PRODIGY: Enabling In-context Learning Over Graphs

Qian Huang [*1] **Hongyu Ren** [*1] **Peng Chen** [1] **Gregor Kržmanc** [2] **Daniel Zeng** [1] **Percy Liang** [1] **Jure Leskovec** [1]

Abstract

In-context learning is the ability of a pretrained model to adapt to novel and diverse downstream tasks by conditioning on prompt examples, without optimizing any parameters. While large language models have demonstrated this ability, how in-context learning could be performed over graphs is unexplored. In this paper, we develop **Pr**etraining **O**ver **D**iverse **I**n-Context **G**raph **Sy**stems (PRODIGY), the first pretraining framework that enables in-context learning over graphs. The key idea of our framework is to formulate in-context learning over graphs with a novel *prompt graph* representation, which connects prompt examples and queries. We then propose a graph neural network architecture over the prompt graph and a corresponding family of in-context pretraining objectives. With PRODIGY, the pretrained model can directly perform novel downstream classification tasks on unseen graphs via in-context learning. We provide empirical evidence of the effectiveness of our framework by showcasing its strong in-context learning performance on tasks involving citation networks and knowledge graphs. Our approach outperforms the in-context learning accuracy of contrastive pretraining baselines with hard-coded adaptation by 18% on average across all setups. Moreover, it also outperforms standard finetuning with limited data by 33% on average with in-context learning.

1. Introduction

In-context learning is a novel and one of the most intriguing capabilities of language models (Brown et al., 2020). It refers to the capability of a pretrained model to perform novel and diverse tasks directly at the prediction time when prompted with just a few examples, without the need to update the model weights. For example, a person may describe the new task (*e.g.*, question answering, machine translation, or code generation) using natural language and demonstrate it to the language model with several prompt examples. The language model then directly without any model training or finetuning performs the task.

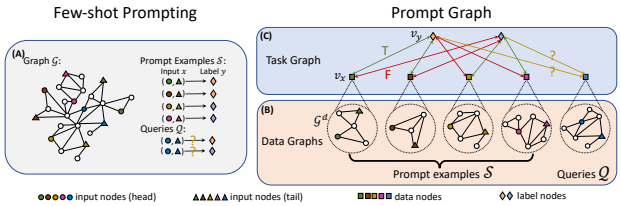

Figure 1. In-context few-shot prompting over graphs with prompt graph for *edge classification* in PRODIGY. (A) Given the source graph $\mathcal{G}$, we provide prompt examples $\mathcal{S}$ that consist of the input head/tail nodes and their labels, as well as the queries. (B) For each datapoint from both prompt examples and the queries, we first construct its data graph $\mathcal{G}^{\text{D}}$ by retrieving context from the source graph $\mathcal{G}$. (C) Then we create a task graph to capture the connection between each datapoint and each label, which includes a data node v_x for each datapoint and a label node v_y for each label in $\mathcal{Y}$. Each pair of data and label nodes are connected with edge attributes corresponding to their binary labels.

However, how to enable in-context learning for diverse graph machine learning tasks, such as identifying misinformation spreader in social networks (Sharma & Sharma, 2021) and product suggestions across online e-commerce websites (Wu et al., 2020), still remain unexplored and challenging. An in-context learner for graphs should be able to solve novel tasks on novel graphs. For example, give music product recommendations on Spotify when being trained on Amazon purchasing graph. The first challenge here is how to formulate and represent node-, edge- and graph-level tasks over graphs with a unified task representation that allows the model to solve diverse tasks without the need for retraining or parameter tuning. In other words, the key challenge is: what is an analog of natural language prompting for graph machine learning tasks? The second challenge is how to design model architecture and pretraining objectives that enable models to achieve in-context learning capability across diverse tasks and diverse graphs in the unified task representation. Existing graph pretraining methods (Hu et al., 2019; You et al., 2020; Hu et al., 2020b; Qiu et al., 2020) only aim to learn a good graph encoder and require fine-tuning to adapt to different tasks, while existing meta-learning methods over graphs (Wang et al., 2022;

[*]Equal contribution [1]Stanford University [2]University of Ljubljana. Correspondence to: Qian Huang <qhwang@cs.stanford.edu>, Hongyu Ren <hyren@cs.stanford.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

Huang & Zitnik, 2020; Chen et al., 2019; Sun et al., 2021; Zhang et al., 2020) only aim to generalize across different tasks within the same graph. On the other hand, achieving in-context learning requires tackling the more difficult setting of generalizing across the graphs *and* tasks without finetuning.

Here we present a general approach for solving these two challenges for classification tasks on graphs: (1) *prompt graph*, an in-context graph task representation, and (2) **Pr**etraining **O**ver **D**iverse **I**n-Context **G**raph **Sy**stems (PRODIGY), a framework for pretraining an in-context learner over prompt graphs.

We propose *prompt graph* (Figure 1) to provide unified way to represent diverse node-, edge- and graph-level machine learning tasks. Prompt graph first contextualizes the input nodes/edges on which we make prediction (including both the prompt examples and the queries), then connects them with additional label nodes, such that the prompt examples are interconnected with queries. Such a unified representation allows us to specify diverse graph machine learning tasks to the same model regardless of the graph size.

PRODIGY then designs both model architecture and pretraining objectives with the prompt graph in-context task formulation, such that the model is pretrained to solve tasks across a wide range of tasks and graphs, and can continue to do so out-of-the-box. We design a graph architecture that utilizes graph neural networks to learn node/edge representations and an attention mechanism to communicate over prompt graph. Furthermore, we propose a family of in-context pretraining objectives over prompt graph. In particular, this includes a novel self-supervised pretraining task, *neighbor matching*, where we classify which neighborhood a node or edge belongs to.

We use PRODIGY framework to pretrain on citation networks (MAG240M (Hu et al., 2021)) and knowledge graphs (Wiki (Xiong et al., 2018b)). We then show that such model (without any retraining) provides strong performance on in-context paper category classification and knowledge graph completion tasks on novel graphs it was never trained on (arXiv, ConceptNet, FB15K-237, NELL) (Hu et al., 2020a; Speer et al., 2017; Xiong et al., 2018a). Specifically, PRODIGY improves upon contrastive pretraining baselines with hard-coded adaptation for in-context setup by 18% on average across all datasets and numbers of labels to classify among. Moreover, it also outperforms standard finetuning with limited data by 32.6% on average with in-context learning. It even outperforms the state-of-the-art few-shot learning methods trained on the testing downstream graph with pure in-context learning. Finally, we further demonstrate that our methods achieve increasingly higher performance with more examples in the prompt even beyond what it was pretrained with, which shows that the model really learns to learn from context.

2. In-context Learning over Graphs

In this work, we specifically focus on in-context learning for node and edge classification tasks on graphs with few-shot prompting, which are the forms of the most standard and important graph machine learning tasks.

2.1. Few-shot Prompting

We define a graph as $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathcal{R})$, where $\mathcal{V}$, $\mathcal{E}$, $\mathcal{R}$ represent the set of nodes, edges and relations. An edge $e=(u,r,v)\in\mathcal{E}$ consists of a subject $u\in\mathcal{V}$, a relation $r\in\mathcal{R}$ and an object $v\in\mathcal{V}$.

Next we define in-context learning setup for classification tasks over graphs with few-shot prompting. For a k-shot prompt with a downstream m-way classification tasks with $|\mathcal{Y}|=m$ classes, we use a small number of input-label pairs $\mathcal{S}=\{(x_i,y_i)\}_{i=1}^{m\cdot k}$ as *prompt examples* of the task specification, such that there are k input-label pairs with label y for each $y\in\mathcal{Y}$. We also give the model a set of queries $\mathcal{Q}=\{x_i\}_{i=1}^{n}$ that we want to predict labels for. Generally, the input x_i can be a subgraph to represent classification tasks over both nodes and edges.

Given the above information as the prompt, the pretrained model should be able to directly output the predicted labels for each datapoint in $\mathcal{Q}$ via in-context learning.

2.2. Prompt Graph Representation

Inspired by (Cao et al., 2022), we propose prompt graph as a unified representation of a k-shot prompt over graphs for an m-way classification task (Figure 1). A prompt graph is composed of *data graphs* and a *task graph*:

Data graph. To construct a prompt graph, we first perform contextualization of each datapoint $x_i=(\mathcal{V}_i,\mathcal{E}_i,\mathcal{R}_i)$ in $\mathcal{S}$ and $\mathcal{Q}$ in the source graph $\mathcal{G}$ to form data graphs. The goal is to gather more information about the x_i from the source graph $\mathcal{G}$ without having to represent the entire source graph explicitly. Here we construct data graph $\mathcal{G}_i^{\text{D}}$ by sampling k-hop neighborhood of $\mathcal{V}_i$ in $\mathcal{G}$. With this data graph $\mathcal{G}_i^{\text{D}}$, we call the node set that corresponds to the nodes in $\mathcal{V}_i$ before contextualization *input node set*, *e.g.*, the target node to classify in node classification task and the pair of nodes in link prediction task.

Task graph. After contextualizing each datapoint to a data graph $\mathcal{G}^{\text{D}}$, we then construct task graph $\mathcal{G}^{\text{T}}$ to better capture the connection and relationship among the inputs and the labels. For each data graph $\mathcal{G}_i^{\text{D}}$ from the previous stage, we have a *data node* v_{x_i} that represents each input; for each label, we have a *label node* v_{y_i}.

Now we add edges between the data nodes and the label nodes: For the query set, since we do not know the labels of each graph, we add single directional edges from all label

nodes to each datapoint in the query set, *i.e.*, each query data node v_{x_i} will be connected to all the label nodes as shown by the yellow edges in Figure 1; For the prompt examples, we connect each data node to all the label nodes, where the edge with the true labels is marked as `T` while the others are marked as `F`, as shown by the green and red edges in Figure 1 respectively.

3. Pretraining to Enable In-context Learning

In this section, we introduce PRODIGY, a general pretraining framework that is designed specifically for enabling in-context learning over downstream classification tasks without any additional finetuning steps on arbitrary new graphs. It has two main components: model architecture over prompt graph and in-context pretraining objectives.

3.1. Message Passing Architecture over prompt graph

Our model architecture over the prompt graph consists of two submodules:

Data graph Message Passing. First, we apply a message passing GNN module M_{D} that learns node representation E for nodes in each $\mathcal{G}^{\text{D}}$.

$$E \in \mathcal{R}^{|\mathcal{V}^{\text{D}}| \times d} = M_{\text{D}}(\mathcal{G}^{\text{D}}) \tag{1}$$

where d is the embedding dimension. M_{D} can be implemented in multiple ways, such as using Graph Convolutional Network (GCN) or Graph Attention Networks (GAT) (Kipf & Welling, 2016; Veličković et al., 2017). We then read out a single embedding G_i for each data graph by pooling the embeddings E of nodes in $\mathcal{V}_i$ (see Appendix E).

Task graph Message Passing. We apply another GNN M_{T} on the task graph to obtain updated representation of data nodes and label nodes.

$$H = M_{\text{T}}(\mathcal{G}^{\text{T}}) \tag{2}$$

where H is the obtained embedding per node. The initial embedding of data node v_{x_i} is G_i and the embedding of label node v_{y_i} can either be initialized with random Gaussian or additional information available about the labels. See the architecture detail in the appendix F.

Prediction Read Out. Finally, we readout the classification logits O_i by taking cosine similarity between each pair of query graph representation and label representation, as in contrastive learning:

$$O_i = [\texttt{cosine_similarity}(H_{x_i}, H_y), \forall y \in \mathcal{Y}] \tag{3}$$

3.2. In-context Pretraining Objectives

We detail our proposed family of in-context pretraining objectives in terms of three components: 1) pretraining task generation as few-shot prompts (*i.e.* Figure 1(A)) and 2) converting generated few-shot prompt to prompt graph format (*i.e.* Figure 1(B,C)) with augmentation. We will only present the high level idea and defer the formal definitions to Appendix C.

3.2.1. Pretraining Task Generation

We propose two methods to generate pretraining tasks from the pretraining graph in the form of few-shot prompts: *neighbor matching* and *multi-task*.

Neighbor Matching. Given the pretraining graph, we construct self-supervised in-context pretraining tasks with the goal of classifying which local neighborhood a node belongs to, where each local neighborhood is defined by the example nodes belonging to that neighborhood. Intuitively, we sample multiple subgraphs from the pretraining graph as the local neighborhoods, and we say a node belongs to a local neighborhood if it is in the sampled subgraph.

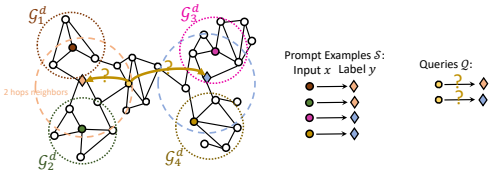

Figure 2. Neighbor matching pretraining task generation. The key idea of neighbor matching is to classify whether a node/edge is in the local neighborhood of a set of sampled nodes (as labels). Given a set of sampled nodes and their two-hop neighbors as prompt examples, we aim to classify whether a query node is also a two-hop neighbor of each sampled node.

Multi-task. When the pretraining graphs have node or edge-level labeling $f(x_i) = y_i \in \mathcal{Y}$ for some $x_i \in \mathcal{V}_{\texttt{pretrain}}$ or $\mathcal{E}_{\texttt{pretrain}}$, we can further leverage this signal to perform supervised pretraining. Similar to neighbor matching, the key is to construct such supervised pretraining tasks in the format of few-shot prompts and corresponding labels. The benefit of such a supervised pretraining objective is that it could directly resemble the format of downstream tasks, compared with neighbor matching objective, which may only serve as a surrogate. However, it requires extra labels if f is not part of pretraining graph, e.g. node classification labels that may not exist for some pretraining graphs.

3.2.2. Prompt graph generation with augmentation

After we obtained the few-shot prompts and labels for either of the two tasks (NeighborMatching and multi-task), we need to construct the prompt graph for pretraining. In addition to the standard construction process described in subsection 2.2, we add an additional augmentation step to augment the data graphs as inspired by Contrastive Learning. The key insight is to corrupt data graph such that the pretrained model learns representation invariant to various corruptions. We adopt graph augmentation techniques for

Table 1. In-context learning accuracy (%) on arXiv paper category classification on 500 sampled test tasks with 3-shot prompts. PRODIGY was pretrained on MAG240M and is then applied in-context to arXiv, which has completely different structure and a different set of paper categories. PG-NM and PG-MT are ablations of PRODIGY.

Classes	NoPretrain	Contrastive	PG-NM	PG-MT	PRODIGY	Finetune
3	33.16 ± 0.30	65.08 ± 0.34	72.50 ± 0.35	65.64 ± 0.33	**73.09 ± 0.36**	65.42 ± 5.53
5	18.33 ± 0.21	51.63 ± 0.29	61.21 ± 0.28	51.97 ± 0.27	**61.52 ± 0.28**	53.49 ± 4.61
10	9.19 ± 0.11	36.78 ± 0.19	46.12 ± 0.19	37.23 ± 0.20	**46.74 ± 0.20**	30.22 ± 3.77
20	4.72 ± 0.06	25.18 ± 0.11	33.71 ± 0.12	25.91 ± 0.12	**34.41 ± 0.12**	17.68 ± 1.15
40	2.62 ± 0.02	17.02 ± 0.07	23.69 ± 0.06	17.19 ± 0.08	**25.13 ± 0.07**	8.04 ± 3.00

this and see details in Appendix C.2.

4. Experiments

4.1. Experimental Setup

Datasets. For pretraining, we use two datasets: `MAG240M` (Hu et al., 2021), and `Wiki`, a knowledge graph (KG) constructed from Wikipedia (Xiong et al., 2018b). After the model is pretrained we evaluate its in-context learning capability on diverse classification tasks over 4 graphs: `arXiv` (Hu et al., 2020a), `ConceptNet` (Speer et al., 2017), `FB15K-237` (Xiong et al., 2018a), `NELL` (Xiong et al., 2018a).

With its original train, validation and test splits, we construct a k-shot prompt for test nodes (or edges) from the test split by randomly selecting k examples per way from these available examples. This allows us to test the model's ability to learn in-context relationships and perform well on classification tasks with truly limited known labels. We use $k=3$.

Methods and Baselines. We consider three versions of PRODIGY: 1) PG-NM, which uses neighbor matching task for pretraining; 2) PG-MT, which employs multi-task pretraining; and 3) full PRODIGY, which combines the two. To augment the data, we use DropNode and MaskNode augmentations with a probability of 0.5 per node for each method.

We consider three baselines for comparison: 1) NoPretrain, which uses a randomly-initialized model with the same architecture as our models; 2) Contrastive (You et al., 2020), which employs a standard contrastive learning method with the same augmentation as above and uses a hard-coded nearest neighbor algorithm to adapt to our in-context learning setting. 3) Finetune (Hu et al., 2019), which trains an additional linear classification head on top of the graph encoder pretrained with contrastive learning, following the standard practice.

4.2. In-Context Learning Results

We evaluate the in-context learning capability for node classification and link prediction with various numbers of ways (i.e. number of classes to classify among) in Table 1 and Table 3. We include ablation and auxiliary results in Appendix H.

Strong in-context learning performance. The results demonstrate that our method PRODIGY consistently outperforms all other baselines in this setting. It achieves the highest average accuracy across all ways on `arXiv`, with an average improvement of 28.6% and up to 48% over the best baseline of Contrastive. Over KGs, PRODIGY also outperforms contrastive learning on average by 12.2%. PRODIGY also demonstrates similar-to-better performance compared to Finetune, which requires additional training on downstream tasks. On `arXiv`, we see an average improvement of 77.7% over all ways. This can be attributed to the diverse set of pretraining tasks incorporated in PRODIGY, which allows the model to avoid overfitting on specific tasks and learn in-context.

Self-supervised pretraining PG-NM bridges different tasks. In particular, we highlight that the pure self-supervised pretraining method PG-NM produces significantly higher in-context learning performance over `arXiv` than baselines, even though the model is pretrained on different tasks from the downstream task. This advantage can be further leveraged by pretraining on even larger-scale unlabeled datasets. On the other hand, PG-MT follows the supervised pretraining objective that directly resembles the format of downstream tasks. On KGs, this allows PG-MT to adapt better to downstream task even sometimes compared to the full PRODIGY (marked by underlines), while PG-NM might have overfitted to the incorrect strategy of only identifying co-occurring nodes. Yet, PG-MT performs worse on `arXiv` potentially due to less diversity. The full PRODIGY, which ensembles the two, achieves more diversity than either single task and therefore achieves the best performance over both worlds.

5. Conclusion

We introduce PRODIGY, the first framework that enables in-context learning on graphs. A model that is pretrained using PRODIGY can seamlessly execute a new classification task over new graphs represented by prompt graph. It markedly surpasses the performance of other baseline models with in-context learning, even those that employ finetuning, in both the node and edge classification tasks.

Acknowledgements

We thank Camilo Ruiz and Michael Moor for discussions and for providing feedback on our manuscript. We also gratefully acknowledge the support of DARPA under Nos. HR00112190039 (TAMI), N660011924033 (MCS); ARO under Nos. W911NF-16-1-0342 (MURI), W911NF-16-1-0171 (DURIP); NSF under Nos. OAC-1835598 (CINES), OAC-1934578 (HDR), CCF-1918940 (Expeditions), NIH under No. 3U54HG010426-04S1 (HuBMAP), Stanford Data Science Initiative, Wu Tsai Neurosciences Institute, Amazon, Docomo, GSK, Hitachi, Intel, JPMorgan Chase, Juniper Networks, KDDI, NEC, and Toshiba. Hongyu Ren is supported by Masason Foundation PhD fellowship, Apple PhD fellowship and Baidu PhD scholarship. Qian Huang is supported by Open Philanthropy AI fellowship.

The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding entities.

References

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Cao, K., You, J., and Leskovec, J. Relational multi-task learning: Modeling relations between data and tasks. In *International Conference on Representation Learning (ICLR)*, 2022.

Chen, M., Zhang, W., Zhang, W., Chen, Q., and Chen, H. Meta relational learning for few-shot link prediction in knowledge graphs. *arXiv preprint arXiv:1909.01515*, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020a.

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

Hu, Z., Dong, Y., Wang, K., Chang, K.-W., and Sun, Y. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1857–1867, 2020b.

Huang, K. and Zitnik, M. Graph meta learning via local subgraphs, 2020. URL https://arxiv.org/abs/2006.07889.

Huang, Q., Ren, H., and Leskovec, J. Few-shot relational reasoning via connection subgraph pretraining. *Advances in neural information processing systems*, abs/2210.06722, 2022.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks, 2016. URL https://arxiv.org/abs/1609.02907.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Qiu, J., Chen, Q., Dong, Y., Zhang, J., Yang, H., Ding, M., Wang, K., and Tang, J. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1150–1160, 2020.

Sharma, S. and Sharma, R. Identifying possible rumor spreaders on twitter: A weak supervised learning approach. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021. doi: 10.1109/IJCNN52387.2021.9534185.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 2020.

Speer, R., Chin, J., and Havasi, C. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Sun, J., Zhou, Y., and Zong, C. One-shot relation learning for knowledge graphs via neighborhood aggregation and paths encoding. *Transactions on Asian and Low-Resource Language Information Processing*, 21(3):1–19, 2021.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

Wang, S., Ding, K., Zhang, C., Chen, C., and Li, J. Task-adaptive few-shot node classification. *Proceedings*

of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2022.

Wu, S., Zhang, W., Sun, F., and Cui, B. Graph neural networks in recommender systems: A survey. *ACM Computing Surveys*, 55:1 – 37, 2020.

Xiong, W., Yu, M., Chang, S., Guo, X., and Wang, W. Y. One-shot relational learning for knowledge graphs. In *EMNLP*, 2018a.

Xiong, W., Yu, M., Chang, S., Guo, X., and Wang, W. Y. One-shot relational learning for knowledge graphs, 2018b. URL https://arxiv.org/abs/1808.09040.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

Zhang, C., Yao, H., Huang, C., Jiang, M., Li, Z., and Chawla, N. V. Few-shot knowledge graph completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3041–3048, 2020.

A. Extension

It is also possible to extend prompt graph to non-classification tasks and free-form text prompting. For example, for numerical regression (e.g. molecular energy prediction) and other free-form generation tasks (e.g. text generation), one can extend our task graph to contain vector values on the edges to represent y_i. Then different label nodes would represent different prediction tasks. To support more general forms of prompting, one can include additional task information and instructions in the feature of label nodes, and additional description paired with each datapoint in the global feature in data graph.

B. Attribute Prediction Loss

For each augmented DataGraph $\mathcal{G}^{aug}$, the certain node features F_v are masked during MaskNode augmentation. Therefore, we can reconstruct them using the learned embedding E_v with a MLP and train with MSE reconstruction Loss.

$$\mathcal{L}_{attr}(\mathcal{G}^{aug}) = \frac{1}{|\mathcal{V}^{\mathrm{D}}|}\sum_{v}\mathtt{MSE}(F_v, \mathtt{MLP}(E_v))$$

C. In-context Pretraining Objectives

We assume access to a pretraining graph $\mathcal{G}_{\mathtt{pretrain}}$ that is independent of the source graph $\mathcal{G}$ for the downstream task.

C.1. Pretraining Task Generation

We propose two methods to generate pretraining tasks from the pretraining graph $\mathcal{G}_{\mathtt{pretrain}}$ in the form of few-shot prompts: *neighbor matching* and *multi-task*.

Neighbor Matching. Given the pretraining graph, we construct self-supervised in-context pretraining tasks with the goal of classifying which local neighborhood a node belongs to, where each local neighborhood is defined by the example nodes belonging to that neighborhood. Intuitively, we sample multiple subgraphs from the pretraining graph $\mathcal{G}_{\mathtt{pretrain}}$ as the local neighborhoods, and we say a node belongs to a local neighborhood if it is in the sampled subgraph.

Formally, we denote $\mathtt{NM}_{k,m}$ as a sampler that generates m-way neighbor matching tasks, where each includes a k-shot prompt $(\mathcal{G}_{\mathtt{pretrain}}, \mathcal{S}_{\mathrm{NM}}, \mathcal{Q}_{\mathrm{NM}})$ (see subsection 2.1 and Figure 1(A)) and the labels of the queries. For simplicity of the notation, we will include the labels in $\mathcal{Q}_{\mathrm{NM}}$ as paired with the inputs:

$$(\mathcal{G}_{\mathtt{pretrain}}, \mathcal{S}_{\mathrm{NM}}, \mathcal{Q}_{\mathrm{NM}}) \sim \mathtt{NM}_{k,m}(\mathcal{G}_{\mathtt{pretrain}}) \tag{4}$$

To generate these, we first sample m nodes from the pretraining graph $\mathcal{G}_{\mathtt{pretrain}}$, where each of the sampled node corresponds to one class.

$$\mathcal{C} = \{c_i\}_{i=1}^{m} \quad c_i \sim \mathit{Uniform}(\mathcal{V}_{\mathtt{pretrain}}) \tag{5}$$

For each sampled node/class c_i, we sample k different nodes from its exact l-hop neighbors. These k nodes serve as examples

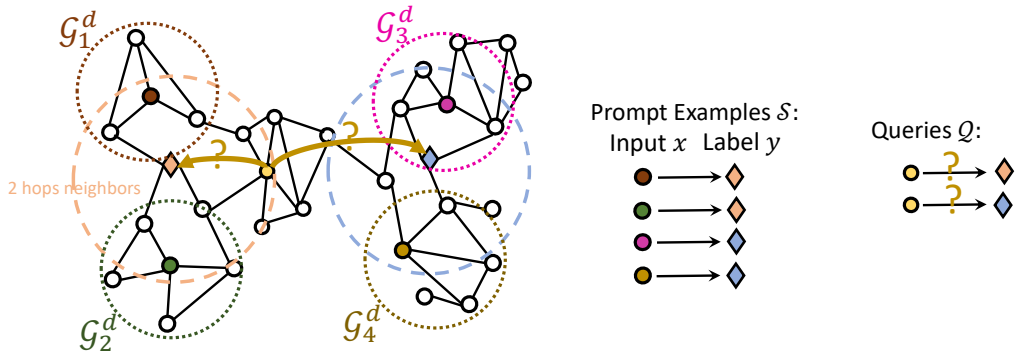

Figure 3. Neighbor matching pretraining task generation. The key idea of neighbor matching is to classify whether a node/edge is in the local neighborhood of a set of sampled nodes (as labels). Given a set of sampled nodes and their two-hop neighbors as prompt examples, we aim to classify whether a query node is also a two-hop neighbor of each sampled node.

of label c_i. We also sample additional $\lceil \frac{n}{m} \rceil$ nodes similarly for each label c_i to form the query set. Formally,

$$N_i = \texttt{Neighbor}(c_i, \mathcal{G}_{\texttt{pretrain}}, l) \tag{6}$$

$$\mathcal{S}_i = \{(x_j, y_j = c_i)\}_{j=1}^{k} \quad x_j \sim \mathit{Uniform}(N_i) \tag{7}$$

$$\mathcal{Q}_i = \{(x_j, y_j = c_i)\}_{j=1}^{\lceil \frac{n}{m} \rceil} \quad x_j \sim \mathit{Uniform}(N_i) \tag{8}$$

In such a way, we constructed a neighbor matching pretraining task sample in the format of a few-shot prompt $(\mathcal{G}_{\texttt{pretrain}}, \mathcal{S}_{\text{NM}} = \bigcup \mathcal{S}_i, \mathcal{Q}_{\text{NM}} = \bigcup \mathcal{Q}_i)$.

The neighbor matching task generation process outlined above is specifically applicable when the downstream tasks are also node classification. When the downstream task is link prediction, we may adapt the above neighbor matching tasks to over edges correspondingly. Specifically, we can expand each sampled input node x_i to an edge by randomly sampling an edge that contains x_i. Then, instead of classifying to which neighborhood a node in the query set belongs, now the neighbor matching task is to classify to which neighborhood an edge in the query set belongs.

Multi-task. When the pretraining graphs have node or edge-level labeling $f(x_i) = y_i \in \mathcal{Y}$ for some $x_i \in \mathcal{V}_{\texttt{pretrain}}$ or $\mathcal{E}_{\texttt{pretrain}}$, we can further leverage this signal to perform supervised pretraining. Similar to neighbor matching, the key is to construct such supervised pretraining tasks in the format of few-shot prompts and corresponding labels.

$$(\mathcal{G}_{\texttt{pretrain}}, \mathcal{S}_{\text{MT}}, \mathcal{Q}_{\text{MT}}) \sim \texttt{MT}_{k,m}(\mathcal{G}_{\texttt{pretrain}}, f) \tag{9}$$

For node classification tasks, we first sample m labels from the whole label set. Then, for each label, we directly sample k nodes as support examples and $\lceil \frac{n}{m} \rceil$ nodes with labels in this set as query examples.

$$\mathcal{C} = \{c_i\}_{i=1}^{m} \quad c_i \sim \mathit{Uniform}(\mathcal{Y}) \tag{10}$$

$$\mathcal{S}_i = \{(x_j, y_j = c_i)\}_{j=1}^{k} \quad x_j \sim \mathit{Uniform}(\{x_i | f(x_i) = c_i\}) \tag{11}$$

$$\mathcal{Q}_i = \{(x_j, y_j = c_i)\}_{j=1}^{\lceil \frac{n}{m} \rceil} \quad x_j \sim \mathit{Uniform}(\{x_i | f(x_i) = c_i\}) \tag{12}$$

We then construct a task with the few-shot prompt as $(\mathcal{G}_{\texttt{pretrain}}, \mathcal{S}_{\text{MT}} = \bigcup \mathcal{S}_i, \mathcal{Q}_{\text{MT}} = \bigcup \mathcal{Q}_i)$. For link prediction, we directly use the edge type function as f, i.e. $f((v_1, v_2)) = r \iff (v_1, r, v_2) \in \mathcal{E}$. With this f, we may directly sample m edge types and construct pretraining tasks in a similar way as above.

The benefit of such a supervised pretraining objective is that it could directly resemble the format of downstream tasks, compared with neighbor matching objective, which may only serve as a surrogate. However, it requires extra labels if f is not part of $\mathcal{G}_{\texttt{pretrain}}$, e.g. node classification labels that may not exist for some pretraining graphs.

C.2. Prompt graph generation with augmentation

After we obtained the few-shot prompts and labels for either of the two tasks (NeighborMatching and multi-task), we need to construct the prompt graph for pretraining. In addition to the standard construction process described in subsection 2.2, we add an additional augmentation step to augment the data graphs as inspired by Contrastive Learning. The key insight is to corrupt data graph such that the pretrained model learns representation invariant to various corruptions.

Here we demonstrate how we adopt graph augmentation techniques during the construction of prompt graph from a few-shot prompt generated from $\mathcal{G}_{\texttt{pretrain}}$. We first still sample the k-hop neighbor subgraph of each sample $\mathcal{G}_i$ in the prompt examples and queries: $\mathcal{G}_i^{\texttt{D}} \sim \bigoplus_{j=1}^{k} \texttt{Neighbor}(\mathcal{G}_i, \mathcal{G}_{\texttt{pretrain}}, j)$. Then we adopt the following two augmentation techniques to create augmented data graph $\mathcal{G}_i^{aug}$, including (1) node dropping, and (2) node feature masking (You et al., 2020). For node dropping, we randomly drop nodes from the k-hop neighbor subgraph and take the remaining graph as $\mathcal{G}_i^{aug} = \texttt{DropNode}(\mathcal{G}_i^{\texttt{D}})$. For node feature masking, we randomly mask the feature of a subset of nodes with value zero to create $\mathcal{G}_i^{aug} = \texttt{MaskNode}(\mathcal{G}_i^{\texttt{D}})$. With the augmented data graphs for each datapoint in the prompt examples and the queries, we may accordingly construct the task graph $\mathcal{G}^{\texttt{T}}$ by creating a data node v_{x_i} for each augmented data graphs and the label node v_{y_i} as introduced in subsection 2.2. Combining data graphs with task graph, we obtain the prompt graph formulation with augmentation for the few-shot prompt.

C.3. Pretraining Loss

Finally, we pretrain the model with the cross-entropy objectives over generated prompt graphs:

$$(\mathcal{G}_{\texttt{pretrain}}, \mathcal{S}_{\text{NM}}, \mathcal{Q}_{\text{NM}}) \sim \texttt{NM}_{k,m}(\mathcal{G}_{\texttt{pretrain}}) \tag{13}$$

$$(\mathcal{G}_{\texttt{pretrain}}, \mathcal{S}_{\text{MT}}, \mathcal{Q}_{\text{MT}}) \sim \texttt{MT}_{k,m}(\mathcal{G}_{\texttt{pretrain}}, f) \tag{14}$$

$$\mathcal{L} = \mathop{\mathbb{E}}_{x_i \in \mathcal{Q}_{\text{NM}}} \texttt{CE}(O_{\text{NM},i}, y_{\text{NM},i}) + \mathop{\mathbb{E}}_{x_i \in \mathcal{Q}_{\text{MT}}} \texttt{CE}(O_{\text{MT},i}, y_{\text{MT},i}) \tag{15}$$

where $O_{\text{NM},i}$ is the logits produced by our model over input of $\mathcal{G}_i^{aug}$ and $\mathcal{G}^{\text{T}}$ produced from $\mathcal{Q}_{\text{NM}}$, as described in subsection 3.1; $y_{\text{NM},i}$ is the corresponding label of x_i in $\mathcal{Q}_{\text{NM}}$; Similar for MT terms.

D. Dataset Statistics

Table 2. Dataset statistics

Dataset	# Nodes	# Edges	# Classes
MAG240M	122M	1.3B	153
Wiki	4.8M	5.9M	639
arXiv	169K	1.2M	40
ConceptNet	791K	2.5M	14
FB15K-237	15K	268K	200
NELL	69K	181K	291

E. Data Graph Pooling

For node classification tasks, we take the updated node representation of the single input node that we aim to predict, *i.e.*:

$$G_i = E_{\mathcal{V}_i} \tag{16}$$

For link prediction tasks, we concatenate the pair of nodes, which we want to predict a link between, as well as a max pooling over all node representations following (Huang et al., 2022) with an additional linear projection layer at the end to convert the embedding size back to d.

$$G_i = W^T(E_{v_1 \in \mathcal{V}_i} || E_{v_2 \in \mathcal{V}_i} || \max(E_i)) + b, \tag{17}$$

where $||$ represents concatenation, $W \in \mathcal{R}^{3d \times d}$ is a learnable weight matrix and b is the learnable bias.

F. Task Graph GNN Architecture

For the GNN over task graph M_{T}, we use an attention-based GNN, where each node performs attention to other nodes at each layer:

$$\beta_{ij} = MLP\left(W_q^T H_i^l || W_k^T H_j^l || e_{ij}\right) \tag{18}$$

$$\alpha_{ij} = \frac{\exp(\beta_{ij})}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp(\beta_{ik})} \tag{19}$$

$$H_i^{l+1} = ReLU\left(BN\left(H_i^l + W_o^T \sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{ij} W_v^T H_j^l\right)\right) \tag{20}$$

G. Related Work

G.1. In-context Learning of Large Language Models

Pretrained large language models can make predictions for diverse downstream tasks directly by prompting with a few examples of the task or more generally any textual instructions. This ability is called in-context learning. Comparing to

Table 3. In-context learning accuracy (%) on ConceptNet, FB15K-237 and NELL (from top to bottom) on 500 sampled test tasks with 3-shot prompts. PRODIGY was pretrained on Wiki, which has completely different node and relation types from graphs it is then applied on in-context.

Classes	NoPretrain	Contrastive	PG-NM	PG-MT	PRODIGY	Finetune
4	30.4 ± 0.63	44.01 ± 0.61	46.94 ± 0.61	51.78 ± 0.63	**53.97 ± 0.63**	53.85 ± 9.29
5	33.54 ± 0.61	81.35 ± 0.58	80.35 ± 0.57	89.15 ± 0.46	**88.02 ± 0.48**	82.01 ± 12.83
10	20.0 ± 0.35	70.88 ± 0.48	71.68 ± 0.45	82.26 ± 0.40	**81.1 ± 0.39**	71.97 ± 6.16
20	9.2 ± 0.18	59.8 ± 0.35	59.9 ± 0.35	73.47 ± 0.32	**72.04 ± 0.33**	64.01 ± 4.66
40	2.5 ± 0.08	49.39 ± 0.23	46.82 ± 0.21	58.34 ± 0.22	**59.58 ± 0.22**	57.27 ± 3.33
5	33.44 ± 0.57	84.08 ± 0.54	80.53 ± 0.58	84.79 ± 0.51	87.02 ± 0.44	**87.22 ± 12.75**
10	18.82 ± 0.31	76.54 ± 0.45	72.77 ± 0.48	78.5 ± 0.44	**81.06 ± 0.41**	71.90 ± 5.90
20	7.42 ± 0.16	66.56 ± 0.35	62.82 ± 0.36	69.82 ± 0.34	**72.66 ± 0.32**	66.19 ± 8.46
40	3.04 ± 0.07	57.44 ± 0.24	49.59 ± 0.22	53.55 ± 0.23	**60.02 ± 0.22**	55.06 ± 4.19

previous language encoder models like BERT (Devlin et al., 2019), it drastically reduces the adaptation effort comparing to fine-tuning, and has demonstrated strong performance in a broad range of models and tasks. Our work extends this success similarly to graph data compared to the current pretrained graph encoders, such that a single pretrained model can be adapted to different classification tasks over different graphs without additional fine-tuning but only few-shot prompting.

G.2. Pretraining on Graphs

There are many existing works on pretraining over graphs(Hu et al., 2019; You et al., 2020; Hu et al., 2020b; Qiu et al., 2020). However, they all follow the general paradigm of learning a good graph encoder that can perform certain pretraining tasks, such as masked feature prediction (Hu et al., 2019) and paired graph classification (You et al., 2020). To adapt to any downstream tasks, it then requires finetuning a classification head on top of the encoder with large amount of task specific data for each downstream task. In contrast, we explore pretraining methods for inducing general in-context learning ability, such that the pretrained model can be directly used for various downstream tasks with no gradient updates.

G.3. Meta Learning on Graphs

Another closely related line of works is meta-learning methods over graphs that aim to address standard few shot learning problems over graphs(Wang et al., 2022; Huang & Zitnik, 2020; Chen et al., 2019; Sun et al., 2021; Zhang et al., 2020). However, existing meta-learning methods are only designed and tested for generalizing across different tasks on the same graph: the methods are trained on a set of training tasks on a graph, then tested over a disjoint but similar set of test tasks over the same graph. They are shown to exhibit optimal performance only when trained on similar curated tasks (Huang et al., 2022). Different from this, our work explicitly focuses on the in-context learning performance, i.e. model performance on graphs and tasks completely different from the pretraining without additional fine-tuning.

H. Additional Experimental Results

H.1. Ablations

Aside from PG-NM and PG-MT, we also conduct ablation studies on various configurations of the self-supervised objective PG-NM as described in 3.2. See the full results in Appendix J and Table 4. Overall, the ablation results reveal that using all of the elements together results in the highest performance. Specifically, attribute prediction (see appendix B) has the greatest impact on PG-NM's performance, as its removal results in an average 7% drop across all ways, shown in the 'No-Attr' column.

H.2. Evaluation using different numbers of in-context examples

We investigate our method's ability to learn from the context by analyzing its performance as the number of prompt examples changes. Figure 4 shows the result on `ConceptNet`. See full results on other datasets in Appendix K. As the number of prompt examples increases, the margin of our proposed PG models over the baseline increases. This supports the hypothesis that the PRODIGY models can more effectively learn the unknown task by reasoning about the common characteristics of prompt examples.

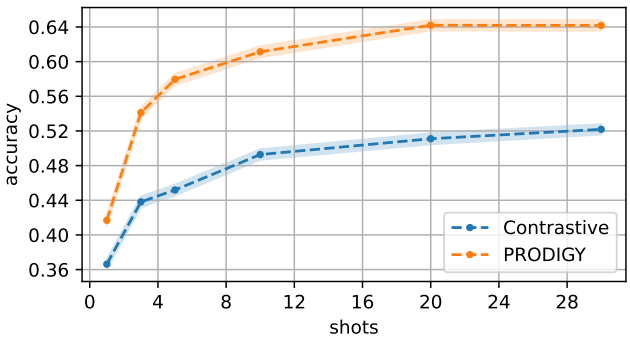

Figure 4. In-context learning accuracy on ConceptNet in a 4-ways setting wrt. the number of prompt examples (shots).

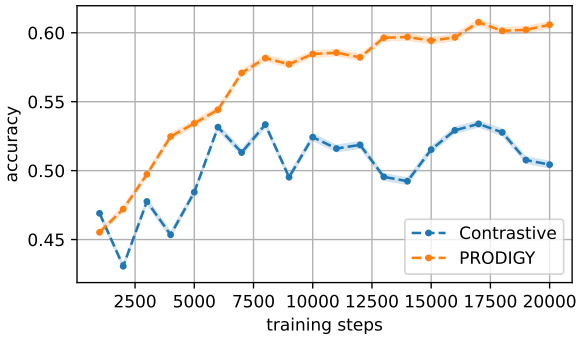

Figure 5. With more training steps and data on arXiv (5-ways), PRODIGY keeps improving while the baseline (Contrastive) saturates.

H.3. Scaling with Data Size

Finally, we explore how the model scales with more pretraining data. The result on arXiv in a 5-ways setting is illustrated in Figure 5. It shows that the Contrastive baseline saturates quickly and its performance fluctuates as trained over more pretraining data. Instead, PRODIGY consistently shows an improvement in performance as more data is pretrained on, since the pretraining tasks are harder and more diverse.

I. Hyperparameters

I.1. Model Architecture, MAG240M and arxiv

We initialize the node features in citation network datasets using a pretrained language model (RoBERTa (Liu et al., 2019) base model trained on NLI and STSB).The architecture of our PromptGraph model in all of our proposed methods for citation network datasets (full PRODIGY, PG-NM, and PG-MT) and the baseline (NoPretrain), consists of two message passing layers, M_{D}, over the DataGraph and one message passing layer, M_{T}, over the TaskGraph. These layers are defined in Section 3.1.

For the Contrastive method, the architecture includes two message passing layers, M_d, over the DataGraph, and a contrastive learning component that is defined in Section 4.1. The mode for Finetune is the same as the Contrastive method, with the addition of a linear layer head over the output of the two M_d layers, also described in Section 4.1.

I.2. Model Architecture, knowledge graph datasets

We initialize node and edge features in knowledge graph datasets using a pretrained language model (MPNet (Song et al., 2020)). The architecture of our PromptGraph model in all of our proposed methods for knowledge graph datasets (full PRODIGY, PG-NM, and PG-MT) and the baseline (NoPretrain), consists of two message passing layers, M_{D}, over the DataGraph, an aggregator as described by Equation 17, and two message passing layers, M_{T}, over the TaskGraph, which only pass messages along the positive and query edges. These layers are defined in Section 3.1.

For the Contrastive method, the architecture includes two message passing layers, M_d, over the DataGraph, an aggregator as described by Equation 17 and a contrastive learning component that is defined in Section 4.1. The mode for Finetune is the same as the Contrastive method, with the addition of a linear layer head over the output of the two M_d layers, also described in Section 4.1.

I.3. Training, MAG240M

The following describes our pretraining setup:

The pretraining task we used consisted of 30 ways, 3 shots, and 4 queries per task. This specific task configuration was carefully selected to strike a balance between complexity and diversity in the training data, without overwhelming the GPU memory.

We checkpoint the model every 500 steps.

Our pretraining setup included a model with an input dimension of 768 and an embedding dimension of 256, batch size

Table 4. Ablation of PG-NM on arXiv.

Ways	PG-NM	3 →1 shot	No Attr	No Aug	No Attr, Aug	No Attr, Aug, M_{T}
3	**72.50** ± **0.35**	69.13 ± 1.09	65.74 ± 1.12	68.98 ± 1.09	66.53 ± 1.12	63.60 ± 1.06
5	**61.21** ± **0.29**	57.49 ± 0.92	52.78 ± 0.90	57.50 ± 0.85	53.89 ± 0.92	51.27 ± 0.69
10	**46.12** ± **0.19**	42.03 ± 0.60	37.99 ± 0.63	42.43 ± 0.64	38.87 ± 0.59	37.62 ± 0.34
20	**33.71** ± **0.11**	30.18 ± 0.38	26.60 ± 0.36	30.89 ± 0.38	27.50 ± 0.36	27.44 ± 0.17
40	**23.69** ± **0.07**	21.44 ± 0.22	18.03 ± 0.21	21.97 ± 0.24	18.52 ± 0.22	19.69 ± 0.08

of 1, and the AdamW optimizer with a learning rate of 1×10^{-3} and weight decay of 1×10^{-3}, a pretraining task with 30 ways, 3 shots, and 4 queries per task, and checkpointing every 500 steps. This consistent configuration was applied across all the methods for fair comparison. Our full PRODIGY setup, on average, involves sampling 1 Neighbor Matching task per 1 multitask pretraining tasks.

For our evaluation process, we computed zero-shot transfer performance of the model on the test set, using the pretraining checkpoint at the 10,000 step of pretraining. The evaluation was conducted on 500 test tasks, with batch size of 5, measured on the downstream task of graph classification accuracy. To maintain consistency, we kept the number of shots and queries constant at 3 for all evaluation tasks.

I.4. Training, Wiki

The following describes our pretraining setup:

Our pretraining setup included a model with an input dimension of 768 and an embedding dimension of 256, the AdamW optimizer with a learning rate of 1×10^{-3} and weight decay of 1×10^{-3}, a pretraining task with 30 ways, 3 shots, and 4 queries per task, using a batch size of 10, and checkpointing every 500 steps. This specific task configuration was carefully selected to strike a balance between complexity and diversity in the training data, without overwhelming the GPU memory. This consistent configuration was applied across all the methods for fair comparison. Our full PRODIGY setup involves sampling one neighbor matching task per 50 multitask pretraining tasks.

For our evaluation process, we computed zero-shot transfer performance of the model on the test set, using the pretraining checkpoint at the 8,000 step of pretraining. The evaluation was conducted on 500 test tasks, with batch size of 1, measured on the downstream task of graph classification accuracy. To maintain consistency, we kept the number of shots and queries constant at 3 for all evaluation tasks. We sample 1-hop neighbourhoods for ConceptNet and FB15K-237 and 2-hop neigbourhoods for NELL and Wiki.

J. Ablation on Table 4 for the PG-NM setting

In Table 4, we ablate on various configurations of the self-supervised objective PG-NM. As described in Section 3.2, PG-NM is composed of an attribute prediction loss, dropnode and zeronode augmentations, with the default setting of sampling 3 shots neighbor matching tasks. Our best setting, referred to as simply "PG-NM", is also shown in Table 1 and comprises of attribute prediction, dropnode and zeronode augmentations, with the default setting of 3 shots.

The ablation results reveal that using all of these elements together results in the highest performance. Specifically, attribute prediction has the greatest impact on PG-NM's performance, as its removal results in an average 7% drop across all ways, as shown in the 'No-Attr' column.

Removing the dropnode and zeronode augmentations results in an average 3% drop across all ways, as shown in the No Aug' column. Removing both attribute prediction and augmentations results in performance that is similar to just removing attribute prediction alone, which is also roughly a 7% drop across all ways, as shown in the 'No Attr, Aug' column. Additionally, we found that decreasing the number of shots to 1 from the default setting of 3 resulted in an average 3.5% drop across all ways, as shown.

K. Evaluation using different numbers of shots

We show evaluation using different numbers of shots, as shown in Figures 4, 6, and 7, 8.

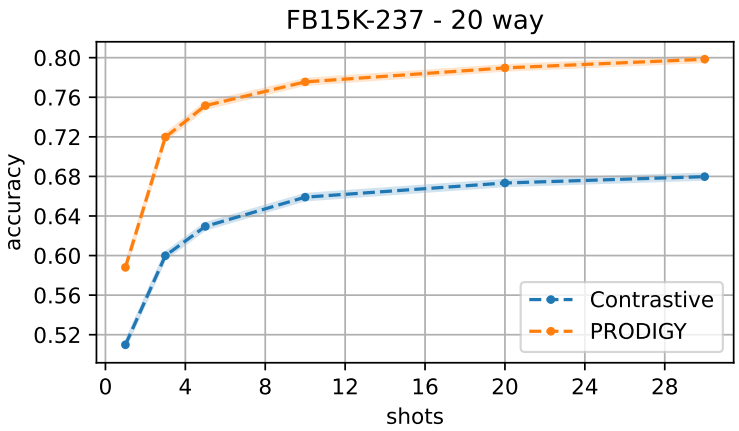

Figure 6. In-context learning accuracy on FB15K-237 in a 20-ways setting wrt. the number of prompt examples (shots).

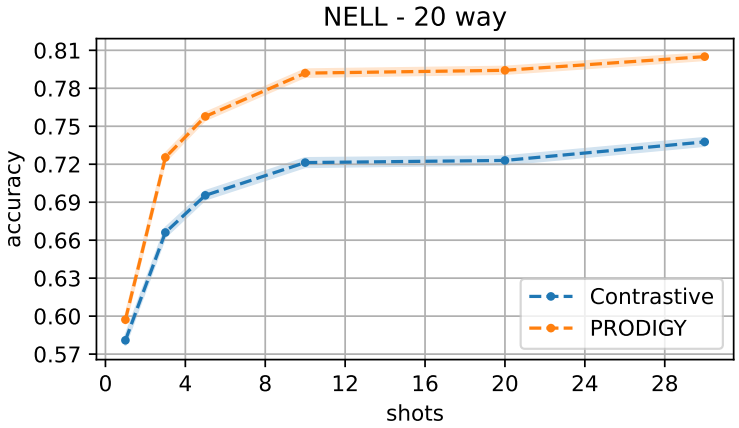

Figure 7. In-context learning accuracy on NELL in a 20-ways setting wrt. the number of prompt examples (shots).

L. Scaling with Data Size

We explore how the model scales with more pretraining tasks. Note that we use the number of train steps as a proxy because the model sees more pretraining tasks as the training proceeds with almost no redundancy (0.20% for 10k tasks). The result on arXiv in a 5-ways setting is illustrated in Figure 5. It shows that the Contrastive baseline saturates quickly and its performance fluctuates given more pretraining tasks. Instead, PRODIGY consistently shows an improvement in performance as more data is pretrained on.

M. Compute

We use one NVIDIA A100-SXM4-80GB GPU for all our experiments. One pretrain run of 10k steps takes 3 to 4 hours.

N. Broader Impacts

Our work aims to extend the success of in-context learning to graphs and start building toward graph foundation models. This would allow cost-effective and accurate predictions, especially in domains where labeled data is scarce and long tail such as network anomaly detection, rare disease diagnosis/treatment, supply chain disruption, and recommendations for new users. However, overreliance on prior knowledge from pretraining could also lead to increased social bias and unfair

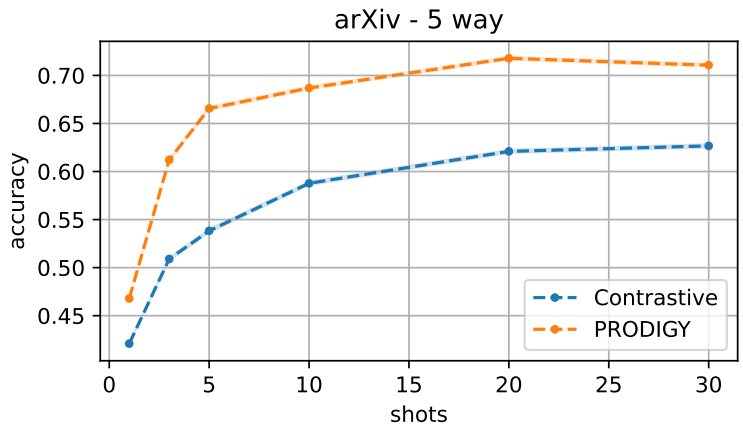

Figure 8. In-context learning accuracy on arXiv in a 5-ways setting wrt. the number of prompt examples (shots).

benefits to the dominate groups. To mitigate this, pretraining data should be diverse and well-balanced, and the pretrained models should be tested on downstream tasks over different groups and subdistributions.