# Zero-Shot Generalization of GNNs over Distinct Attribute Domains

Yangyi Shen[1] Beatrice Bevilacqua[2] Joshua Robinson[1] Charilaos Kanatsoulis[1] Jure Leskovec[1]
Bruno Ribeiro[2]

## Abstract

There are no known graph machine learning methods that can zero-shot generalize across attributed graphs with very different node attribute domains and consistently outperform methods that ignore node attributes. For instance, no method can significantly outperform structure-only predictions in zero-shot link prediction by pretraining on online appliance store datasets (with node attributes such as brand, model, capacity, dimension, has ice maker, energy rating for refrigerators) and zero-shot at test on an electronics store dataset for smartphones (with attributes such as processor type, display type, storage, and battery capacity). In this work, we leverage concepts in statistical theory to design STAGE, a universally applicable approach for encoding node attributes in *any GNN* that facilitates such generalization. Empirically, we show that STAGE outperforms its natural baselines and can accurately make predictions when presented with completely new feature domains.

## 1. Introduction

Zero-shot generalization refers to a model's ability to handle new, unseen data without additional training (Larochelle et al., 2008; Xian et al., 2017; Wang et al., 2022). Achieving this requires the model to learn prediction rules that can be used across entirely different sets of features at test time.

In the context of attributed graph data, zero-shot generalization presents unique challenges. First, node features can vary widely between graphs in different domains. Unlike text data, where tokenization (Samuel & Øvrelid, 2023) standardizes text into a fixed format enabling zero-shot generalization, processing graph data requires methods that can handle heterogeneous and high-dimensional node features. Second, the relational dependencies between nodes, encoded in the edges, can be rather complex and context-dependent, requiring graph learning methods that can capture both the topology and node attributes. These challenges make it harder to define a unified input space that can be leveraged for zero-shot generalization in attributed graphs.

For these reasons, it has proven very challenging to pretrain general purpose graph models. This is reflected in the fact that, in graph ML, generalizing to new data formats is rarely tackled by directly applying popular pretrained models without altering model weights. Initial steps in this direction include ISDEA+ and ULTRA (Gao et al., 2023; Galkin et al., 2024), models that can generalize across different relation types, without however including node features, or PRODIGY (Huang et al., 2023), which can perform different tasks on text-attributed graphs, but does not consider generalization to new data with different features.

**Contributions.** (a) Our first contribution is the development of STAGE (**S**tatistical **T**ransfer for **A**ttributed **G**raph **E**mbeddings), a method for feature encoding on attributed graphs that can be trained on a set of attributed graphs and applied to a completely different graph with a completely different feature domain. STAGE embeds input node features belonging to arbitrary continuous or discrete spaces into a single embedding space common to all feature domains. The main insight of STAGE is to capture distributional features within the node endpoints of graph edges. Since this encoding works in the space of probability functions, it is a common space across any feature domains. Intuitively, for each edge, we construct a fully connected weighted graph that encodes the probability distributions of the node features of its endpoints. This graph is then processed to obtain an edge embedding, which can be used by subsequent layers in place of node features.

(b) The second contribution of our work is a set of graph datasets with rich and distinct node features in different domains, which are used to evaluate our approach. This is essential since existing datasets are often homogenized to have the same feature spaces or are stripped down to binary features, capturing mostly the topological information. To bridge this gap, we leverage LLMs to reverse-engineer detailed features across various product categories in a set

[1]Department of Computer Science, Stanford University, Stanford, USA [2]Department of Computer Science, Purdue University, West Lafayette, USA. Correspondence to: Yangyi Shen <pyyshen@stanford.edu>.

Published at ICML 2024 Workshop on Foundation Models in the Wild. Copyright 2024 by the author(s).

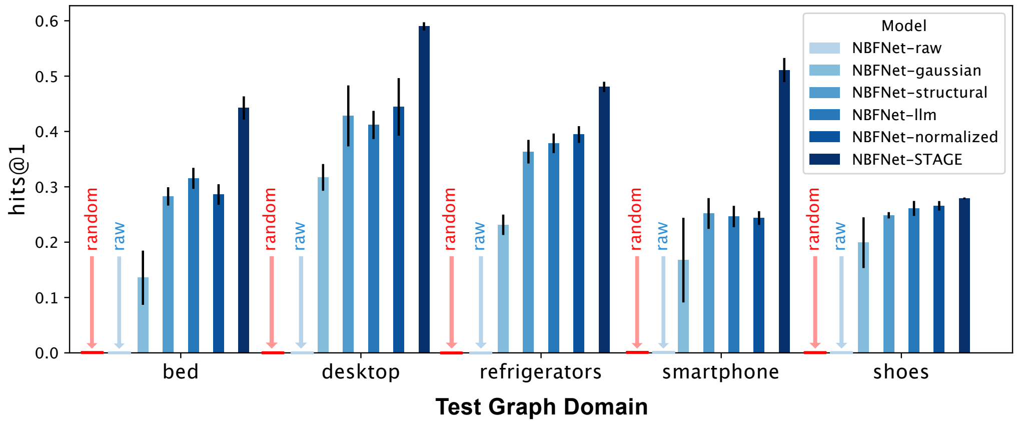

*Figure 1.* Zero-shot Hits@1 (higher is better) of STAGE and its baselines. Given different product categories, each with its own set of features, we choose a test graph domain (x-axis) and pre-train the models on all the remaining domains. The error bars corresponds to training with different seeds. On average, the zero-shot performance of our model is better than any other baselines.

of e-commerce datasets (Kechinov, 2020). This enables us to construct more realistic and diverse datasets, where each dataset corresponds to a different set of product categories in distinct e-commerce stores, each with its own set of node features. As shown in Figure 1, STAGE consistently outperforms all baseline models in zero-shot testing scenarios, where the test graph domain is entirely unseen during training.

## 2. Our approach: STAGE

GNNs typically assume features have the same semantics across different nodes. This limits the applicability of GNNs to new graphs at test time, which do not necessarily conform to the same feature domains seen in the training data. To address this, our goal is to design an architecture capable of processing graphs with distinct node feature spaces.

### 2.1. Method

We are given an attributed graph $G = (V, E, X)$ where $X = (X^v)_{v\in V}$ and describes a set of features for each node in the graph with $X_v$ belonging to any measurable space of dimension $d \geq 1$.

To design a model capable of generalizing to test graphs that may have node features living in a different space than $X$, we design a projection map that transforms the node features $(X^u, X^v)$ of an edge $(u, v) \in E$ into a fixed dimensional pairwise embedding

$$\mathcal{P} : (X^u, X^v) \mapsto \boldsymbol{r}^{uv} \in \mathbb{R}^k, \quad k \geq 1. \tag{1}$$

In order to obtain the mapping $\mathcal{P}$, we start with building a graph based on the following pairwise *pdf* feature descriptors. First, we define the random variable $\tilde{X}_i$ as the $i$-th feature of a randomly chosen node in $G$, $i \in \{1, \ldots, d\}$. We will use $\tilde{X}_i$ to define a conditional probability $\mathbb{P}(X^u_i | X^v_j)$, $i, j \in \{1, \ldots, d\}$, that accounts for mix of discrete and continuous features:

- $\mathbb{P}(X^u_i | X^v_j) := P(\tilde{X}_i \leq X^u_i | \tilde{X}_j \leq X^v_j, (u, v) \in E)$, if $\tilde{X}_i$ and $\tilde{X}_j$ are continuous.
- $\mathbb{P}(X^u_i | X^v_j) := P(\tilde{X}_i = X^u_i | \tilde{X}_j \leq X^v_j, (u, v) \in E)$, if $\tilde{X}_i$ is discrete and $\tilde{X}_j$ is continuous.
- $\mathbb{P}(X^u_i | X^v_j) := P(\tilde{X}_i \leq X^u_i | \tilde{X}_j = X^v_j, (u, v) \in E)$, if $\tilde{X}_i$ is continuous and $\tilde{X}_j$ is discrete.
- $\mathbb{P}(X^u_i | X^v_j) := P(\tilde{X}_i = X^u_i | \tilde{X}_j = X^v_j, (u, v) \in E)$, if $\tilde{X}_i$ and $\tilde{X}_j$ are discrete.

For $i, j \in \{1, \ldots, 2d\}$, $i \neq j$, we define the following matrix

$$\mathbf{X}^{uv}_{ij} = \begin{cases} \mathbb{P}(X^u_i \mid X^u_j) & \text{if } i \leq d \text{ and } j \leq d, \\ \mathbb{P}(X^v_{i-d} \mid X^v_{j-d}) & \text{if } d \leq i \leq 2d \text{ and } d \leq j \leq 2d, \\ \mathbb{P}(X^u_i \mid X^v_{j-d}) & \text{if } i \leq d \text{ and } d < j \leq 2d, \\ \mathbb{P}(X^v_{i-d} \mid X^u_j) & \text{if } d < i \leq 2d \text{ and } j \leq d. \end{cases}$$

and for the diagonal $i = j$ we define,

$$\mathbf{X}^{uv}_{ij} = \begin{cases} \mathbb{P}(X^u_i) & \text{if } i \leq d, \\ \mathbb{P}(X^v_i) & \text{if } i > d, \end{cases}$$

where $\mathbb{P}(X^u_i) := P(\tilde{X}_i = X^u_i)$ if $X^u_i$ is discrete and $\mathbb{P}(X^u_i) := P(\tilde{X}_i \leq X^u_i)$ if $X^u_i$ is continuous.

We now build a fully connected weighted directed graph denoted $\mathbb{G}(\mathbf{X}^{uv})$ with $2d$ nodes, where node $i$ has scalar attribute $\mathbf{X}^{uv}_{ii}$ and edge $(i, j)$ has scalar attribute $\mathbf{X}^{uv}_{ij}$. We choose to focus on edge-wise relations instead of node-wise features in order to maximize the expressivity for entity and link prediction tasks.

For each edge $(u, v) \in E$ in the original graph $G$, we process $\mathbb{G}(\mathbf{X}^{uv})$ with a message-passing neural network to produce its graph embedding, denoted $\boldsymbol{r}^{uv}$ in Equation (1). Finally, a GNN (the base model) is trained end-to-end to solve the task at hand using $(V, E, \{\boldsymbol{r}^{uv}\}_{(u,v)\in E})$, i.e., the original $V$, $E$, and these new edge features (which replace the original node features). For entity prediction tasks our experiments use NBFNet as our base model (Zhu et al., 2021), but any GNN that can accommodate edge embeddings can be used as the base model.

The key probabilistic insight behind our design is that random variables (node features) are completely characterized by their Radon–Nikodym derivatives (Kallenberg, 1997) (i.e., their *probability density functions* (*pdf* s)). This allows variables belonging to different spaces to be reasoned about in the same space (the space of measurable densities). Using this insight, STAGE learns a projection map from a *pdf* description of the features into a constant-size embedding.

Specifically, consider that the feature vector of a node is a realization (sample) of a random vector $\tilde{\mathbf{X}} = \left[\tilde{X}_1, \ldots, \tilde{X}_d\right]$ that is drawn from a joint distribution $f_{\tilde{\mathbf{X}}}$. Estimating $f_{\tilde{\mathbf{X}}}$ would provide complete knowledge of the different features, however it is usually infeasible as it requires a large number of samples. Instead we estimate the marginal probabilities of each feature, i.e., $P(\tilde{X}_i)$, and the conditional probabilities for feature pairs, i.e., $P(\tilde{X}_i \mid \tilde{X}_j)$. As we discuss later in the paper, pairwise probabilities are sufficient to estimate the joint distribution of the features under certain conditions.

Using the estimated probabilities and the observed feature values for each node we generate edge features as follows: In the *pdf* feature descriptors $\mathbf{X}^{uv}$, the diagonal encodes the marginal likelihood of $X_u$ and $X_v$, and the off-diagonals encodes conditional likelihood between features of $X_u$ and itself, and between features of $X_u$ and $X_v$.

By focusing on the likelihood of observing a feature value given other features, we remove the need to process the feature itself by focusing only on pair-wise relations between features. This allows us to apply a fixed edge-level model to the weighted graph $G(\mathbf{X}^{uv})$, dubbed *edge-feature graph*, *independently* of the original feature $X^u$ and $X^v$. Our model can generalize to completely unseen feature types by reasoning by analogy from its relations to other features.

**Higher-order extension.** In general, random variables are fully characterized by higher-order interactions between variables. Our pairwise formulation can be naturally extended to triple- and higher-order conditional probabilities by defining a *hypergraph*, which has features $X_S$ for subsets of nodes $S \subseteq V$ with $|S| \geq 2$, and processed with hypergraph neural networks (Feng et al., 2019). In practice, accurately estimating higher-order conditional distributions requires significant amounts of data and we leave further exploration of this to future work.

**Expressivity.** A critical decision for the generalization of our approach is to transform the node-level feature representations into feature representations between pairs of nodes (edge-level feature representations). This enables our model to be trained and executed on graphs with varying feature numbers and/or types. To that end, we model the node features as random variables, characterized entirely by their probability density or mass functions, and empirically estimate the true marginal and conditional distributions based on observed data. Note that this is doable, since the sample complexity for reliable estimation of the joint probabilities, and thus conditionals and marginals, of discrete random variables is relatively low. Specifically, to achieve $\left|P\left(\tilde{X}_i, \tilde{X}_j\right) - \hat{P}\left(\tilde{X}_i, \tilde{X}_j\right)\right| \leq \epsilon$ with a probability greater than $1-\delta$, $\mathcal{O}\left(\epsilon^{-2}\log\left(\frac{1}{\delta}\right)\right)$ samples are needed. Using pairwise probabilities allows us to generate powerful representations for the edges of the graph. Their expressivity is theoretically grounded since the identification of the joint probability mass function from pairwise joint probabilities can be guaranteed under certain conditions (Ibrahim et al., 2019). It is also notable that GNN models can effectively generalize when the features are derived from probability functions. In fact, it can be shown that such features are invariant across different graphs under certain conditions on their distributions, but this formalization is left to a full version of this work.

**Choosing pairwise relations.** $\mathbf{X}^{uv}$ is only computed for *edges* $(u, v)$, and so can only model pairwise relations between nodes connected by an edge. In some cases, such as bipartite graphs, we find it beneficial to add extra edges between nodes of the same type (see Section 3 for details).

## 3. Experiments

We perform a preliminary set of experiments to answer two main questions: (Q1) How does STAGE compare to its natural baselines, which can be obtained, for instance, by disregarding node features or by employing an LLM to obtain initial node embeddings? (Q2) How does the number of training graphs impact the performance? In the following we present our results and refer to Appendix B for additional experimental results for different setup variations.

**Dataset creation.** We consider a dataset of e-commerce users and products (Kechinov, 2020), from which we create graphs, where an edge indicates that a user has bought, carted, uncarted, or seen a certain product. We split the data into different product categories: shoes, refrigerators, desktop, smartphone, and bed. These will be our graph domains. We select a subset of product categories to form our training graph and use unseen product categories to form our test graph. To ensure that each product category has its own set of features describing a product in that category, we use GPT-4 (OpenAI, 2023) to retrieve information about the product as it would have been in 2019 (the year

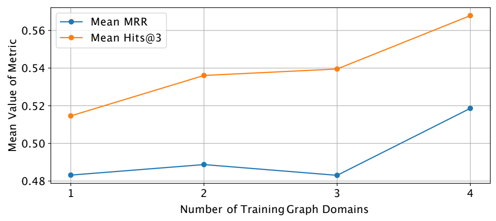

*Figure 2.* Zero-shot performance of STAGE on a fixed test graph (bed e-commerce domain) with increasing number of graphs in pre-training. We report results averaged across seeds. Increasing the number of training graph domains leads to better performance.

of the e-commerce dataset we built upon). This allows us to retrieve different features for different categories such as *display type* for smartphone, and *ankle height* for shoes. We refer the reader to Appendix A for additional details. Once the training and test graphs are constructed, we include additional product-product edges, representing how many users have bought both products. Since users do not have any node feature, we create our edge graphs $\mathbb{G}(\mathbf{X}^{uv})$ (Section 2.1) only for the product-product edges. Finally, we consider the entity prediction task, where the goal is to predict the tail entity given a head entity and a relation type.

**Baselines.** We compare STAGE to different feature encoding methods, which serve as our baselines: (1) RAW, which projects each node feature into a fixed dimensional space; (2) GAUSSIAN, which uses a Gaussian random noise vector instead of the node features (Sato et al., 2021; Abboud et al., 2021; Murphy et al., 2019); (3) STRUCTURAL, which completely disregards node features; (4) LLM, which textualizes the node-feature semantics and their values and pass them through an text-encoding language model, similar to the approach taken in PRODIGY (Huang et al., 2023); (5) NORMALIZED, which only keeps the continuous features and normalizes them into the same feature space. For a fair comparison, in all the baselines, as well as in our method, we employ the same NBFNet architecture (Zhu et al., 2021), changing only the input feature encoding method.

**Results.** To test the performance of STAGE and compare it with its baselines, we consider each graph domain (product category) as the test domain, while using all the remaining four graph domains (product categories) for training. This approach allows us to evaluate how well STAGE and its baselines generalize to unseen graph domains in our zero-shot scenario. As shown in Figure 1, STAGE significantly outperforms all its baselines, with the largest margin obtained when testing on the smartphone domain. In the shoes domain, however, STAGE performs similarly (slightly better) to some of its baselines. We conjecture that this is due to the rich features and complex structure of the shoes graph domain, which are difficult to capture when training on the remaining graph domains.

We further test the meta-learning capabilities of STAGE by assessing whether increasing the number of training domains improves inference performance on a fixed new test graph domain. We set the bed graph domain as the test graph domain and train the model sequentially with one, two, three, and four graph domains. This evaluation helps us understand how well STAGE can leverage additional training data from diverse graph domains to enhance its zero-shot generalization performance on a new domain. The results are presented in Figure 2, showing that the performance increases with the number of domains, demonstrating the effectiveness of STAGE in utilizing diverse training data.

## 4. Related Work

Foundation models for graph data aim to create versatile graph models capable of generalizing across different graphs and tasks. Despite significant interest, achieving a truly universal graph foundation model remains challenging due to the complexities in designing a suitable graph vocabulary that ensures transferability across datasets and tasks. Current research emphasizes the importance of building models that can adapt to the diverse structural patterns present in different graph data, often overlooking the handling of different features across graph domains (Mao et al., 2024).

Most current methods convert attributed graphs into texts. GRAPHTEXT (Zhao et al., 2023) represents graph structures within a textual space, using advanced language models to treat graph reasoning as text generation. Unigraph (He & Hooi, 2024) also merges NLP with graph learning, utilizing a unified graph tokenizer to generalize across various domains. PRODIGY (Huang et al., 2023) encodes the textual features with an LLM and focuses on employing a prompt graph representation to connect examples and queries for generalization to new tasks.

## 5. Conclusions

In this work we introduced STAGE, the first method specially designed to allow GNNs to generalize to graphs with input features in different feature spaces, including different dimensional node features and features with different semantics. STAGE works by modelling the pairwise probabilities between individual node features of edge endpoints, and encoding these probabilities with a neural network that is invariant to the order of the features (a graph neural network of the *edge-feature graph*). We find that this approach significantly outperforms other natural approaches to graph feature encoding such as viewing the raw features as text and using an LLM to produce node embeddings from node features. Our future work includes expanding our theoretical analysis to a theory on the transferability of universal feature embeddings for graphs in distinct domains. We will also train and test on more diverse graph domains, and include other GNN models besides NBFNet along with additional tasks (e.g. node and graph-level) besides link prediction.

## References

Abboud, R., Ceylan, İ. İ., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artifical Intelligence (IJCAI)*, 2021.

Feng, Y., You, H., Zhang, Z., Ji, R., and Gao, Y. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 3558–3565, 2019.

Galkin, M., Yuan, X., Mostafa, H., Tang, J., and Zhu, Z. Towards foundation models for knowledge graph reasoning. In *The Twelfth International Conference on Learning Representations*, 2024.

Gao, J., Zhou, Y., Zhou, J., and Ribeiro, B. Double equivariance for inductive link prediction for both new nodes and new relation types. *arXiv preprint arXiv:2302.01313*, 2023.

He, Y. and Hooi, B. Unigraph: Learning a cross-domain graph foundation model from natural language. *ArXiv*, abs/2402.13630, 2024.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

Huang, Q., Ren, H., Chen, P., Kržmanc, G., Zeng, D., Liang, P. S., and Leskovec, J. Prodigy: Enabling in-context learning over graphs. *Advances in Neural Information Processing Systems*, 36, 2023.

Ibrahim, S., Fu, X., Kargas, N., and Huang, K. Crowdsourcing via pairwise co-occurrences: Identifiability and algorithms. *Advances in neural information processing systems*, 32, 2019.

Kallenberg, O. *Foundations of modern probability*, volume 2. Springer, 1997.

Kechinov, M. ecommerce behavior data from multi category store, 2020. URL www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store.

Larochelle, H., Erhan, D., and Bengio, Y. Zero-data learning of new tasks. In *AAAI*, volume 1, pp. 3, 2008.

Mao, H., Chen, Z., Tang, W., Zhao, J., Ma, Y., Zhao, T., Shah, N., Galkin, M., and Tang, J. Graph foundation models. In *arXiv preprint arXiv:2402.02216*, 2024.

Murphy, R., Srinivasan, B., Rao, V., and Ribeiro, B. Relational pooling for graph representations. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 4663–4673, 2019.

OpenAI. Gpt-4 technical report. https://openai.com/research/gpt-4, 2023. Accessed: 2023-05-28.

Samuel, D. and Øvrelid, L. Tokenization with factorized subword encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, Toronto, Canada, 2023. Association for Computational Linguistics.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM*, 2021.

Wang, T., Roberts, A., Hesslow, D., Scao, T. L., Chung, H. W., Beltagy, I., Launay, J., and Raffel, C. What language model architecture and pretraining objective works best for zero-shot generalization? In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 22964–22984. PMLR, 17–23 Jul 2022.

Xian, Y., Schiele, B., and Akata, Z. Zero-shot learning-the good, the bad and the ugly. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4582–4591, 2017.

Zhao, J., Zhuo, L., Shen, Y., Qu, M., Liu, K., Bronstein, M., Zhu, Z., and Tang, J. Graphtext: Graph reasoning in text space, 2023.

Zhu, Z., Zhang, Z., Xhonneux, L.-P., and Tang, J. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34:29476–29490, 2021.

## A. Dataset Construction

To test the model's generalization to new input feature spaces, we consider a dataset of e-commerce users and products (Kechinov, 2020). There are 29,228,809 different product categories, such as smartphones, shoes, and computers. During training, we select a subset of product categories and form an input graph from those products and connected users. At test time, we create an entirely different graph containing unseen products, *from new unseen categories* and associated users and test the zero-shot (i.e., frozen model) performance on the test data. We focus on the single task of predicting links between users and products, with links indicating a user purchasing a product.

To ensure that the test graph has different feature types than those of the training graph, we use GPT-4 to retrieve information specific to each category. Specifically, the information retrieval process involves prompting GPT-4 with the following content:

```
"According to the following information regarding an e-commerce purchase, give
    information about the product in the following asked format."
"First, the product is purchased at time: " + row["event_time"] + "."
"Second, the category of the product is " + row["category_code"] + "."
"Third, the brand of the product is " + row["brand"] + "."
"Last, the price of the product is " + str(row["price"]) + "."
"Please provide information about the product in the following json format."
"{json_prototype}"
```

The JSON prototype is different for different categories, and contains features that are specific for the category being prompted. That is, the JSON prototype for smartphones contains, for instance, features like *display type*, which is not a feature for shoes, containing instead features such as *ankle height*. In the following, we report the JSON prototype for all categories.

smartphone

```
{
    "display_type": <select from ['OLED', 'LCD']>,
    "display_size": <give float in inches>,
    "display_resolution": <give int in pixels>,
    "processor_type": <give string>,
    "ram": <give int in GB>,
    "storage_options": <give int in GB>,
    "rear_camera_primary_resolution": <give int in MP>,
    "front_camera_resolution": <give int in MP>,
    "operating_system": <select from ['Android', 'iOS', 'HarmonyOS', 'KaiOS', 'Tizen
        ', 'Ubuntu Touch', 'PureOS', 'Sailfish OS', 'Plasma Mobile']>,
    "Battery_capacity": <give int in mAh>,
    "Has_gps":  <select from ['True', 'False']>,
    "has_nfc":  <select from ['True', 'False']>
    }
```

shoes

```
{
    "type": <select from ['Running', 'Casual', 'Formal', 'Sports', 'Boots', 'Sandals
        ', 'Slippers', 'Hiking', 'Dress', 'Work', 'Safety']>,
    "material": <select from ['Leather', 'Synthetic', 'Textile', 'Rubber', 'Canvas',
         'Mesh', 'Suede', 'Patent Leather', 'Nubuck', 'Faux Leather']>,
    "color": <give string>,
    "size": <give float in UK sizes>,
    "gender": <select from ['Men', 'Women', 'Unisex', 'Children', 'Infants']>,
    "closure_type": <select from ['Laces', 'Velcro', 'Slip-on', 'Buckle', 'Zip', '
```

```
        Hook and Loop', 'None']>,
    "sole_material": <select from ['Rubber', 'Synthetic', 'PVC', 'EVA', 'Leather', '
        TPU (Thermoplastic Polyurethane)', 'TPR (Thermoplastic Rubber)']>,
    "water_resistant": <select from ['True', 'False']>,
    "ankle_height": <select from ['Low-top', 'Mid-top', 'High-top', 'Over the ankle
        ']>,
    "breathability": <select from ['High', 'Medium', 'Low']>,
    "weight": <give float in grams>,
    "origin_country": <give string>,
    "seasonality": <select from ['All-season', 'Summer', 'Winter', 'Rainy', 'Spring
        ', 'Autumn']>,
    "eco_friendly": <select from ['True', 'False']>
}
```

desktop

```
{
    "processor_type": <select from ['Intel Core i3', 'Intel Core i5', 'Intel Core i7
        ', 'Intel Core i9', 'AMD Ryzen 3', 'AMD Ryzen 5', 'AMD Ryzen 7', 'AMD Ryzen
        9', 'Apple M1', 'ARM other']>,
    "ram_gb": <give int>,
    "storage_type_hdd_size_gb": <give int>,
    "storage_type_ssd_size_gb": <give int>,
    "storage_type_hybrid_size_gb": <give int>,
    "graphics_card": <select from ['NVIDIA GeForce GTX 1660', 'NVIDIA GeForce RTX
        2060', 'NVIDIA GeForce RTX 2070', 'NVIDIA GeForce RTX 2080', 'AMD Radeon RX
        570', 'AMD Radeon RX 580', 'AMD Radeon RX 590', 'AMD Radeon RX 5700', 'AMD
        Radeon RX 5700 XT']>,
    "operating_system": <select from ['Windows 10', 'macOS', 'Linux Ubuntu', 'Linux
        Fedora', 'Linux Mint', 'Debian', 'FreeBSD']>,
    "power_supply_watts": <give int>,
    "cooling_system": <select from ['Air cooling', 'Liquid cooling', 'Passive
        cooling']>,
    "has_bluetooth": <select from ['True', 'False']>
}
```

refrigerators

```
{
    "energy_rating": <select from ['A+++', 'A++', 'A+', 'A', 'B', 'C']>,
    "capacity_liters": <give int>,
    "refrigerator_type": <select from ['Top Freezer', 'Bottom Freezer', 'Side-by-
        Side', 'French Door', 'Mini Fridge', 'Commercial']>,
    "defrost_type": <select from ['Manual', 'Frost Free', 'Automatic Defrost']>,
    "has_ice_maker": <select from ['True', 'False']>,
    "has_water_dispenser": <select from ['True', 'False']>,
    "has_smart_technology": <select from ['True', 'False']>,
    "is_energy_efficient": <select from ['True', 'False']>,
    "height_cm": <give float>,
    "width_cm": <give float>,
    "depth_cm": <give float>
}
```

bed

```
{
    "type": <select from ['Twin', 'Twin XL', 'Full', 'Queen', 'King', 'California
       King']>,
    "material": <select from ['Wood', 'Metal', 'Upholstered', 'Bamboo', 'Particle
       Board', 'Composite']>,
    "bed_frame_included": <select from ['True', 'False']>,
    "headboard_included": <select from ['True', 'False']>,
    "footboard_included": <select from ['True', 'False']>,
    "mattress_included": <select from ['True', 'False']>,
    "box_spring_required": <select from ['True', 'False']>,
    "weight_capacity_lbs": <give int in lbs>,
    "bed_size_length_inches": <give float in inches>,
    "bed_size_width_inches": <give float in inches>,
    "bed_size_height_inches": <give float in inches>
}
```

## B. Additional Experiments and Details

In this section, we explore variations of the experiments reported in the main text. Specifically, we measure the performance of STAGE and its baselines using additional metrics within the same experimental setup presented in Section 3, where, for each product category selected as the test category, we train on all remaining categories. Results for Hits@1, Hits@3, and Hits@10 are presented in Figures 3 to 5, respectively.

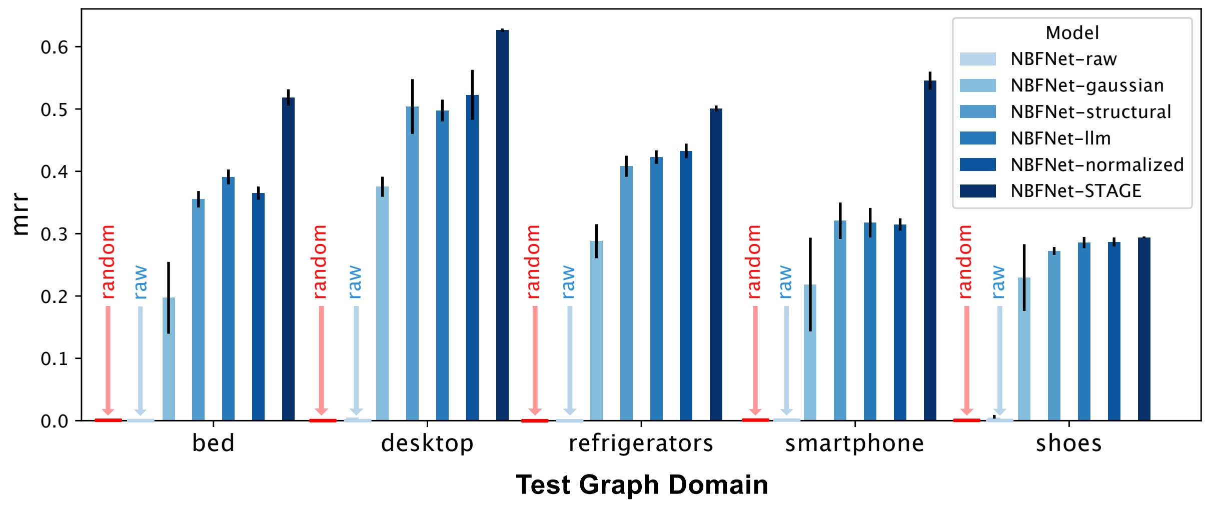

*Figure 3.* MRR (higher is better) of STAGE and baseline models.

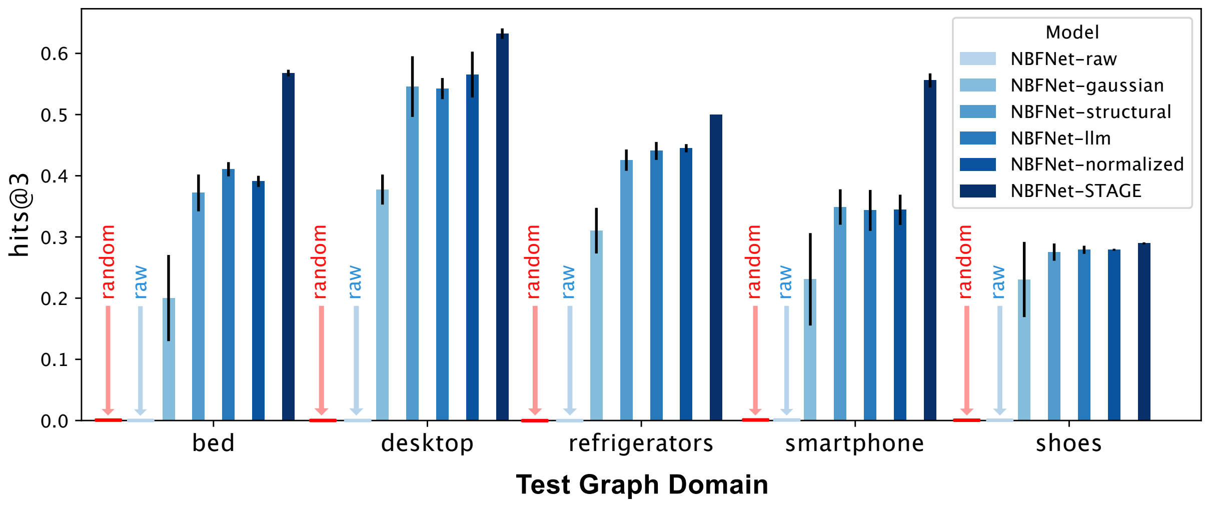

*Figure 4.* Hits@3 (higher is better) of STAGE and baseline models.

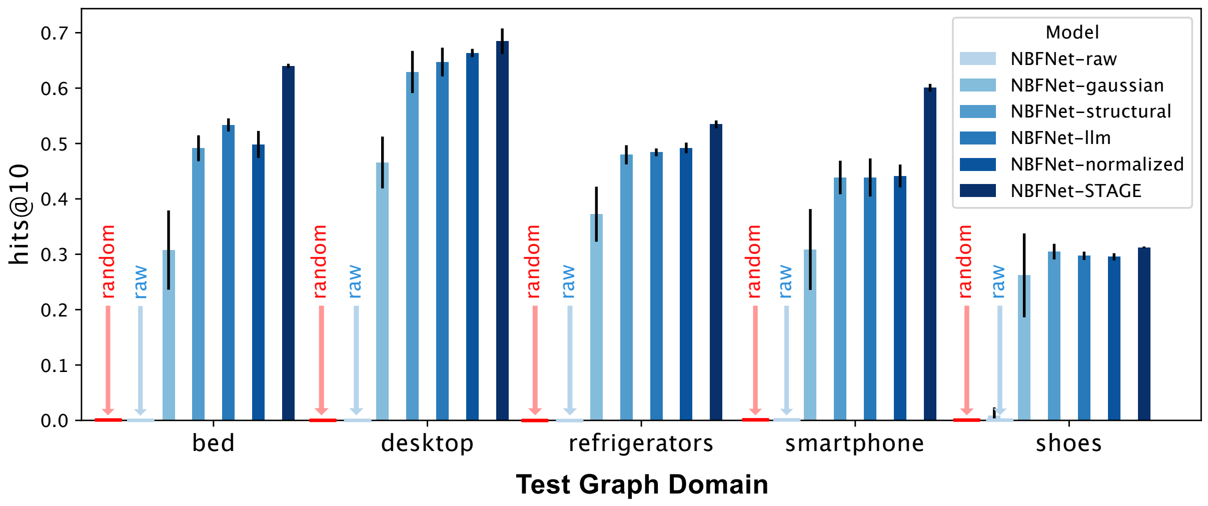

*Figure 5.* Hits@10 (higher is better) of STAGE and baseline models.

We also conduct an additional experiment to test the meta learning capabilities of STAGE, complementing the one presented in Figure 2 in Section 3. In particular, given the total number of categories $C$, we fix the number of training categories $k$ (x-axis), and average the performance of STAGE on all test categories $(C - k)$, when training on all combinations of categories $(\binom{C}{k})$. Results for different metrics are reported in Figures 6 to 9, showing that increasing the number of categories generally leads to better performance for STAGE.

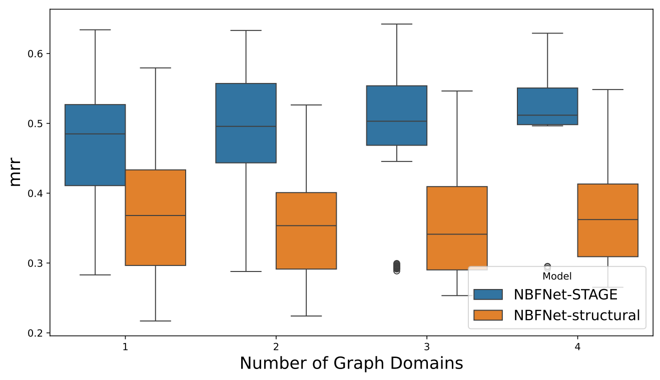

*Figure 6.* STAGE shows meta-learning: # graphs in pre-training (# Domains) vs. MRR values of zero-shot performance on the remaining graph domains. We see a clearer trend of increase in the overall MRR span in our method, relative to other baseline models (e.g. structural).

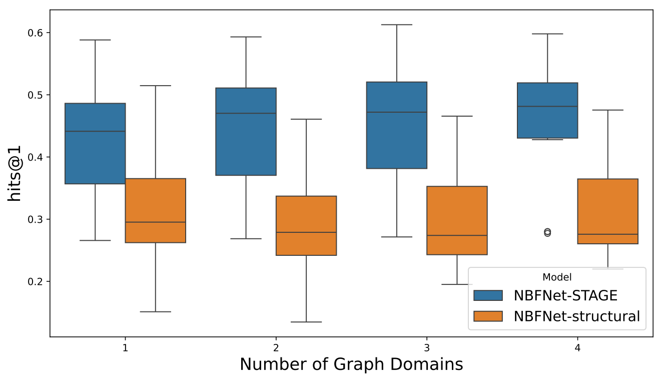

*Figure 7.* STAGE shows meta-learning: # graphs in pre-training (# Domains) vs. Hits@1 values of zero-shot performance on the remaining graph domains. We see a clearer trend of increase in the overall Hits@1 span in our method, relative to other baseline models (e.g. structural).

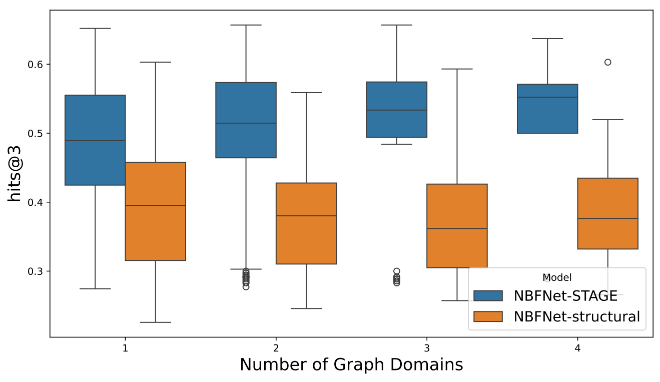

*Figure 8.* STAGE shows meta-learning: # graphs in pre-training (# Domains) vs. Hits@3 values of zero-shot performance on the remaining graph domains. We see a clearer trend of increase in the overall Hits@3 span in our method, relative to other baseline models (e.g. structural).

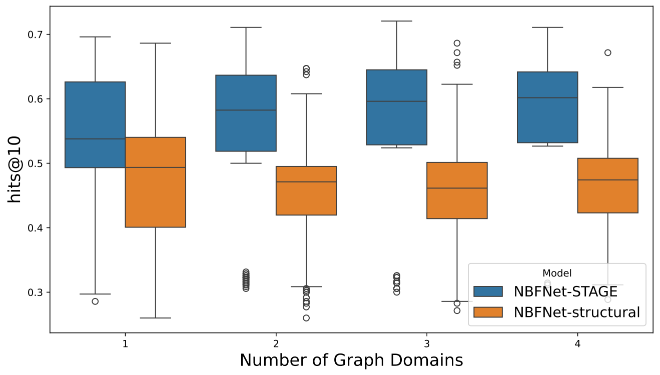

*Figure 9.* STAGE shows meta-learning: # graphs in pre-training (# Domains) vs. Hits@10 values of zero-shot performance on the remaining graph domains. We see a clearer trend of increase in the overall Hits@10 span in our method, relative to other baseline models (e.g. structural).

Finally, we report in Table 1 the categories used in training for Figure 2, where we fix the test category (bed) and train with increasing number of categories (from 1 to 4), by including a new category to the mixture.

*Table 1.* Graphs in different pre-training mixtures in Figure 2. We fix the test category as Bed.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| desktop | ✓ | ✓ | ✓ | ✓ |
| shoes | | ✓ | ✓ | ✓ |
| smartphone | | | ✓ | ✓ |
| refrigerators | | | | ✓ |
| # epochs | 30 | 30 | 30 | 30 |

**Experimental Details.** We trained all our models on NVIDIA A100 GPUs. For all experiments, we utilized the following configuration to evaluate the performance of all models, namely our STAGE, llm, structural, normalized, raw, and gaussian. The input dimension is set to 256, corresponding to the feature dimension (k) as described in Equation (1). We employ NBFNet (Zhu et al., 2021) as our GNN model, with six layers of GeneralizedRelationalConv (Zhu et al., 2021), each with hidden dimension equal to 256. For the remaining hyperparameters, we keep the default choices in NBFNet, namely the message passing function being DistMult, the aggregation function being PNA, with both shortcut connections and layer normalization.

For the baseline methods, no additional parameters are needed. However, for our STAGE, we specify an edge embedding dimension of 256 and employ a single layer of GINEConv (Hu et al., 2020), followed by sum pooling to obtain the edge graph embedding.

The task configuration includes generating 64 negative samples with strict negative sampling and an adversarial temperature of 1. The evaluation metrics used are mean reciprocal rank (MRR), hits@1, hits@3, and hits@10. The optimizer chosen for training is Adam with a learning rate of 5.0e-3. During training, the batch size is set to 32, and the model is trained for 30 epochs.