PROOF-AUGMENTED RETRIEVAL AND REASONING: SUPERVISING LANGUAGE MODEL FOR KNOWLEDGE GRAPH COMPLETION WITH INTERPRETABLE LINK PREDICTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Proof-Augmented Retrieval and Reasoning (PARR), a general framework for training language models for Knowledge Graph Completion (KGC). PARR leverages proof paths sampled from interpretable link predictors (1) to augment retrieval database for enhanced sub-graph retrieval, (2) as ground-truth signals to train a REWRITER LLM for KG-based query rewritings, and (3) as a mean to "distill" the structural knowledge captured from pre-trained link predictors with structural prior to a REASONER LLM through chain-of-thoughts. PARR achieves state-of-the-art performance across multiple KGC datasets under both transductive and inductive settings, while being generative, scalable, and interpretable.

1 INTRODUCTION

Knowledge Graph Completion (KGC) is a fundamental task in machine learning that seeks to infer missing relations between entities in large-scale knowledge graphs. KGC enables downstream applications in diverse domains such as recommendation systems, scientific discovery, and healthcare.

Most traditional KGC methods fall into two categories. Knowledge graph embedding (KGE) models (Bordes et al., 2013; Trouillon et al., 2016; Sun et al., 2019) learn vector representations of entities and relations for efficient link prediction. GNN-based approaches (Vashishth et al., 2020; Zhu et al., 2021; 2023) leverage message passing to capture local graph structure. These models are lightweight and effective, but real-world knowledge-intensive domains often prioritize accuracy, interpretability, and interactive reasoning over raw inference latency. Moreover, both KGE and GNN models are inherently discriminative: predictions are made by scoring candidate entities, offering limited explainability and weaker generalization outside training distributions.

Large Language Models (LLMs) (lla, 2024; Yang et al., 2024) offer appealing generative and conversational abilities, but they do not naturally excel at KGC. Unlike KGE or GNN models built around structural priors, LLMs operate in the much larger space of natural language. Despite their internal knowledge, augmenting KGs with LLM-generated background information yields little improvement (Jiang et al., 2024). By contrast, methods such as NBFNet and ULTRA (Zhu et al., 2021; Galkin et al., 2024) achieve strong link prediction solely by modeling relational and structural patterns, without explicit entity embeddings. This suggests that background knowledge of entities is not the key ingredient for KGC. Prior attempts to directly fine-tune LLMs on KGC also underperform traditional link predictors significantly. (Yao et al., 2025; Zhu et al., 2024). Another line of work repurposes LLMs as discriminative encoders, trained with standard KGE objectives such as binary cross-entropy loss and negative sampling (Yao et al., 2019; Guo et al., 2024). While these methods improve accuracy, they reduce LLMs to specialized link predictors, sacrificing the generative and interactive capabilities that make them attractive for explainable KGC.

In this work, we identify three key challenges that have limited prior LLM-based approaches:

(1) Retrieval. Unlike natural language retrieval, where information is naturally grouped into paragraphs or documents, knowledge graphs lack inherent clustering patterns. Moreover, as there is no ground-truth retrieval labels for KGs, prior LLM-based link predictors often rely on simple

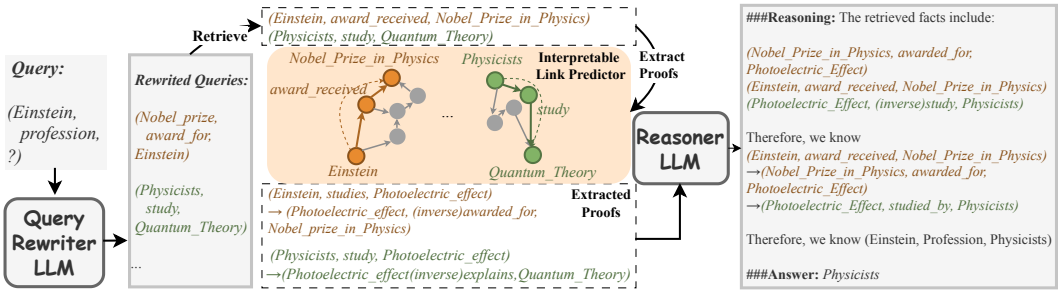


Figure 1: Overall pipeline of PARR. Given a query such as *(Einstein, profession, ?)*, the Rewriter LLM produces a set of semantically varied yet logically related sub-queries (e.g., *(Nobel_prize, award_for, Einstein)*) and *(Physicists, study, Quantum_Theory)* to broaden retrieval. Pre-trained interpretable link predictors (e.g., NBFNet (Zhu et al., 2021)) then provide proofs for retrieved facts, enabling targeted subgraph retrieval. The collected evidence is passed to the REASONER LLM to derive the final answer.

similarity-based strategies. For example, KICGPT (Wei et al., 2023) retrieves all triplets sharing the same head or tail entity, which is constrained to local neighbors while missing global context.

(2) Reasoning. Although LLMs demonstrate strong general reasoning ability, they are prone to hallucination in KG reasoning when exposed to noisy or irrelevant retrievals. As shown in Figure 3, LLM performance drops sharply as irrelevant context accumulates, highlighting the need for more robust reasoning mechanisms that can operate under imperfect retrieval.

(3) Structural understanding. Existing LLM-based KGC methods (Yao et al., 2025; Zhu et al., 2024) often reduce link prediction to a one-step task: given a subject and relation, directly predict the object. However, this formulation makes it difficult for LLMs to capture the underlying relational and structural patterns of the graph.

To address these challenges, we introduce Proof-Augmented Retrieval and Reasoning (PARR). PARR leverages proof supervision from interpretable link predictors to fine-tune native LLM-based generative link predictors, enhancing both retrieval and reasoning in three complementary ways:

(1) Proof-augmented retrievals. At retrieval time, we retrieve not only relevant triplets but also their associated proofs. Proofs serve as natural clusters that capture both local and global subgraph structure, yielding more comprehensive yet compact retrievals.

(2) Proof-guided Query Rewriting. Proofs could also act as pseudo ground-truth signals for retrieval. Inspired by query rewriting in natural language RAG (Ma et al., 2023), we fine-tune a REWRITER LLM to decompose queries into sub-queries, improving non-local retrieval and recall.

(3) Proof-guided reasoning. Finally, we fine-tune a REASONER LLM with Chain-of-Thought (CoT) supervision derived from proofs. This improves robustness to noisy retrievals and distills the structural knowledge of pre-trained link predictors into the LLM. The REASONER can be seen as performing graph traversal in the language space, with the next node determined by token probabilities.

The overall PARR framework is illustrated in Figure 1. Through extensive experiments, we show PARR achieves SOTA H@1 performance across all tested datasets, under both transductive and inductive settings. To our knowledge, this is the first native, fully generative LLM-based KGC framework to match the accuracy of discriminative link predictors.

The rest of the paper is organized as follows: Section 3 introduces background on link prediction and proof extraction. Section 4 presents the PARR framework. Section 5.2 reports experimental results across benchmark datasets and settings. Section 5.3 provides detailed ablation studies on each component of our framework. More results and ablations are provided in the appendix.

2 RELATED WORK

2.1 LINK PREDICTION ON KNOWLEDGE GRAPHS

Knowledge graph completion (KGC) is a long-standing task that seeks to infer missing links between entities. Knowledge Graph Embedding (KGE) (Bordes et al., 2013; Trouillon et al., 2016; Balazevic

et al., 2019) is an effective method for KGC. These models embed entities and relations into continuous spaces, but typically operate as black-box functions without interpretability.

Interpretable link prediction. Another line of works explored KG reasoning via GNN-based or rule-based methods. Notably, NBFNet (Zhu et al., 2021) solves link prediction as a shortest path finding problem and parametrize Bellman-Ford algorithm with GNN. A*Net (Zhu et al., 2023) improves the scalability of NBFNet by incorporating A* algorithm to select important nodes and edges. Neural Theorem Provers (Rocktäschel & Riedel, 2017; Cui et al., 2025) are rule-based neuro-symbolic methods that extends backward chaining algorithm into continuous space.

Link prediction with Language Models Leveraging pre-trained language models for link predictions has been explored for years. Earlier works leverage BERT as GNN encoder to encode triplets (Guo et al., 2022; Chen et al., 2021), while others use BERT to encode textual information of KG entities and relations (Yao et al., 2019; Wang et al., 2021; Lin et al., 2023; Youn & Tagkopoulos, 2023). These works mostly follow the same training regime used in KGE methods.

Recent works have explored LLMs for link prediction through fine-tuning (Yao et al., 2025; Zhu et al., 2024), but their performance often lags behind traditional KGE methods. On the other hand, KICGPT (Wei et al., 2023) achieves strong results by using a KGE model to generate candidate entities and asking an LLM to rerank them. However, it depends on proprietary models (ChatGPT), external KGE systems, and costly multi-round QA. MKGL (Guo et al., 2024) modifies the embedding and output spaces of LLMs to act as specialized link predictors. While effective, this approach reduces the LLM to a task-specific discriminative model, sacrificing its generative capacity.

In contrast, PARR is a fully generative, LLM-native framework that preserves the conversational ability of LLMs while achieving competitive link prediction accuracy. This generality also makes it easily adaptable to broader KG tasks such as node classification or complex KG-based QA.

2.2 Retrieval-Augmented KG Reasoning

Retrieval-Augmented Generation. Retrieval-augmented generation (RAG) (Lewis et al., 2021) equips LLMs with external memory by retrieving relevant facts to support reasoning in knowledge-intensive tasks. Works focus on jointly pre-training or fine-tuning LLMs with retrieval modules to boost accuracy (Guu et al., 2020; Izacard & Grave, 2021; Borgeaud et al., 2022; Izacard et al., 2022).

Query rewriting for RAG. Query rewriting improves retrieval by transforming complex queries into simpler sub-queries, or to incorporate more information to bridge retrieval asymmetry. HyDE (Gao et al., 2022) uses LLMs to generate synthetic document for document retrieval. Ma et al. (2023); Mao et al. (2024) train query rewriters on top of retrieve-then-read systems with preference fine-tuning, while LeRet (Hsu et al., 2025) introduces an iterative rewriting strategy for multi-hop QA.

Retrieval-Augmented KGQA. A related line of work comes from knowledge graph question answering (KGQA), where models answer complex natural-language questions over a KG. (Das et al., 2022) retrieves nearest-neighbor subgraphs and transfers their latent reasoning patterns to new questions via a GNN, enabling efficient multi-hop reasoning without explicit supervision. RoG (Luo et al., 2024) supervises its planning and reasoning modules using retrieved paths, while GNN-RAG (Mavromatis & Karypis, 2024) uses a GNN retriever to gather subgraphs and verbalized paths as evidence for an LLM. Although these approaches share a retrieve–reason intuition, they rely on supervision such as complete paths and ground-truth retrieval, that are not available in KGC datasets. In contrast, PARR obtains the necessary structural supervision from interpretable link predictors.

3 Preliminaries

3.1 Knowledge Graph and Link Prediction

Knowledge Graph. A Knowledge Graphs (KG) is a directed, multi-relation graph expressed as a set of triplets $(s, r, o) \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where $\mathcal{E}$ and $\mathcal{R}$ denote the set of entities and relations in the KG. Link prediction is one of the most important task in the domain of KG, whose goal is to predict missing object(subject) given subject(object) entity and relation, i.e. $(s, r, ?)$ or $(?, r, o)$.

Proofs. For simple link prediction tasks, given any pair of entities (s, o), its proof $p(s, o)$ can be defined as chain-like logical rules in the form of: $p(s, o) \leftarrow r_1(s, z_1) \wedge r_2(z_1, z_2) \wedge \cdots \wedge r_n(z_n, o)$, where $r_1, \cdots r_n$ are relations in the given KG. For path-based link predictors, the proofs can be reformulated as the paths take to reach object o from subject s, i.e. $p(s, o) = r_1(s, z_1) \rightarrow r_2(z_1, z_2) \rightarrow \cdots \rightarrow r_n(z_n, o)$.

3.2 Proof extractions with Interpretable Link Predictors

We consider two SOTA link prediction models with interpretable path formulation: NBFNet (Zhu et al., 2021) and A*Net (Zhu et al., 2023). Below we briefly describe how proofs can be extracted.

NBFNet parametrize Bellman-Ford path-finding algorithm with GNN. Given a triplet (s, r, o), we can sample top-*k* proofs $P_1, P_2, \cdots, P_k$ from a pre-trained NBFNet through a linear model((Baehrens et al., 2009)) (*i.e.* 1st order Taylor expansion), where each proof's score can be modeled by the partial derivative of the prediction $p(s, r, o)$ *w.r.t.* the proof path (Zhu et al., 2021):

$$P_1, P_2, \cdots, P_k = \underset{P \in \mathcal{P}_(s,o)}{\text{top-k}} \frac{\partial p(s, r, o)}{P}$$

In practice we use the average edge importance to approximate path score, which can be efficiently computed by auto differentiation.

A*Net learns a neural priority function to select a subset of nodes and edges on top of NBFNet to improve scalability. The A* algorithm provides a natural and distinct way of extracting proofs (Zhu et al., 2023). Given a triplet (s, r, o) and the pre-trained node priority function $s^t_{(s,r)}(x)$, we can sample top-*k* proofs by their average node importance $s(P)$:

$$s(P) = \frac{1}{|P|} \sum_{t=1, P^{(t)}=(x,r,y)}^{|P|} \frac{s^{(t-1)}_{s,r}(x)}{S^{(t-1)}_{s,r}}, \quad S^{(t-1)}_{s,r} = \max_{x \in \mathcal{E}^{(t-1)}} s^{(t-1)}_{s,r}(x).$$

4 Proof-Augmented Retrieval and Reasoning

We introduce Proof-Augmented Retrieval and Reasoning (PARR), a native LLM-based framework for link prediction. PARR is composed of three modules: a Rewriter, a Reasoner, and a non-trainable Retriever. We leverage proofs extracted from pre-trained interpretable link predictors to supervise the Rewriter LLM for KG query rewriting, and the Reasoner LLM for structure-awared reasoning given large retrieved context. Figure 1 provides an overview of our framework.

4.1 Proof Extraction

We start off by extracting top-*k* proofs for each triplet using pre-trained link predictors. To avoid model bias from the pre-trained link predictor, we sample proofs using a mixture of experts (MoE) of different models (NBFNet and A*Net), where each model has a distinct method on proof extraction 3. We further train each link predictors several times with different configurations and random seeds, and aggregate the resulting proofs. Finally, we filter out proofs with duplications or cycles. For triplets in the training set, we also filter out proofs containing the triplet itself, as it represents an existing edge for which the model is likely to give the highest score. The resulting RAG database $\mathcal{D}$ is then a set of triplet-proofs pairs: $\mathcal{D} = \left\{ \left((s, r, o), \mathcal{P}_{s,r} \right) \right\}$, where $\mathcal{P}_{s,r} = \{p_1, \cdots, p_k\}$ denotes a set of proofs that deduce the triplet. For visualized dataset samples, please refer to Appendix E.

4.2 Retriever

We consider three types of fixed, non-trainable retrieval interfaces. **(1) Sparse retrieval.** Similar to KICGPT, we consider triplets that share the same subject/object with the query. To constrain retrieval size, we only select top-*k* triplets based on the degree of their entities. **(2) Dense retrieval.** We encode triplets with text embedding models. We consider two embedding approaches: separate entity/relation embedding followed by principal neighborhood aggregation, and sentence embedding by converting triplets to natural language sentences. **(3) Hybrid retrieval.** We first perform dense retrieval, then run breadth-first-search with a predefined depth on each of retrieved proofs.

4.3 REWRITER

Problem Statement. To improve KG retrieval, we aim to train a REWRITER LLM to decompose the original query (s, r) into a set of sub-queries whose individual retrievals collectively cover the triplets in ground-truth proof. Formally, let f_θ be the REWRITER that maps the query (s, r) into a set of rewritten triplets:

$$\mathcal{Q} = f_\theta(s, r) = \big\{(s^{(1)}, r^{(1)}, o^{(1)}), \cdots, (s^{(m)}, r^{(m)}, o^{(m)})\big\}.$$

For each rewrited triplet $q_i = (s^{(i)}, r^{(i)}, o^{(i)}) \in \mathcal{Q}$, we retrieve the top-$k$ most similar KG triplets using the fixed retrieval interface detailed in Sec. 4.2, denoted as $\texttt{Retrieve}(\cdot, \cdot)$:

$$\mathcal{G} = \bigcup_{i=1}^{m} \mathcal{G}_i, \;\; \mathcal{G}_i = \texttt{Retrieve}(q_i, k).$$

We say the rewriting $\mathcal{Q}$ successfully covers the ground-truth if there exists *at least* one proof path $p \in \mathcal{P}$ such that $p \subseteq \mathcal{G}$. That is, all triplets in some valid proofs are retrieved by the composed sub-queries $\mathcal{G}$. This forms our retrieval success criterion.

Constructing minimal rewriting sets. To generate rewriting data for finetuning the REWRITER, we extract Minimal Rewriting Sets (MRS) that satisfy the proof coverage constraint discussed above. The extraction algorithm is provided in Appendix 1. For each proof path $p_j \in \mathcal{P}$, we solve a set-cover problem to find a minimal set of rewritten queries $\mathcal{Q}$ such that:

$$p_j \subseteq \text{argmin}_{|\mathcal{Q}|} \bigcup_{q \in \mathcal{Q}} \texttt{Retrieve}(q, k).$$

Since there may exist multiple possible rewriting combinations that fully cover a given proof path, we run a beam-search style iteration on top of the set-cover algorithm to obtain top-n MRS $\mathfrak{Q}^*$ for each proof. We are therefore able to obtain a much larger training dataset (as compared to the original training set size for link prediction) for REWRITER of size $|N| \cdot |\mathcal{P}| \cdot |\mathfrak{Q}^*|$, where $|N|$ is the number of training triplets, $|\mathcal{P}|$ is the number of proofs per triplet, and $|\mathfrak{Q}^*|$ is the number of minimal rewriting sets per proof. The resulting dataset is composed by query-rewriting pairs $\big((s, r), \mathcal{Q}\big)$. This expanded training set allows the REWRITER to learn a more comprehensive mapping between the query and the rewriting sets. The detailed statistics for the resulting dataset can be found in Appendix F.

Finetuning REWRITER. Given a query (s, r), we want to model the space of all valid query decompositions that lead to a successful coverage. Formally, let $\mathfrak{Q}^*$ denote the set of all valid rewrite sets $\mathcal{Q} = \{q_1, \cdots, q_m\}$ such that the retrieval result $\mathcal{G}(\mathcal{Q})$ covers at least one proof path in $\mathcal{P}_{s,r}$, that is $\mathfrak{Q}^* = \big\{\mathcal{Q} : \exists p \in \mathcal{P} \text{ s.t. } p \subseteq \mathcal{G}(\mathcal{Q})\big\}$. We want to maximize the conditional distribution:

$$\mathcal{L}_f(\theta) = -\log \sum_{\mathcal{Q} \in \mathfrak{Q}^*} f_\theta(\mathcal{Q}|s, r).$$

which is optimized with standard negative log-likelihood (NLL) loss in LLM finetuning. Sample prompt and LLM output can be found in Appendix F.

4.4 REASONER

While modern LLMs are trained on vast amounts of reasoning data, they remain ineffective for link prediction due to two factors: (1) limited robustness when retrievals contain irrelevant or noisy context (Fig. 3), and (2) insufficient understanding of the structural and relational patterns in knowledge graphs. To address both issues, we fine-tune the REASONER with Chain-of-Thought (CoT) supervision using retrieved context and extracted proofs.

Problem Statement. Given a link prediction query (s, r), and the retrieved set $\mathcal{G}$, REASONER g_ϕ predicts the missing object o after producing an intermediate CoT:

$$g_\phi : \big((s, r), \mathcal{G}\big) \mapsto \big(p, o\big), p = [(s, r^{(1)}, o^{(1)}), (o^{(1)}, r^{(2)}, o^{(2)}), \cdots, (o^{(n-1)}, r^{(n)}, o)].$$

During inference, the finetuned REASONER is essentially performing graph traversal over the subgraph provided by the RETRIEVER, by predicting the next most probable token (node).

Constructing reasoning dataset. Since each fact can be proved in multiple ways, and each proof can be supported by different rewriting sets, we expand the training data analogously to the rewriting dataset (Sec. 4.3). The resulting dataset consists of quadruplets $\big((s, r), \mathcal{G}, p, o\big)$, with size $|N| \cdot |\mathcal{P}| \cdot |\mathfrak{Q}^*|$, matching that of the rewriting training set. Since retrieval is often imperfect, we randomly drop a portion of ground-truth retrievals during REASONER fine-tuning. This encourages the model to infer missing facts from incomplete context, improving robustness to noisy or partial retrievals.

4.5 EXTENDING TO MULTI-ANSWER LINK PREDICTION

The framework described so far addresses single-answer link prediction, where the model is trained to retrieve and reason towards a single correct object o for a query (s, r). To align with the standard evaluation setting, which considers top-k answers (e.g., $k = 10$), we extend both retrieval and reasoning to support multi-answer supervision. This contrasts with prior LLM-based approaches (Yao et al., 2025), which are restricted to top-1 prediction.

Sampling top-k answers and proofs. Since for each (s, r) pair, there may be less than k ground-truth tail entity t, we use the score distribution from interpretable link predictors (the same models that generate proofs) to obtain additional candidate ts with scores above a preset threshold. Specifically, given a query (s, r), the link predictor returns a ranked list $\mathcal{O}_{\text{top-}k} = o_1, \ldots, o_k$. For each candidate o_i, we extract an associated proof set $\mathcal{P}_i = p_1^i, \ldots, p_m^i$.

Extending rewriter and reasoner. In the multi-answer setting, REWRITER outputs a set of subqueries such that the resulting retrieved triplets $\mathcal{G}(Q)$ cover at least one proof path for each candidate:

$$\forall i \in \{1, \cdots, k\}, \exists p \in \mathcal{P}_i \text{ s.t. } p \subseteq \mathcal{G}(Q).$$

Similarly, the REASONER is trained not only to produce a single object but to output a ranked list of candidates, each accompanied by its proof as chain-of-thought (CoT).

5 EXPERIMENTS

5.1 EXPERIMENT SETUP

Datasets. We evaluate PARR on FB15k-237 (Toutanova & Chen, 2015) and WN18RR (Dettmers et al., 2018), two major datasets for link prediction. We consider both transductive and inductive setting, following standard splits in (Trouillon et al., 2016) and (Teru et al., 2020). Dataset statistics can be found in Appendix E.1.

Evaluation. We evaluate with the standard HITS@k metrics: HITS@1, HITS@3 and HITS@10. We do not consider mean reciprocal recall (MRR) because we do not produce the full score distribution.

Implementation Details. We employ Llama3-8B Instruct (lla, 2024) (abbr. as Llama3) and Qwen3-8B (Yang et al., 2024) (abbr. as Qwen3) as the base LLM. We finetune using LoRA (Hu et al., 2021) with rank and alpha being 32 for one epoch. By default we use dense retrieving and sentence embedding with Jina V3 (Sturua et al., 2024) embedding model as the fixed RETRIEVER for our REWRITER model. We employ NBFNet (Zhu et al., 2021) and A*Net (Zhu et al., 2023) as the interpretable expert models. For each model, we train separately twice with different message functions and random seeds. Please refer to Appendix E.1 for full implementation details.

Baselines. We compare PARR against embedding based models such as TransE (Bordes et al., 2013), RotatE (Sun et al., 2019), ComplEx (Trouillon et al., 2016), and TuckER (Balazevic et al., 2019); GNN-based methods like CompGCN (Vashishth et al., 2020) and NBFNet (Zhu et al., 2021); path-based methods such including NeuralLP (Yang et al., 2017) and A*Net (Zhu et al., 2023); and methods that utilize pre-trained language models including KG-BERT (Yao et al., 2019), StAR (Wang et al., 2021), KGLM (Youn & Tagkopoulos, 2023), FTL-LM (Lin et al., 2023), DET (Guo et al., 2022), KG-Llama (Yao et al., 2025), KICGPT (Wei et al., 2023) and MKGL (Guo et al., 2024).

Table 1: Transductive link prediction on FB15K-237, WN18RR and NELL995. We use Llama3/Qwen3 to refer to Llama3-8B-instruct/Qwen3-8B throughout the rest of the paper. Best/2nd-best results are in **Bold**/underlined.

Model	FB15K-237			WN18RR			NELL995	
	HITS@1	HITS@3	HITS@10	HITS@1	HITS@3	HITS@10	HITS@1	HITS@10
TransE	0.218	0.345	0.495	0.061	0.366	0.522	-	-
RotatE	0.241	0.375	0.533	0.428	0.492	0.571	0.448	0.608
ComplEx+RP	0.298	0.425	0.568	0.443	0.505	0.578	-	-
TuckER	0.266	0.394	0.544	0.443	0.526	0.526	-	-
CompGCN	0.264	0.39	0.535	0.443	0.494	0.546	0.257	0.544
NeuralLP	-	-	0.362	0.371	0.434	0.566	-	-
Red-GNN	0.283	-	0.558	0.485	-	0.624	0.476	0.651
NBFNet	0.321	0.454	**0.599**	0.497	0.573	0.666	0.485	0.655
A*Net	0.321	0.453	0.586	0.495	0.573	0.659	0.479	0.652
KG-BERT	-	-	0.420	0.041	0.302	0.524	-	-
StAR	0.205	0.322	0.482	0.243	0.491	0.709	-	-
KGLM	0.200	0.314	0.468	0.330	0.538	0.741	-	-
FTL-LM	0.253	0.386	0.521	0.452	0.637	**0.773**	-	-
MKGL	0.325	0.454	0.591	0.500	0.577	0.656	-	-
KG-Llama-7b	-	-	-	0.242	-	-	-	-
GPT 3.5 Turbo	0.267	-	-	0.212	-	-	-	-
KICGPT	0.327	0.448	0.554	0.474	**0.585**	0.641	-	-
PARR-Llama3 (ours)	0.344	0.453	0.588	0.496	0.573	0.641	0.514	0.655
PARR-Qwen3 (ours)	**0.352**	**0.465**	0.593	**0.513**	0.584	0.653	**0.519**	**0.658**

Table 2: Transductive setting on YAGO3-10 dataset. Baselines are from Zhu et al. (2023).

Method	YAGO3-10		
	HITS@1	HITS@3	HITS@10
DistMult	0.24	0.38	0.54
ComplEx	0.26	0.40	0.55
RotatE	0.402	0.550	0.670
BoxE	0.400	0.472	0.541
HAKE	0.452	0.516	0.582
NFBNet	0.480	0.612	**0.708**
A*Net	0.470	0.611	0.707
KG-LLaMA-13B	0.133	-	-
PARR-Qwen3	**0.494**	**0.620**	0.691

Table 3: Performance on inductive KG reasoning on FB15k-237-ind (v1) and WN18RR-ind (v1). Baseline results are from (Zhu et al., 2021; Guo et al., 2024).

Model	FB15K-237-ind		WN18RR-ind	
	H@1	H@10	H@1	H@10
NeuralLP	0.243	0.468	0.592	0.772
DRUM	0.247	0.474	0.613	0.777
GraIL	0.302	0.483	0.653	0.769
RED-GNN	0.302	0.483	0.653	0.8
NBFNet	0.335	0.574	0.695	**0.826**
MKGL	0.400	**0.595**	0.700	0.822
ChatGPT [42]	0.288	-	0.279	-
PARR-Llama3	0.394	0.541	0.711	0.813
PARR-Qwen3	**0.412**	0.566	**0.718**	0.824

5.2 Main Results

Table 1 summarizes the results on KG link prediction under the transductive setting. PARR shows competitive performance against existing methods on both datasets. Particularly, PARR-Qwen3 outperforms previous SOTA (MKGL) on HITS@1 for both FB15K-237 and WN18RR. For instance, PARR-Qwen3 achieves 0.348 HITS@1 on FB15K-237, 2.3% above the previous SOTA (MKGL).

Table 3 shows results under the inductive setting. PARR-Qwen3 noticeably outperforms all the other methods under HITS@1 for both datasets. For example, PARR-Qwen3 scores 0.412 HITS@1 on FB15K-237-ind, surpassing previous SOTA by 1.2%. In Table 2 we show additional results on YAGO3-10 Mahdisoltani et al. (2015) with PARR-Qwen3. We can observe PARR achieves better performance on HITS@1 and HITS@3, and is comparable to SOTA methods on HITS@10.

In both transductive and inductive settings, we observe PARR achieves stronger performance on metrics with small ks, such as HITS@1 and HITS@3, while relatively fall short on HITS@10 as compared to SOTA methods such as NBFNet and MKGL. This is due to existing methods such as NBFNet and MKGL directly learn a score distribution over all entities, which is in turn more advantageous when considering large ks.

5.3 Ablation Studies

Evaluating Retrieval. In Figure 2 we show average recall and different number of retrievals per sample (controlled by top-k retrieval). We can see the REWRITER outperforms all other retrieving

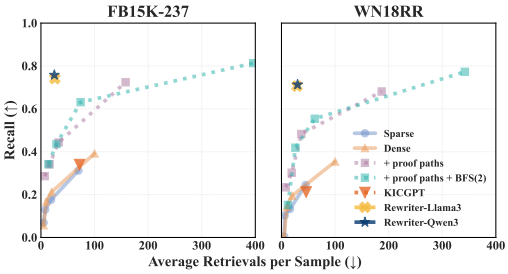
Table 4: Retrieval performance of PARR REWRITER under different top-k, and their effect on the downstream REASONER's performance. The recall is calculated as the highest among n ground-truth paths. top-k for REWRITER refers to top-k retrievals given each rewrited sub-queries. Avg. Retr. refers to average number of retrievals per sample; Avg. Rew. refers to the average number of rewrited sub-queries per sample. Avg. Retr. is caluculated from Avg. Rew. $\times$ k $\times$ Path-Lengths. Best/2nd-best results are in **Bold**/underlined.

top-k	Base LLM	Avg. Retr.	Avg. Rew.	Recall(↑)	HITS@1	HITS@3	HITS@10
			FB15K-237				
2	Llama3	23	4	0.452	0.288	0.288	0.371
2	Qwen3	20	4	0.485	0.319	0.413	0.406
5	Llama3	26	2	0.741	0.341	0.449	0.579
5	Qwen3	25	2	**0.758**	**0.348**	**0.460**	**0.586**
15	Llama3	48	1	0.724	0.312	0.426	0.534
15	Qwen3	50	1	0.743	0.325	0.437	0.548
			WN18RR				
2	Llama3	27	5	0.388	0.251	0.352	0.325
2	Qwen3	26	5	0.395	0.262	0.378	0.403
5	Llama3	29	2	0.706	0.488	0.566	0.632
5	Qwen3	30	2	**0.713**	**0.507**	**0.581**	**0.644**
15	Llama3	44	1	0.662	0.462	0.514	0.541
15	Qwen3	44	1	0.683	0.476	0.533	0.590

methods significantly in terms of Recall, while requiring much fewer number of retrievals, for both base LLMs and datasets. Meanwhile, Dense + proof paths noticeably outperform Dense without proof paths in all scenarios, demonstrating the effect of augmenting proof paths into retrieval. On the other hand, KICGPT's retrieval shows similar performance to the baseline sparse/dense RETRIEVER under our implementation, but falls significantly behind Dense + proof paths and REWRITER. In case of the non-trainable RETRIEVER, we observe dense mode to slightly outperform sparse retrieval, and hybrid RETRIEVER being marginally better than dense RETRIEVER.

Effect of Retrieval Performance on Reasoner. In Table 4 we show ablations on the effect of top-k on our REWRITER, and the effect of the retrieval performance on the REASONER LLM. We can observe best retrieval performance are achieved at $k = 5$ under all scenarios, with $k = 2$'s recall lags significantly behind. We conjecture this is because when k is low, the number of rewrited sub-queries are noticeably increased. This imposes a harder task on the REWRITER, which has to learn a much more complex mapping between the query and the rewrited sub-queries, leading to low recall. On the other hand, we can also observe the downstream REASONER's performance is directly tied to the quality of the retrieval.

Figure 2: Retrieval performance on FB15K-237 and WN18RR as a function of Average Retrievals per Sample (x-axis) and Average Recall(y-axis. + proof paths denotes dense RETRIEVER with proofs included in the retrieval. + proof paths + BFS(2) refers to the hybrid RETRIEVER with Breadth-First-Search depth of 2.



Ablations on Reasoner. We start by evaluating the inherent logical reasoning ability of LLMs. In Figure 3 we show zero-shot performance on link prediction, where ground-truth retrievals (grounding paths extracted from pretrained link predictors) are provided. To simulate noisy retrievals with false positives, we randomly sample and inject irrelevant triplets. We can see the model's performance lags largely behind traditional link predictors, even with ground-truth retrieval and no false positives (*i.e.* recall=precision=1). The model performance quickly degrade as the number of injected irrelevant triplets increase. For FB15K-237, both LLMs's HITS@1 quickly drop to near 0 with 16 false positives. This shows the need for a model with better and more robust logical reasoning capacity.

In Table 5 we show the impact of each component on the performance of the REASONER. We consider two LLM-based baselines: (1) LLM with naive finetuning (without CoT and retrieval). Similar to KG-Llama, we perform finetuning to predict missing entities, without retrieval. (2) LLM with RETRIEVER. In this setting we use the fixed RETRIEVER (Sec. 4.1).

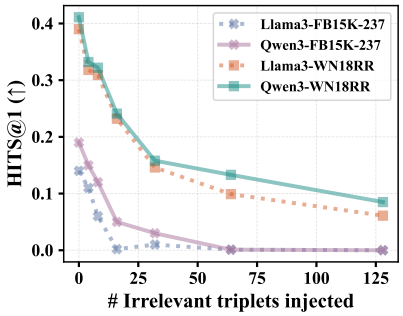
Table 5: Ablations on the performance of REASONER with respect to each component: Chain of Thought (**CoT**), retrieval method (**Retr.**), and **Retr. Dropout**: whether to apply random dropout on ground-truth retrievals while finetuning the REASONER. Dense refers to fixed dense retrieval with $k = 50$. REW refers to our fine-tuned REWRITER. H@k refers to HITS@k.

Method	CoT	Retr.	Retr. Dropout	FB15K-237			WN18RR		
				H@1	H@3	H@10	H@1	H@3	H@10
NBFNet	-	-	-	0.321	0.454	**0.599**	0.497	0.573	**0.666**
KICGPT	-	-	-	0.327	0.448	0.554	0.474	**0.585**	0.641
KG-Llama-7B	-	-	-	-	-	-	0.242	-	-
Llama3	×	×	-	0.077	-	-	0.171	-	-
Qwen3	×	×	-	0.085	-	-	0.184	-	-
Llama3	×	Dense	-	0.143	-	-	0.279	-	-
Qwen3	×	Dense	-	0.155	-	-	0.285	-	-
PARR-Llama3	×	REW	×	0.293	0.395	0.412	0.451	0.413	0.479
PARR-Qwen3	×	REW	×	0.302	0.388	0.48	0.467	0.499	0.544
PARR-Llama3	✓	REW	×	0.316	0.405	0.534	0.479	0.491	0.538
PARR-Qwen3	✓	REW	×	0.335	0.413	0.521	0.493	0.524	0.603
PARR-Llama3	✓	REW	✓	0.341	0.449	0.579	0.488	0.566	0.632
PARR-Qwen3	✓	REW	✓	**0.348**	**0.460**	0.586	**0.507**	0.581	0.644

Table 6: Ablation on MoE effect for RETRIEVER recall and REASONER's HITS@1 on FB15k-237 and WN18RR.

	Retriever (Recall)		Reasoner (HITS@1)	
w/ MoE	FB15k-237	WN18RR	FB15k-237	WN18RR
×	0.683	0.629	0.324	0.456
✓	0.741	0.706	0.341	0.488

Figure 3: Zero-Shot performance on FB15k-237 and WN18RR. We provide ground-truth retrievals in the context with varying amount of randomly sampled irrelevant triplets, simulating the false positive retrievals.



We can first observe that naive finetuning (*i.e.* directly finetuning using training triplets without retrieval or CoT) results in significantly lower accuracy in all scenarios. For instance, Llama3 only achieves 0.077 HITS@1 on FB15K-237, and 0.171 on WN18RR. With dense RETRIEVER included, the accuracy for both datasets are considerably improved, but still noticeably lag behind other models.

At the bottom of Table 5 we show each component's effect on the performance of REASONER. We observe improved performance with CoT and retrieval dropout included during training. The full PARR performs significantly better than LLM baselines, and is on par or better with existing SOTA methods.

5.4 ABLATION ON MOE LINK PREDICTORS

To understand the effect of sampling proofs from MoE link predictors (Sec. 4.1), we conduct ablation study on the effect of using MoE-sampled proofs versus proof sampled from a single link predictor. As shown in Tab. 6, we can observe that with MoE we can achieve notable improvement for both retrieval and link prediction accuracy, with over 6% improvement on recall, and 2% on HITS@1.

6 CONCLUSION

In this paper we introduce Proof-Augmented Retrieval and Reasoning (PARR) for KG completion. We effectively leverage proofs extracted from interpretable link predictors such as NBFNet and A*Net to (1) augment the RAG database for better sub-graph retrieval, (2) serve as golden retrieval for supervising a REWRITER LLM for query rewriting, and (3) supervise a REASONER LLM as CoT data. By experimenting on different datasets and task settings, we show PARR achieves competitive performance compared to SOTA link prediction models. Finally, we conduct extensive ablations to examine the effect and performance of each component of PARR.

Ethics Statement

We reviewed the ICLR Code of Ethics carefully and do not observe potential concerns for our work.

Reproducibility Statement

We made our best efforts to comprehensively document the implementation details. Training hyperparameters and model architectures are discussed in Section 5.1. We include the dataset construction details including all the example prompts we used in Section E.1 and Section F.

References

The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Mueller. How to explain individual classification decisions, 2009. URL https://arxiv.org/abs/0912.1128.

Ivana Balazevic, Carl Allen, and Timothy Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 2019. doi: 10.18653/v1/d19-1522. URL http://dx.doi.org/10.18653/v1/D19-1522.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pp. 2787–2795, Red Hook, NY, USA, 2013. Curran Associates Inc.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens, 2022. URL https://arxiv.org/abs/2112.04426.

Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. Hitter: Hierarchical transformers for knowledge graph embeddings, 2021. URL https://arxiv.org/abs/2008.12813.

Xuanming Cui, Chionh Wei Peng, Adriel Kuek, and Ser-Nam Lim. Improving soft unification with knowledge graph embedding methods, 2025. URL https://openreview.net/forum?id=OOqvY9yvVG.

Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Robin Jia, Manzil Zaheer, Hannaneh Hajishirzi, and Andrew McCallum. Knowledge base question answering by case-based reasoning over subgraphs, 2022. URL https://arxiv.org/abs/2202.10610.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings, 2018. URL https://arxiv.org/abs/1707.01476.

Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning, 2024. URL https://arxiv.org/abs/2310.04562.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels, 2022. URL https://arxiv.org/abs/2212.10496.

Lingbing Guo, Qiang Zhang, and Huajun Chen. Unleashing the power of transformer for graphs, 2022. URL https://arxiv.org/abs/2202.10581.

Lingbing Guo, Zhongpu Bo, Zhuo Chen, Yichi Zhang, Jiaoyan Chen, Yarong Lan, Mengshu Sun, Zhiqiang Zhang, Yangyifei Luo, Qian Li, Qiang Zhang, Wen Zhang, and Huajun Chen. Mkgl: Mastery of a three-word language, 2024. URL https://arxiv.org/abs/2410.07526.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020. URL https://arxiv.org/abs/2002.08909.

Sheryl Hsu, Omar Khattab, Chelsea Finn, and Archit Sharma. Grounding by trying: LLMs with reinforcement learning-enhanced retrieval. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=BPAZ6yW3K7.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering, 2021. URL https://arxiv.org/abs/2007.01282.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models, 2022. URL https://arxiv.org/abs/2208.03299.

Pengcheng Jiang, Lang Cao, Cao Xiao, Parminder Bhatia, Jimeng Sun, and Jiawei Han. KG-FIT: Knowledge graph fine-tuning upon open-world knowledge. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=rDoPMODpki.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Qika Lin, Rui Mao, Jun Liu, Fangzhi Xu, and Erik Cambria. Fusing topology contexts and logical rules in language models for knowledge graph completion. *Inf. Fusion*, 90(C):253–264, February 2023. ISSN 1566-2535. doi: 10.1016/j.inffus.2022.09.020. URL https://doi.org/10.1016/j.inffus.2022.09.020.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning, 2024. URL https://arxiv.org/abs/2310.01061.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models, 2023. URL https://arxiv.org/abs/2305.14283.

Farzaneh Mahdisoltani, Joanna Asia Biega, and Fabian M. Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *Conference on Innovative Data Systems Research*, 2015. URL https://api.semanticscholar.org/CorpusID:6611164.

Shengyu Mao, Yong Jiang, Boli Chen, Xiao Li, Peng Wang, Xinyu Wang, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. RaFe: Ranking feedback improves query rewriting for RAG. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 884–901, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.49. URL https://aclanthology.org/2024.findings-emnlp.49/.

Costas Mavromatis and George Karypis. Gnn-rag: Graph neural retrieval for large language model reasoning, 2024. URL https://arxiv.org/abs/2405.20139.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving, 2017. URL https://arxiv.org/abs/1705.11040.

Saba Sturua, Isabelle Mohr, Mohammad Kalim Akram, Michael Günther, Bo Wang, Markus Krimmel, Feng Wang, Georgios Mastrapas, Andreas Koukounas, Nan Wang, and Han Xiao. jina-embeddings-v3: Multilingual embeddings with task lora, 2024. URL https://arxiv.org/abs/2409.10173.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space, 2019. URL https://arxiv.org/abs/1902.10197.

Komal K. Teru, Etienne Denis, and William L. Hamilton. Inductive relation prediction by subgraph reasoning, 2020. URL https://arxiv.org/abs/1911.06962.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In Alexandre Allauzen, Edward Grefenstette, Karl Moritz Hermann, Hugo Larochelle, and Scott Wen-tau Yih (eds.), *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 57–66, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-4007. URL https://aclanthology.org/W15-4007/.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction, 2016. URL https://arxiv.org/abs/1606.06357.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BylA_C4tPr.

Bo Wang, Tao Shen, Guodong Long, Tianyi Zhou, Ying Wang, and Yi Chang. Structure-augmented text representation learning for efficient knowledge graph completion. In *Proceedings of the Web Conference 2021*, WWW '21, pp. 1737–1748. ACM, April 2021. doi: 10.1145/3442381.3450043. URL http://dx.doi.org/10.1145/3442381.3450043.

Yanbin Wei, Qiushi Huang, Yu Zhang, and James Kwok. Kicgpt: Large language model with knowledge in context for knowledge graph completion. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 8667–8683. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.580. URL http://dx.doi.org/10.18653/v1/2023.findings-emnlp.580.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning, 2017. URL https://arxiv.org/abs/1702.08367.

Liang Yao, Chengsheng Mao, and Yuan Luo. Kg-bert: Bert for knowledge graph completion, 2019. URL https://arxiv.org/abs/1909.03193.

Liang Yao, Jiazhen Peng, Chengsheng Mao, and Yuan Luo. Exploring large language models for knowledge graph completion, 2025. URL https://arxiv.org/abs/2308.13916.

Jason Youn and Ilias Tagkopoulos. Kglm: Integrating knowledge graph structure in language models for link prediction, 2023. URL https://arxiv.org/abs/2211.02744.

Yuqi Zhu, Xiaohan Wang, Jing Chen, Shuofei Qiao, Yixin Ou, Yunzhi Yao, Shumin Deng, Huajun Chen, and Ningyu Zhang. Llms for knowledge graph construction and reasoning: Recent capabilities and future opportunities, 2024. URL https://arxiv.org/abs/2305.13168.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhaocheng Zhu, Xinyu Yuan, Mikhail Galkin, Sophie Xhonneux, Ming Zhang, Maxime Gazeau, and Jian Tang. A*net: A scalable path-based reasoning approach for knowledge graphs, 2023. URL https://arxiv.org/abs/2206.04798.

Table 7: Training time comparison using default settings for each model.

Model	WN18RR		FB15k-237		YAGO	
	Time (hr)	Max GPU (G)	Time (hr)	Max GPU (G)	Time (hr)	Max GPU (G)
NBFNet	21	27	32	19	1141	27
AstarNet	15	5	29	14	503	15
MKGL (Llama2-7B)	20	58	60	67	3789	54
PARR (Llama3-8B)	73	75	126	68	923	72

Table 8: Evaluation throughput (seconds per batch) and maximum GPU memory allocation (in GB).

Model	WN18RR		FB15k-237		YAGO	
	s/batch	Max GPU (G)	s/batch	Max GPU (G)	s/batch	Max GPU (G)
NBFNet	1.3	1	1.4	0.6	3.2	5
AstarNet	1.3	0.4	1.4	0.6	1.7	4
MKGL	3.1	39	4.6	28	5.6	31
KICGPT	158	–	176	–	–	–
PARR (Llama3-8B) + vLLM	3.2	74	3.5	74	3.3	74

A Computation Analysis

Here we profile the training/evaluation's peak GPU memory consumption and time spent for PARR and SOTA methods. All experiments are run on the same compute node with one Nvidia A100 GPU. Due to time limit, we estimate training time by average time per step $\times$ steps per epoch $\times$ n epochs using parameters from models' default configs. For evaluation, we measure the average time (in second) spent for one batch with batch size equals to 8.

Justification on the system complexity. While the overall PARR framework exhibits more complexity than traditional link predictors, we would like to note:

(1) PARR targets a fundamentally different use case than traditional link predictors. While traditional link predictors such as KGE models are discriminative classifiers, PARR is built towards a generative, conversational KG agent. This means PARR must model the full distribution of the KG over natural language's space, which is substantially more challenging than training a discriminative KGE with a binary classification loss.

(2) PARR offers significantly better inference efficiency than prompt-based LLM approaches. Unlike recent SOTA prompt-based LLM KGC method such as KICGPT (Wei et al., 2023), which require repeated calls to large proprietary LLMs, PARR is much more efficient at inference time. As shown in Table 8, on FB15k-237, KICGPT takes 176 seconds to evaluate 8 samples, whereas PARR takes only 3.5 seconds—with better accuracy.

(3) PARR is scalable. GNN-based link predictors often incur at least polynomial (exponential for exhaustive path search algorithms like PathCon) time/memory complexity w.r.t. to the number of entities, edges, and embedding dimensions. In contrast, both Rewriter and Reasoner in PARR have near constant time and memory consumption, irrespective of graph size. The Retriever can also achieve constant or logarithmic time complexity via precomputed retrieval tables or fast retrieval libraries like FAISS (Johnson et al., 2019). As shown in Tab. 7, on YAGO3-10, MKGL requires 3700 GPU hours, while PARR achieves better performance with only 920 GPU hours.

(4) Heavy data curation, straightforward deployment. Despite the complexity of PARR, more than half of the efforts are for data curation. This, however, has become a normal practice in the era of LLMs, where data curation has become the most important and time-consuming process. On the other hand, PARR is straightforward to deploy in real-world, thanks to the development in LLM acceleration tools (e.g. vLLM) and scalable retrieval index (e.g. FAISS).

Table 9: Training with Llama3-8B (total hours spent).

Dataset	Rewriter	Retriever	Reasoner
WN18RR	15	–	58
FB15k-237	23	–	103
YAGO3-10	88	–	835

Table 10: Evaluation with Llama3-8B (seconds per batch, with vLLM).

Dataset	Rewriter	Retriever	Reasoner
WN18RR	1.3	–	1.9
FB15k-237	1.3	–	2.2
YAGO3-10	1.3	–	2.0

Table 11: Performance on inductive knowledge graph reasoning. V1-v4 refer to the 4 standard splits.

Method	v1		v2		v3		v4	
	HITS@1	HITS@10	HITS@1	HITS@10	HITS@1	HITS@10	HITS@1	HITS@10
FB15k-237								
GraIL	0.205	0.429	0.202	0.424	0.165	0.424	0.143	0.389
NeuralLP	0.243	0.468	0.286	0.586	0.309	0.571	0.289	0.593
DRUM	0.247	0.474	0.284	0.595	0.308	0.571	0.309	0.593
NBFNet	0.335	0.574	0.421	0.685	0.384	0.637	0.360	0.627
RED-GNN	0.302	0.483	0.381	0.629	0.351	0.603	0.340	0.621
A*Net	0.381	0.589	0.419	0.672	0.389	0.629	0.365	**0.645**
MKGL	0.400	**0.595**	0.417	0.681	0.392	**0.643**	0.374	**0.645**
PARR-Qwen3	**0.412**	0.586	**0.433**	**0.688**	**0.397**	0.633	**0.386**	0.633
WN18RR								
GraIL	0.554	0.760	0.542	0.776	0.278	0.409	0.443	0.687
NeuralLP	0.592	0.772	0.575	0.749	0.304	0.476	0.583	0.706
DRUM	0.613	0.777	0.595	0.747	0.330	0.477	0.586	0.702
NBFNet	0.695	**0.826**	0.651	0.798	0.392	0.568	0.608	0.694
RED-GNN	0.653	0.799	0.633	0.780	0.368	0.524	0.606	0.721
A*Net	0.682	0.810	0.649	**0.803**	0.386	0.544	0.616	**0.743**
MKGL	0.700	0.822	0.662	0.799	0.406	0.559	**0.620**	0.741
PARR-Qwen3	**0.718**	0.824	**0.675**	0.788	**0.414**	**0.572**	**0.627**	0.725

B More results

In Table 11 we show full results on the inductive settings for FB15k-237 and WN18RR. Given the better performance of Qwen3 over Llama3, we only run experiments based on Qwen3. We can see PARR achieves better or comparable performance on all the splits for both FB15k-237 and WN18RR, demonstrating its strong generalizability over unseen entities.

We further evaluate PARR on ogbl-wikikg2, a large-scale knowledge graph derived from Wikidata. The results, reported in terms of MRR, are shown in Table 12. Notably, PARR attains the strongest overall performance, despite representing a lower bound on MRR (because PARR predicts only the top-10 candidate entities, any correct entity ranked outside the top-10 receives zero reciprocal rank). Still, PARR surpasses all baselines, demonstrating the effectiveness and scalability of our proof-augmented retrieval and reasoning framework on large KGs.

To further demonstrate robustness across graph sizes and relational structures, we also evaluate on three small but widely used KGs—Kinship, Nations, and UMLS. As shown in Table 13, PARR consistently outperforms all existing systems, achieving new state-of-the-art results on all datasets across both HITS@1 and HITS@10 metrics.

Table 12: MRR results on `ogbl-wikikg2`. PARR achieves the best performance despite representing a lower bound, as it only predicts top-10 entities.

	PARR-Qwen3	TransE	ComplEx	RotatE	PairRE	ComplEx+RP	A*Net
MRR	**0.7013**	0.4256	0.4027	0.4332	0.5208	0.6392	0.6767

Table 13: Results on three small, statistical KGs: Kinship, UMLS, and Nations. PARR achieves state-of-the-art performance across all datasets.

Method	Kinship		UMLS		Nations	
	H@1	H@10	H@1	H@10	H@1	H@10
NeuralLP	0.475	0.912	0.643	0.862	–	–
MINERVA	0.605	0.924	0.728	0.968	–	–
DRUM	0.367	0.885	0.546	0.935	–	–
NBFNet	0.632	0.966	0.721	0.971	0.633	0.951
LERP	0.500	0.931	0.646	0.942	–	–
PARR-Qwen3	**0.656**	**0.973**	**0.748**	**0.983**	**0.672**	**0.960**

C More Analysis

C.1 Ablation on the robustness of PARR on Noisy Proofs

Here we conduct a robustness analysis on how variations in proof quality (e.g., using weaker or partial proofs) could affect end-to-end link prediction performance. To do so, we add $n\%$ of noisy proof (proofs with bottom 5% path scores) to the retrieval. In Tab. 14 we show the performance (HIT@1) of Reasoner (LLama3-8B) during inference time, where the retrieval is randomly perturbed by $n\%$. We can see the Reasoner maintains decent performance (e.g. less than 0.015% drop) even when perturbation rate is 25% for both datasets.

Table 14: Reasoner performance (HITS@1) under different perturbation ratios.

Dataset	**0%**	**5%**	**10%**	**25%**	**50%**
FB15k-237	0.341	0.340	0.334	0.325	0.296
WN18RR	0.488	0.487	0.482	0.476	0.436

C.2 Decoupling Pretrained Link Predictors from Inference

To further improve test-time efficiency, we evaluate a variant of PARR in which no proofs are used during retrieval (PARR-Qwen3 w/o proofs). This removes all dependence on pre-trained link predictors during inference, making PARR lighter while preserving the training benefits of proof supervision. As shown in Table 15, this variant yields only a minor drop in performance on both FB15k-237 and WN18RR, demonstrating that PARR remains highly effective even without proof-based retrieval at test time.

Table 15: Ablation on test-time proof usage. Removing proofs during retrieval (PARR-Qwen3 w/o proofs) leads to only minimal degradation.

Method	FB15k-237	WN18RR
ComplEx+RP	0.298	0.443
NBFNet	0.321	0.497
MKGL	0.325	0.500
PARR-Qwen3	**0.352**	**0.513**
PARR-Qwen3 (w/o proofs)	0.346	0.508

C.3 Convergence of PARR

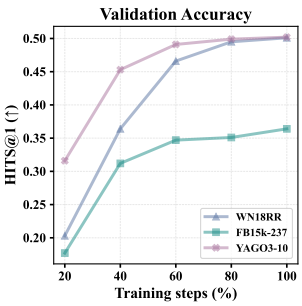


Figure 4: Validation accuracy (HITS@1) on 100 samples sampled from the validation set.

As shown in Figure 4, we observe that the PARR reasoner converges rapidly, particularly on larger KGs such as YAGO3-10, where substantial overlap arises between the constructed rewriting and reasoning sets. This property enables an efficient trade-off between training time and final performance: the model can be trained for significantly fewer steps with only minimal degradation.

D CONSTRUCTING MINIMAL REWRITING SETS

Given a proof and a fixed retrieval interface RETRIEVER, we want to find a minimal set of rewrited sub-queries that, after each individual retrieval, can cover the proof. We solve this with a set-cover algorithm. Moreover, since there may exist multiple (semi-)minimal rewriting sets that cover a proof, we apply a beam-search style iteration on top of the set-cover algorithm. Algorithm 1 shows the detailed procedure.

Algorithm 1 Find N Minimal Rewriting Sets

Require: Target triplet $\mathcal{T}$; target proof p; precomputed retrievals mapping given triplets: $\mathcal{M}$; inverse of retrieval mapping $\mathcal{M}^{-1}$; maximum number of solutions n; beam width w
Ensure: A set of rewriting sets $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_n\}$, each of which covers p

Initialize beam list: $\mathcal{B} \leftarrow \{(\emptyset, p)\}$
Initialize solution set: $\mathcal{S} \leftarrow [\,]$
Initialize seen set: $s \leftarrow \emptyset$
while $\mathcal{B} \neq \emptyset$ **and** $|\mathcal{S}| < n$ **do**
  $\mathcal{B} \leftarrow [\,]$
  for all $(\mathcal{Q}, \mathcal{U}) \in \mathcal{B}$ **do**
    if $\mathcal{U} = \emptyset$ **then**
      $k \leftarrow \texttt{sorted}(\mathcal{Q})$
      if $k \notin s$ **then**
        $s \leftarrow s \cup \{k\}$
        $S \leftarrow s \cup \{\mathcal{Q}\}$
      end if
      continue
    end if
    $r \leftarrow \emptyset$ ▷ Gather relevant triplets r that will cover part of the remaining proof
    for all $q \in \mathcal{U}$ **do**
      $r \leftarrow r \cup \mathcal{M}^{-1}[q]$
    end for
    $r \leftarrow r \setminus \mathcal{Q} \setminus \{\mathcal{T}\}$
    $\texttt{Scored} \leftarrow [\,]$ ▷ Greedily score candidate queries by how many remaining they cover
    for all $q \in r$ **do**
      $\mathcal{C}_q \leftarrow \mathcal{M}[q] \cap \mathcal{U}$
      if $\mathcal{C}_q \neq \emptyset$ **then**
        Append $(|\mathcal{C}_q|, q, \mathcal{C}_q)$ to $\texttt{Scored}$
      end if
    end for
    $\texttt{Scored} \leftarrow \texttt{sorted}(\texttt{Scored}, \text{reverse=True})$
    for $i = 1$ **to** $\min(w, |\texttt{Scored}|)$ **do**
      $(\_, q, \mathcal{C}_q) \leftarrow \texttt{Scored}[i]$
      $\mathcal{Q}_{\text{new}} \leftarrow \mathcal{Q} \cup \{q\}$ ▷ Extend query set
      $\mathcal{U}_{\text{new}} \leftarrow \mathcal{U} \setminus \mathcal{C}_q$
      Append $(\mathcal{Q}_{\text{new}}, \mathcal{U}_{\text{new}})$ to $\mathcal{B}$
    end for
  end for
  $\mathcal{B} \leftarrow \mathcal{B}$
end while
return $\mathcal{S}$

E DATASETS AND IMPLEMENTATIONS DETAILS

E.1 DATASET STATISTICS

In Table 16 and 17 we show dataset statistics for the transductive and inductive setting. We follow the standard splits in their original works.

Table 16: Dataset statistics for the transductive setting. # Rewriter Train denotes the number of training samples used for training REWRITER (and the REASONER).

Dataset	#Entity	#Relation	# Train	# Validation	# Test	# Rewriter Train	Avg. Rewrites
FB15k-237	14,541	237	272,115	17,535	20,466	3,682,046	4
WN18RR	40,943	11	86,835	3,034	3,134	1,149,304	2
YAGO3-10	123,182	37	1,079,040	5000	5000	16,628,192	5

Table 17: Dataset statistics for the inductive setting.

Dataset	**# Relation**	**Train**		**Valid**			**Test**		
		# Entity	# Triplet	# Entity	# Evaluation	# Fact	# Entity	# Evaluation	# Fact
FB15k-237-ind-v1	180	1,594	4,245	1,594	489	4,245	1,093	205	1,993
FB15k-237-ind-v2	200	2,608	9,739	2,608	1,166	9,739	1,660	478	4,145
FB15k-237-ind-v3	215	3,668	17,986	3,668	2,194	17,986	2,501	865	7,406
FB15k-237-ind-v4	219	4,707	27,203	4,707	3,352	27,203	3,051	1,424	11,714
WN18RR-ind-v1	9	2,746	5,410	2,746	630	5,410	922	188	1,618
WN18RR-ind-v2	10	6,954	15,262	6,954	1,838	15,262	2,757	441	4,011
WN18RR-ind-v3	11	12,078	25,901	12,078	3,097	25,901	5,084	605	6,327
WN18RR-ind-v4	9	3,861	7,940	3,861	934	7,940	7,084	1,429	12,334

Table 18: Dataset samples of the rewriting training dataset for WN18RR. Entity in red refers to the target entity, which is missing from the query.

Query	Rewrited sub-queries
(subdivision pinophytina, hypernym, class)	*(order coniferales, (inverse) member meronym, subdivision coniferophytina), (subdivision coniferophytina, hypernym, class)*
(nippon, has part, nagasaki)	*(volcano islands, instance hypernym, archipelago), (kyushu, has part, nagasaki), (pacific ocean, has part, volcano islands)*
(enfeeble, hypernym, weaken)	*(weaken, (inverse) hypernym, nullify), (enfeeble, drf, exhaustion)*

Table 19: Dataset samples of the REASONERs training dataset for WN18RR. *drf* refers to *Derivationally Related Form*.

Query	**Retrievals**	**Target**	**Proof**
(screen, hypernym, ?)	*(hood, hypernym, protective covering), (motorcar, has part, hood), (plane, has part, hood) (windshield, (inverse) has part, plane), (screen, (inverse) hypernym, windshield), ..., (hood, (inverse) has part, motorcar)*	*protective covering*	*(hood, hypernym, protective covering) → (hood, (inverse) has part, motorcar) → (screen, (inverse) hypernym, windshield) → (windshield, (inverse) has part, plane) → (plane, has part, hood) → (motorcar, has part, hood)*
(stick in, drf, ?)	*(inclosure, (inverse) drf, shut in), (stick in, (inverse) drf, inclosure), (inclosure, drf, stick in), ..., (stick in, (inverse) drf, inset)*	*inset*	*(stick in, (inverse) drf, inclosure) →(inclosure, drf, stick in) →(stick in, (inverse) drf, inset)*
(carry, verb group, ?)	*(carry, hypernym, make up), (make up, (inverse) hypernym, carry), (carry, (inverse) verb group, carry), (right, also see, make up), ..., (make up, (inverse) also see, right)*	*carry*	*(carry, hypernym, make up) →(make up, (inverse) hypernym, carry) →(carry, (inverse) verb group, carry) →(right, also see, make up) →(make up, (inverse) also see, right)*

Table 20: Hyper-parameters for supervised finetuning on REWRITER and REASONER, for both Llama3-8B Instruct and Qwen3-8B.

	Dataset	LoRA Rank/Alpha	Epochs	Learning Rate	Batch Size	Gradient Accumulation Steps	Optimizer
REWRITER	FB15k-237	32	1	$1e^{-5}$	8	2	AdamW
	WN18RR	32	1	$1e^{-5}$	8	2	AdamW
	YAGO3-10	32	1	$1e^{-5}$	8	2	AdamW
REASONER	FB15k-237	32	1	$1e^{-5}$	2	8	AdamW
	WN18RR	32	1	$1e^{-5}$	2	8	AdamW
	YAGO3-10	32	1	$1e^{-5}$	2	8	AdamW

E.2 DATASET SAMPLES

In Table 18 and 19 we show sample data from the rewriting dataset and reasoning dataset from WN18RR Dettmers et al. (2018).

E.3 IMPLEMENTATION DETAILS

Table 21: Hyperparameters for NBFNet and A*Net on all datasets. We use the same set of hyper-parameters for NBFNet Zhu et al. (2021) and A*Net Zhu et al. (2023), except for parameters **Priority Function**, which are only used for A*Net.

Hyperparameter		**FB15k-237**		**WN18RR**		**YAGO3-10**
		transductive	inductive	transductive	inductive	transductive
Message Passing	#step (T)	6	6	6	6	6
	hidden dim.	32	32	32	32	32
	message	DistMult	DistMult	DistMult	DistMult	DistMult
	aggregation	PNA	sum	PNA	sum	PNA
Priority Function	$g(\cdot)$ #layer	1	1	1	1	1
	$f(\cdot)$ #layer	2	2	2	2	2
	hidden dim.	64	64	64	64	64
	node ratio α	10%	50%	10%	5%	10%
	degree ratio β	100%	100%	100%	100%	100%
Learning	optimizer	Adam	Adam	Adam	Adam	Adam
	batch size	256	256	256	256	40
	learning rate	$5e^{-3}$	$5e^{-3}$	$5e^{-3}$	$5e^{-3}$	$5e^{-3}$
	#epoch	20	20	20	20	0.4
	adv. temperature	0.5	0.5	1	1	0.5
	#negative	32	32	32	32	32

Proof Extraction. We utilize NBFNet and A*Net in this work to extract proofs. Hyperparameters are shown in Table 21. By default we adopt the same hyperparameters as original works.

RETRIEVER. For dense and hybrid retriever, we use Jina-V3 Sturua et al. (2024) text embedding model to compute and store the normalized embeddings, and retrieve using FAISS Johnson et al. (2019) with IndexFlatIP index, which is equivalent to the cosine-similarity metric given pre-normalized embeddings. For dense retrieval, we observe similar performance when using Principle Neighboring Aggregation (PNA) and sentence embedding for triplet embedding calculation. For consistency, we use the latter throughout the paper.

Supervised Finetuning. We conduct experiments on Llama3-8B Instruct lla (2024) and Qwen3-8B Yang et al. (2024), two state-of-the-art LLMs. We show hyper-parameters in Table 20. We use the same set of hyper-parameters between Llama3-8B and Qwen3-8B. Experiments for FB15K-237 and WN18RR are run on 4 Nvidia H100 GPUs, and experiments for YAGO3-10 are run on 8 Nvidia H100 GPUs.

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a knowledgeable assistant that performs query rewriting for retrieving relevant context for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation,?), you will rewrite (expand) the query into a list of triplets. Each of the expanded triplets is then used to retrieve relevant triplets from a database. Your goal is to provide a list of rewrited triplets that, after retrieval, will provide complete information needed for proving the original query, while keeping the number of rewrited triplets minimum.
<leot_idl><|start_header_idl>user<|end_header_idl>

Query:
(veratrum, hypernym,?)

Please expand the query into a python list of triplets in the form of [(head, relation, tail),...]. Please strictly follow the format of the output as it will be parsed by a program.<leot_idl><|start_header_idl>assistant<|end_header_idl>

Answer:[('polianthes', 'hypernym', 'liliid monocot genus'), ('hellebore', '(inverse) member meronym','veratrum')]<leot_idl>

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a knowledgeable assistant that performs query rewriting for retrieving relevant context for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation,?), you will rewrite (expand) the query into a list of triplets. Each of the expanded triplets is then used to retrieve relevant triplets from a database. Your goal is to provide a list of rewrited triplets that, after retrieval, will provide complete information needed for proving the original query, while keeping the number of rewrited triplets minimum.
<leot_idl><|start_header_idl>user<|end_header_idl>

Query:
(family compositae, member meronym,?)

Please expand the query into a python list of triplets in the form of [(head, relation, tail),...]. Please strictly follow the format of the output as it will be parsed by a program.<leot_idl><|start_header_idl>assistant<|end_header_idl>

Answer:[('vegetable oyster', 'has part','vegetable oyster'), ('genus ageratum', '(inverse) member meronym', 'family compositae')]<leot_idl>

Figure 5: Sample prompt and answer(in dark green) for Llama3-based REWRITER.

F PROMPT SAMPLES

In Figure 5 and 7 we show sample prompt and answer from REWRITER LLM for WN18RR. In Figure 6 and Figure 9 we show sample inputs and generated outputs from REASONER LLMs, respectively.

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a knowledgeable assistant that performs link prediction for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), and a retrieved set of relevant triplets, each with the corresponding reasoning steps that prove the triplet using other triplets from the database, you will first provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.
<leot_idl><|start_header_idl>user<|end_header_idl>

Query:
(carry, verb group, ?)

Retrieved triplets:
('land reform', 'also see', 'land reform'), ('psychopathology', '(inverse) synset domain topic of', 'defense reaction'), ('offset', 'derivationally related form', 'make up'), ('make up', '(inverse) also see', 'even out'), ('compensation', 'derivationally related form', 'overcompensate'), ('carry', 'hypernym', 'make up'), ('trim', 'hypernym', 'equilibrize'), ('carry', 'verb group', 'carry'), ('make up', 'hypernym', 'equilibrize'), ('make up', '(inverse) hypernym', 'carry'), ('even out', 'also see', 'make up'), ('carry', '(inverse) verb group', 'carry'), ('equalizer', 'derivationally related form', 'make up'), ('defense reaction', '(inverse) hypernym', 'compensation'), ('make up', '(inverse) derivationally related form', 'equalizer'), ('overcompensation', 'synset domain topic of', 'psychopathology'), ('overcompensate', 'derivationally related form', 'overcompensation'), ('right', 'also see', 'make up'), ('overcompensate', 'hypernym', 'make up'), ('right', 'derivationally related form', 'recompense'), ('make up', '(inverse) also see', 'right'), ('make up', '(inverse) hypernym', 'overcompensate'), ('make up', '(inverse) derivationally related form', 'offset'), ('recompense', '(inverse) derivationally related form', 'make up')

What is an appropriate object entity that completes the query? Please provide a rigorous reasoning process before the final answer.<leot_idl>
<|start_header_idl>assistant<|end_header_idl>

Reasoning:
Based on the retrieved context, we have:
('carry', 'hypernym', 'make up') -> ('make up', '(inverse) hypernym', 'carry') -> ('carry', '(inverse) verb group', 'carry') -> ('right', 'also see', 'make up') -> ('make up', '(inverse) also see', 'right')

Therefore, we know:
('carry', 'verb group', 'carry')

Answer:carry<leot_idl>

<|begin_of_text|><|start_header_id|>system<|end_header_id|>

You are a knowledgeable assistant that performs link prediction for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), and a retrieved set of relevant triplets, each with the corresponding reasoning steps that prove the triplet using other triplets from the database, you will first provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.
<leot_idl><|start_header_idl>user<|end_header_idl>

Query:
(thread, hypernym, ?)

Retrieved triplets:
('threader', 'derivationally related form', 'thread'), ('run', '(inverse) verb group', 'thread')]
{('yarn', 'derivationally related form', 'thread'), ('thread', 'verb group', 'run'), ('run', 'also see', 'wipe'), ('run', 'verb group', 'run'), ('thread', '(inverse) derivationally related form', 'threader'), ('threader', 'derivationally related form', 'thread'), ('thread', '(inverse) derivationally related form', 'yarn'), ('threader', '(inverse) derivationally related form', 'thread'), ('thread', 'derivationally related form', 'threader'), ('run', '(inverse) verb group', 'thread'), ('wipe', '(inverse) also see', 'run'), ('thread', 'derivationally related form', 'yarn'), ('run', 'verb group', 'thread'), ('thread', '(inverse) verb group', 'run'), ('run', '(inverse) verb group', 'run')

What is an appropriate object entity that completes the query? Please provide a rigorous reasoning process before the final answer.<leot_idl>
<|start_header_idl>assistant<|end_header_idl>

Reasoning:
Based on the retrieved context, we have:
('thread', '(inverse) derivationally related form', 'threader') -> ('thread', '(inverse) verb group', 'run') -> ('threader', 'derivationally related form', 'thread')

Therefore, we know:
('thread', 'hypernym', 'run')
</think>

Answer:run<leot_idl>

Figure 6: Sample prompt and answer(in dark green) for Llama3-based REASONER.

<|im_start|>system
You are a knowledgeable assistant that performs query rewriting for retrieving relevant context for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), you will rewrite (expand) the query into a list of triplets. Each of the expanded triplets is then used to retrieve relevant triplets from a database. Your goal is to provide a list of rewrited triplets that, after retrieval, will provide complete information needed for proving the original query, while keeping the number of rewrited triplets minimum.
<lim_endl>
<lim_startl>user
Query:
(capitalization, (inverse) derivationally related form, ?)

Please expand the query into a python list of triplets in the form of [(head, relation, tail), ...]. Please strictly follow the format of the output as it will be parsed by a program.<lim_endl>
<lim_startl>assistant
<think>

</think>

Answer:[('win', '(inverse) hypernym', 'steal'), ('profits', 'derivationally related form', 'turn a profit')]<lim_endl>

<|im_start|>system
You are a knowledgeable assistant that performs query rewriting for retrieving relevant context for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), you will rewrite (expand) the query into a list of triplets. Each of the expanded triplets is then used to retrieve relevant triplets from a database. Your goal is to provide a list of rewrited triplets that, after retrieval, will provide complete information needed for proving the original query, while keeping the number of rewrited triplets minimum.
<lim_endl>
<lim_startl>user
Query:
(therapeutic, (inverse) derivationally related form, ?)

Please expand the query into a python list of triplets in the form of [(head, relation, tail), ...]. Please strictly follow the format of the output as it will be parsed by a program.<lim_endl>
<lim_startl>assistant
<think>

</think>

Answer:[('medicine', 'synset domain topic of', 'practice of medicine'), ('therapeutic', 'derivationally related form', 'heal'), ('practice of medicine', 'derivationally related form', 'medical')]<lim_endl>

Figure 7: Sample prompt and answer(in dark green) for Qwen3-based REWRITER.

<|im_start|>system
You are a knowledgeable assistant that performs link prediction for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), and a retrieved set of relevant triplets, each with the corresponding reasoning steps that prove the triplet using other triplets from the database, you will first provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.
<lim_endl>
<lim_startl>user
Query:
(carry, verb group, ?)

Retrieved triplets:
('land reform', 'also see', 'land reform'), ('psychopathology', '(inverse) synset domain topic of', 'defense reaction'), ('offset', 'derivationally related form', 'make up'), ('make up', '(inverse) also see', 'even out'), ('compensation', 'derivationally related form', 'overcompensate'), ('carry', 'hypernym', 'make up'), ('trim', 'hypernym', 'equilibrize'), ('carry', 'verb group', 'carry'), ('make up', 'hypernym', 'equilibrize'), ('make up', '(inverse) hypernym', 'carry'), ('even out', 'also see', 'make up'), ('carry', '(inverse) verb group', 'carry'), ('equalizer', 'derivationally related form', 'make up'), ('defense reaction', '(inverse) hypernym', 'compensation'), ('make up', '(inverse) derivationally related form', 'equalizer'), ('overcompensation', 'synset domain topic of', 'psychopathology'), ('overcompensate', 'derivationally related form', 'overcompensation'), ('right', 'also see', 'make up'), ('overcompensate', 'hypernym', 'make up'), ('right', 'derivationally related form', 'recompense'), ('make up', '(inverse) also see', 'right'), ('make up', '(inverse) hypernym', 'overcompensate'), ('make up', '(inverse) derivationally related form', 'offset'), ('recompense', '(inverse) derivationally related form', 'make up')

What is an appropriate object entity that completes the query? Please provide a rigorous reasoning process before the final answer.<lim_endl>
<lim_startl>assistant
<think>
Based on the retrieved context, we have:
('carry', 'hypernym', 'make up') -> ('make up', '(inverse) hypernym', 'carry') -> ('carry', '(inverse) verb group', 'carry') -> ('right', 'also see', 'make up') -> ('make up', '(inverse) also see', 'right')

Therefore, we know:
('carry', 'verb group', 'carry')
</think>

Answer:carry<lim_endl>

<|im_start|>system
You are a knowledgeable assistant that performs link prediction for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), and a retrieved set of relevant triplets, each with the corresponding reasoning steps that prove the triplet using other triplets from the database, you will first provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.
<lim_endl>
<lim_startl>user
Query:
(thread, hypernym, ?)

Retrieved triplets:
('threader', 'derivationally related form', 'thread'), ('run', '(inverse) verb group', 'thread')]
{('yarn', 'derivationally related form', 'thread'), ('thread', 'verb group', 'run'), ('run', 'also see', 'wipe'), ('run', 'verb group', 'run'), ('thread', '(inverse) derivationally related form', 'threader'), ('threader', 'derivationally related form', 'thread'), ('thread', '(inverse) derivationally related form', 'yarn'), ('threader', '(inverse) derivationally related form', 'thread'), ('thread', 'derivationally related form', 'threader'), ('run', '(inverse) verb group', 'thread'), ('wipe', '(inverse) also see', 'run'), ('thread', 'derivationally related form', 'yarn'), ('run', 'verb group', 'thread'), ('thread', '(inverse) verb group', 'run'), ('run', '(inverse) verb group', 'run')

What is an appropriate object entity that completes the query? Please provide a rigorous reasoning process before the final answer.<lim_endl>
<lim_startl>assistant
<think>
Based on the retrieved context, we have:
('thread', '(inverse) derivationally related form', 'threader') -> ('thread', '(inverse) verb group', 'run') -> ('threader', 'derivationally related form', 'thread')

Therefore, we know:
('thread', 'hypernym', 'run')
</think>

Answer:run<lim_endl>

Figure 8: Sample prompt and answer(in dark green) for Qwen3-based REASONER.

<|im_start|>system
You are a knowledgeable assistant that performs multi-answer link prediction for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), and a retrieved set of relevant triplets, each with the corresponding reasoning steps that prove the triplet using other triplets from the database, you will first provide top-10 most probable reasoning step by step based on the retrieved triplets, and provide the corresponding top-10 appropriate object entity that completes the query.<lim_endl>
<lim_startl>user
Query:
(carry, verb group, ?)

Retrieved triplets:
('land reform', 'also see', 'land reform'), ('psychopathology', '(inverse) synset domain topic of', 'defense reaction'), ('offset', 'derivationally related form', 'make up'), ('make up', '(inverse) also see', 'even out'), ('compensation', 'derivationally related form', 'overcompensate'), ('carry', 'hypernym', 'make up'), ('trim', 'hypernym', 'equilibrize'), ('carry', 'verb group', 'carry'), ('make up', 'hypernym', 'equilibrize'), ('make up', '(inverse) hypernym', 'carry'), ('even out', 'also see', 'make up'), ('carry', '(inverse) verb group', 'carry'), ('equalizer', 'derivationally related form', 'make up'), ('defense reaction', '(inverse) hypernym', 'compensation'), ('make up', '(inverse) derivationally related form', 'equalizer'), ('overcompensation', 'synset domain topic of', 'psychopathology'), ('overcompensate', 'derivationally related form', 'overcompensation'), ('right', 'also see', 'make up'), ('overcompensate', 'hypernym', 'make up'), ('right', 'derivationally related form', 'recompense'), ('make up', '(inverse) also see', 'right'), ('make up', '(inverse) hypernym', 'overcompensate'), ('make up', '(inverse) derivationally related form', 'offset'), ('recompense', '(inverse) derivationally related form', 'make up'), ...

What are the top-10 appropriate object entities that completes the query? Please provide a rigorous reasoning process for each before the final answer.<lim_endl>
<lim_startl>assistant
<think>
Based on the retrieved context, we can deduce below top-10 proofs:
Proof 1:('carry', 'hypernym', 'make up') -> ('make up', '(inverse) hypernym', 'carry') -> ('carry', '(inverse) verb group', 'carry') -> ('right', 'also see', 'make up') -> ('make up', '(inverse) also see', 'right')
Proof 2: ...

Therefore, we know:
Answer 1: ('carry', 'verb group', 'carry')
Answer 2: ...
...
</think>

Answer:[carry, ...]<lim_endl>

<|im_start|>system
You are a knowledgeable assistant that performs multi-answer link prediction for knowledge graph completion tasks. Given a query in the form of a partially missing triplet (subject, relation, ?), and a retrieved set of relevant triplets, each with the corresponding reasoning steps that prove the triplet using other triplets from the database, you will first provide top-10 most probable reasoning step by step based on the retrieved triplets, and provide the corresponding top-10 appropriate object entity that completes the query.<lim_endl>
<lim_startl>user
Query:
(thread, hypernym, ?)

Retrieved triplets:
('threader', 'derivationally related form', 'thread'), ('run', '(inverse) verb group', 'thread')]
{('yarn', 'derivationally related form', 'thread'), ('thread', 'verb group', 'run'), ('run', 'also see', 'wipe'), ('run', 'verb group', 'run'), ('thread', '(inverse) derivationally related form', 'threader'), ('threader', 'derivationally related form', 'thread'), ('thread', '(inverse) derivationally related form', 'yarn'), ('threader', '(inverse) derivationally related form', 'thread'), ('thread', 'derivationally related form', 'threader'), ('run', '(inverse) verb group', 'thread'), ('wipe', '(inverse) also see', 'run'), ('thread', 'derivationally related form', 'yarn'), ('run', 'verb group', 'thread'), ('thread', '(inverse) verb group', 'run'), ('run', '(inverse) verb group', 'run'),...

What is an appropriate object entity that completes the query? Please provide a rigorous reasoning process before the final answer.<lim_endl>
<lim_startl>assistant
<think>
Based on the retrieved context, we have:
Proof 1: ('thread', '(inverse) derivationally related form', 'threader') -> ('thread', '(inverse) verb group', 'run') -> ('threader', 'derivationally related form', 'thread')
Proof 2: ...

Therefore, we know:
Answer 1: ('thread', 'hypernym', 'run')
Answer 2: ...
</think>

Answer:[run,...]<lim_endl>

Figure 9: Sample prompt and answer(in dark green) for Qwen3-based REASONER with multi-answer extension.