Symbolic Prompt Tuning Completes the App Promotion Graph

Zhongyu Ouyang[1], Chunhui Zhang[2], Shifu Hou[1], Shang Ma[1], Chaoran Chen[1], Toby Li[1], Xusheng Xiao[3], Chuxu Zhang[4], Yanfang Ye[1] ✉

[1] University of Notre Dame, USA
{zouyang2,shou,sma5,cchen25,toby.j.li,yye7}@nd.edu
[2] Dartmouth College, USA chunhui.zhang.gr@dartmouth.edu
[3] Arizona State University, USA xusheng.xiao@asu.edu
[4] Brandeis University, USA chuxuzhang@brandeis.edu

Abstract. Recent mobile applications (i.e., apps) have been extensively implanted with paid advertisements that promote other mobile apps, including malware that raises alarming concerns in cybersecurity. Excavating the app promotion patterns in the app-promoting ecosystem allows for early interceptions of malware installment, and hence has gained more attention in recent research. However, related data in the app-promoting ecosystem such as app developers and categories is often scarce, especially when the data is collected from a single data source. The scarce data is insufficient in training effective deep and complex models for app promotion pattern mining, and targeting the data scarcity problem is therefore the key to advancing research in app promotion pattern mining. Therefore, we aim to complete data in the app-promoting ecosystem to pave the way for app-promoting pattern mining. We present SymPrompt, a language model-based framework that leverages the symbolic prompts to complete the missing data in the app-promoting ecosystem. The symbolic prompts are tokens that provide extra contextual information that assists the model in completing the missing data. We devise two sets of symbolic prompts containing contextual information from the perspectives of data structure and data semantics to assist the model prediction. Through extensive experiments, we demonstrate SymPrompt's effectiveness in completing the missing in the app-promoting ecosystem. Code: `https://github.com/zyouyang/SymPrompt`

1 Introduction

Mobile applications (i.e., apps) are extensively implanted with paid advertisements (i.e., ads) as a means of product promotion. It has been reported that over 57% of all apps in Google Play contain ad libraries, and this percentage reaches up to two-thirds within popular apps [24]. Among these ads, the app-promotion ads are widely used by Android app developers to promote other mobile apps. These app-promotion ads play a crucial role in helping users discover new apps, demonstrated by the research showing that 33% of users discover new apps

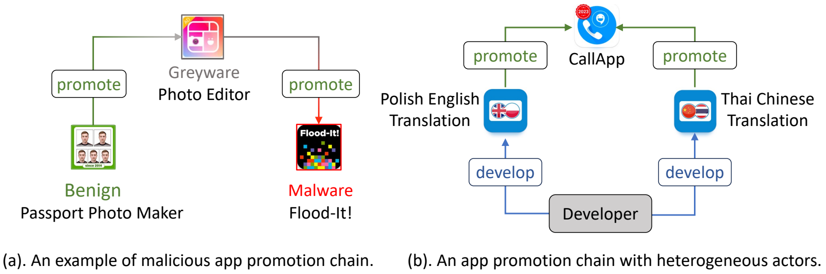


Fig. 1. App promotion chain examples in the app-promoting ecosystem.

through ads in other apps [19]. However, concerns arise regarding the trustfulness of the apps promoted through these ads, given the competitive nature of the industry and the potential for web technology with risks of invading personal privacy [18,13].

To prevent users from being promoted to malicious apps, researchers start to mine promoting patterns in the app-promoting ecosystem in order to provide early interventions for suspicious app installments. Some of the previous studies focus on analyzing the behaviors of ad libraries within the app promotion ecosystem [8,12], and some others analyze app promotion behaviors based solely on ways the apps are presented (web view or image view) [11,6]. These studies either primarily examine the behaviors of ad libraries, or analyze app promotion behaviors based solely on the app representations (web views or image views). Therefore, they pay too little attention to app propagation in terms of how massive individuals exploit the app promotion ecosystem and possess limited abilities to capture the interaction patterns with other apps in the promotion network. For instance, Figure 1(a) demonstrates an app promotion chain where a popular benign app “Passport Photo Maker” promotes a greyware app “Photo Editor”, which in turn promotes malware “Flood-It!”, a strategy game capable of scanning the local network and stealing sensitive phone information. Promotion chains among apps, as exampled above, hence cannot be detected and exploited by previous studies for app promoting behavior analysis.

Furthermore, prior studies lack a comprehensive understanding of the app-promoting ecosystem, which involves multiple heterogeneous actors beyond apps, such as app markets, security vendors, and developers. These actors collectively contribute to the promotion of specific apps. For example, Figure 1(b) provides a promotion chain path that could explain why an online messaging app, “Polish English Translation”, promotes “CallApp”. The underlying behaviors indicate that “Polish English Translation” shares the same developer as another translation app “Thai Chinese Translation”, which has been observed to promote “CallApp”. Hence, a more holistic approach that learns the intrinsic connections among these various entities is necessary to deeply understand the complexities of the app-promoting ecosystem, as well as its implications for society and online commerce. Such an in-depth understanding of the app-promoting ecosystem has the potential for various positive extensions, such as improving trust in app recommender systems and detecting malicious apps.

A natural way to holistically exploit the complex connections among the heterogeneous actors is to model the collected app-promoting data as a heterogeneous graph, and directly analyze the graph for app promotion pattern mining. Current graph representation learning [9,37,26,7,14,33,36] have shown their effectiveness in various domains such as recommender systems [16], node/graph classfication [5,17,22,32], knowledge engieering [20,34,28,10,27], and graph completion [29,15]. They are designed to capture high-order connectivity relationships between multiple entities. However, a huge obstacle blocking the exploitation of the app-promoting data with heterogeneous actors is the data scarcity problem. Around 15% of our collected app-promoting data in Google Play contains missing entries such as app developers and app categories. Also, data scarcity is one of the major challenges in training deep and complex graph learning models with exceptional performance, including graph learning methods [35]. Therefore, targeting the data scarcity problem is the key to advancing studies in app promotion pattern mining.

To address the above limitations, we first model the app-promoting data as a relational heterogeneous graph and center around the graph completion task. Specifically, each heterogeneous actor (e.g., an app developer, a visited URL) is abstracted to an entity in the graph, and each interaction between the actors (e.g., a developer develops an app, an app belongs to a category) is abstracted to a relation in the graph. Then, we center around the heterogeneous graph completion task, where given the observed graph and a query containing an entity and a relation type (e.g., a query contains the entity $developer_1$, and the relation type `developer-develop-app`), we aim to predict un-observed true positive entities that answer the query (e.g, the apps that are developed by the developer in the query).

Nevertheless, learning to complete a graph collected from the wild in our focused app promotion network is non-trivial. Existing methods for graph completion are either too simplistic for modeling the highly complex networks with scarce relations and entities [1,21,30], or heavily rely on rich semantic information to train a heavy model with massive parameters [25,15,31], which contradicts the data scarcity issue in app-promoting data collected from the wild. To target the limitations of existing techniques, we propose a framework that leverages a pre-trained language model (e.g., BERT) to model the complex relation, and incorporate two different symbolic prompts to compensate for the lack of semantic information closely related to the query. We first derived symbolic prompts from existing embedding-based methods like DistMult [30] to provide contextual information from the geometrical embeddings, and then generate the other set of symbolic prompts based on the correlation between the queried relation and metapaths. The two sets of symbolic prompts are concatenated with the query tokens to construct the input tokens to our framework. Our contributions are:

– We propose a novel framework that addresses the challenge of modeling complex connectivity patterns in the app promotion graph by leveraging the pre-trained language model. Our approach additionally incorporates two sets of symbolic prompts for further learning guidance.

- We collect a real-world dataset in the app promotion ecosystem, and demonstrate the effectiveness of our approach through extensive experiments. The results show that our approach outperforms existing techniques in terms of both accuracy and generalities.
- We provide a deeper understanding of the app promotion ecosystem, its complexities, and its implications for societal trust. Our work sheds light on the potential applications of heterogeneous graph completion methods and pre-trained language models in detecting malicious app promotions.

2 Background and Related Work

2.1 Definitions

Definition 1 (Heterogeneous Graph). *A heterogeneous graph (HG)* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ *consists of a entity set* $\mathcal{V}$*, an relation set* $\mathcal{E}$*, and the optional entity and relation features:* $\mathcal{X} = (\mathcal{X}_{\mathcal{V}}, \mathcal{X}_{\mathcal{E}})$*. The types of entities and relations are mapped through the type mapping functions* $\phi : \mathcal{V} \to \mathcal{A}$ *and* $\psi : \mathcal{E} \to \mathcal{R}$*, where* $\mathcal{A}$ *and* $\mathcal{R}$ *denotes the entity and relation type set respectively. Each relation is directional, and is represented as a triple* $(h, r, t) \in \mathcal{E}$ *where* $h, t \in \mathcal{V}, r \in \mathcal{R}$*. For a heterogeneous graph, there exists the constrain* $|\mathcal{A}| + |\mathcal{R}| > 2$*.*

Definition 2 (Metapath). *In a heterogeneous graph, a metapath is a predefined sequence of entity types and relation types that capture the semantic relationships between entities. Formally, a metapath* $\mathcal{P}$ *is denoted as* $e_1 \xrightarrow{r_1} e_2 \xrightarrow{r_2} ...e_L \xrightarrow{r_L} e_{L+1}$*, where* $r_i \in \mathcal{R}, e_i \in \mathcal{A}, r = r_1 \cdot r_2 \cdot ... \cdot r_L$ *is the composite relation between entity type* e_1 *and* e_{L+1}*, and* L *is the length of the metapath.*

Definition 3 (Heterogeneous Graph Completion). *For a query* (h, r) *where* $h \in \mathcal{V}$ *and* $r \in \mathcal{R}$*, the heterogeneous graph completion (HGC) task refers to discovering answers* $\mathcal{T} \subset \mathcal{V}$*, such that for all* $t \in \mathcal{T}$*,* $(h, r, t) \in \mathcal{E}$ *in reality.*

Table 1. Numbers and types for entities.

Ent.Type	Signature	VT Engine	Category	Developer	URL
Ent. #	185	65	36	3139	18870
Ent. Type	**Manifest**	**Benign**	**Greyware**	**Malware**	**Total**
Ent. #	10269	3961	1143	363	38031

2.2 Collected App Promotion Dataset

To exploit the complex relational patterns in the app promotion ecosystem, we collect data from Google Play and construct the app promotion heterogeneous graph (APHG) for the completion task.

Entities. The APHG encapsulates various entities derived from the following attributes: application, developer, application category, manifest, VirusTotal Engine, digital signature, and URL. Given the unique promotional behaviors demonstrated by benign, grey, and malicious apps, we further classify the application entity into three discrete classes - benign, grey, and malicious, and extend the aggregated count of entity types to nine. We provide the statistics of the entities in the APHG in Tab. 1. In total, there are nine types of entities.

Relations. The relations of interests are demonstrated in Fig. 2. In total, we consider twenty-nine classes of relations by further categorizing the applications into benign, greyware, and malware. Note that all the relations are directional. Despite the potential to gather additional information, neither the entities nor the relations are associated with any features. Therefore, our APHG is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. We provide detailed descriptions of the relations in the supplementary materials.

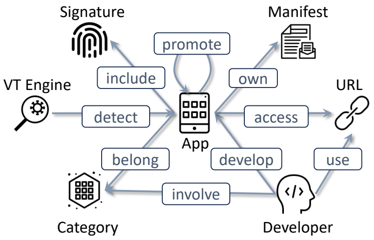


Fig. 2. The schema of the APHG.

2.3 APHG Completion Task

The APHG completion task refers to answering a query that contains an entity type and a relation type based on the observed APHG. For example, for a query $q = (developer_1,$ `developer-develop-app`), the model is expected to output the possible apps that $developer_1$ develops. We additionally note that the queried relations only associate with one of the constructed directional relations, excluding the reverse relations. For example, we query ($developer_1$, `developer-develop-app`), rather than (app_1, `app-developed-developer`).

2.4 Related Work

Embedding-based Methods. Knowledge graph embedding (KGE) methods employ geometric operations in the vector space to capture the underlying semantics of the graph, such as translation [1], bilinear transformation [30], rotation [21]. Other methods design embeddings from different perspectives. For

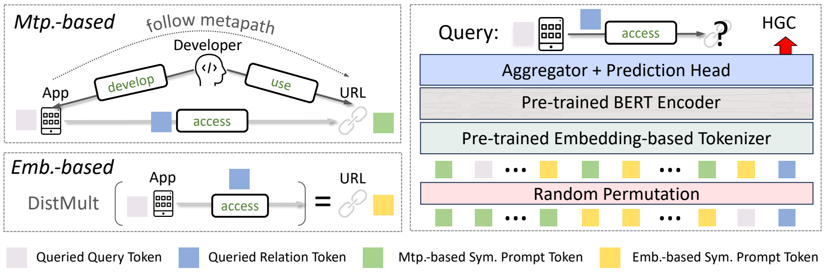


Fig. 3. Overall framework: the metapath-based and embedding-based symbolic prompts are pre-computed, and are concatenated with the queried entity and relation to construct the symbolic inputs.

instance, CompLEX [23] leverages compositionality to model the complex relationships between entities. ConvE [3] utilizes multi-layer convolutional networks on the 2D grid abstracted from the knowledge graph to encode local dependencies. Although conceptually straightforward, these methods encode each entity and relation's embedded information through a simple vector. The inherent simplicity of embedding-based methods can present challenges in scenarios involving complex reasoning and scarcity of information.

Transformer-based Methods. Taking account of the relatively weak expression power of the embedding-based methods, several recent works utilize transformers for additional enhanced contextual information encoding. Some works take the triple as the input and perform tasks such as triple classification and link prediction. For example, KG-BERT [31] treats triples as textual sequences to inject semantic information and exploits pre-trained BERT to learn context-aware embeddings. PKGC [15] leverages the entity's semantic information and converts them into natural prompt sentences to address the closed-world assumption (CWA) and incoherent issue. However, the above methods require the scoring of all possible triples in inference, therefore introducing some unnecessary calculation overheads. On the other hand, some other works are designed to directly output the candidate entities. For example, StAR [25] designs a structure-aware and structure-augmented framework for efficient KGC inference. HittER [2] extracts context neighbors for the source entity and introduces the additional masked entity prediction task for balanced contextualization. GenKGC [29] introduces relation-aware demonstration and entity-ware hierarchical decoding for better representation learning. Despite the progress made so far, we notice some implementation gaps in applying the above methods to a knowledge graph and a heterogenous graph: First, entities in a knowledge graph naturally entitle semantic information, while this is not always true for a heterogeneous graph; Second, the above methods left out the entity/node type information provided in a heterogeneous graph, therefore leaving considerable space for performance improvement. In contrast, our model is designed to not only straightly output the candidate entities, which eliminates the calculation overhead but also fully utilize the entity and relation type information for better prompting.

3 SymPrompt

Our SymPrompt approach leverages the pre-trained BERT [4] as a language model to process the tokenized inputs. The reasons are tri-folded: (i) BERT reduces the computational overhead by directly outputing the probability distribution over the entities, where the corresponding probabilities represent the ranking scores. In comparison, geometrical-based KGE methods require further similarity calculation to obtain the ranking scores; (ii) The bidirectional attention learning mechanism in BERT allows for complex input sequence processing, which is essential in aligning the query tokens with the symbolic prompt tokens to the completion task; (iii) BERT accepts flexible length of the input tokens, which include the query tokens and the symbolic prompt tokens. It supports various numbers of symbolic prompts. This feature further enhances the practical applicability of our framework. Subsequently, the encoded tokens are aggregated and decoded by a two-layer MLP to output the final result. The overall framework is depicted in Fig. 3. The input tokens are composed of three parts, as demonstrated in Fig. 3: (i) the query tokens, including the queried entity and relation token; (ii) the symbolic prompts generated based on embedding-based methods, such as DistMult; (iii) the symbolic prompts generated based on the correlation between the metapaths and the queried relation. In the following content, we provide details regarding these symbolic prompts.

3.1 Embedding-based Symbolic Prompts

Prior embedding-based models have demonstrated remarkable performance on various benchmark datasets. These models possess inherent simplicity that renders them proficient tokenizers, effectively mapping entity and relation tokens to a shared semantic space. In this paper, we select DistMult [30] as the pre-trained embedding-based method to tokenize the entities and relations. Note that this is a designer's choice and can be substituted with any other methods that fit our framework. Let n be the size of the embedding-based symbolic prompts, which are defined as the top-n predicted entities by the pre-trained embedding-based methods according to the predicted scores. For example, when considering the query (app_1, `app-access-URL`), the embedding-based symbolic prompts are represented as $S_e = [url_1,\ url_2,\ ...,\ url_n]$, where S_e includes the top-n URLs predicted by DistMult that may be accessed by app_1.

3.2 Metapath-based Symbolic Prompts

Previous embedding-based methods rely on geometric operations to model relations among entities, resulting in symbolic prompts that share similarities from a geometric perspective. Meanwhile, semantic relational information among entities can also be extracted from the metapaths of the HG. Therefore, we additionally provide the model with metapath-based symbolic prompts from the semantic perspective, based on the assumption that relations are semantically correlated with metapaths to different extents. We first introduce the measure of the semantic correlation between a metapath and a relation and then illustrate how to utilize the correlation to create the metapath-based symbolic prompts.

Relation-Metapath Correlation We first define the functions $src(\cdot)$ and $dst(\cdot)$ as the source and destination entity type mapping functions for a relation r respectively. We make the following definitions:

Definition 4 (r-valid Metapath). *A metapath* $p = e_1 \xrightarrow{r_1} e_2 \xrightarrow{r_2} ...e_L \xrightarrow{r_L} e_{L+1}$ *is* r*-valid if and only if* $src(r) = e_1$ *and* $dst(r) = e_{L+1}$.

Definition 5 (p-Hit). *For a triple* (h, r, t)*, where* r *is the relation,* h *and* t *are the source and destination entity respectively, we say the triple is* p*-Hit if and only if there exists at least one path from* h *to* t *such that this path is an instance of the metapath* p.

Definition 6 (p-Hit Ratio). *For a* p*-Hit triple* (h, r, t)*, the corresponding* p*-hit ratio is defined as the ratio of* t *among all the entities reached by following the metapath* p *starting from entity* h*; if* t *cannot be reached, the* p*-hit ratio is zero.*

Definition 7 (r-p Ratio). *For a relation* r*, a* r*-valid metapth* p*, and triples* $\mathcal{E}_r = \{(h, r, t) \in \mathcal{E}\}$*, the corresponding* r*-*p *ratio is defined as the averaged* p*-Hit Ratio in* $\mathcal{E}_r$.

Example. For simplicity, we abbreviate relation `benign-access-URL` as `access`, `developer-develop-app` as `develop`, and `developer-use-URL` as `use`. For relation $r =$ `access`, one of the metapaths $p = benign \xleftarrow{\texttt{develop}} developer \xrightarrow{use} URL$ is an r-valid metapath. For the triple ($benign_1$, `access`, url_1), if there exists a path "$benign_1 \xleftarrow{\texttt{develop}} developer_1 \xrightarrow{\texttt{use}} url_1$", then we say the triple is p-Hit. Furthermore, if starting from $benign_1$ and following p reaches to a set of nodes $\mathcal{T}_h^p$ where $url_1 \in \mathcal{T}_h^p$, and $\mathcal{T}_h^r \in \mathcal{T}_h^p$ represent the entities that are connected with $benign_1$ with relation r, then the p-Hit Ratio is $1/|(\mathcal{T}_h^p \setminus \mathcal{T}_h^r| + 1)$. Since the r-p Ratio is defined as the probability of finding the answer t by following metapath p starting from the entity h, we utilize it as the correlation indicator and select top-k metapaths that are most correlated with relation r, denoted as $\mathcal{P}_r$.

Entity Candidate Refining Even if metapaths in $\mathcal{P}_r$ are selected based on the correlation with the relation, the reachable entities may still contain noise regarding the queried relation, especially when the path passes a high-degree entity, which results in a significant size of the entity candidates with little and even noisy information. To further refine the entity candidate set, we first categorize the metapaths $p \in \mathcal{P}_r$ as one leading to either a large or small size of candidates. For those who lead to small-sized candidates, we *union* the candidates, and for large-sized candidates, we *intersect* them. The rationale is as follows: if a relation is weakly related to a metapath, then the candidate size is large and we rely on the *intersect* operation to filter out the noise; if some relations are broadly related to more than one metapath, then the candidate size is small and the *union* operation gathers all the possible candidates. To further reduce the size of the entity candidates, we lastly utilize an embedding-based method to select the top-n entities among the candidates as the final refined metapath-based symbolic prompts.

3.3 Combined Input Tokens

The final input of a query (h, r) is defined as the concatenation of the embedding-based symbolic prompt tokens, the metapath-based symbolic prompts tokens, as well as the query tokens. Prior to the language model, we randomly permute the input tokens. This step is essential in forcing the language model to learn the intrinsic connection between the query and the answer, rather than consulting the positional information as the shortcut. We validate the necessity of this step in the following experiments. Subsequently, we utilize an embedding-based method as the tokenizer (rather than the BERT's original tokenizer) to project the tokens into the embedding space for encoding. Finally, we adopt the binary cross entropy loss to train the language model for the HGC task.

4 Experiment

4.1 Setup

For baseline models, we carefully select DistMult [30], ComplEX [23], ConvE [3], HittER [2], and LTE [38] as the baselines, for they can be easily adapted to our HGC task. We evaluate the models with two key metrics, mean reciprocal rank (MRR) and Hits@K, where MRR provides an absolute measure of ranking performance via the average reciprocal rank of the correct candidates for each test example, and Hits@K measures the proportion of test examples for which the correct candidate is ranked within the top-K predicted candidates. Higher values of MRR and Hits@K indicate better performance in accurately ranking the correct candidates for the graph completion task. We select a pre-trained DistMult [30] as the tokenizer to tokenize the entity and relation tokens and utilize a pre-trained ComplEX [23] to refine the symbolic prompts as described in § 3.2. Note that the above choices are a matter of preference, and can be substituted with other embedding-based methods such as TransE [1].

4.2 Performance on App Promotion HGC

The performance comparison with the baselines is shown in Tab. 2 (based on the three types of input token components, i.e., the query tokens, the embedding-based and the metapath-based symbolic prompt tokens as well as the adoption

Table 2. Performance comparison with the baseline models on the app promotion dataset. Best results are bolded, and runner-ups are underlined.

Model	Hit@1	Hit@3	Hit@5	Hit@10	MRR
DisMult [30]	.6040	.7280	.7550	.8350	.6840
ComplEX [23]	.6680	.7780	.8180	.8650	.7370
ConvE [3]	.6400	.7460	.7950	.8490	.7110
HittER [2]	.5505	.6758	.7227	.7862	.6312
ConvE-LTE [38]	.6350	.7444	.7918	.8506	.6602
Distmult-LTE [38]	.6381	.7651	.8083	.8677	.7174
Base	.7246	.7610	.7729	.7895	.7481
Emb.-based Only	.7786	.8272	.8447	.8672	.8096
Mtp.-based Only	.4567	.4740	.4843	.5082	.4795
Ours w/o. Rand. Perm.	.7383	.7817	.7940	.8118	.7653
Ours	**.8393**	**.8710**	**.8802**	**.8922**	**.8587**

of token random permutation/shuffle, we consider five settings under our framework, as shown in Tab. 3). The results in Tab. 2 demonstrate that our model outperforms the other baselines by a significant margin. This is because while our model utilizes DistMult as the tokenizer to project the tokens in the embedding space, it does not merely rely on the simple multiplication operation as in DistMult for query answering; instead, the attention mechanism in the deep layers of the language model captures the complex interaction between the query and the symbolic prompt tokens. We also observe that as the value of K in Hit@K increases, the performance gap between our model and the baselines gradually diminishes. This phenomenon could be explained by the two-step inference process followed by our model: (i) the model processes the symbolic prompts and attempts to identify the answers out of the input tokens. If the correct answer exists within the symbolic prompts, the model confidently outputs it with a high probability, leading to higher hit ratios with a small K. This *answer identification* step is relatively straightforward; (ii) if the answer is not included in the symbolic prompts, the model instead attempts to generate the answer token. We term the second step the *answer generation* step, as it requires the model to deduce the interactive patterns among the input tokens wrt the query, and is, therefore, more challenging compared with the first one. The two-step inference process reveals that our model excels at both answer identification and generation. Additionally, the Hit@1 in all settings but the *Mtp. Only* is significantly improved upon DistMult, suggesting our framework's effective combination of the answer identification and generation steps for query answering. The downgrade Hit@1 in *Mtp. Only* results from the tilted focus to the more challenging answer generation step, for the massive noise in the symbolic prompts. We detailedly analyze the two-step inference process in § 4.3.

4.3 Component Analysis

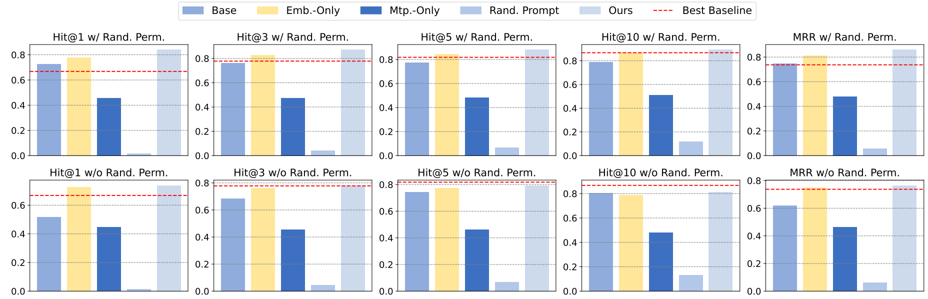


Fig. 4. Results of all component-differed variant models. The model variants in the first row randomly permute the input tokens, and the ones in the second row do not. The input tokens are constructed as indicated by the legend.

We analyze the effectiveness of each component in our framework to confirm the necessity of constructing our model as designed and provide supportive evidence for the two-step inference process conducted by our model. Additionally, we add another model variant named *Random-Prompt*, where the symbolic prompts are replaced with randomly sampled entity tokens. The input component settings of the model variants are shown in Tab. 4.

Table 3. Detailed settings of model variants in Tab. 2. Emb., Mtp., and Query represent embedding-based prompt tokens, metapath-based prompt tokens only, and query tokens respectively. Shuffle represents the random permutation layer.

Variant	Emb.	Mtp.	Query	Shuffle
Base	✗	✗	✓	✓
Emb.-based Only	✓	✗	✓	✓
Mtp.-based Only	✗	✓	✓	✓
Ours w/o Rand. Perm.	✓	✓	✓	✗
Ours (*SymPrompt*)	✓	✓	✓	✓

Performance Comparison. We combine the adoption of the random permutation layer and the components of the input tokens to expand the model variants, and compare the performance with the best baseline in Fig. 4:

For the *Base* model which shares the same input tokens and embedding space with DistMult, it relies on the language model (BERT) and the two-layer MLP decoder for query answering, substituting the simple matrix multiplication employed in DistMult. This substitution, while allowing the model to yield better Hit@1 performance than the baselines, fails to achieve consistent improvement across all evaluation metrics. This is because it is challenging to enforce a complex language model encoder and a decoder to fill the role of the multiplication operation. Therefore, we detour the functionality of our model from the replication of matrix multiplication to mining the complex interactive patterns between the query tokens and the symbolic prompt tokens.

Table 4. The token components discussed in § 4.3. Emb. represent embedding-based prompt tokens, Mtp. represents metapath-based prompt tokens, Query represents query tokens, and Rand. represents random prompt tokens.

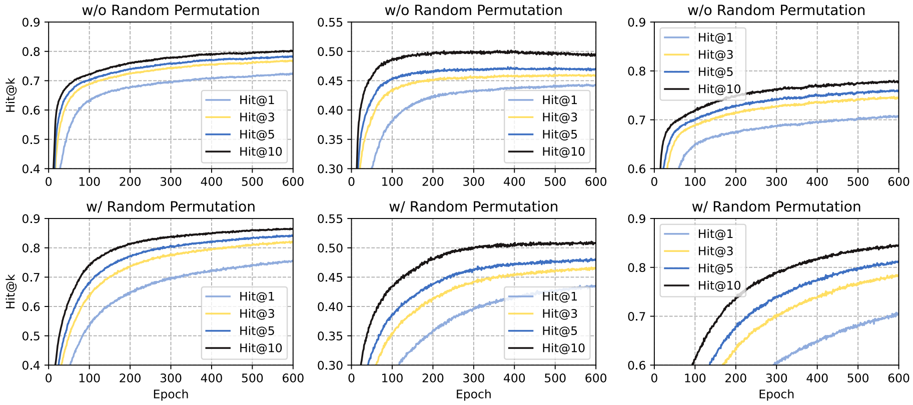
Variant	Emb.	Mtp.	Query	Rand.
Base	✗	✗	✓	✗
Emb.-based Only	✓	✗	✓	✗
Mtp.-based Only	✗	✓	✓	✗
Rand. Prompt	✗	✗	✓	✓
SymPrompt (ours)	✓	✓	✓	✗

For the *Rand. Prompt* model, the addition of randomly generated symbolic prompts completely collapses the model, regardless of the employment of random permutation. This is because the model is overwhelmed with the massive noise brought by the random tokens, and cannot identify the query tokens for the downstream task. It suggests that the symbolic prompt tokens should be crafted with very limited noise to avoid model collapse.

The *Embedding-based Only* variant yields superior performance when combined with the random permutation layer, especially for hit ratios with small K's. This suggests that the embedding-based symbolic prompts are effective in providing both possible candidates and additional information regarding the query. The inferior performance when not adopting the random permutation layer results from the positional shortcut taken by the model, which weakens the model's ability in answer generation.

The *Metapath-based Only* model underperforms the *Base* model due to the massive noise introduced in the prompts. Although the *Embedding-based Only* model also accepts extra symbolic prompt tokens as the input, these prompt tokens are geometrically similar in the embedding space and therefore introduce less noise when providing information related to the query. However, the noises in the metapath-based symbolic prompts introduced by following the correlated metapaths are intractable, which inevitably results in inferior performance.

Our model outperforms all other variants under the two settings, suggesting the necessity of combining the two sets of symbolic prompts. Furthermore, we discover that as K increases, the gap between our variants and the baselines decreases, and decreases faster under the *w/o Rand. Perm.* setting. The decrement is observed because as K increases, our model relies more on the more challenging answer generation step to improve the metrics, while the baselines consistently generate the answers (i.e., conducting the answer generation step) through their designed geometrical operations. In addition, the random permutation layer mingles the input tokens together, forcing our model to focus on the intrinsic connection between the input tokens rather than the one hooked by the token positions. This deprivation of position shortcut enhances our model's ability to generate more possible answers, therefore resulting in slower performance gap decrement as K increases, compared with the baselines.



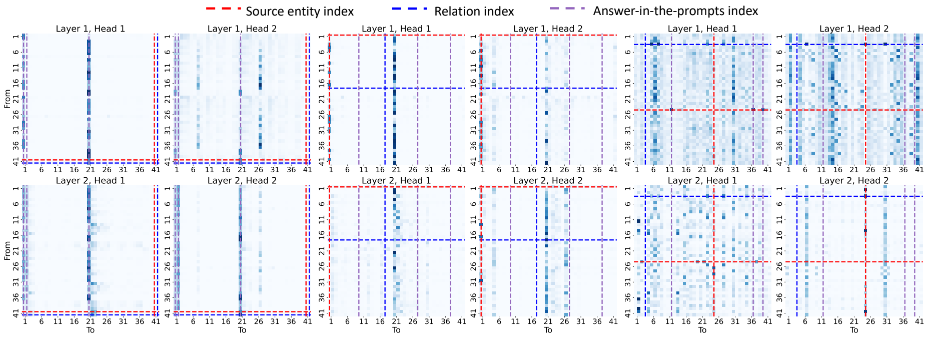
(a). Full Prompt (Ours) (b). Metapath-based Only (c). Embedding-based Only

Fig. 5. The training dynamics of models with full symbolic prompt tokens (ours), metapath-based symbolic prompts only, and embedding-based symbolic prompts only.

Training Dynamics Comparison. We analyze the training dynamics of the *Embedding-based Only* model, *Metapath-based Only* model, and our model under two settings (i.e., adopt random permutation and not), and show the training dynamics in Fig. 5. We observe that models converge slower under *w/ Rand.Perm.* than those under *w/o Rand.Perm.* This is because models with consistently positioned input tokens tend to opt for the shortcut solution like memorizing positions, rather than learning the intrinsic relations between the tokens. Identifying the shortcut token's positions, compared with generating the answer based on the learned interactive patterns, is less challenging, leading to faster model convergence. The phenomenon aligns with the two-step inference process, where the model first tackles an easy task by attempting to identify existing answers in the symbolic prompt tokens and then engages in the more complex task of generating answers. Additionally, we notice that the performance gaps in Hit@K for different K's are larger under *w/ Rand. Perm.*, indicating better generality for the HGC task. Our model neither saturates as quickly as *Metapath-based Only* due to less noise introduced in the symbolic prompt, nor takes excessively long to achieve performance improvement like *Embedding-based Only*, which relies heavily on accessible shortcuts that hinder generality.

4.4 Random Permutation on Model Learning

To analyze how random permutation effects model learning, we showcase the normalized attention scores heatmaps in Fig. 6 under the following three conditions : (a) train and test the model under *w/o. Rand. Perm.*.; (b) train the model *w/o. Rand. Perm.*, but test it under *w/ Rand. Perm.*; (c) train and test the model under *w/ Rand. Perm.* (our model). The correct answer entity ranks 1, 133, and 1 under the three conditions respectively. Under condition (a), we



(a). Train and test w/o Rand. Perm. (b). Train w/o and test w/ Rand. Perm. (c). Train and test w/ Rand. Perm (ours).

Fig. 6. The attention heatmaps in the layers of the model under three settings.

see the model consistently pays heavy attention to the 1^{st} and 21^{st} tokens. This is because we set the number of embedding-based and metapath-based symbolic prompts as 20, and the two positions correspond to the most probable answers. Without random permutation, the model easily identifies the position shortcut and pays less attention to the query tokens (indexed by the red and blue dotted lines). Under condition (b), the model fails to assign a high rank to the correct answer (133): the model rigidly assigns heavy attention to the 1^{st} and 21^{st} tokens when they are no longer the most probable answers due to random permutation. The model trained and tested under *w/ Rand. Perm.* as we designed, on the other hand, assigns much more attention to the input sequence. Under condition (c), our model learns to dynamically assign attention to the potential answers (red and purple line intersections in Layer 1, Head 1), the source entity token (red vertical dotted line in Layer 2, Head 2), and any other important information it deems important (tokens indexed by 7, 31, etc.). The comparison showcases that random permutation stimulates the model to learn the interactive patterns between the input tokens, increasing its power in generalized query answering.

5 Conclusion

In this work, we focus on completing the missing data in the context of app promotion to combat the information scarcity problem and propose a language model-based approach named SymPrompt that leverages symbolic prompts to provide valuable hints to answer the query. This research advances the understanding of app promotion networks, and we direct future works toward explainable information completion in the context of the app promotion ecosystem.

Acknowledgements

This work was partially supported by the NSF under grants IIS-2321504, IIS-2334193, IIS-2340346, IIS-2203262, IIS-2217239, CNS-2203261, and CMMI-2146076. Any opinions, findings, and conclusions or recommendations expressed

in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

1. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NeurIPS (2013)
2. Chen, S., Liu, X., Gao, J., Jiao, J., Zhang, R., Ji, Y.: Hitter: Hierarchical transformers for knowledge graph embeddings. In: EMNLP (2021)
3. Dettmers, T., Pasquale, M., Pontus, S., Riedel, S.: Convolutional 2d knowledge graph embeddings. In: AAAI (2018)
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL (2019)
5. Guo, Z., Zhang, C., Fan, Y., Tian, Y., Zhang, C., Chawla, N.V.: Boosting graph neural networks via adaptive knowledge distillation. In: AAAI (2023)
6. Hao, S., Liu, B., Nath, S., Halfond, W.G., Govindan, R.: Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In: Annual International Conference on Mobile systems, Applications, and Services (2014)
7. Jia, Y., Zhang, C., Vosoughi, S.: Aligning relational learning with lipschitz fairness. In: ICLR (2024)
8. Jin, L., He, B., Weng, G., Xu, H., Chen, Y., Guo, G.: Madlens: Investigating into android in-app ad practice at api granularity. IEEE Transactions on Mobile Computing (2021)
9. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
10. Li, J., Zhang, C., Zhang, C.: Heterogeneous temporal graph neural network explainer. In: CIKM (2023)
11. Liu, B., Nath, S., Govindan, R., Liu, J.: Decaf: Detecting and characterizing ad fraud in mobile apps. In: USENIX Symposium on Networked Systems Design and Implementation (2014)
12. Liu, T., Wang, H., Li, L., Luo, X., Dong, F., Guo, Y., Wang, L., Bissyandé, T., Klein, J.: MadDroid: Characterizing and detecting devious ad contents for android apps. In: WWW (2020)
13. Liu, Z., Dou, G., Tian, Y., Zhang, C., Chien, E., Zhu, Z.: Breaking the trilemma of privacy, utility, and efficiency via controllable machine unlearning. In: The Web Conference (2024)
14. Liu, Z., Zhang, C., Tian, Y., Zhang, E., Huang, C., Ye, Y., Zhang, C.: Fair graph representation learning via diverse mixture-of-experts. In: The Web Conference (2023)
15. Lv, X., Lin, Y., Cao, Y., Hou, L., Li, J., Liu, Z., Li, P., Zhou, J.: Do pre-trained models benefit knowledge graph completion? a reliable evaluation and a reasonable approach. In: Findings of the Association for Computational Linguistics (2022)
16. Ouyang, Z., Zhang, C., Hou, S., Zhang, C., Ye, Y.: How to improve representation alignment and uniformity in graph-based collaborative filtering? In: International AAAI Conference on Web and Social Media (2024)
17. Qian, Y., Zhang, C., Zhang, Y., Wen, Q., Ye, Y., Zhang, C.: Co-modality graph contrastive learning for imbalanced node classification. In: NeurIPS (2022)
18. Rafieian, O., Yoganarasimhan, H.: Targeting and privacy in mobile advertising. Marketing Science (2021)

19. Research, G.: How people discover, use, and stay engaged with apps. Think with Google (2023)
20. Rizun, M.: Knowledge graph application in education: a literature review. Acta Universitatis Lodziensis. Folia Oeconomica (2019)
21. Sun, Z., Deng, Z.H., Nie, J.Y., Tang, J.: Rotate: Knowledge graph embedding by relational rotation in complex space. In: ICLR (2018)
22. Tian, Y., Dong, K., Zhang, C., Zhang, C., Chawla, N.V.: Heterogeneous graph masked autoencoders. In: AAAI (2023)
23. Trouillon, T., Welbl, J., Riedel, S., Gaussier, E., Bouchard, G.: Complex embeddings for simple link prediction. In: ICML (2016)
24. Viennot, N., Garcia, E., Nieh, J.: A measurement study of google play. In: ACM international conference on Measurement and modeling of computer systems (2014)
25. Wang, B., Shen, T., Long, G., Zhou, T., Wang, Y., Chang, Y.: Structure-augmented text representation learning for efficient knowledge graph completion. In: WWW (2021)
26. Wen, Q., Ouyang, Z., Zhang, C., Qian, Y., Zhang, C., Ye, Y.: GCVR: Reconstruction from cross-view enable sufficient and robust graph contrastive learning. In: The 40th Conference on Uncertainty in Artificial Intelligence (2024)
27. Wen, Q., Ouyang, Z., Zhang, J., Qian, Y., Ye, Y., Zhang, C.: Disentangled dynamic heterogeneous graph learning for opioid overdose prediction. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2022)
28. Wu, J., Zhang, C., Liu, Z., Zhang, E., Wilson, S., Zhang, C.: Graphbert: Bridging graph and text for malicious behavior detection on social media. In: ICDM (2022)
29. Xie, X., Zhang, N., Li, Z., Deng, S., Chen, H., Xiong, F., Chen, M., Chen, H.: From discrimination to generation: knowledge graph completion with generative transformer. In: WWW (2022)
30. Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: ICLR (2015)
31. Yao, L., Mao, C., Luo, Y.: Kg-bert: Bert for knowledge graph completion. arXiv preprint arXiv:1909.03193 (2019)
32. Yuan, X., Zhang, C., Tian, Y., Ye, Y., Zhang, C.: Mitigating emergent robustness degradation on graphs while scaling up. In: ICLR (2024)
33. Yue, H., Zhang, C., Zhang, C., Liu, H.: Label-invariant augmentation for semi-supervised graph classification. In: NeurIPS (2022)
34. Zhang, C., Huang, C., Li, Y., Zhang, X., Ye, Y., Zhang, C.: Look twice as much as you say: Scene graph contrastive learning for self-supervised image caption generation. In: CIKM (2022)
35. Zhang, C., Huang, C., Tian, Y., Wen, Q., Ouyang, Z., Li, Y., Ye, Y., Zhang, C.: When sparsity meets contrastive models: Less graph data can bring better class-balanced representations. In: ICML (2023)
36. Zhang, C., Liu, H., Li, J., Ye, Y., Zhang, C.: Mind the gap: Mitigating the distribution gap in graph few-shot learning. Transactions on Machine Learning Research (2023)
37. Zhang, C., Tian, Y., Ju, M., Liu, Z., Ye, Y., Chawla, N., Zhang, C.: Chasing all-round graph representation robustness: Model, training, and optimization. In: ICLR (2023)
38. Zhang, Z., Wang, J., Ye, J., Wu, F.: Rethinking graph convolutional networks in knowledge graph completion. In: WWW (2022)

A Limitations

While our work showcases its superior performance, we do not test the limit of our framework by trialing various language model backbones (in our work, we use the representative BERT), which also effectively encode the input tokens by considering their interactions. In addition, we design the framework specifically for a heterogeneous graph, which is not generalized to knowledge graphs due to the design of metapath-based symbolic prompts. We expect the symbolic prompts to have the same effect in the case of knowledge graph-based question answering, and also project the generalization as a future research direction. Lastly, our model is evaluated specifically in the context of app promotion. It is worthwhile to trial its effectiveness in other context and domains that also encounter the problem of information scarcity.

B The App Promotion Dataset

B.1 Data Collection

The app promotion dataset is gathered following three steps. Initially, for each app, the package name, developer information, and category of each app are crawled from Google Play. Subsequently, an analysis is conducted on the app using VirusTotal to examine the flags associated with its security level, along with the corresponding URLs. Lastly, the manifest and signature of each app are inferred through the process of reverse engineering (e.g., interesting strings provided by the VirusTotal report). The promotion actions between apps are discovered by checking whether the clickable widgets in an UI from the source apps lead to the download page of the sink app. If so, then a `source_app <promotes> sink_app` relation is identified. The collected raw data is then used to construct the following App Promotion Heterogeneous Graph (APHG).

B.2 Detailed Relations

The descriptions of each relation in APHG are detailed as follows: (1) R1: an `app-promote-app` relation indicates that there exists a promotion link from the subject app to the object app; (2) R2: an `app-include-signature` relation means that a digital signature can be used to verify the authenticity and integrity of the app package; (3) R3: an `engine-detect-app` relation indicates that a VT engine marks an app with a specific flag (e.g., adware or Trojan); (4) R4: an `app-belong-category` represents that an app belongs to a specific app category categorized by Google Play; (5) R5: a `developer-involve-category` relation suggests that an app created by the developer is categorized into a specific app category; (6) R6: a `developer-develop-app` relation signifies that a developer develops an app; (7) R7: an `app-access-URL` relation denotes that an app has access to a specific URL; (8) R8: a `developer-use-URL` relation indicates that the app developed by the developer may access a specific URL;

(9) R9: an `app-own-manifest` relation represents that an app is associated with a specific manifest file. Since apps with different security levels follow different behavior patterns, we further classify them into three sub-classes. For example, the relation `app-belong-category` is extended to: `benign-belong-category`, `grey-belong-category`, and `mal-belong-category` In total, the above relations are extended to twenty-nine classes of relation types.

C Empirical Evaluation for Entity Candidate Refining

Table 5. Empirical evaluation results of the correlation between metapath and relation.

Relation	p_r	s_r	b_r
`mal-belong-category`	0.922	4.932	0.137
`benign-access-URL`	0.879	129.329	0.007
`developer-use-URL`	0.457	42.905	0.002
`grey-promote-grey`	0.660	154.786	0.135

We here empirically validate the correlation between the queried relation and the refined entity candidates obtained by following the metapaths. Particularly, for relation r, denote s_r as the average size of the filtered candidates for all r-related triples. We also let p_r be the probability that the candidates contain the correct answer, and let $b_r = s_r/|dst(r)|$ be the proportion of refined candidates to all possible entity answers. Table 5 presents the statistics of the selected relations due to the large size of $\mathcal{R}$. From the table, we can see that: (1) for the relation `mal-belong-category`, where the average candidate size is 5, there is a 0.922 probability that the correct answer is within these candidates. Relations like `mal-belong-category`, which has high p_r and small s_r, demonstrate their strong correlation with the metapaths; (2) for some relations whose candidate sizes are significantly reduced by the refining process (demonstrated by small b_r), p_r is positively related to s_r (e.g., `benign-access-URL` has higher p_r and s_r than `developer-use-URL`). Those relations have intermediate strengths of correlation with the metapaths; (3) for relations like `grey-promote-grey`, the refining process can neither reduce the candidate size significantly (155 is not small enough), nor provides rich information about the query (low p_r). These relations are therefore weakly related to the metapaths. Nevertheless, the results confirm that the metapaths are correlated with the relations to different extents, and this correlation is effective in generating informative symbolic prompts.

D Experimental Setup

Compared Baselines. We compare our approach against several baselines commonly used in the graph completion task:

- **DistMult** [30]: DistMult represents entities and relations as low-dimensional vectors and utilizes a bilinear dot product scoring function for link prediction.

- **ComplEX** [23]: ComplEX extends DistMult by using complex-valued embeddings, which allows for a more expressive representation. The calculation remains linear in both space and time.
- **ConvE** [3]: ConvE employs a convolutional neural network architecture to encode entities and relations. It operates on 2D tensors to capture local patterns and dependencies within the knowledge graph.
- **HittER** [2]: HittER utilizes hierarchical transformers to learn knowledge graph embeddings, balancing the contextual relational information and the information from the training entity.
- **LTE** [38]: LTE extends embedding-based methods by equipping existing knowledge graph embedding models with linearly transformed entity embeddings. It mines semantic information from entity representations to enhance the model performance. In this paper, we select DistMult and ConvE as the backbones, denoted as DistMult-LTE and ConvE-LTE respectively.

Implementation Details. We use a pre-trained DistMult [30] as the backbone model to tokenize the entities and relations as low-dimensional vectors. Note that this choice is a matter of preference, and can be substituted with other embedding-based methods such as TransE [1]. ComplEX [23] is utilized for prompt filtering, and can also be replaced by any other graph completion methods. We encode the input sequence with a two-layer BERT model, and utilize the sum operation to aggregate the encoded sequence. Finally, a two-layer MLP is applied as the prediction head (decoder) for the HGC task. The train/dev/test split ratio is 0.8/0.1/0.1, and we split the full data by randomly mask the instances of each type of relations in the APHG. During training, we employ the AdamW optimizer and use binary cross-entropy as the loss function. In the pre-trained phase of DistMult, the learnable parameters are initialized randomly, while BERT is loaded with pre-trained weight parameters. The training and testing process is conducted on an NVIDIA RTX 3090 GPU with 24 GB of memory.