bar{z}_1(v)^\top, \ldots, \bar{z}_M(v)^\top \right]^\top, \tag{19}$$

and stack the support representations as the rows of $H_{\mathcal{S}}$. Let $Y_{\mathcal{S}}$ be the one-hot support-label matrix and define the remaining support error as $R_{\mathcal{S}} = Y_{\mathcal{S}} - p_{q^\star}(Y \mid \mathcal{S})$. Let $B \in \mathbb{R}^{Md \times C}$ map the joint representation to a class-wise correction. On the support set, this correction should approximate the remaining prediction error:

$$H_{\mathcal{S}} B \approx R_{\mathcal{S}}. \tag{20}$$

With regularization strength $\kappa > 0$, the fitted correction has the closed-form solution

$$B^\star_{\mathcal{S}} = H_{\mathcal{S}}^\top \left(H_{\mathcal{S}} H_{\mathcal{S}}^\top + \kappa I \right)^{-1} R_{\mathcal{S}}. \tag{21}$$

The corresponding objective and derivation are given in Appendix C.4. The corrected target logits are

$$a^\star(v) = \log p_{q^\star}(\cdot \mid v, \mathcal{S}) + h(v)^\top B^\star_{\mathcal{S}}. \tag{22}$$

No encoder parameter or vocabulary entry is updated during adaptation.

3.5 Vocabulary Selection and Task-Score Approximation

Having described the complete GLEX pipeline, we finally connect the vocabulary-selection objective to downstream score approximation under a finite-output linear-task assumption.

Assumption 1 (Finite-output linear task)**.** *For the target task under consideration, each class-score vector is $f_y = U_y(\sum_{c\in\overline{\mathcal{C}}(G)} \gamma_{y,c} e_c)$, where the fixed linear readout U_y has operator norm at most one. The description-weighted coefficient norm $\|\Gamma\|^2_{\mu^{-1}} = \sum_{y,c} \gamma^2_{y,c}/\mu_G(c)$ is bounded by R^2_{task}.*

Proposition 2 (Task-score approximation)**.** *Under Assumption 1, let $\Pi_{\mathcal{P}}$ project onto the selected output span. Then*

$$\left(\sum_{y=1}^{C} \|f_y - U_y \Pi_{\mathcal{P}} \sum_c \gamma_{y,c} e_c\|_2^2\right)^{1/2} \leq \|\Gamma\|_{\mu^{-1}} \sqrt{D_G(\mathcal{P})}. \tag{23}$$

The bound connects the vocabulary reconstruction objective to task-score approximation within the retained execution space under Assumption 1; proofs are provided in Appendix E.

4 Experiments

We evaluate GLEX through five research questions covering classification performance, component contributions, target-adaptation efficiency, hyperparameter sensitivity, and program selection and use.

RQ1: How effective is GLEX for few-shot node and graph classification? (Section 4.2)
RQ2: How much does each component contribute? (Section 4.3)
RQ3: How efficient is target adaptation in GLEX? (Section 4.4)
RQ4: How sensitive is GLEX to its hyperparameters? (Section 4.5)
RQ5: Which programs are selected and used across target graphs? (Section 4.6)

4.1 Experimental Setup

Datasets. We evaluate on seven target graphs spanning citation, co-purchase, and social-network benchmarks. Cora, CiteSeer, and PubMed (Sen et al., 2008) are citation networks; Amazon Photo and Amazon Computers (Shchur et al., 2019) are co-purchase graphs; and Facebook Page-Page and LastFM Asia (Rozemberczki et al., 2021) are social-network graphs. We abbreviate the latter two as Facebook and LastFM. Wiki-CS (Mernyei & Cangea, 2020) is a web-link graph of computer-science articles, used only as an additional source graph for training.

Baselines. We compare GLEX with representative methods from four groups. (1) *Supervised GNNs*: GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018) learn from target labels without a separate graph-pretraining stage and provide task-specific reference points. (2) *Self-supervised graph pretraining*: DGI (Veličković et al., 2019), GraphCL (You et al., 2020), InfoGraph (Sun et al., 2020), and MVGRL (Hassani & Khasahmadi, 2020) learn representations from unlabeled graph objectives before downstream adaptation. (3) *Prompt-based graph adaptation*: GPPT (Sun et al., 2022), GraphPrompt (Liu et al., 2023), and GPF (Fang et al., 2023) adapt pretrained graph models through task-specific prompts. (4) *Multi-domain graph pretraining*: GCOPE (Zhao et al., 2024), SAMGPT (Yu et al., 2025), and BRIDGE (Yuan et al., 2025a) transfer knowledge learned across multiple source graphs to an unseen target, making them the closest comparisons to GLEX.

Evaluation protocol. For each target graph, we exclude it entirely from source pretraining and train on the remaining source graphs only. The pretrained encoder is then frozen, and the unseen target is used only for vocabulary induction and m-shot adaptation, where m labeled examples per class form the support set and a separate held-out subset forms the query set. For graph classification, each labeled node defines a node-centered ego-graph inheriting the center-node label. We report mean accuracy over five fixed support/query splits.

Table 1: One-shot node-classification accuracy. Best is bold and second-best is underlined.

Method	Cora	CiteSeer	PubMed	Photo	Computers	Facebook	LastFM	Avg.
GCN (Kipf & Welling, 2017)	29.53 ± 7.56	26.29 ± 6.50	23.32 ± 11.56	26.96 ± 12.94	24.40 ± 5.62	20.45 ± 5.62	9.21 ± 3.11	22.88
GAT (Veličković et al., 2018)	24.27 ± 9.26	21.56 ± 8.09	22.28 ± 9.76	17.85 ± 10.22	23.03 ± 12.12	29.27 ± 6.47	9.01 ± 2.61	21.04
DGI (Veličković et al., 2019)	33.40 ± 10.48	25.80 ± 8.27	47.22 ± 9.50	30.89 ± 10.54	25.75 ± 12.45	34.36 ± 9.57	14.14 ± 6.31	30.22
GraphCL (You et al., 2020)	27.72 ± 9.37	35.02 ± 8.46	48.89 ± 9.03	34.78 ± 11.56	23.79 ± 12.28	34.85 ± 7.07	18.93 ± 7.32	32.00
GPPT (Sun et al., 2022)	27.18 ± 4.88	25.90 ± 4.68	39.82 ± 8.79	31.58 ± 10.27	19.94 ± 9.61	34.73 ± 3.99	20.98 ± 3.98	28.59
GraphPrompt (Liu et al., 2023)	28.26 ± 12.68	32.51 ± 8.73	47.47 ± 9.15	48.11 ± 9.89	42.82 ± 11.67	40.44 ± 9.68	19.84 ± 7.23	37.06
GPF (Fang et al., 2023)	32.17 ± 6.56	36.79 ± 7.70	41.28 ± 8.14	47.47 ± 8.19	35.75 ± 7.12	40.45 ± 6.34	27.26 ± 5.50	37.31
MVGRL (Hassani & Khasahmadi, 2020)	33.35 ± 6.93	33.66 ± 7.24	39.87 ± 8.16	48.48 ± 7.07	39.99 ± 7.91	37.70 ± 5.79	27.16 ± 4.94	37.17
GCOPE (Zhao et al., 2024)	35.62 ± 11.93	38.33 ± 9.28	45.38 ± 9.87	52.87 ± 9.19	45.65 ± 10.69	40.63 ± 8.50	28.84 ± 7.59	41.05
SAMGPT (Yu et al., 2025)	$\underline{47.80 \pm 11.88}$	36.38 ± 9.10	50.25 ± 10.43	58.71 ± 8.69	48.22 ± 8.17	42.70 ± 8.73	$\underline{33.36 \pm 8.11}$	45.35
BRIDGE (Yuan et al., 2025a)	35.99 ± 6.86	$\mathbf{44.33 \pm 6.11}$	$\underline{55.40 \pm 6.42}$	$\underline{63.68 \pm 9.09}$	$\underline{54.13 \pm 4.18}$	$\underline{43.07 \pm 10.58}$	29.04 ± 5.73	$\underline{46.52}$
GLEX	$\mathbf{48.78 \pm 13.19}$	$\underline{38.93 \pm 9.31}$	$\mathbf{59.20 \pm 7.90}$	$\mathbf{73.38 \pm 4.29}$	$\mathbf{57.67 \pm 8.14}$	$\mathbf{45.38 \pm 6.15}$	$\mathbf{47.29 \pm 6.09}$	**52.95**

Table 2: One-shot graph-classification accuracy. Best is bold and second-best is underlined.

Method	Cora	CiteSeer	PubMed	Photo	Computers	Facebook	LastFM	Avg.
GCN (Kipf & Welling, 2017)	30.64 ± 10.31	26.90 ± 7.15	38.84 ± 11.82	15.60 ± 8.77	21.94 ± 14.51	31.33 ± 9.47	28.83 ± 9.60	27.73
GAT (Veličković et al., 2018)	27.80 ± 7.85	27.50 ± 7.13	21.66 ± 8.70	15.74 ± 7.62	16.02 ± 13.46	21.20 ± 7.31	27.80 ± 7.85	22.53
InfoGraph (Sun et al., 2020)	34.98 ± 10.15	35.87 ± 9.84	48.67 ± 12.29	25.70 ± 11.73	19.02 ± 14.09	31.26 ± 9.65	23.29 ± 7.99	31.26
GraphCL (You et al., 2020)	42.70 ± 10.64	36.66 ± 8.67	47.53 ± 11.52	33.07 ± 12.31	16.02 ± 13.47	21.99 ± 13.00	21.30 ± 10.45	31.32
GraphPrompt (Liu et al., 2023)	37.38 ± 14.03	36.66 ± 9.19	49.55 ± 10.25	50.79 ± 12.31	43.09 ± 11.45	41.71 ± 10.61	32.62 ± 8.54	41.69
GPF (Fang et al., 2023)	39.62 ± 8.52	36.73 ± 7.66	45.08 ± 10.36	47.57 ± 10.16	35.70 ± 8.71	34.84 ± 5.14	34.31 ± 7.05	39.12
MVGRL (Hassani & Khasahmadi, 2020)	36.86 ± 10.74	35.78 ± 8.80	43.97 ± 13.27	41.55 ± 13.08	29.49 ± 13.86	35.57 ± 9.00	25.39 ± 8.14	35.52
GCOPE (Zhao et al., 2024)	38.85 ± 10.99	39.93 ± 9.82	47.05 ± 11.74	53.93 ± 9.74	45.60 ± 10.96	40.26 ± 9.53	34.68 ± 7.70	42.90
SAMGPT (Yu et al., 2025)	$\mathbf{55.35 \pm 13.62}$	38.75 ± 9.40	48.69 ± 10.16	58.75 ± 11.67	48.72 ± 11.18	43.71 ± 9.54	$\underline{48.28 \pm 9.72}$	48.89
BRIDGE (Yuan et al., 2025a)	41.43 ± 9.97	$\mathbf{46.20 \pm 2.85}$	$\underline{56.00 \pm 13.63}$	$\underline{68.85 \pm 7.38}$	$\underline{55.33 \pm 6.57}$	$\underline{43.93 \pm 13.17}$	$\mathbf{52.06 \pm 4.69}$	$\underline{51.97}$
GLEX	$\underline{51.09 \pm 11.53}$	$\underline{43.07 \pm 5.90}$	$\mathbf{61.67 \pm 9.03}$	$\mathbf{73.31 \pm 3.24}$	$\mathbf{57.93 \pm 8.79}$	$\mathbf{49.20 \pm 6.42}$	38.54 ± 4.76	**53.54**

4.2 RQ1: How effective is GLEX for few-shot node and graph classification?

4.2.1 One-shot Classification.

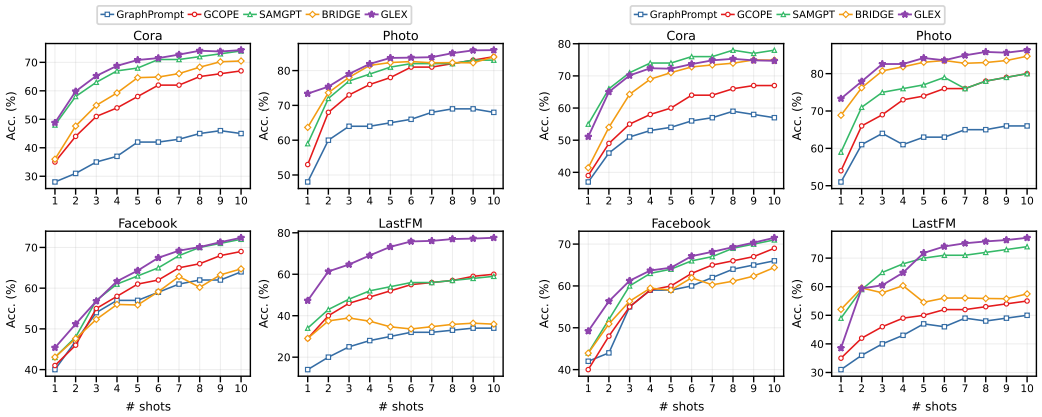
Node classification. Table 1 shows that GLEX achieves the highest accuracy on six of seven targets and the best average (52.95%), outperforming BRIDGE (46.52%) and SAMGPT (45.35%). The consistent gains across targets may reflect the vocabulary's ability to expose reusable local graph computations. **Graph classification.** Table 2 shows that GLEX ranks first on four targets and achieves the highest average (53.54%), compared with 51.97% for BRIDGE. The smaller margin may reflect greater sensitivity in graph classification to target-specific global structure and readout.

4.2.2 m-shot Classification.

Figure 2 evaluates $m = 1, \ldots, 10$. Performance generally improves with more support examples, while relative ranking remains target- and task-dependent. GLEX shows the most consistent gains on node classification, maintaining a clear advantage across the full shot range on several targets. Graph classification is more mixed: additional labels improve all methods, but no method dominates every target. This suggests that increasing the support budget reduces estimation error without eliminating differences in how useful the induced program predictors are for each target. Results on the remaining targets are reported in Appendix I.2.

4.3 RQ2: How much does each component contribute?

Figure 3 compares controlled variants under the same splits and pretraining protocol. Graph variants use a common radius-one mean-pooling profile, whereas Table 2 uses validation-selected graph profiles. Replacing the induced multi-program vocabulary with a one-entry raw-feature vocabulary reduces average one-shot node accuracy from 52.95% to 47.74% (5.21 points), while removing both predictor reweighting and residual correction lowers it to 47.82%. Replacing the induced vocabulary with a fixed four-program vocabulary reduces one-shot node accuracy more modestly, from 52.95% to 52.05%. Since this comparison also changes vocabulary size, it does not isolate graph-specific selection alone. For graph classification, removing each component causes a smaller decrease of 0.83–1.03 points, possibly because the graph-level readout already captures part of the structural information provided by the vocabulary and support adaptation. Per-target ablation results are in Appendix H.



(a) Node classification **(b) Graph classification**

Figure 2: m-shot node- and graph-classification accuracy on four representative target graphs.

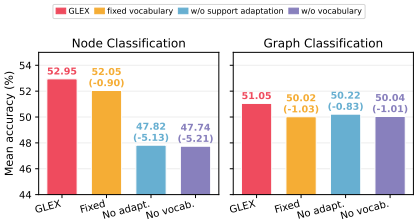


Figure 3: One-shot ablations (mean).

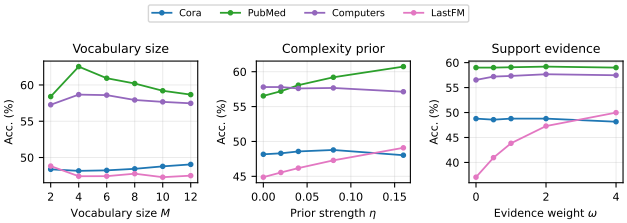


Figure 4: One-shot node-classification sensitivity.

4.4 RQ3: How efficient is target adaptation in GLEX?

For vocabulary size M, representation dimension d, C classes, and support size $s = mC$, cached program representations and predictor outputs give a support-set fitting cost of $O(s^2Md+s^3+sMdC)$ time and $O(sMd + s^2)$ memory, governed by the small support set rather than the target size. Table 3 reports five-shot CiteSeer graph-classification time averaged over five splits. All methods exclude source pretraining and include target loading, adaptation, and evaluation. GLEX takes 4.52 seconds per split, making its target-side procedure 1.7–23.7 times faster.

Table 3: Runtime (s/split).

Method	Time (s)
GLEX	**4.52**
GraphPrompt	7.80
SAMGPT	13.39
GCOPE	107.03

4.5 RQ4: How sensitive is GLEX to its hyperparameters?

Figure 4 varies the vocabulary size M, complexity-prior strength η, and support-evidence weight ω on four representative targets; η and ω enter the predictor-reweighting objective in equation 16. Increasing M yields no consistent gain beyond a compact vocabulary: most curves plateau, while PubMed peaks at $M = 4$. Larger η benefits PubMed and LastFM but has little effect on Cora and Computers, showing that the value of favoring simpler predictors depends on the target. The same contrast is stronger for ω: Cora, PubMed, and Computers remain stable, whereas LastFM improves by 12.93 points over the tested range. Complete results are given in Appendix I.1.

4.6 RQ5: Which programs are selected and used across target graphs?

Figure 5 compares the induced vocabularies and their one-shot program weights across seven targets. We obtain each program weight by summing $q^\star$ over its direct and propagated predictors, then renormalizing after excluding the label-propagation candidate. The vocabularies have a mean pairwise Jaccard similarity of 0.709 and five distinct forms, revealing a shared core with graph-specific additions. The weights are also target-dependent: most targets favor one- and two-hop propagation,

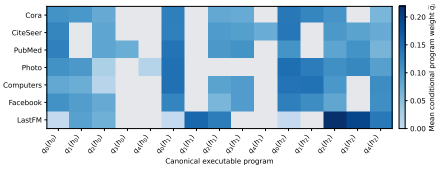


Figure 5: Induced programs and one-shot program weights (gray: absent).

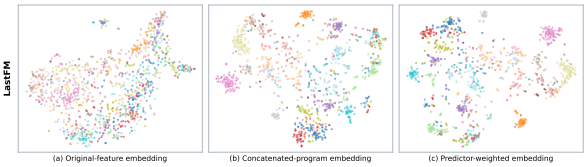


Figure 6: LastFM original-feature, concatenated-program, and predictor-weighted representations.

whereas LastFM emphasizes nonlinear two-hop programs. Vocabulary induction thus determines the available computations, and support labels determine their contributions.

Figure 6 visualizes the LastFM node representations. Classes overlap heavily with the original features, become more distinguishable after concatenating the program representations, and form clearer clusters after predictor weighting. The vocabulary entries therefore provide complementary class structure, while the support-conditioned weights emphasize the more useful predictors.

5 Related Work

Graph pretraining and cross-domain foundation models. Graph pretraining learns transferable representations through contrastive, generative, and reconstruction objectives (Veličković et al., 2019; Sun et al., 2020; Hassani & Khasahmadi, 2020; You et al., 2020; Qiu et al., 2020; Hu et al., 2020a;b; Thakoor et al., 2022; Hou et al., 2022; 2023; Rong et al., 2020). Recent graph foundation models further address transfer across heterogeneous graphs through shared semantics, structural interfaces, or feature alignment (Wang et al., 2025; Liu et al., 2024; He et al., 2025; Tang et al., 2024). Text-enhanced approaches such as OFA and UniGraph use natural language to provide a shared semantic space (Liu et al., 2024; He et al., 2025), while text-free methods such as SAMGPT and BRIDGE transfer through structural prompting, feature alignment, and selective source knowledge (Yu et al., 2025; Yuan et al., 2025a). Unlike these approaches, GLEX transfers executable graph computations rather than aligned representations or semantic coordinates.

Graph prompting and few-shot adaptation. Graph prompting adapts pretrained graph representations through task templates, feature perturbations, prompt subgraphs, or learnable graph tokens (Sun et al., 2022; Liu et al., 2023; Fang et al., 2023; Sun et al., 2023; Huang et al., 2023; Zhao et al., 2024; Zi et al., 2024; Long et al., 2024). Most of them require target-side prompt parameters or gradient-based optimization. GLEX instead adapts fixed program predictors through support-conditioned reweighting and residual correction.

Graph vocabularies and reusable structural units. Graph representation learning has long relied on reusable structural units such as neighborhoods, computation trees, motifs, and learned subgraphs (Hamilton et al., 2017; Xu et al., 2019; Narayanan et al., 2017). More recently, GRAVER explicitly learns structure-feature vocabularies for transfer (Yuan et al., 2025b). GLEX differs by defining each vocabulary entry as an executable computation whose syntax is shared across graphs while its execution is graph-specific.

A more comprehensive discussion of related work is provided in Appendix J.

6 Conclusion

We introduced GLEX, a text-free graph foundation model that transfers executable computations through graph-specific program vocabularies. Across seven heterogeneous targets, GLEX achieves strong few-shot performance on both node and graph classification from one to ten shots, including a 6.43-point average gain over the strongest baseline in one-shot node classification. Ablation and diagnostic studies show how vocabulary induction determines the available computations and support-conditioned adaptation determines their target-task contributions, while additional analyses characterize efficiency and sensitivity.

AI USE STATEMENT

Generative AI tools were used for manuscript polishing and AI-assisted code drafting during implementation. All scientific content was created and verified by the authors, who take full responsibility for the final text.

ETHICS STATEMENT

This work studies text-free graph foundation models on established benchmark datasets and does not involve direct interaction with users or collection of additional personal data. We do not identify inherent ethical risks specific to the proposed method.

REPRODUCIBILITY STATEMENT

The main text formalizes the proposed method with equations, while the Appendix provides detailed implementation settings, including datasets, hyperparameters, training procedures, evaluation protocols, and experiment configurations. The source code will be made publicly available upon acceptance.

REFERENCES

Haibo Chen, Xin Wang, Zeyang Zhang, Haoyang Li, Ling Feng, and Wenwu Zhu. AutoGFM: Automated graph foundation model with adaptive architecture customization. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=fCPB0qRJT2.

Taoran Fang, Yunchao Zhang, Yang Yang, Chunping Wang, and Lei Chen. Universal prompt tuning for graph neural networks. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

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

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Yufei He, Yuan Sui, Xiaoxin He, and Bryan Hooi. UniGraph: Learning a unified cross-domain foundation model for text-attributed graphs. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '25, pp. 448–459, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712456. doi: 10.1145/3690624.3709277. URL https://doi.org/10.1145/3690624.3709277.

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

Zhenyu Hou, Yufei He, Yukuo Cen, Xiao Liu, Yuxiao Dong, Evgeny Kharlamov, and Jie Tang. GraphMAE2: A decoding-enhanced masked self-supervised graph learner. In *Proceedings of the ACM Web Conference 2023*, WWW '23, pp. 737–746, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394161. doi: 10.1145/3543507.3583379. URL https://doi.org/10.1145/3543507.3583379.

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

Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. GPT-GNN: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 1857–1867, New York, NY, USA, 2020b. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403237. URL https://doi.org/10.1145/3394486.3403237.

Qian Huang, Hongyu Ren, Peng Chen, Gregor Kržmanc, Daniel Zeng, Percy Liang, and Jure Leskovec. PRODIGY: enabling in-context learning over graphs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

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.

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*, 2024.

Zemin Liu, Xingtong Yu, Yuan Fang, and Xinming Zhang. Graphprompt: Unifying pre-training and downstream tasks for graph neural networks. In *Proceedings of the ACM Web Conference 2023*, WWW '23, pp. 417–428, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394161. doi: 10.1145/3543507.3583386. URL https://doi.org/10.1145/3543507.3583386.

Qingqing Long, Yuchen Yan, Peiyan Zhang, Chen Fang, Wentao Cui, Zhiyuan Ning, Meng Xiao, Ning Cao, Xiao Luo, Lingjun Xu, Shiyue Jiang, Zheng Fang, Chong Chen, Xian-Sheng Hua, and Yuanchun Zhou. Towards graph prompt learning: A survey and beyond. 2024. URL https://arxiv.org/abs/2408.14520.

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

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs, 2017. URL https://arxiv.org/abs/1707.05005.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 1150–1160, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403168. URL https://doi.org/10.1145/3394486.3403168.

Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Benedek Rozemberczki, Carl Allen, Rik Sarkar, and Thilo Gross. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(1):1–22, 2021. doi: 10.1093/comnet/cnab014.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, September 2008. ISSN 0738-4602. doi: 10.1609/aimag.v29i3.2157. URL https://doi.org/10.1609/aimag.v29i3.2157.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation, 2019. URL https://arxiv.org/abs/1811.05868.

Yangyi Shen, Jincheng Zhou, Beatrice Bevilacqua, Joshua Robinson, Charilaos Kanatsoulis, Jure Leskovec, and Bruno Ribeiro. Zero-shot generalization of GNNs over distinct attribute domains, 2025. URL https://openreview.net/forum?id=5btqauRdz0.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12(null):2539–2561, November 2011. ISSN 1532-4435.

Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2020.

Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. GPPT: Graph pre-training and prompt tuning to generalize graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, pp. 1717–1727, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539249. URL https://doi.org/10.1145/3534678.3539249.

Xiangguo Sun, Hong Cheng, Jia Li, Bo Liu, and Jihong Guan. All in one: Multi-task prompting for graph neural networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 2120–2131, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599256. URL https://doi.org/10.1145/3580305.3599256.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. GraphGPT: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 491–500, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657775. URL https://doi.org/10.1145/3626772.3657775.

Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=0UXT6PpRpW.

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.

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rklz9iAcKQ.

Zehong Wang, Chuxu Zhang, Jundong Li, Nitesh Chawla, and Yanfang Ye. Graph foundation models: Challenges, methods, and open questions. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, pp. 6184–6194, New York, NY, USA, 2025. Association for Computing Machinery. doi: 10.1145/3711896.3736568. URL https://doi.org/10.1145/3711896.3736568.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

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

Xingtong Yu, Zechuan Gong, Chang Zhou, Yuan Fang, and Hui Zhang. SAMGPT: Text-free graph foundation model for multi-domain pre-training and cross-domain adaptation. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, pp. 1142–1153, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712746. doi: 10.1145/3696410.3714828. URL https://doi.org/10.1145/3696410.3714828.

Haonan Yuan, Qingyun Sun, Junhua Shi, Xingcheng Fu, Bryan Hooi, Jianxin Li, and Philip S. Yu. How much can transfer? BRIDGE: Bounded multi-domain graph foundation model with generalization guarantees. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 73604–73644, 13–19 Jul 2025a.

Haonan Yuan, Qingyun Sun, Junhua Shi, Xingcheng Fu, Bryan Hooi, Jianxin Li, and Philip S. Yu. GRAVER: Generative graph vocabularies for robust graph foundation models fine-tuning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=tyERwC5520.

Haihong Zhao, Aochuan Chen, Xiangguo Sun, Hong Cheng, and Jia Li. All in one and one for all: A simple yet effective method towards cross-domain graph pretraining. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 4443–4454, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671913. URL https://doi.org/10.1145/3637528.3671913.

Chenyi Zi, Haihong Zhao, Xiangguo Sun, Yiqing Lin, Hong Cheng, and Jia Li. ProG: a graph prompt learning benchmark. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

A NOTATION

Table 4 collects the symbols used in the method and its theoretical analysis. A subscript t marks the held-out target graph, j indexes source graphs, i indexes selected programs, r is a polynomial degree, l is an encoder layer, and y is a class. We use c for a candidate program and a for a candidate predictor.

Table 4: Notation for GLEX.

Symbol	Meaning
Graphs and few-shot tasks	
$G = (V, E)$, G_j, G_t	Graph with node set V and edge set E; G_j is source graph j, and G_t is the unseen target graph.
$A, X, \tilde{A}$	Adjacency, node features, and row-normalized self-loop adjacency.
$\mathcal{S}_t, \mathcal{Q}_t, m$	Target support set, query set, and number of labeled support examples per class.
C, d	Number of target classes and dimension of one encoded program representation.
Program construction and vocabulary selection	
T, K, R	Graph transport $\mathsf{T}(h) = \tilde{A}h$, maximum transport depth, and maximum polynomial degree.
$h_k, \mathcal{S}_G, s$	k-step propagated node features, seed set, and one seed.
$\widehat{F}_s, u_s$	Empirical coordinate-wise CDF and rank coordinate $2\widehat{F}_s(s) - 1$.
$P_r, q_r(s)$	Degree-r Legendre polynomial and the corresponding rank-transformed atom; $q_0(s) = \mathrm{std}(s)$.
$\mathcal{C}(G)$	Candidate programs constructed on graph G.
$\overline{\mathcal{C}}(G)$	Reduced candidate set after merging programs with similar outputs.
$p_i, \mathcal{P}_t$	Program i and vocabulary after greedy step t.
$\mathcal{P}(G), M$	Final vocabulary and its size budget.
v_c, e_c	Vectorized output of candidate program c, and its centered, sampled, unit-normalized execution vector.
$\ell(c), \beta$	Description length of c, and coefficients used to reconstruct e_c from selected executions.
$B_{\mathcal{P}}$	Matrix of normalized outputs from vocabulary $\mathcal{P}$.
$r(c \mid \mathcal{P})$	Reconstruction residual of candidate c under vocabulary $\mathcal{P}$.
$\mu_G(c)$	Candidate weight normalized from its description length.
$D_G(\mathcal{P})$	Weighted reconstruction error of vocabulary $\mathcal{P}$.
$\Delta_G(c \mid \mathcal{P})$	Reduction in D_G obtained by adding candidate c.
τ	Tolerance for matching programs with similar normalized outputs.
ϵ_{dep}	Numerical dependence threshold for admitting a program.
$\Pi_{\mathcal{P}}$	Orthogonal projection onto the selected output span.
Shared encoder	
g_ϕ, ϕ	Shared graph encoder and its source-trained parameters, frozen on the target.
$H_i^{(0)}, h_v^{(0)}$	Input view $p_i(A, X)$ of program i, and its row for node v.
$h_v^{(l)}$	Hidden state of node v at encoder layer l.
L_{enc}	Number of message-passing layers in the shared encoder.
$\mathcal{N}^+(v)$	Node v and its neighbors, including the self-loop.
Q_l, K_l	Layer-l query and key maps used to compute edge gates.
V_l, S_l	Layer-l message-value and self-transformation maps.
$d_g, \sigma, a_{vu}^{(l)}$	Gate-vector dimension, logistic sigmoid, and unnormalized edge gate.
$\alpha_{vu}^{(l)}$	Edge weight after normalizing gates over the neighbors of v.
$m_v^{(l)}$	Aggregated neighbor message at layer l.
$z_i(v), \bar{z}_i(v)$	Unit-normalized program representation; $\bar{z}_i = z_i$ in node classification.
$\mathcal{L}_{\mathrm{align}}$	Contrastive term aligning each program representation with the identity-view representation.
d_ψ, d_x	Shared source-only linear decoder and dimension of an input program view.
$\mathcal{B}_j$	Sampled node batch from source graph j.
$\mathcal{L}_{\mathrm{pred}}^{(j)}$	Next-view prediction loss on source j.
λ_{pred}	Fixed weight of the prediction loss.
$\mathcal{L}_j$	Pretraining objective for source graph j.
Predictive candidates and reweighting	

Table 4 (continued)

Symbol	Meaning
$\mathcal{S}_y, c_{i,y}$	Support nodes of class y, and their normalized prototype under program i.
$\mathcal{A}$	Set of candidate class-score predictors used for reweighting.
$a_{i,0}, a_{i,1}$	Program prototype scores and their propagated version.
a_{lab}	Logits obtained by transporting the observed support labels.
$\widetilde{Y}_{\mathcal{S}}, \mathsf{T}_G$	Matrix containing one-hot support labels and zeros elsewhere, and the fixed graph-score propagation operator $\mathsf{T}_G(B) = \widetilde{A}B$.
$\mathcal{E}(a; \mathcal{S})$	Support cross-entropy of candidate predictor a.
$\ell(a), \widetilde{\ell}(a)$	Predictor description cost, including any fixed propagation cost, and its standardized value.
$\eta, \pi_\eta(a)$	Complexity-prior strength and prior weight assigned to candidate a.
$\omega, T(\mathcal{S})$	Support-evidence weight and support-dependent temperature.
$\mathrm{KL}(q\|\pi_\eta)$	Kullback–Leibler divergence $\sum_{a\in\mathcal{A}} q(a)\log[q(a)/\pi_\eta(a)]$.
$\mathcal{J}_{\mathcal{S}}(q)$	Support objective balancing candidate errors against departure from the complexity prior.
$q, q^\star$	Candidate-weight distribution and its optimum.
$p_{q^\star}$	Predictive probability mixture under weights $q^\star$.
Residual correction and theory	
$h(v), H_{\mathcal{S}}$	Concatenated program representations and matrix stacking support representations.
$Y_{\mathcal{S}}, R_{\mathcal{S}}$	One-hot support labels and their residual relative to $p_{q^\star}$.
$B, B^\star_{\mathcal{S}}, \kappa$	Linear residual map, its regularized fit, and regularization strength.
$a^\star(v)$	Final class-score vector after adding the residual correction.
f_y, U_y, Γ	Theoretical class-score vector, fixed linear readout, and execution coefficients $\gamma_{y,c}$.
R_{task}	Assumed bound on $\|\Gamma\|_{\mu^{-1}}$.

B Program-Construction Details

B.1 Bounded Candidate Construction

Let $h_0 = X$ and $h_{k+1} = \tilde{A}h_k$ for $k < K$. The implementation forms $\mathcal{S}_G = \{h_k\}_{k=0}^K \cup \{h_j - h_i\}_{0\le i<j\le K}$. For each scalar coordinate of seed s, let $u_s = 2\widehat{F}_s(s) - 1$, where $\widehat{F}_s$ is the within-graph empirical CDF. The atom bank is

$$q_0(s) = \mathrm{std}(s), \qquad q_1(s) = u_s, \tag{24}$$

$$q_2(s) = \tfrac{1}{2}(3u_s^2 - 1), \qquad q_3(s) = \tfrac{1}{2}(5u_s^3 - 3u_s), \tag{25}$$

$$q_4(s) = \tfrac{1}{8}(35u_s^4 - 30u_s^2 + 3). \tag{26}$$

For every seed and atom, the implementation enumerates $q_r(s)$, $\tilde{A}^t q_r(s)$ for $1 \le t \le K$, and $q_r(s) \odot \tilde{A}^t q_r(s)$. The same rules are applied on every graph and generate the candidate set $\mathcal{C}(G)$, not the final vocabulary.

B.2 Vocabulary-Selection Implementation

We implement the vocabulary-selection objective in equation 5 using deterministic subsampling. Each program view is flattened and restricted to at most 512 evenly spaced entries before centering and unit normalization. For redundancy reduction, we set $\tau = 10^{-3}$ and retain the shorter program whenever two normalized views have absolute inner product at least $1 - \tau$.

Starting from the original-feature program $q_0(h_0)$, we apply the greedy rule in Section 3.2.2 to candidates in $\overline{\mathcal{C}}(G)$. Projection gains are computed incrementally using a twice-reorthogonalized basis, avoiding repeated least-squares solutions. Candidates with $r(c \mid \mathcal{P}) \le \epsilon_{\mathrm{dep}}$, where $\epsilon_{\mathrm{dep}} = 10^{-3}$, are skipped. Selection continues until M programs have been chosen or no admissible candidate remains. The same sampling rule, thresholds, and vocabulary budget are used for every dataset.

C Encoder and Adaptation Details

C.1 Complexity Prior

The description length $\ell(p)$ counts the operations used to construct program p, including its transformations and propagation steps. Vocabulary selection assigns it the weight $w(p) = 2^{-\ell(p)}$, favoring programs with fewer operations. For predictor reweighting, a prototype predictor inherits the description length of its vocabulary entry, a propagated predictor adds one fixed propagation cost, and the support-label predictor is assigned a fixed cost. We standardize these costs as $\widetilde{\ell}(a) = (\ell(a) - \bar{\ell})/s_\ell$, where $\bar{\ell}$ and s_ℓ are their mean and standard deviation over $\mathcal{A}$, and use the complexity prior defined in Section 3.4. The prior is fixed before the support labels are used.

C.2 Source Pretraining Objective

The encoder and shared decoder are trained on source graphs using equation 13; no source labels are used, and the decoder is discarded after pretraining. To balance the source graphs, a no-gradient pass at the start of each epoch estimates their losses $\widehat{\mathcal{L}}_j$. Let $\overline{\mathcal{L}}$ and $\mathrm{sd}(\widehat{\mathcal{L}})$ be the mean and population standard deviation of these losses. We compute

$$\rho_j = \frac{\widehat{\mathcal{L}}_j - \overline{\mathcal{L}}}{\mathrm{sd}(\widehat{\mathcal{L}}) + 10^{-6}}, \qquad w_j = \frac{e^{\lambda_{\mathrm{src}}\rho_j}}{\sum_{k=1}^{S} e^{\lambda_{\mathrm{src}}\rho_k}}, \qquad \lambda_{\mathrm{src}} = 1, \tag{27}$$

where S is the number of source graphs. A second pass updates the encoder and decoder using $\sum_{j=1}^{S} w_j \mathcal{L}_j$. Sources with larger current losses receive more weight, while w_j remains fixed during the gradient update.

C.3 Support Evidence and Predictor Reweighting

The support losses $\mathcal{E}(a; \mathcal{S})$ in Section 3.4 determine how strongly the predictor weights should depart from the complexity prior. We measure how clearly the support set distinguishes the candidates by

$$\Delta(\mathcal{S}) = \left[|\mathcal{A}|^{-1} \sum_{a \in \mathcal{A}} \mathcal{E}(a; \mathcal{S}) - \min_{a \in \mathcal{A}} \mathcal{E}(a; \mathcal{S}) \right]_+, \tag{28}$$

$$T(\mathcal{S}) = T_0 \max\{T_{\min}, \exp[-\rho \Delta(\mathcal{S})]\}. \tag{29}$$

Here, $\Delta(\mathcal{S})$ is the gap between the average candidate loss and the lowest candidate loss. When this gap is small, the candidates perform similarly, meaning that the limited support labels do not clearly identify which predictor is better. The resulting larger temperature therefore keeps their weights closer to the prior. When the gap is large, the support set provides clearer evidence that some predictors are better than others; the lower temperature then places more weight on those predictors. The constants T_0, $T_{\min}$, and ρ set the overall scale, relative lower bound, and rate of decrease of the temperature, respectively. Substituting $T(\mathcal{S})$ into equation 16 gives the weights in equation 17.

C.4 Residual-Correction Objective

The residual correction in Section 3.4 minimizes

$$B^\star_{\mathcal{S}} = \arg\min_B \|H_{\mathcal{S}} B - R_{\mathcal{S}}\|_F^2 + \kappa \|B\|_F^2. \tag{30}$$

Setting its gradient with respect to B to zero gives

$$H_{\mathcal{S}}^\top (H_{\mathcal{S}} B - R_{\mathcal{S}}) + \kappa B = 0, \qquad (H_{\mathcal{S}}^\top H_{\mathcal{S}} + \kappa I) B = H_{\mathcal{S}}^\top R_{\mathcal{S}}. \tag{31}$$

Since $\kappa > 0$, the solution is unique and has the primal form

$$B^\star_{\mathcal{S}} = (H_{\mathcal{S}}^\top H_{\mathcal{S}} + \kappa I)^{-1} H_{\mathcal{S}}^\top R_{\mathcal{S}}. \tag{32}$$

The primal expression above requires inverting an $Md \times Md$ matrix. The identity $(H^\top H + \kappa I)^{-1} H^\top = H^\top (H H^\top + \kappa I)^{-1}$ rewrites the same solution as Equation 21, which instead inverts an $s \times s$ matrix. Because the support set is small, $s \ll Md$, this second computation is substantially cheaper. After obtaining $B^\star_{\mathcal{S}}$, equation 22 uses $h(v)^\top B^\star_{\mathcal{S}}$ as the correction to the original log probabilities. This calculation does not update any neural parameters, and all reported experiments use $\kappa = 1$.

D END-TO-END ALGORITHM

Algorithm 1 summarizes vocabulary construction, source-domain representation learning, and support-conditioned target adaptation. Vocabulary induction uses unlabeled program views, while target support labels enter the predictor-weighting and residual-correction stages.

Algorithm 1: GLEX training and few-shot adaptation

Input: source graphs $\{G_s = (A_s, X_s)\}_{s=1}^{S}$; target graph $G_t = (A_t, X_t)$; support set $\mathcal{S}_t$; construction bounds (K, R); vocabulary budget M.
Output: predictions for the target queries.

Program construction and source pretraining
for each graph $G \in \{G_1, \ldots, G_S\}$ **do**
Construct $\mathcal{C}(G)$ from (A, X) using equation 2.
Normalize program views; merge candidates whose absolute view correlation is at least $1 - \tau$, retaining the shorter description.
Initialize $\mathcal{P}_1(G)$ with the original-feature program.
for $t = 2, \ldots, M$ **do**
If no candidate satisfies $r(c \mid \mathcal{P}_{t-1}) > \epsilon_{\text{dep}}$, stop selection.
Among $c \in \overline{\mathcal{C}}(G) \setminus \mathcal{P}_{t-1}(G)$ with $r(c \mid \mathcal{P}_{t-1}) > \epsilon_{\text{dep}}$, select
$p_t = \arg\max_c \Delta_G(c \mid \mathcal{P}_{t-1})/\ell(c)$, and set $\mathcal{P}_t(G) = \mathcal{P}_{t-1}(G) \cup \{p_t\}$.
end for
end for
Train g_ϕ and source-only decoder d_ψ with the alignment and cross-view prediction losses in equation 13, weighted across sources by equation 27; discard d_ψ and freeze ϕ.

Target adaptation
Construct $\mathcal{P}(G_t)$ from the unlabeled target graph using the same program-construction and vocabulary-selection steps above.
Execute and encode every vocabulary entry $p_i \in \mathcal{P}(G_t)$ with the frozen g_ϕ; for graph classification, pool the representations as in Appendix F. Form the program prototype predictors.
For node classification, also form propagated scores and the support-label propagation candidate; collect the task-specific predictors in $\mathcal{A}$.
Compute $\mathcal{E}(a; \mathcal{S}_t)$ and $\pi_\eta(a)$ for each $a \in \mathcal{A}$, and compute $T(\mathcal{S}_t)$.
Compute $q^\star(a)$ by equation 17.
Form $p_{q^\star}(y \mid v, \mathcal{S}_t) = \sum_{a \in \mathcal{A}} q^\star(a)\,\mathrm{softmax}(a(v))_y$.
Form $h(v)$ using equation 19, and stack its support rows as $H_{\mathcal{S}_t}$.
Compute residuals $R_{\mathcal{S}_t} = Y_{\mathcal{S}_t} - p_{q^\star}(Y \mid \mathcal{S}_t)$.
Compute $B^\star_{\mathcal{S}_t} = H_{\mathcal{S}_t}^\top (H_{\mathcal{S}_t} H_{\mathcal{S}_t}^\top + \kappa I)^{-1} R_{\mathcal{S}_t}$.
return $\arg\max_y a_y^\star(v)$ using equation 22 for each query v.

E PROOFS OF THE THEORETICAL RESULTS

Stability of redundancy reduction. Let $e_c, e_{c'}$ be unit-normalized executions and suppose $|e_c^\top e_{c'}| \geq 1 - \tau$. Choosing $\sigma = \mathrm{sign}(e_c^\top e_{c'})$ gives

$$\|e_c - \sigma e_{c'}\|_2^2 = 2 - 2|e_c^\top e_{c'}| \leq 2\tau. \tag{33}$$

Let $\bar{e}_{[c]}$ be the retained representative assigned to c, with its sign chosen to match e_c. By the triangle inequality, a full-candidate score $f = \sum_c b_c e_c$ and its representative score $\bar{f} = \sum_c b_c \sigma_c \bar{e}_{[c]}$ satisfy

$$\|f - \bar{f}\|_2 \leq \sum_c |b_c| \|e_c - \sigma_c \bar{e}_{[c]}\|_2 \leq \sqrt{2\tau}\|b\|_1. \tag{34}$$

For class-indexed scores, the same bound applies after stacking the coefficients across classes. Thus, replacing redundant executions changes the resulting score vector by at most $\sqrt{2\tau}\|b\|_1$, provided that every original output is within the threshold of its final representative. The implemented sequential replacement does not guarantee this condition after multiple replacements, so Proposition 2 is stated only on the reduced candidate set.

Proof of Proposition 1: strict reconstruction-error decrease. Let e_c be a normalized retained output and let ϵ_{dep} be the dependence threshold used during greedy selection. If $r(c \mid \mathcal{P}) > \epsilon_{\text{dep}}$,

appending its normalized residual direction makes the residual of e_c zero. Since every term of D_G is nonnegative and projection onto a larger space cannot increase it,

$$D_G(\mathcal{P}) - D_G(\mathcal{P} \cup \{c\}) \geq \mu_G(c) r(c \mid \mathcal{P}) > \mu_G(c)\epsilon_{\text{dep}} > 0.$$

The first inequality holds because the candidate's own residual becomes zero and every other residual is nonincreasing. The selected candidate maximizes the actual decrease per code length among admissible candidates. This proves strict progress and excludes dependent executions at each step.

Proof of Proposition 2: task-score approximation. Write $u_c = (I - \Pi_{\mathcal{P}})e_c$, and stack the C class-score residuals. Because $\|U_y\|_{\text{op}} \leq 1$, the triangle inequality followed by weighted Cauchy–Schwarz yields

$$\left(\sum_y \left\| U_y \sum_c \gamma_{y,c} u_c \right\|_2^2 \right)^{1/2} \leq \sum_c \left(\sum_y \gamma_{y,c}^2 \right)^{1/2} \|u_c\|_2 \tag{35}$$

$$\leq \left(\sum_{y,c} \frac{\gamma_{y,c}^2}{\mu_G(c)} \right)^{1/2} \left(\sum_c \mu_G(c) \|u_c\|_2^2 \right)^{1/2}. \tag{36}$$

By the definition of $r(c \mid \mathcal{P})$, the weighted reconstruction error in equation 5 is

$$D_G(\mathcal{P}) = \sum_c \mu_G(c) \|u_c\|_2^2.$$

Substitution proves equation 23. The result is conditional on Assumption 1 over the sampled retained execution vectors.

Predictor-weight solution. The predictor-weight objective is strictly convex over distributions q when $\pi_\eta(a) > 0$ for every $a \in \mathcal{A}$. Let α enforce the constraint $\sum_{a \in \mathcal{A}} q(a) = 1$. Setting the derivative with respect to each $q(a)$ to zero gives

$$\omega \mathcal{E}(a; \mathcal{S}) + T(\mathcal{S}) \left(\log \frac{q(a)}{\pi_\eta(a)} + 1 \right) + \alpha = 0, \qquad a \in \mathcal{A}. \tag{37}$$

Rearranging shows that $q(a) \propto \pi_\eta(a) \exp[-\omega \mathcal{E}(a; \mathcal{S})/T(\mathcal{S})]$. Normalizing these values to sum to one gives $q^\star(a)$ in equation 17. The solution is unique because $T(\mathcal{S}) > 0$.

F Implementation and Model Selection

We use one global configuration for GLEX, while selecting only the input rank and graph-level readout settings on validation splits. Specifically, the construction bounds, ten-program budget $M = 10$, prediction-loss weight $\lambda_{\text{pred}} = 0.10$, source-weighting strength $\lambda_{\text{src}} = 1$, complexity-prior strength $\eta = 0.08$, support-evidence weight $\omega = 2$, residual regularization $\kappa = 1$, encoder architecture, and source-training schedule are fixed across all targets, shot counts, and prediction tasks.

Each split separates the support, validation, and query sets. The validation set contains 30 labeled examples per class in addition to the support budget; these labels are used for configuration selection, not for fitting target predictors. Query examples are sampled separately from the remaining nodes (approximately 300 in total, balanced across classes). The remaining target-specific choices for the main and m-shot comparisons are determined only from validation splits. Heterogeneous raw feature dimensions are mapped to a common rank by deterministic SVD, with the rank selected from $\{16, 32, 64\}$ for each target and task and then fixed across shot counts. For graph classification, the ego radius and readout (mean or center-attention pooling) are selected in the same way.

For each program, node representations within each ego-graph are pooled and unit-normalized to form the graph-example representation. Prototype prediction and residual correction then operate on these pooled representations. Graph classification uses the program prototype candidates without score or label propagation between ego-graph examples. The controlled graph ablations instead use the same radius-one mean-pooling profile for all variants and the same input rank within each target;

consequently, their full-model values need not equal those in Table 2. No query labels are used for configuration selection, vocabulary induction, source selection, or stopping.

Support/query splits are generated once and shared across all GLEX variants. Our evaluation follows the same protocol as SAMGPT, allowing us to directly use the baseline results reported in it. BRIDGE adopts a different evaluation setting, so we reproduce its results using the released code under our protocol. For the direct m-shot comparisons, all reruns use our fixed split manifests without changing the released model definitions.

G DATASETS AND BASELINES

This section complements Section 4.1 with a brief description of datasets and baselines.

G.1 DATASETS

Citation networks. **Cora** is a citation graph of machine-learning papers; nodes are papers, edges are citations, and the prediction task assigns each paper to one of seven research topics. **CiteSeer** has the same paper–citation structure but covers six computer-science topics. **PubMed** contains biomedical papers about diabetes, with citation links and three paper categories. These graphs test transfer among related network types with different feature spaces and label taxonomies (Sen et al., 2008).

Co-purchase networks. **Amazon Photo** and **Amazon Computers** represent products as nodes and frequent co-purchases as edges. Their labels are product categories: eight for Photo and ten for Computers. The two graphs share an e-commerce setting but differ in product content and graph structure (Shchur et al., 2019).

Social networks. **Facebook Page-Page** connects verified Facebook pages through mutual likes; the four node labels identify page categories. **LastFM Asia** connects music-service users through social ties, and its 18 labels identify users' countries. These targets contrast page-level and user-level social graphs with the citation and co-purchase networks (Rozemberczki et al., 2021).

Additional source graph. **Wiki-CS** links Wikipedia pages about computer science. It contributes a web-link network to source pretraining but is not used as a target in the seven-dataset evaluation (Mernyei & Cangea, 2020).

G.2 BASELINES

Supervised graph neural networks. **GCN** aggregates neighboring node features through graph convolutions (Kipf & Welling, 2017). **GAT** learns attention weights that vary the contribution of each neighbor (Veličković et al., 2018). They provide supervised reference points without a separate self-supervised pretraining stage.

Self-supervised graph pretraining. **DGI** trains node representations by contrasting local embeddings with a graph-level summary (Veličković et al., 2019). **GraphCL** learns graph representations from contrasting augmented views (You et al., 2020). **InfoGraph** maximizes agreement between graph representations and their substructures (Sun et al., 2020). **MVGRL** contrasts representations from adjacency-based and diffusion-based graph views (Hassani & Khasahmadi, 2020). These methods differ in their pretraining signals before target adaptation.

Prompt-based graph adaptation. **GPPT** reformulates node classification to match a link-prediction pretraining task (Sun et al., 2022). **GraphPrompt** uses a shared task template and learns a task-specific prompt for node or graph classification (Liu et al., 2023). **GPF** adapts a pretrained graph model through learnable input-feature prompts (Fang et al., 2023).

Multi-domain graph pretraining. **GCOPE** uses domain-specific graph coordinators to support joint pretraining across source graphs and transfers the resulting model to a target task (Zhao et al.,

Table 5: One-shot node-classification ablations.

Variant	Cora	CiteSeer	PubMed	Photo	Computers	Facebook	LastFM	Avg.	Δ
GLEX	48.78 ± 13.19	38.93 ± 9.31	59.20 ± 7.90	73.38 ± 4.29	57.67 ± 8.14	45.38 ± 6.15	47.29 ± 6.09	52.95	–
GLEX (fixed vocabulary)	48.78 ± 11.10	35.47 ± 6.48	55.80 ± 4.68	72.30 ± 3.49	58.00 ± 8.28	43.56 ± 5.90	50.42 ± 8.49	52.05	−0.90
GLEX w/o support adaptation	45.85 ± 10.03	37.60 ± 7.62	57.33 ± 3.23	71.82 ± 4.12	58.67 ± 7.69	39.27 ± 5.22	24.18 ± 1.72	47.82	−5.13
GLEX w/o program vocabulary	46.84 ± 10.17	34.87 ± 5.96	54.60 ± 4.41	73.99 ± 3.31	56.73 ± 7.87	40.38 ± 8.11	26.79 ± 2.70	47.74	−5.21

Table 6: One-shot graph-classification ablations.

Variant	Cora	CiteSeer	PubMed	Photo	Computers	Facebook	LastFM	Avg.	Δ
GLEX	50.14 ± 8.63	40.07 ± 8.32	57.00 ± 6.48	70.74 ± 3.46	57.53 ± 8.49	43.18 ± 6.54	38.68 ± 7.06	51.05	–
GLEX (fixed vocabulary)	49.70 ± 6.93	37.80 ± 6.16	55.13 ± 3.46	73.18 ± 4.89	60.47 ± 6.58	41.33 ± 7.06	32.52 ± 1.78	50.02	−1.03
GLEX w/o support adaptation	48.98 ± 10.07	39.73 ± 8.39	56.60 ± 6.41	69.07 ± 4.33	56.67 ± 9.07	43.22 ± 6.20	37.26 ± 6.73	50.22	−0.83
GLEX w/o program vocabulary	50.77 ± 6.81	36.73 ± 6.01	55.20 ± 3.10	73.24 ± 5.00	57.73 ± 6.49	41.18 ± 7.12	35.40 ± 1.79	50.04	−1.01

2024). **SAMGPT** aligns structural knowledge across source graphs with shared structure tokens and adapts to a target with holistic and specific prompts (Yu et al., 2025). **BRIDGE** aligns heterogeneous source features and selects transferable source knowledge for target adaptation (Yuan et al., 2025a).

H Additional Experimental Results

Node-classification ablations. Table 5 expands the averages in Figure 3. Replacing the induced vocabulary with a fixed one reduces the average by only 0.90 points, indicating that the fixed vocabulary remains competitive on several targets. In contrast, removing support adaptation or the program vocabulary lowers the average by 5.13 and 5.21 points, respectively, with the largest drops on LastFM. These results show that node classification benefits mainly from combining multiple vocabulary entries with support-conditioned prediction.

Graph-classification ablations. Table 6 reports the corresponding graph-classification results. The three variants reduce the seven-target average by 0.83–1.03 points, a smaller gap than for node classification. This suggests that ego-graph pooling already captures part of the structural information supplied by the program views, although the full model remains the most reliable overall.

I Additional Diagnostics

I.1 Full Hyperparameter Sensitivity

Figure 7 reports the complete node-classification sensitivity study on all seven target domains. The M, η, ω sweeps change exactly one quantity around $M = 10, \eta = 0.08, \omega = 2$ at one shot. Data splits, input rank, source pool, encoder, and transfer protocol remain fixed. The results reinforce the main trends: increasing vocabulary size eventually saturates or reduces accuracy, moderate complexity-prior strengths are stable, and the effect of support evidence varies across targets.

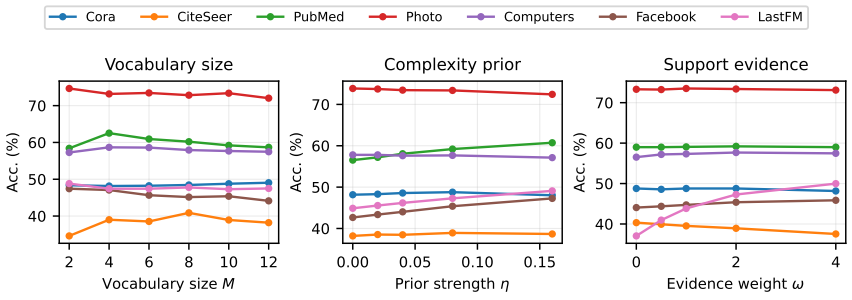
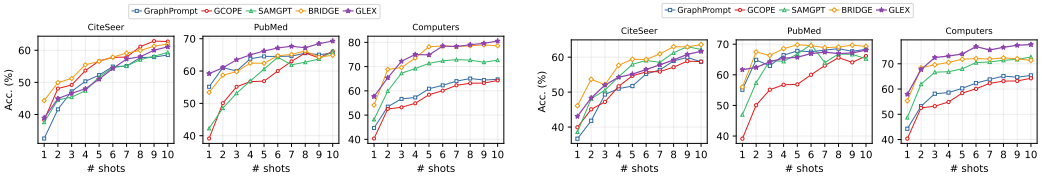


Figure 7: One-shot node-classification sensitivity on seven targets.



(a) Node classification **(b) Graph classification**

Figure 8: Direct m-shot accuracy on CiteSeer, PubMed, and Computers (five-split means).

I.2 Additional m-shot Results

Figure 8 completes the node- and graph-classification m-shot study on CiteSeer, PubMed, and Computers, the three targets not shown in Figure 2. Accuracy generally increases with the support size, but the relative gains remain target- and task-dependent. For node classification, GLEX ranks first at 9 of 10 shot counts on PubMed and 7 of 10 on Computers; on CiteSeer, BRIDGE and GCOPE lead at six and four shot counts, respectively. For graph classification, GLEX leads at 9 of 10 shot counts on Computers, whereas BRIDGE is strongest on most CiteSeer and PubMed settings. Overall, GLEX extends its one-shot gains across multiple support sizes on Computers and node-level PubMed, although this advantage does not hold uniformly across targets and tasks.

J Expanded Related Work

This section expands related work in Section 5 and relates GLEX to recent work on cross-domain graph interfaces and few-shot adaptation.

Graph pretraining and cross-domain foundation models. Contrastive methods differ in what constitutes two views: DGI contrasts local and global summaries, InfoGraph maximizes graph–substructure agreement, MVGRL contrasts diffusion and adjacency views, GraphCL searches over graph augmentations, and GCC transfers through subgraph discrimination (Veličković et al., 2019; Sun et al., 2020; Hassani & Khasahmadi, 2020; You et al., 2020; Qiu et al., 2020). Generative approaches predict context or reconstruct masked attributes and structure, including Strategies for Pre-training GNNs, GPT-GNN, GROVER, GraphMAE, and GraphMAE2 (Hu et al., 2020a;b; Rong et al., 2020; Hou et al., 2022; 2023); BGRL removes negative pairs through bootstrapping (Thakoor et al., 2022). These objectives learn transferable representations, but they do not explicitly specify a shared transfer interface across graphs with different feature spaces, semantics, and structural statistics.

Cross-domain GFMs address this interface explicitly. OFA converts heterogeneous graph tasks into text-attributed graph classification, while UniGraph and GraphGPT align graph structure with natural language (Liu et al., 2024; He et al., 2025; Tang et al., 2024). These approaches gain a shared semantic coordinate system. Among text-free methods, SAMGPT learns transferable structural tokens and BRIDGE aligns heterogeneous features before selectively assembling source knowledge (Yu et al., 2025; Yuan et al., 2025a). STAGE transfers statistical dependencies rather than raw attributes across distinct feature domains, while AutoGFM searches architectures tailored to each domain and task (Shen et al., 2025; Chen et al., 2025). GLEX instead keeps one shared encoder and transfers explicit computations: common construction rules produce graph-specific vocabularies whose predictors are reweighted by target support labels.

Graph prompting and few-shot adaptation. GPPT recasts node classification in a pretraining-compatible prompt form, GraphPrompt uses a unified template for node- and graph-level tasks, and GPF learns a universal feature prompt (Sun et al., 2022; Liu et al., 2023; Fang et al., 2023). All in One represents prompts as learnable graph tokens and uses meta-learning across tasks, PRODIGY performs graph in-context learning from demonstrations, and GCOPE introduces structural prompt information for cross-domain transfer (Sun et al., 2023; Huang et al., 2023; Zhao et al., 2024). ProG systematizes this area by distinguishing prompts represented as graphs from prompts represented as tokens and benchmarking both families (Zi et al., 2024). These methods show that lightweight adaptation can replace full-model fine-tuning, but their prompt parameters, initialization, or optimization

remain part of the target procedure. GLEX instead conditions its program predictors on the target support set through reweighting and a regularized residual correction.

Transferable graph patterns and vocabularies. The search for reusable graph units predates GFMs. Neighborhood aggregation represents nodes through rooted computation trees (Hamilton et al., 2017; Xu et al., 2019); Weisfeiler–Lehman kernels compare iterative neighborhood refinements, and graph2vec learns distributed representations of rooted subgraphs (Shervashidze et al., 2011; Narayanan et al., 2017). Modern pretraining methods often preserve such patterns implicitly inside an encoder. GRAVER instead learns factor-aware structure–feature vocabularies and generates target-aligned support augmentations (Yuan et al., 2025b). GLEX also makes the transferred units explicit, but an element denotes a computation rather than a subgraph token.