LEARNING GENERATIVE KNOWLEDGE GRAPH COMPLETION VIA PATH SUPERVISION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Path-Augmented Retrieval and Reasoning (PARR), a simple, effective, and generalizable framework for training generative language models for Knowledge Graph Completion (KGC). A unique challenge in adopting native LLMs[1] for KGC is the lack of retrieval and reasoning supervision in KGC datasets, which has caused all previous efforts to underperform traditional KGC methods by a significant margin. In this work, we propose to leverage grounding paths extracted from pretrained interpretable link predictors as supervision for LLM-based KGC. Specifically, we use these paths (1) to augment retrieval for enhanced subgraph retrieval, (2) as ground-truth retrieval signals to supervise KG query rewriting, and (3) as a means to *"distill"* the structural knowledge into the LLM through chain-of-thought reasoning. In PARR, we propose to supervise a single LLM to perform KGC-based rewrite-retrieve-reason, trained end-to-end with Group Sequence Policy Optimization over tree-structured rollouts with soft verifiable rewards tailored for KGC. PARR is the first native LLM-based framework that significantly surpasses traditional discriminative KGC methods, achieving superior performance over existing state-of-the-art methods across 19 KGs under both transductive and inductive settings, while being generalizable, scalable, and interpretable.

1 INTRODUCTION

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

Traditional KGC methods typically 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. While these models are lightweight, **real-world knowledge-intensive domains often prioritize accuracy, interpretability, and conversational reasoning over efficiency**. Moreover, both KGE and GNN models are inherently discriminative: predictions are made by scoring candidate entities within a fixed set, offering limited generalization and restricting existing KGC methods to highly specialized expert models, hindering their accessibility to general users.

On the other hand, LLMs with Retrieval-Augmented Generation (RAG) have become the *de facto* paradigm for knowledge-intensive natural language QA (Yang et al., 2018) and KGQA (Yih et al., 2016; Talmor & Berant, 2018; Luo et al., 2024). Compared with specialized discriminative link predictors, LLMs offer greater accessibility through their generalizability and natural-language interface. However, directly extending this paradigm to KGC remains challenging. Existing attempts to adopt native LLMs for KGC (Yao et al., 2025; Zhu et al., 2024) still substantially underperform strong link predictors. We argue that this performance gap stems largely from the **absence of fine-grained supervision for retrieval and reasoning in the domain of KGC**. Unlike natural language QA or KGQA (Yih et al., 2016; Talmor & Berant, 2018), where golden retrieval or reasoning traces are either provided or relatively easy to construct (Zhang et al., 2022; Luo et al., 2024), KGC operates

[1]Here, *native* means that the LLM is used without modifying its architecture or adding components to it (e.g., graph encoders or entity-scoring heads), and keeps its original autoregressive next-token prediction at inference.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

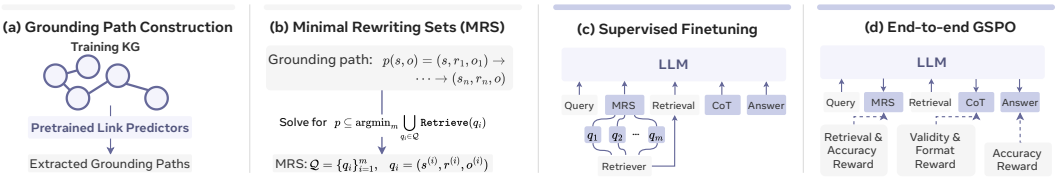

Figure 1: Overall training pipeline of PARR. (a) Grounding paths are extracted from pretrained interpretable link predictors. (b) For each path, we construct Minimal Rewriting Sets (MRS), minimum number of sub-queries whose retrievals jointly cover the path. (c) We perform Supervised Finetuning (SFT) as stage 1 training. (d) The same LLM is then trained end-to-end with GSPO.

over large-scale, domain-specific, and incomplete KGs. As a result, simple graph traversal utilized in KGQA (Zhang et al., 2022; Luo et al., 2024) cannot reliably produce reasoning supervision for KGC, since extracted paths may be spurious or unrelated to the target relation. Existing native LLM-based KGC methods therefore either rely on heuristic neighbor retrieval (Wei et al., 2023), which yields local and often suboptimal evidence, or bypass retrieval and reasoning altogether by directly predicting the missing entity from a subject–relation pair (Yao et al., 2025; Zhu et al., 2024). In both cases, LLMs receive little explicit supervision for the structural knowledge that is known to be essential for link prediction (Galkin et al., 2024).

To address this supervision bottleneck, we propose PARR, a Rewrite-Retrieval-Reason (RRR) framework for training native generative LLMs for KGC. Our key insight is that pretrained interpretable link predictors already encode useful structural knowledge about how missing links are supported by relational paths. Instead of treating these models only as discriminative baselines, PARR repurposes their extracted grounding paths as supervision for LLM-based KG retrieval and reasoning. These paths serve as compact and interpretable structural evidence, which encodes multi-hop relational structures, providing effective supervision for retrieval and Chain-of-Thought (CoT) reasoning. Specifically, we leverage the extracted paths in three complementary ways:

(1) Path-augmented retrieval. Rather than retrieving isolated triplets, PARR retrieves triplets together with their associated grounding paths. These paths act as natural structural clusters, allowing the model to access compact evidence that better captures non-local relational dependencies.

(2) Path-guided KG query rewriting. We use grounding paths as pseudo retrieval supervision to supervise the LLM to act as a KGC REWRITER. Given a query, it decomposes the query into a set of sub-queries, improving retrieval beyond static neighborhood heuristics.

(3) Path-supervised reasoning. The same LLM then acts as a KGC REASONER over the retrieved triplets. Supervised finetuning on path-derived rewriting and CoT data is first performed to teach the model to reason over noisy retrieved triplets and to distill structural knowledge from pretrained link predictors into the LLM. To further enhance KG reasoning capability and to avoid cascaded errors propagated from pretrained link predictors, we subsequently train the whole RRR trajectory end-to-end with GSPO (Zheng et al., 2025), using verifiable, fine-grained soft rewards tailored to KGC. Under this formulation, the REASONER can be viewed as performing graph traversal in token space, where each reasoning step is sampled through the LLM's next-token distribution.

The overall PARR's training framework is shown in Figure 1. Our contributions include:

(1) We propose PARR, the first RRR framework for LLM-based KGC, and the first to leverage grounding paths from pretrained link predictors to supervise native LLM-based link prediction.

(2) We are the first to explore Reinforcement Learning with Verifiable Rewards (RLVR) for link prediction, training the full RRR pipeline end-to-end with GSPO over tree-structured rollouts. We show such design can significantly improve the performance over SFT with only 5K training samples.

(3) We evaluate PARR on 19 KGs of varying scales under three settings: transductive, inductive with unseen entities, and inductive with unseen entities and relations. PARR achieves SOTA performance and is, to our knowledge, the first native LLM-based KGC to surpass strong traditional discriminative methods. In the fully inductive setting, it also outperforms KG foundation models (*e.g.* Galkin et al. (2024) and Zhang et al. (2025)), which are pretrained on multiple KGs. More broadly, PARR enables KGC to be integrated into native LLMs without any modification to LLMs' architecture or inference procedure, opening new opportunities for general-purpose, multi-task, and multi-dataset KG reasoning, as well as conversational KGQA that are more accessible to non-expert users.

2 PRELIMINARIES

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

Grounding Paths. Given a pair of entities (s, o), its proof $p(s, o)$ can be defined as a chain-like logical rule of the form: $p(s, o) \leftarrow r_1(s, z_1) \wedge r_2(z_1, z_2) \wedge \cdots \wedge r_n(z_{n-1}, o)$, where $r_1, \cdots, r_n$ are relations in the given KG. For path or GNN-based link predictors, the proofs can be viewed as the grounding paths taken to reach object o from subject s, i.e., $p(s, o) = r_1(s, z_1) \rightarrow \cdots \rightarrow r_n(z_{n-1}, o)$.

3 PATH-AUGMENTED RETRIEVAL AND REASONING

PARR trains a single LLM π_θ to solve KGC through a rewrite–retrieve–reason trajectory (Figure 1). Given a query (s, r), π_θ first acts as a REWRITER and generates a set of sub-queries; a fixed RETRIEVER returns the KG triplets matching each sub-query; and π_θ then acts as a REASONER, producing CoT reasoning paths over the retrieved triplets followed by the answers. In PARR, REWRITER and REASONER denote these two roles of the same model rather than separate models. Below we describe how grounding paths are extracted and turned into supervision for each role, and how the model is trained with SFT followed by end-to-end RL.

3.1 PATH EXTRACTION FROM LINK PREDICTORS

We consider two SOTA link prediction models with interpretable path formulations: NBFNet (Zhu et al., 2021) and A*Net (Zhu et al., 2023). We choose them because (a) they consistently deliver strong and stable link prediction performance, and (b) they naturally support the extraction of grounding (proof) paths, each through a distinct mechanism.

NBFNet parametrizes the Bellman-Ford algorithm with a GNN. Given a triplet (s, r, o), we can sample top-k proofs $P_1, P_2, \cdots, P_k$ from a pretrained NBFNet through a linear model (Baehrens et al., 2010) (*i.e.*, first-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 grounding path (Zhu et al., 2021):

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

where $\mathcal{P}_{(s,o)}$ denotes all paths from s to o. We use the average edge importance to approximate path score, which can be efficiently computed by an autograd engine (Paszke et al., 2019).

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 paths (Zhu et al., 2023). Given a triplet (s, r, o) and the pretrained node priority function $s^{(t)}_{s,r}(x)$, we can sample top-k paths by their average node importance $s(P)$:

$$s(P) = \frac{1}{|P|} \sum\nolimits_{t=1, P^{(t)}=(x,r,y)}^{|P|} \frac{s^{(t-1)}_{s,r}(x)}{S^{(t-1)}_{s,r}}, \quad S^{(t-1)}_{s,r} = \max\nolimits_{x \in \mathcal{E}^{(t-1)}} \left[s^{(t-1)}_{s,r}(x) \right].$$

Constructing Path Data with Mixture of Experts. To avoid model bias from a single pretrained model, we sample paths using a mixture of experts (MoE) of different models, where each model has a distinct path extraction method, and aggregate the resulting paths. Finally, we filter out paths with duplications or cycles. For triplets in the training set, we also filter out paths containing the triplet itself, as it represents an existing edge for which the model is likely to give the highest score. The resulting retrieval KB $\mathcal{D}$ is then a set of triplet-paths 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 paths that ground the triplet.

Multi-Answer Link Prediction. To align with the standard top-k evaluation in KGC (e.g., $k = 10$), PARR supports multi-answer supervision for both retrieval and reasoning. This contrasts with prior LLM-based approaches (Yao et al., 2025), which are restricted to top-1 prediction. Since a query (s, r) may have fewer than k ($k = 10$ in our case) ground-truth objects, we fill the remaining slots

with the top-ranked candidates of the interpretable link predictors (the same models that generate paths). Specifically, given a query (s, r), we obtain a ranked list $\mathcal{O}_{\text{top-}k} = \{o_1, \ldots, o_k\}$, and for each candidate o_i, we extract an associated path set $\mathcal{P}_i = \{p_1^i, \ldots, p_m^i\}$.

3.2 Non-parametric KGC Retriever

We consider three types of non-parametric KG retriever. **(1) Sparse retrieval.** Similar to KICGPT (Wei et al., 2023), we consider triplets sharing the same entities with the query. To constrain retrieval size, we only select top-*k* triplets based on the degree of their entities. **(2) Dense retrieval.** We first convert triplets to natural language sentences and encode them with a sentence embedding model. **(3) Hybrid retrieval.** We first perform dense retrieval, then run breadth-first search on each of the retrieved proofs.

3.3 Path-Guided Query Rewriting

To improve KG retrieval, the Rewriter decomposes the original query (s, r) into a set of sub-queries whose individual retrievals collectively cover the triplets in ground-truth grounding paths. Formally, acting as the Rewriter, π_θ maps the query (s, r) into a set of rewritten triplets:

$$\mathcal{Q} \sim \pi_\theta(\cdot \mid s, r), \quad \mathcal{Q} = \left\{(s^{(1)}, r^{(1)}, o^{(1)}), \cdots, (s^{(m)}, r^{(m)}, o^{(m)})\right\}.$$

For each rewritten triplet $q_i = (s^{(i)}, r^{(i)}, o^{(i)}) \in \mathcal{Q}$, we retrieve the top-*k* most similar KG triplets $\mathcal{G}_i$ using the fixed Retriever detailed in Sec. 3.2, denoted as $\texttt{Retrieve}(\cdot, \cdot)$:

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

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

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

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

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 path. We are therefore able to obtain a much larger rewriting dataset (as compared to the original training set size for link prediction) of size $|N| \cdot |\mathcal{P}| \cdot |\mathfrak{Q}^*|$, where $|N|$ is the number of training triplets, $|\mathcal{P}|$ is the number of paths per triplet, and $|\mathfrak{Q}^*|$ is the number of minimal rewriting sets per path. The resulting dataset is composed of 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.4.

3.4 Path-Supervised Reasoning

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, and (2) insufficient understanding of the structural patterns in knowledge graphs. As shown in Figure 2, zero-shot LLMs fall well below traditional link predictors even with oracle retrieval and degrade rapidly as irrelevant triplets are injected. To address both issues, we train the Reasoner with Chain-of-Thought (CoT) supervision using retrieved context and extracted grounding paths.

Formally, given a link prediction query (s, r), its rewrite $\mathcal{Q}$, and the retrieved set $\mathcal{G}$, π_θ acts as the Reasoner and predicts the missing object o after producing an intermediate CoT:

$$\pi_\theta : \big((s,r),\mathcal{Q},\mathcal{G}\big) \mapsto \big(p,o\big), \quad p = [(s, r^{(1)}, o^{(1)}), \cdots, (o^{(n-1)}, r^{(n)}, o)].$$

The finetuned REASONER can be seen as 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. 3.3). The resulting dataset consists of tuples $\big((s,r),\mathcal{Q},\mathcal{G},p,o\big)$, where $\mathcal{Q}$ is an MRS and $\mathcal{G}$ its retrieval, with size $|N| \cdot |\mathcal{P}| \cdot |\mathfrak{Q}^*|$, the same as the rewriting set. Each tuple forms a complete rewrite–retrieve–reason trajectory for SFT. While we utilize ground-truth retrieval during SFT, the retrieval is often imperfect at test time. To close this gap, we randomly perturb a portion of ground-truth retrievals during SFT. This encourages the model to infer missing facts from incomplete context, improving robustness to noisy or partial retrievals.

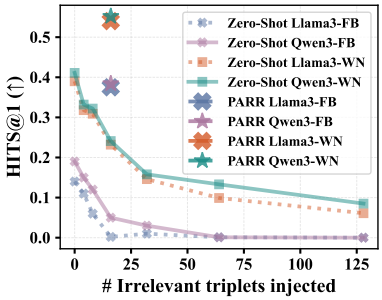

Figure 2: HITS@1 with ground-truth retrievals and injected irrelevant triplets, simulating the false positive retrievals.

3.5 TRAINING STAGE 1: SUPERVISED FINETUNING

We first perform supervised finetuning (SFT) on the trajectories $\mathcal{D}_{\text{SFT}}$ constructed above. Each trajectory consists of the rewrite $\mathcal{Q}$, the retrieved triplets $\mathcal{G}$, and the reasoning output $y = (p, o)$, *i.e.*, the CoT path followed by the answer (in the multi-answer setting, y is the ranked sequence of path–answer pairs). Since the retrieved triplets are inserted by the RETRIEVER, the NLL loss is computed only over the generated tokens:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{((s,r),\mathcal{Q},\mathcal{G},y)\sim\mathcal{D}_{\text{SFT}}}\Big[\underbrace{\log \pi_\theta\big(\mathcal{Q} \mid s, r\big)}_{\text{query rewriting}} + \underbrace{\sum\nolimits_{t=1}^{|y|} \log \pi_\theta\big(y_t \mid y_{<t}, (s,r), \mathcal{Q}, \mathcal{G}\big)}_{\text{path-supervised reasoning}}\Big].$$

3.6 TRAINING STAGE 2: END-TO-END RL WITH GSPO

Tree-structured rollouts. After SFT, we train π_θ end-to-end with GSPO. Since the final answer depends on both the rewrite and the reasoning over its retrieval, we sample rollouts as a two-level tree. Given a query $x = (s, r)$, the model first samples K rewrites $\{\mathcal{Q}_j\}_{j=1}^{K} \sim \pi_\theta(\cdot \mid x)$. For each rewrite, the RETRIEVER returns $\mathcal{G}_j$, and the model samples M reasoning rollouts $\{y_{j,m}\}_{m=1}^{M} \sim \pi_\theta(\cdot \mid x, \mathcal{Q}_j, \mathcal{G}_j)$, giving $K \times M$ trajectories per query. The tree lets us credit rewriting and reasoning separately: reasoning rollouts that share a rewrite are compared under the same retrieval, while rewrites are rewarded by their retrieval accuracy and the outcomes of their reasoning rollouts.

Reasoning rewards. Each reasoning rollout receives a fine-grained, soft reward composed of:

(1) *Accuracy reward* (R_{acc}) evaluates whether the model predicts the correct missing entities. We adopt NDCG@10 to credit multiple ranked answers, and only count unique answers to avoid reward hacking by repeating answers.

(2) *Format reward* (R_{format}) measures whether each generated path conforms to the predefined format $(s, r^{(1)}, o^{(1)}) \to \cdots \to (o^{(n-1)}, r^{(n)}, o)$.

(3) *Validity reward* (R_{val}) checks the validity of each path in three aspects: (a) $R_{\text{val-}c}$: the path is contiguous, *i.e.*, the tail of each hop is the head of the next; (b) $R_{\text{val-}v}$: each triplet is a valid edge in the training KG; and (c) $R_{\text{val-}l}$: the path contains no loops. We set $R_{\text{val}} = \frac{1}{3}(R_{\text{val-}c} + R_{\text{val-}v} + R_{\text{val-}l})$.

R_{format} and R_{val} are path-wise rewards. As each response contains k paths (one per answer), we average them over all paths, which provides smooth and continuous feedback to the policy:

$$R_{\text{rea}} = \frac{1}{2} R_{\text{acc}} + \frac{1}{4} \cdot \frac{1}{k} \sum_{i=1}^{k} \big(R^i_{\text{format}} + R^i_{\text{val}}\big).$$

Rewrite reward. A good rewrite should retrieve the evidence needed to answer the query. We reward each rewrite $\mathcal{Q}_j$ by (i) the mean accuracy of its reasoning rollouts, $\bar{R}_{\text{acc}}(\mathcal{Q}_j) = \frac{1}{M}\sum_{m=1}^{M} R_{\text{acc}}(y_{j,m})$, a Monte Carlo estimate of how useful its retrieval is for answering, and (ii) its coverage of the extracted grounding paths, $R_{\text{cov}}(\mathcal{Q}_j) = \max_{p\in\mathcal{P}_{s,r}} |p \cap \mathcal{G}_j|/|p|$, *i.e.*, the largest fraction of any grounding path that its retrieval recovers. The rewrite reward is $R_{\text{rw}}(\mathcal{Q}_j) = \bar{R}_{\text{acc}}(\mathcal{Q}_j) + R_{\text{cov}}(\mathcal{Q}_j)$. The coverage term provides dense feedback even when none of the reasoning rollouts is correct, while the accuracy term rewards rewrites that lead to correct answers even if they deviate from the extracted paths. The detailed calculation of all rewards is provided in Appendix F.2.

Group Sequence Policy Optimization (GSPO). We optimize both the rewriting and reasoning segments with GSPO (Zheng et al., 2025), which stabilizes GRPO-style (Shao et al., 2024) training by using sequence-level rather than token-level importance ratios. Advantages are computed group-relatively at each level of the tree: each rewrite is normalized against the K rewrites of the same query, and each reasoning rollout against the M rollouts that share its rewrite:

$$A_j = \frac{R_{\text{rw}}(\mathcal{Q}_j) - \text{mean}\big(\{R_{\text{rw}}(\mathcal{Q}_{j'})\}_{j'=1}^{K}\big)}{\text{std}\big(\{R_{\text{rw}}(\mathcal{Q}_{j'})\}_{j'=1}^{K}\big)}, \quad A_{j,m} = \frac{R_{\text{rea}}(y_{j,m}) - \text{mean}\big(\{R_{\text{rea}}(y_{j,m'})\}_{m'=1}^{M}\big)}{\text{std}\big(\{R_{\text{rea}}(y_{j,m'})\}_{m'=1}^{M}\big)}.$$

For a generated segment z (a rewrite or a reasoning rollout) with context c, GSPO uses the length-normalized sequence-level importance ratio $\rho_z(\theta) = \big(\pi_\theta(z \mid c)/\pi_{\text{old}}(z \mid c)\big)^{1/|z|}$. The objective is

$$\mathcal{J}_{\text{GSPO}}(\theta) = \mathbb{E}\Big[\frac{1}{K}\sum_{j=1}^{K} \ell\big(\rho_{\mathcal{Q}_j}(\theta), A_j\big) + \frac{1}{KM}\sum_{j=1}^{K}\sum_{m=1}^{M} \ell\big(\rho_{y_{j,m}}(\theta), A_{j,m}\big)\Big] - \beta\, \mathbb{D}_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}}),$$

where $\ell(\rho, A) = \min\big(\rho A, \text{clip}(\rho, 1-\epsilon, 1+\epsilon)A\big)$ with clipping range ϵ, π_{old} denotes the sampling policy, and π_{ref} is the SFT model used as a fixed reference. Retrieved triplets are excluded from the loss, as they are not generated by the policy.

4 Experiments

4.1 Experiment Setup

Baselines. We compare PARR against embedding-based, GNN-based, and path-based link predictors, KG foundation models (*e.g.*, ULTRA (Galkin et al., 2024) and TRIX (Zhang et al., 2025)), and methods built on pretrained language models, including the LLM-based KG-Llama (Yao et al., 2025), KICGPT (Wei et al., 2023), MKGL (Guo et al., 2024), and KRLM (Zhuo et al., 2026). The full list of baselines is given in Appendix F.1.

Datasets. We evaluate PARR primarily on FB15k-237 (Toutanova & Chen, 2015) and WN18RR (Dettmers et al., 2018), two major datasets for link prediction. We consider both transductive and inductive settings, following the standard splits in Toutanova & Chen (2015); Dettmers et al. (2018) and Teru et al. (2020). We also consider additional datasets such as NELL995 (Mitchell et al., 2018; Xiong et al., 2017), YAGO3-10 (Mahdisoltani et al., 2015), the large-scale ogbl-wikikg2 (Hu et al., 2020), and small-scale statistical KGs such as Kinship, Nations, and UMLS (Kemp et al., 2006). Due to space limits, some results are provided in Appendix D.

Evaluation. We report the standard HITS@k metrics ($k \in \{1, 3, 10\}$). We do not report mean rank (MR) or mean reciprocal rank (MRR): as a generative method, PARR outputs only a ranked list of top-k candidates rather than a score over all entities, so gold entities outside the top-k have no well-defined rank. This follows prior generative LLM-based KGC methods (Yao et al., 2025; Zhu et al., 2024). The only exception is ogbl-wikikg2, whose official metric is MRR; therefore, we assign zero reciprocal rank to gold entities outside the top-k, so the reported MRR is a lower bound.

Implementation Details. We employ Llama3-8B-Instruct (Grattafiori et al., 2024) (abbr. as Llama3) and Qwen3-8B (Yang et al., 2025) (Qwen3) as the base LLM, with a single model serving as both Rewriter and Reasoner. We finetune using LoRA (Hu et al., 2022) with rank and alpha being 32 and 64. We use dense retrieval with the Jina V3 (Sturua et al., 2025) embedding model. For end-to-end GSPO, we sample $K = 4$ rewrites per query and $M = 4$ reasoning rollouts per rewrite (16 trajectories per query), and set the KL divergence coefficient (β) to 0.04 by default and the learning rate to 1×10^{-6}. We use 5K training samples for GSPO training. All models are trained for one epoch. Please refer to Appendix F.2 for full details and Appendix C for the complexity and computational cost analysis.

Table 1: Transductive link prediction on FB15k-237, WN18RR and NELL995. Best/2nd-best results are in **bold**/underlined. Baseline refers to the setting with dense retrieval $k = 50$ and an LLM finetuned without path supervision (*i.e.*, no rewriting or CoT).

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	0.482	0.526	-	-
CompGCN	0.264	0.390	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
ULTRA	-	-	0.564	-	-	0.614	-	0.660
TRIX	-	-	0.559	-	-	0.611	-	0.648
KG-ICL	-	-	0.538	-	-	0.637	-	-
MOTIF	-	-	0.550	-	-	0.628	-	-
SEMMA	-	-	0.574	-	-	0.656	-	-
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**	-	-
CSProm-KG	0.269	-	0.538	0.522	-	0.678	-	-
KG-FIT	0.275	-	0.572	-	-	-	-	-
GLTW-7B	0.351	-	0.614	0.556	-	0.690	-	-
SAT	0.319	-	-	-	-	-	-	-
MKGL	0.325	0.454	0.591	0.500	0.577	0.656	-	-
KRLM	-	-	0.568	-	-	0.659	-	-
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	-	-
Baseline-Llama3	0.143	-	-	0.279	-	-	0.301	-
PARR_{SFT}-Llama3	0.335 (+.19)	0.424	0.551	0.463 (+.18)	0.559	0.638	0.482 (+.18)	0.644
PARR_{RL}-Llama3	0.381 (+.24)	0.495	0.619	0.549 (+.27)	0.636	0.683	0.566 (+.27)	0.712
Baseline-Qwen3	0.155	-	-	0.285	-	-	0.313	-
PARR_{SFT}-Qwen3	0.341 (+.19)	0.436	0.573	0.488 (+.20)	0.573	0.632	0.501 (+.19)	0.642
PARR_{RL}-Qwen3	**0.387** (+.23)	**0.524**	**0.639**	**0.564** (+.28)	**0.656**	0.702	**0.577** (+.26)	**0.725**

Table 2: Results on inductive (e) KG reasoning

Model	FB15k-237-ind		WN18RR-ind	
	HITS@1	HITS@10	HITS@1	HITS@10
Finetuned Setting				
NeuralLP	0.243	0.468	0.592	0.772
DRUM	0.247	0.474	0.613	0.777
RED-GNN	0.302	0.483	0.653	0.799
NBFNet	0.335	0.574	0.695	0.826
KG-ICL	-	0.700	-	0.827
MOTIF	-	0.702	-	0.806
MKGL	0.400	0.595	0.700	0.822
KRLM	-	0.701	-	0.800
ChatGPT-3.5	0.288	-	0.279	-
Zero-shot Setting				
ULTRA	-	0.656	-	0.768
TRIX	-	0.682	-	0.791
SEMMA	-	0.655	-	0.816
KRLM	-	0.708	-	0.783
PARR_{SFT}-Llama3	0.377	0.564	0.684	0.802
PARR_{RL}-Llama3	0.488	0.693	0.773	0.856
PARR_{SFT}-Qwen3	0.384	0.574	0.702	0.810
PARR_{RL}-Qwen3	**0.522**	**0.713**	**0.775**	**0.866**

Table 3: Performance (HITS@10) on inductive setting (e, r). Baseline results are from Lee et al. (2023); Zhang et al. (2025); Zhuo et al. (2026).

Method	HITS@10		
	FB-100	WK-100	NL-100
Finetuned Setting			
CompGCN	0.025	0.009	0.014
DRUM	0.077	0.019	0.138
NBFNet	0.154	0.026	0.199
RED-GNN	0.263	0.136	0.385
InGram	0.371	0.169	0.506
ULTRA	0.643	0.286	0.684
TRIX	0.633	0.299	0.691
KRLM	0.655	0.313	0.688
Zero-shot Setting			
ULTRA	0.642	0.286	0.651
TRIX	0.635	0.299	0.676
KRLM	0.647	0.320	0.667
PARR_{RL}-Llama3	0.661	0.325	0.732
PARR_{RL}-Qwen3	**0.688**	**0.349**	**0.744**

4.2 Main Results

Transductive Setting. Tables 1 and 4 show results on five transductive datasets of varying scales. PARR_{RL} achieves SOTA performance on most datasets, and improves over finetuning without path supervision (Baseline) by over 20% absolute HITS@1. Notably, it also surpasses NBFNet and A*Net, the link predictors it learns from. Since PARR_{RL} also outperforms baselines built on the same or stronger LLMs, its gains cannot be trivially attributed to memorization in the base LLM alone.

Inductive Setting. Table 2 shows results where entities can be unseen during training. Evaluated zero-shot, PARR_{RL}-Qwen3 outperforms all methods on both datasets, including those finetuned on

Figure 3: Retrieval recall (y-axis) as a function of the average number of retrieved triplets per sample (x-axis). BFS(2) refers to the hybrid RETRIEVER with Breadth-First-Search depth of 2.

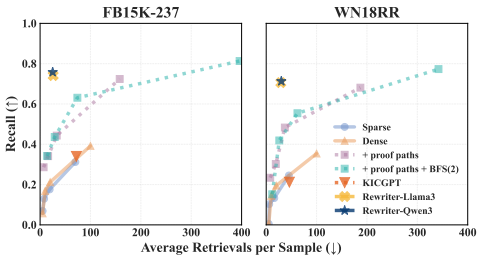

Table 5: REWRITER and REASONER performance under different top-k retrieval. # Rew.: # rewritten sub-queries per sample; # Retr.: # retrievals per sample.

top-k	Backbone	# Retr.	# 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

the inductive splits, and its margin over NBFNet, one of its supervising link predictors, is larger than in the transductive setting, as is the RL gain over SFT on FB15k-237-ind.

Table 3 further evaluates the harder setting where both entities and relations can be unseen. PARR_{RL} achieves SOTA performance on all three datasets, surpassing KG foundation models (ULTRA and TRIX) trained with more data, the LLM-based KRLM, and, by a large margin, NBFNet, which supplies part of its path supervision, showing strong generalization to unseen entities and relations.

REWRITER performance. As shown in Figure 3, the REWRITER achieves substantially higher recall than all baselines at a comparable retrieval budget, including KICGPT's heuristic retrieval. Adding proof paths to dense retrieval also improves recall at every budget.

Table 4: Results on YAGO3-10 and ogbl-wikikg2.

Method	**YAGO3-10** HITS@1	HITS@3	HITS@10	**ogbl-wikikg2** MRR
DistMult	0.240	0.380	0.540	0.373
TransE	-	-	-	0.425
ComplEx	0.260	0.400	0.550	0.402
RotatE	0.402	0.550	0.670	0.433
BoxE	0.400	0.472	0.541	-
HAKE	0.452	0.516	0.582	-
PairRE	-	-	-	0.520
CompGCN	0.392	-	0.577	-
NBFNet	0.480	0.612	0.708	-
A*Net	0.470	0.611	0.707	0.677
ULTRA	0.531	-	0.710	-
KG-ICL	-	-	0.688	-
MOTIF	-	-	0.735	-
SEMMA	-	-	0.567	-
CSProm-KG	0.451	-	-	-
KG-FIT	0.474	-	0.718	-
SAT	0.525	-	-	-
KG-Llama-13B	0.133	-	-	-
PARR_{SFT}-Qwen3	0.515	0.626	0.724	0.673
PARR_{RL}-Qwen3	**0.598**	**0.707**	**0.773**	**0.755**

4.3 ABLATION STUDIES

Ablation setting. The two roles of the unified PARR share parameters and are trained jointly, so changing one stage also shifts the other, confounding each component's effect. Our ablations (and Figure 3) therefore use separately trained REWRITER and REASONER copies of one base LLM with identical data and RETRIEVER, varying one stage while fixing the other (Appendix E). Variants are thus compared within each table, not with the main results; only the rewrite-reward rows of Table 7 use the unified model.

How does retrieval impact reasoning accuracy? Table 5 varies the retrieval top-k. Retrieval quality peaks at $k = 5$ and drops markedly at $k = 2$, likely because smaller k requires more sub-queries and hence a harder query-to-subquery mapping for the REWRITER. REASONER accuracy closely tracks retrieval quality, showing a strong dependency between retrieval and reasoning.

How does each component impact REASONER performance? Table 6 ablates the components of the REASONER. Naive finetuning without retrieval or CoT performs poorly. Fixed dense retrieval helps, replacing it with the REWRITER helps far more, and CoT supervision and retrieval perturbation bring further gains, showing that each design is necessary.

How does the reward design impact end-to-end GSPO? Table 7 ablates the reward components. As the accuracy reward, NDCG@10 yields much larger gains than HITS@10, which does not enforce correct ranking, and the validity and format rewards bring further consistent gains. Training end-to-end with the rewrite reward brings the largest gains, and the path-coverage term helps further, showing the benefit of jointly training rewriting and reasoning. Lastly, we observe that RL performance is robust to how the three validity checks are weighted (Appendix Table 16).

Table 6: Ablations on the performance of REASONER (after the SFT stage) with respect to each component: Chain of Thought (**CoT**), retrieval method (**Retr.**), and **Retr. Perturb**: whether to apply random perturbation on ground-truth retrievals while finetuning the REASONER. Dense refers to fixed dense retrieval with $k = 50$. REW refers to our finetuned REWRITER.

Method	w/ CoT	w/ Retr.	w/ Retr. Perturb	FB15k-237			WN18RR		
				HITS@1	HITS@3	HITS@10	HITS@1	HITS@3	HITS@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	-	-
Baseline-Llama3	×	×	-	0.077	-	-	0.171	-	-
Baseline-Qwen3	×	×	-	0.085	-	-	0.184	-	-
Baseline-Llama3	×	Dense	-	0.143	-	-	0.279	-	-
Baseline-Qwen3	×	Dense	-	0.155	-	-	0.285	-	-
PARR_{SFT}-Llama3	×	REW	×	0.293	0.395	0.412	0.451	0.413	0.479
PARR_{SFT}-Qwen3	×	REW	×	0.302	0.388	0.480	0.467	0.499	0.544
PARR_{SFT}-Llama3	✓	REW	×	0.316	0.405	0.534	0.479	0.491	0.538
PARR_{SFT}-Qwen3	✓	REW	×	0.335	0.413	0.521	0.493	0.524	0.603
PARR_{SFT}-Llama3	✓	REW	✓	0.341	0.449	0.579	0.488	0.566	0.632
PARR_{SFT}-Qwen3	✓	REW	✓	**0.348**	**0.460**	0.586	**0.507**	0.581	0.644

Table 7: Effect of the RL reward on link prediction performance. R_{acc}, R_{format}, and R_{val} are the reasoning rewards; $\bar{R}_{\text{acc}}$ and R_{cov} are the downstream-accuracy and path-coverage terms of the rewrite reward.

Training Stage	R_{acc}	R_{format}	R_{val}	$\bar{R}_{\text{acc}}$	R_{cov}	FB15k-237			WN18RR		
						HITS@1	HITS@3	HITS@10	HITS@1	HITS@3	HITS@10
SFT	–	–	–	–	–	0.341	0.436	0.573	0.488	0.573	0.632
GSPO	HITS@10	×	×	×	×	0.344 (+.003)	0.457 (+.021)	0.588 (+.015)	0.492 (+.004)	0.583 (+.010)	0.648 (+.016)
	NDCG@10	×	×	×	×	0.353 (+.012)	0.472 (+.036)	0.601 (+.028)	0.516 (+.028)	0.591 (+.018)	0.661 (+.029)
	NDCG@10	×	✓	×	×	0.358 (+.017)	0.476 (+.040)	0.612 (+.039)	0.518 (+.030)	0.596 (+.023)	0.668 (+.036)
	NDCG@10	✓	✓	×	×	0.367 (+.026)	0.486 (+.050)	0.618 (+.045)	0.527 (+.039)	0.603 (+.030)	0.679 (+.047)
	NDCG@10	✓	✓	✓	×	0.381 (+.040)	0.509 (+.073)	0.633 (+.060)	0.550 (+.062)	0.622 (+.049)	0.693 (+.061)
	NDCG@10	✓	✓	✓	✓	**0.387** (+.046)	**0.524** (+.088)	**0.639** (+.066)	**0.564** (+.076)	**0.656** (+.083)	**0.702** (+.070)

Table 8: MoE effect for RETRIEVER and REASONER.

w/ MoE	Retriever (Recall)		Reasoner (HITS@1)	
	FB15k-237	WN18RR	FB15k-237	WN18RR
×	0.683	0.629	0.324	0.456
✓	0.741 (+.058)	0.706 (+.077)	0.341 (+.017)	0.488 (+.032)

Table 9: Effect of noisy training paths on FB15k-237.

Model	% Noisy Paths				
	0%	25%	50%	75%	100%
PARR_{SFT}-Qwen3	0.348	0.329	0.255	0.211	0.203
PARR_{RL}-Qwen3	0.384	0.372	0.341	0.316	0.308

How does MoE sampling impact performance? Table 8 compares sampling paths from MoE link predictors (Sec. 3.1) against a single link predictor (NBFNet). MoE sampling consistently improves both retrieval recall and link prediction accuracy.

How does PARR handle noisy training paths? To test robustness to errors in pretrained link predictors, we replace $x\%$ of the path-supervision examples with low-confidence paths (Sec. 3.1; Table 9). While SFT degrades as noise increases, the RL stage consistently mitigates this cascaded error propagation, as its rule-based accuracy and validity rewards do not rely on the extracted paths.

5 CONCLUSION

In this paper we introduce Path-Augmented Retrieval and Reasoning (PARR) for KG completion. We propose to leverage grounding paths extracted from interpretable link predictors to supervise both retrieval and reasoning for LLM-based KGC. In addition, we train the resulting single-LLM rewrite–retrieve–reason pipeline end-to-end with GSPO, the first exploration of RL-based finetuning for KGC, with carefully designed rewards for both rewriting and reasoning, achieving notable gains while being data efficient. By experimenting on a wide range of datasets and task settings, we show PARR achieves superior performance against prior SOTA on most tested KGs with diverse scales.

AI USE STATEMENT

LLMs are the subject of this work: PARR finetunes Llama3-8B-Instruct and Qwen3-8B, and several baselines are built on LLMs. In preparing the manuscript, we used LLM-based assistants to polish the writing and to help format tables and figures. All research ideas, methods, experiments, and analyses are the authors' own; the authors reviewed all LLM-assisted text and take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

Sec. 3 describes every component of PARR: grounding-path extraction, the construction of the rewriting and reasoning data (Sec. 3.3 and 3.4), and SFT and end-to-end GSPO with all reward definitions (Sec. 3.5 and 3.6). Appendix F details the baselines, the evaluation protocol, the configurations and hyperparameters of SFT (Table 18), RL, the link predictors (Table 19), and the RETRIEVER, the reward computation, the construction of minimal rewriting sets, and dataset statistics and samples; Appendix G provides the full prompts with sample trajectories. All datasets are public benchmarks with standard splits, and all base LLMs, link predictors, and the embedding model we use are publicly available.

REFERENCES

Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A box embedding model for knowledge base completion. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9649–9661, 2020.

Arvindh Arun, Sumit Kumar, Mojtaba Nayyeri, Bo Xiong, Ponnurangam Kumaraguru, Antonio Vergari, and Steffen Staab. SEMMA: A semantic aware knowledge graph foundation model. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 31825–31848, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1621.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(61):1803–1831, 2010.

Ivana Balažević, 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)*, pp. 5185–5194. Association for Computational Linguistics, 2019. doi: 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 *Advances in Neural Information Processing Systems*, volume 26, pp. 2787–2795. Curran Associates, Inc., 2013.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, et al. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022.

Linlin Chao, Jianshan He, Taifeng Wang, and Wei Chu. PairRE: Knowledge graph embeddings via paired relation vectors. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4360–4369, 2021.

Chen Chen, Yufei Wang, Aixin Sun, Bing Li, and Kwok-Yan Lam. Dipping PLMs sauce: Bridging structure and text for effective knowledge graph completion via conditional soft prompting. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 11489–11503, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.729.

Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. HittER: Hierarchical transformers for knowledge graph embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10395–10407, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.812.

Xuanming Cui, Chionh Wei Peng, Adriel Kuek, and Ser-Nam Lim. Improving soft unification with knowledge graph embedding methods. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 11543–11562. PMLR, 2025.

Yuanning Cui, Zequn Sun, and Wei Hu. A prompt-based knowledge graph foundation model for universal in-context reasoning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7095–7124. Curran Associates, Inc., 2024. doi: 10.52202/079017-0227.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations*, 2018.

Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Manzil Zaheer, Hannaneh Hajishirzi, Robin Jia, and Andrew McCallum. Knowledge base question answering by case-based reasoning over subgraphs. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4777–4793. PMLR, 2022.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pp. 1811–1818, 2018. doi: 10.1609/aaai.v32i1.11573.

Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning. In *The Twelfth International Conference on Learning Representations*, pp. 31598–31619, 2024. URL https://openreview.net/forum?id=jVEoydFOl9.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1762–1777, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.99.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Lingbing Guo, Qiang Zhang, and Huajun Chen. Unleashing the power of transformer for graphs. *arXiv preprint arXiv:2202.10581*, 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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 140509–140534. Curran Associates, Inc., 2024. doi: 10.52202/079017-4460.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3929–3938. PMLR, 2020.

Chi Han, Qizheng He, Charles Yu, Xinya Du, Hanghang Tong, and Heng Ji. Logical entity representation in knowledge-graphs for differentiable rule learning. In *International Conference on Learning Representations*, 2023.

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*, pp. 26136–26152, 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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133. Curran Associates, Inc., 2020.

Xingyue Huang, Pablo Barceló, Michael M. Bronstein, Ismail Ilkan Ceylan, Mikhail Galkin, Juan L. Reutter, and Miguel Romero Orth. How expressive are knowledge graph foundation models? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 25021–25058. PMLR, 2025.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 874–880, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.74.

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. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

Pengcheng Jiang, Lang Cao, Cao Xiao, Parminder Bhatia, Jimeng Sun, and Jiawei Han. KG-FIT: Knowledge graph fine-tuning upon open-world knowledge. In *Advances in Neural Information Processing Systems*, volume 37, pp. 136220–136258. Curran Associates, Inc., 2024. doi: 10.52202/079017-4328.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572.

Charles Kemp, Joshua B. Tenenbaum, Thomas L. Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pp. 381–388. AAAI Press, 2006.

Jaejun Lee, Chanyoung Chung, and Joyce Jiyoung Whang. InGram: Inductive knowledge graph embedding via relation graphs. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 18796–18809. PMLR, 2023.

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. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Qika Lin, Rui Mao, Jun Liu, Fangzhi Xu, and Erik Cambria. Fusing topology contexts and logical rules in language models for knowledge graph completion. *Information Fusion*, 90:253–264, 2023. doi: 10.1016/j.inffus.2022.09.020.

Ben Liu, Jihai Zhang, Fangquan Lin, Cheng Yang, and Min Peng. Filter-then-generate: Large language models with structure-text adapter for knowledge graph completion. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 11181–11195, Abu Dhabi, UAE, 2025a. Association for Computational Linguistics.

Yu Liu, Yanan Cao, Xixun Lin, Yanmin Shang, Shi Wang, and Shirui Pan. Enhancing large language model for knowledge graph completion via structure-aware alignment-tuning. In *Proceedings*

of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 20970–20984, Suzhou, China, 2025b. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1061.

Kangyang Luo, Yuzhuo Bai, Cheng Gao, Shuzheng Si, Zhu Liu, Yingli Shen, Zhitong Wang, Cunliang Kong, Wenhao Li, Yufei Huang, Ye Tian, Xuantang Xiong, Lei Han, and Maosong Sun. GLTW: Joint improved graph transformer and LLM via three-word language for knowledge graph completion. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11328–11344, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.591.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *International Conference on Learning Representations*, 2024.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5303–5315, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.322.

Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. YAGO3: A knowledge base from multilingual Wikipedias. In *7th Biennial Conference on Innovative Data Systems Research (CIDR 2015)*, Asilomar, California, USA, 2015.

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 *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 884–901, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.49.

Costas Mavromatis and George Karypis. GNN-RAG: Graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16682–16699, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.856.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018. doi: 10.1145/3191513.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pp. 8026–8037, 2019.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. DRUM: End-to-end differentiable rule mining on knowledge graphs. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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 text encoder with low-rank adaptations. In *Advances in Information Retrieval: 47th European Conference on Information Retrieval, ECIR 2025*, volume 15576 of *Lecture Notes in Computer Science*, pp. 123–129. Springer, 2025. doi: 10.1007/978-3-031-88720-8_21.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 641–651. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-1059.

Komal K. Teru, Etienne Denis, and William L. Hamilton. Inductive relation prediction by subgraph reasoning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9448–9457. PMLR, 2020.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 57–66, Beijing, China, 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-4007.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2071–2080. PMLR, 2016.

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, 2021a. doi: 10.1145/3442381.3450043.

Hongwei Wang, Hongyu Ren, and Jure Leskovec. Relational message passing for knowledge graph completion. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1697–1707, 2021b.

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, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.580.

Wenhan Xiong, Thien Hoang, and William Yang Wang. DeepPath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 564–573. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1060.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6575.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, volume 30, pp. 2319–2328. Curran Associates, Inc., 2017.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259.

Liang Yao, Chengsheng Mao, and Yuan Luo. KG-BERT: BERT for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 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. In *ICASSP 2025 – 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2025. doi: 10.1109/ICASSP49660.2025.10889242.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 201–206, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-2033.

Jason Youn and Ilias Tagkopoulos. KGLM: Integrating knowledge graph structure in language models for link prediction. In *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pp. 217–224, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.starsem-1.20.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5773–5784. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.396.

Yongqi Zhang and Quanming Yao. Knowledge graph reasoning with relational digraph. In *Proceedings of the ACM Web Conference 2022*, pp. 912–924. ACM, 2022. doi: 10.1145/3485447.3512008.

Yucheng Zhang, Beatrice Bevilacqua, Mikhail Galkin, and Bruno Ribeiro. TRIX: A more expressive model for zero-shot domain transfer in knowledge graphs. In *Proceedings of the Third Learning on Graphs Conference*, volume 269 of *Proceedings of Machine Learning Research*, pp. 12:1–12:28. PMLR, 2025.

Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3065–3072, 2020.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

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. *World Wide Web*, 27(5):58, 2024. doi: 10.1007/s11280-024-01297-w.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural Bellman-Ford networks: A general graph neural network framework for link prediction. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29476–29490, 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. In *Advances in Neural Information Processing Systems*, volume 36, pp. 59323–59336. Curran Associates, Inc., 2023. doi: 10.52202/075280-2591.

Xingrui Zhuo, Jiapu Wang, Gongqing Wu, Zhongyuan Wang, Jichen Zhang, Shirui Pan, and Xindong Wu. Knowledge reasoning language model: Unifying knowledge and language for inductive knowledge graph reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026.

A LIMITATIONS

While PARR achieves superior performance and generalizability, it is currently limited to a single-task link predictor, and is limited to each individual KG dataset. Future directions can resolve these limitations by extending PARR to a multi-dataset, multi-task foundational KG LLM that can perform any KG-related task across different KG domains.

B RELATED WORK

B.1 KNOWLEDGE GRAPH COMPLETION

Knowledge Graph Embedding. 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; Balažević et al., 2019) is an effective and efficient method for KGC. KGE models embed entities and relations into continuous spaces, but typically operate as black-box functions without interpretability.

Interpretable link prediction. Another line of work explores KG reasoning via GNN-based or rule-based methods. NBFNet (Zhu et al., 2021) solves link prediction as a shortest path finding problem and parametrizes the Bellman-Ford algorithm with a GNN. A*Net (Zhu et al., 2023) improves the scalability of NBFNet by incorporating the 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 extend the backward chaining algorithm into continuous space. KG foundation models such as ULTRA (Galkin et al., 2024), TRIX (Zhang et al., 2025), KG-ICL (Cui et al., 2024), MOTIF (Huang et al., 2025), and SEMMA (Arun et al., 2025) are pretrained on multiple KGs to generalize zero-shot to unseen entities and relations, but remain discriminative scorers over a fixed entity set.

Link prediction with Language Models. Leveraging pretrained Language Models (LMs) for link prediction has been extensively explored. Earlier works leverage BERT as a GNN encoder to encode triplets (Guo et al., 2022; Chen et al., 2021), while others use BERT-like LMs to encode textual information of KG entities and relations (Yao et al., 2019; Wang et al., 2021a; Lin et al., 2023; Youn & Tagkopoulos, 2023). These works mostly follow the same training regime used in KGE methods, which turn generative LMs into discriminative link predictors.

Recent works have explored LLMs for link prediction through finetuning (Yao et al., 2025; Zhu et al., 2024), but their performance lags significantly behind traditional KGE methods. Other methods combine structural encoders or embeddings with PLMs/LLMs (Chen et al., 2023; Jiang et al., 2024; Luo et al., 2025; Liu et al., 2025b); these improve accuracy, but none of them generates explicit reasoning paths. 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 (e.g., ChatGPT), external KGE systems, and costly multi-round QA. Such LLM-based reranking (Wei et al., 2023; Liu et al., 2025a) is a separate stage applied after candidate generation, and is thus complementary to PARR. MKGL (Guo et al., 2024) modifies the embedding and output spaces of LLMs to act as specialized link predictors, and KRLM (Zhuo et al., 2026) further couples an LLM with NBFNet-based knowledge encoders and a GNN-based next-entity predictor that scores all entities of the KG. While effective, these approaches reduce 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.

B.2 RETRIEVAL-AUGMENTED REASONING

Retrieval-Augmented Generation. Retrieval-augmented generation (RAG) (Lewis et al., 2020) 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., 2023).

Query rewriting for RAG. Query rewriting improves retrieval by transforming complex queries into simpler sub-queries, or incorporating more information to bridge retrieval asymmetry. HyDE (Gao et al., 2023) uses LLMs to generate synthetic documents 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. Recent works further train LLMs to interleave search and reasoning end-to-end with outcome-based RL (Jin et al., 2025). PARR similarly trains rewriting and reasoning jointly in a single LLM, but supervises and rewards retrieval with grounding paths from link predictors, since KGC provides no retrieval labels.

Retrieval-Augmented KGQA. Another related field is 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, 2025) 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 ground-truth retrieval and proof paths, which are not available in KGC datasets. In contrast, PARR obtains the necessary structural supervision from interpretable link predictors.

C Complexity and Computation Analysis

C.1 Complexity

Although PARR introduces additional system components, its overall memory and inference cost scale sub-linearly *w.r.t.* the KG size. In contrast, the complexity of many existing GNN-based (Zhu et al., 2021) and LLM-based (Guo et al., 2024) link prediction methods grows **at least** linearly with the size of KGs (Zhu et al., 2021). Here we focus on the complexity of PARR itself. Pretrained link predictors are used only for data curation and never run at inference; their paths can also be dropped from test-time retrieval with minor loss (Table 15).

(1) Retriever. We adopt a FAISS (Johnson et al., 2021) IVF index for dense retrieval. The resulting retrieval cost is therefore sub-linear w.r.t. the number of stored triplets.

(2) LLM. A single LLM serves as both Rewriter and Reasoner and generates the whole rewrite–retrieve–reason trajectory. The input length (i.e., the query) is fixed, so the cost of the trajectory is determined by (i) the number of rewritten triplets, (ii) the number of retrieved triplets inserted into the context, and (iii) the number of generated output triplets. All three are governed by the average length of grounding paths and the fixed retrieval budget k, instead of the size of the KG. Empirically, as shown in Table 5, datasets of substantially different sizes exhibit nearly identical numbers of rewrites and retrievals, confirming that the per-query cost is effectively decoupled from KG scale. Since the two roles share one model, only a single LLM needs to be deployed.

Table 11 benchmarks inference latency and memory usage across datasets of varying scales. The results show that PARR achieves nearly $30\times$ lower latency than recent LLM-based methods such as KICGPT, while maintaining nearly constant inference time and memory consumption *w.r.t.* KG size, confirming the scalability of PARR.

C.2 Computational Cost

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 eight Nvidia A100 GPUs. We estimate training time as time per step $\times$ steps per epoch $\times$ number of epochs, using each model's default configuration. For evaluation, we measure the average time (in seconds) spent on one batch with a batch size of 8.

Discussion of system complexity. Although PARR has more components than traditional link predictors, we note the following:

(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

Table 10: Training time and memory consumption.

Model	WN18RR		FB15k-237		YAGO	
	t (hr)	Max Mem (G)	t (hr)	Max Mem (G)	t (hr)	Max Mem (G)
NBFNet	2.6	27	3	19	11	27
A*Net	1.9	5	1.2	14	4	15
MKGL	2.5	58	7.5	67	134	54
PARR	7.8	77	10.2	78	13	75

Table 11: Evaluation latency and memory usage.

Model	WN18RR		FB15k-237		YAGO	
	s/batch	Max Mem (G)	s/batch	Max Mem (G)	s/batch	Max Mem (G)
NBFNet	1.3	1	1.4	0.6	3.2	5
A*Net	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 + vLLM	5.4	74	5.9	75	6.1	74

Table 12: Results on three small, statistical KGs: Kinship, UMLS, and Nations. PARR_{RL} 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_{RL}**-Qwen3**	**0.665**	**0.977**	**0.753**	**0.981**	**0.670**	**0.973**

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 methods 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 11, on FB15k-237, KICGPT takes 176 seconds to evaluate 8 samples, whereas PARR takes only 5.9 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 (Wang et al., 2021b)) time/memory complexity w.r.t. the number of entities, edges, and embedding dimensions. In contrast, both the rewriting and reasoning stages of 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., 2021). As shown in Tab. 10, on YAGO3-10, MKGL requires 134 hours of training, while PARR achieves better performance with only 13 hours of training.

(4) Heavy data curation, straightforward deployment. A substantial part of the cost of PARR lies in data curation, which takes 9–11 hours per dataset. This, however, has become a normal practice in the era of LLMs, where data curation has become the most important and time-consuming process. We reuse pretrained link-predictor checkpoints whenever available and only train them otherwise; for YAGO3-10 and ogbl-wikikg2, we train A*Net for half of its default epochs, which already yields near-converged validation accuracy. On the other hand, PARR is straightforward to deploy in the real world, thanks to the development in LLM acceleration tools (e.g. vLLM) and scalable retrieval index (e.g. FAISS).

D MORE RESULTS

In Table 13 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.

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 12, 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 13: Performance on inductive knowledge graph reasoning. v1–v4 refer to the four 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
KRLM	-	0.701	-	**0.752**	-	0.680	-	0.699
PARR_{RL}-Qwen3	**0.522**	**0.713**	**0.534**	0.741	**0.466**	**0.697**	**0.441**	**0.712**
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
KRLM	-	0.800	-	0.799	-	0.595	-	0.738
PARR_{RL}-Qwen3	**0.775**	**0.866**	**0.719**	**0.824**	**0.449**	**0.610**	**0.648**	**0.752**

Table 14: REASONER inference 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

E MORE ANALYSIS

Setting of component ablations. To isolate the effect of individual components, the component ablations and analyses (Tables 5, 6, 8, 9, 14, 15, 16, and 17, Figures 3 and 4, and the first four GSPO rows of Table 7, which use only the reasoning rewards) are conducted in a more controlled setting, where the REWRITER and the REASONER are separately trained copies of the same base LLM: the REWRITER is trained with SFT on the MRS targets, and the REASONER with SFT on the CoT targets followed by RL on the reasoning rewards R_{acc}, R_{format}, and R_{val}. Both use the same grounding paths, training data, and RETRIEVER as PARR, but share no parameters and are not optimized jointly. A component of one stage (*e.g.*, the retrieval method, CoT supervision, the path source, retrieval noise, or a reasoning reward) can thus be varied while the other stage is held fixed, so that its effect is attributed to that component alone; in the unified model, a change to one stage also affects the other through the shared parameters and the end-to-end rollouts. These ablations therefore compare design choices within each table, and their absolute numbers are not directly comparable with the main results (Tables 2, 3, 1, 4, and 13), which are all obtained with the unified model. The last two rows of Table 7, which add the rewrite reward, are also obtained with the unified model.

Ablation on the inference robustness of PARR against Noisy Retrieval. Here we conduct a robustness analysis on how variations in path quality (e.g., using weaker or partial paths) could affect end-to-end link prediction performance. To do so, we add $n\%$ of noisy paths (grounding paths with bottom 5% path scores) to the retrieval. In Tab. 14 we show the performance (HITS@1) of the REASONER (Llama3-8B) at inference time, where the retrieval is randomly perturbed by $n\%$. The REASONER maintains decent performance (at most 1.6 points of HITS@1 drop) even when the perturbation rate is 25% for both datasets.

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_{SFT}-Qwen3 w/o paths). 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 path usage (HITS@1). Removing paths during retrieval (PARR_{SFT}-Qwen3 w/o paths) leads to only minimal degradation, thanks to the effective rewriting stage. This allows us to de-couple pre-trained link predictors from the inference stage.

Method	FB15k-237	WN18RR
ComplEx+RP	0.298	0.443
NBFNet	0.321	0.497
MKGL	0.325	0.500
PARR_{SFT}-Qwen3	**0.352**	**0.513**
PARR_{SFT}**-Qwen3 (w/o paths)**	0.346	0.508

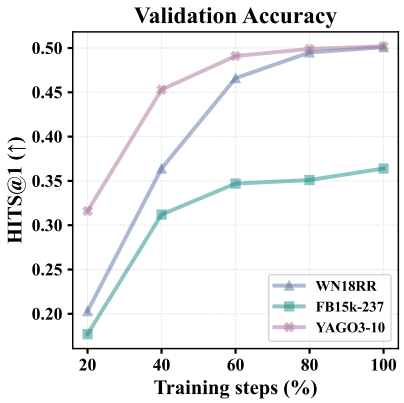

Figure 4: Validation accuracy (HITS@1) on 100 samples sampled from the validation set during SFT training.

Training convergence. As shown in Figure 4, we observe that PARR converges rapidly during SFT, 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 minimal degradation.

Ablation on validity reward weights. We study the sensitivity of our method to different configurations of validity reward weights in Table 16. Across a range of weight allocations, the performance remains consistently stable, with only minor variations in HITS@1/3/10. We observe that assigning non-zero weights to both $R_{\text{val-}c}$ and $R_{\text{val-}v}$ tends to yield slightly stronger results, suggesting that these components contribute more significantly to the overall validity signal. However, no single configuration dominates, and all settings perform within a narrow performance band. This indicates that our approach is robust to the exact weighting scheme of the validity reward, especially when compared with the choice of accuracy reward (Table 7).

Table 16: Ablation on validity reward weight configurations on FB15k-237. We report HITS@1/3/10.

$R_{\text{val-}c}$	$R_{\text{val-}v}$	$R_{\text{val-}l}$	H@1	H@3	H@10
1/3	1/3	1/3	0.384	0.501	0.633
1/2	1/2	0	0.365	0.483	0.613
1/2	0	1/2	0.363	0.481	0.613
0	1/2	1/2	0.360	0.477	0.610

Number of predicted answers. By default, PARR predicts a ranked list of $k = 10$ answers to support HITS@k evaluation (Sec. 3.1). Table 17 shows that training PARR to predict fewer answers further improves HITS@1, as expected, since producing a ranked list of ten answers is a harder task. This is also why prior native LLM-based methods restrict themselves to a single answer. Even in the default top-10 setting, PARR outperforms these single-answer baselines at HITS@1 by a large margin (Table 1).

Table 17: HITS@1 of PARR_{RL}-Qwen3 when trained to predict top-k answers.

Top-k	FB15k-237	WN18RR
$k = 10$ (default)	0.384	0.552
$k = 3$	0.413	0.580
$k = 1$	0.427	0.589

Table 18: Hyper-parameters for supervised finetuning, for both Llama3-8B Instruct and Qwen3-8B.

Dataset	LoRA Rank/Alpha	Epochs	Learning Rate	Batch Size	Gradient Accumulation Steps	Optimizer
FB15k-237	32/64	1	$1e^{-5}$	2	8	AdamW
WN18RR	32/64	1	$1e^{-5}$	2	8	AdamW
YAGO3-10	32/64	1	$1e^{-5}$	2	8	AdamW

Table 19: 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 the **Priority Function** parameters, 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

F Experimental Details

F.1 Baselines

We compare PARR against embedding-based models (Bordes et al., 2013; Yang et al., 2015; Trouillon et al., 2016; Sun et al., 2019; Balažević et al., 2019; Abboud et al., 2020; Zhang et al., 2020; Chao et al., 2021; Lee et al., 2023); GNN-based methods (Teru et al., 2020; Vashishth et al., 2020; Zhang & Yao, 2022; Zhu et al., 2021; Wang et al., 2021b); rule- and path-based methods (Das et al., 2018; Yang et al., 2017; Sadeghian et al., 2019; Han et al., 2023; Zhu et al., 2023); KG foundation models including ULTRA (Galkin et al., 2024), TRIX (Zhang et al., 2025), KG-ICL (Cui et al., 2024), MOTIF (Huang et al., 2025), and SEMMA (Arun et al., 2025); and methods that utilize pretrained language models, including KG-BERT (Yao et al., 2019), StAR (Wang et al., 2021a), KGLM (Youn & Tagkopoulos, 2023), FTL-LM (Lin et al., 2023), CSProm-KG (Chen et al., 2023), KG-FIT (Jiang et al., 2024), GLTW (Luo et al., 2025), SAT (Liu et al., 2025b), KG-Llama (Yao et al., 2025), KICGPT (Wei et al., 2023), MKGL (Guo et al., 2024), and KRLM (Zhuo et al., 2026). Apart from KICGPT, we do not compare against methods that use LLMs only to rerank the candidates of another model (*e.g.*, FtG (Liu et al., 2025a)), since reranking is a separate stage that is complementary to PARR.

F.2 IMPLEMENTATION DETAILS

Path Extraction. We utilize NBFNet and A*Net in this work to extract proofs. Hyperparameters are shown in Table 19. By default we adopt the same hyperparameters as the original works. For each model, we train separately twice with different message functions and random seeds.

RETRIEVER. For dense and hybrid retrievers, we use Jina-V3 (Sturua et al., 2025) text embedding model to compute and store the normalized embeddings, and retrieve using FAISS (Johnson et al., 2021) with IndexFlatIP index, which is equivalent to the cosine-similarity metric given pre-normalized embeddings. For dense retrieval, we observe similar performance when using Principal Neighbourhood 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 (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025), two state-of-the-art LLMs. We show hyper-parameters in Table 18. 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.

RL-based Finetuning. For end-to-end RL, we use GSPO (Zheng et al., 2025) with tree-structured rollouts (Sec. 3.6), sampling $K = 4$ rewrites per query and $M = 4$ reasoning rollouts per rewrite. We set the KL divergence coefficient (β) to 0.04 by default, and learning rate to be $1e^{-6}$. We sample 5K training samples for all tested transductive datasets, and 2.5K for inductive settings, where convergence is typically observed. All models are trained for one epoch.

Reward Calculation. Here we provide details on reward calculation. (1)–(3) are the reasoning rewards and (4) is the rewrite reward.

(1) R_{acc}: we use NDCG@10 as our accuracy reward.

$$R_{\text{acc}} = \frac{1}{\text{IDCG@10}} \sum_{i=1}^{10} \frac{\mathbb{I}[\hat{a}_i \in \mathcal{A}^\star]}{\log_2(i+1)}, \quad \text{IDCG@10} = \sum_{i=1}^{\min(10,\,|\mathcal{A}^\star|)} \frac{1}{\log_2(i+1)}$$

where $\hat{a}_i$ is the i-th unique predicted answer ranked by the REASONER; $\mathcal{A}^\star$ denotes the set of ground-truth answers; $\mathbb{I}[\cdot]$ returns 1 if true, or 0 otherwise. Counting only unique answers prevents reward hacking by repeating answers.

(2) R_{format}: R_{format} measures whether each generated path conforms to the proper format $(s, r^{(1)}, o^{(1)}) \to \cdots \to (o^{(n-1)}, r^{(n)}, o)$, for each individual path in the response. It returns 1 or 0 for each path. The final reward is calculated as:

$$R_{\text{format}} = \frac{1}{k} \sum_{i=1}^{k} R^i_{\text{format}}, \quad R^i_{\text{format}} \in \{0, 1\},$$

where k is the number of paths (answers) generated in the response.

(3) R_{val}: R_{val} is composed of three elements, each averaged over the k paths in the response:

- $R_{\text{val-}c}$: each path is contiguous, *i.e.*, the tail of each hop is the head of the next hop.[2] $R_{\text{val-}c}$ has the same setup as R_{format}:

$$R_{\text{val-}c} = \frac{1}{k} \sum_{i=1}^{k} R^i_{\text{val-}c}, \quad R^i_{\text{val-}c} \in \{0, 1\}.$$

- $R_{\text{val-}v}$: each triplet is a valid edge from the training KG. For each individual path, we measure the percentage of valid triplets against path length, and aggregate the results.
- $R_{\text{val-}l}$: each path does not contain loops. This also has the same setup as R_{format} and $R_{\text{val-}c}$, where we return 0 for a path if it contains a loop, or 1 otherwise. We then take the mean across all paths in one response.

[2]For simple link prediction, we only consider reasoning without intersection, union, or negation.

Table 20: Dataset statistics for the transductive setting. # SFT Train denotes the number of SFT trajectories.

Dataset	#Entity	#Relation	# Train	# Validation	# Test	# SFT 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	5,000	5,000	16,628,192	5

Table 21: 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

By default, the three elements are weighted equally:

$$R_{\text{val}} = \frac{1}{3}(R_{\text{val-}c} + R_{\text{val-}v} + R_{\text{val-}l}).$$

Table 16 shows that performance is robust to these weights.

The overall reasoning reward is then defined as:

$$R_{\text{rea}} = \frac{1}{2}R_{\text{acc}} + \frac{1}{4}\big(R_{\text{format}} + R_{\text{val}}\big).$$

(4) R_{rw}: for the j-th rewrite $\mathcal{Q}_j$ of query (s, r), with retrieval $\mathcal{G}_j$ and reasoning rollouts $\{y_{j,m}\}_{m=1}^{M}$, the rewrite reward is

$$R_{\text{rw}}(\mathcal{Q}_j) = \frac{1}{M}\sum_{m=1}^{M} R_{\text{acc}}(y_{j,m}) + \max_{p \in \mathcal{P}_{s,r}} \frac{|p \cap \mathcal{G}_j|}{|p|},$$

where $\mathcal{P}_{s,r}$ is the set of grounding paths extracted for (s, r), and $|p \cap \mathcal{G}_j|$ is the number of triplets of p retrieved by $\mathcal{Q}_j$.

F.3 Constructing Minimal Rewriting Sets

Given a proof and a fixed retrieval interface RETRIEVER, we want to find a minimal set of rewritten 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.

F.4 Dataset statistics

In Tables 20 and 21 we show dataset statistics for the transductive and inductive settings, respectively. We follow the standard splits in their original works.

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**
                $s \leftarrow s \cup \{k\}$
                $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{Q}\}$
            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 proof triplets 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 $\mathcal{S}$

Table 22: Samples of the rewriting part of the SFT data for WN18RR. Entity in red refers to the target entity, which is missing from the query.

Query	Rewritten 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 23: Samples of the reasoning part of the SFT data 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)*

F.5 Dataset samples

To provide a clearer understanding of the structure and supervision signals used in our framework, we present representative examples of both the rewriting and reasoning data derived from WN18RR (Dettmers et al., 2018) in Table 22 and Table 23, respectively.

The rewriting data (Table 22) decomposes a missing-entity query into a set of logically connected sub-queries over WordNet relations. Given an incomplete triple where the target entity is masked (highlighted in red), the model is trained to generate intermediate relational paths that recover the missing semantic information. These sub-queries often involve relation composition (e.g., hypernym chains), inverse relations, and auxiliary semantic links, which together form a structured decomposition of the original query. This encourages the model to explicitly reason over multi-hop relational structure rather than relying on single-step inference.

In contrast, the reasoning data (Table 23) provides richer supervision in the form of retrieved candidate facts and explicit proof chains. Given a query and a large set of potentially relevant facts retrieved from the knowledge graph, the model is trained to construct a coherent reasoning path that leads to the correct target entity. The proof traces illustrate how intermediate relational steps connect retrieved evidence to the final prediction, including the use of inverse relations and derivationally related forms (e.g., *drf*). This setup allows the model to learn structured multi-hop reasoning while grounding each inference step in observed facts.

Together, the two parts of each SFT trajectory provide complementary supervision: the rewriting part focuses on decomposing complex queries into executable sub-queries, while the reasoning part emphasizes grounded inference over retrieved evidence chains.

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, ?), you will first 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. Given the retrieved set of relevant triplets, you will then provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.

User:
Query:
(carry, verb group, ?)
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.

Assistant:
Rewrite:[('carry', 'hypernym', 'make up'), ('right', 'also see', 'make up')]

User:
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.

Assistant:
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

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, ?), you will first 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. Given the retrieved set of relevant triplets, you will then provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.

User:
Query:
(thread, 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.

Assistant:
Rewrite:[('threader', 'derivationally related form', 'thread'), ('run', '(inverse) verb group', 'thread')]

User:
Retrieved triplets:
('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.

Assistant:
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')

Answer:run

Figure 5: Sample rewrite–retrieve–reason trajectory of Llama3-based PARR. Generated segments are shown in dark green and triplets inserted by the RETRIEVER in blue.

G SAMPLE PROMPTS AND GENERATED RESPONSES

Since PARR uses a single LLM for both stages, each sample is one multi-turn trajectory: the model first generates the rewritten triplets, the RETRIEVER then inserts the retrieved triplets as a new user turn, and the same model continues with the reasoning and the final answer. Figures 5 and 6 show such trajectories on WN18RR for Llama3- and Qwen3-based PARR, respectively, and Figure 7 shows a trajectory of Qwen3-based PARR with the multi-answer extension.

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, ?), you will first 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. Given the retrieved set of relevant triplets, you will then provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.

User:
Query:
(carry, verb group, ?)
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.

Assistant:
Rewrite:[('carry', 'hypernym', 'make up'), ('right', 'also see', 'make up')]

User:
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.

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

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, ?), you will first 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. Given the retrieved set of relevant triplets, you will then provide the reasoning step by step based on the retrieved triplets, and provide the appropriate object entity that completes the query.

User:
Query:
(thread, 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.

Assistant:
Rewrite:[('threader', 'derivationally related form', 'thread'), ('run', '(inverse) verb group', 'thread')]

User:
Retrieved triplets:
('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.

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

Figure 6: Sample rewrite–retrieve–reason trajectory of Qwen3-based PARR. Generated segments are shown in dark green and triplets inserted by the RETRIEVER in blue.

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, ?), you will first 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. Given the retrieved set of relevant triplets, you will then provide the top-10 most probable reasoning step by step based on the retrieved triplets, and provide the corresponding top-10 appropriate object entities that complete the query.

User:
Query:
(carry, verb group, ?)
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.

Assistant:
Rewrite:[('carry', 'hypernym', 'make up'), ('right', 'also see', 'make up')]

User:
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 complete the query? Please provide a rigorous reasoning process for each before the final answer.

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, ...]

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, ?), you will first 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. Given the retrieved set of relevant triplets, you will then provide the top-10 most probable reasoning step by step based on the retrieved triplets, and provide the corresponding top-10 appropriate object entities that complete the query.

User:
Query:
(thread, 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.

Assistant:
Rewrite:[('threader', 'derivationally related form', 'thread'), ('run', '(inverse) verb group', 'thread')]

User:
Retrieved triplets:
('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 are the top-10 appropriate object entities that complete the query? Please provide a rigorous reasoning process for each before the final answer.

Assistant:
<think>
Based on the retrieved context, we can deduce below top-10 proofs:
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, ...]

Figure 7: Sample rewrite–retrieve–reason trajectory of Qwen3-based PARR with the multi-answer extension. Generated segments are shown in dark green and triplets inserted by the RETRIEVER in blue.