GROUNDED LATENT REASONING FOR KNOWLEDGE GRAPH QUESTION ANSWERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Knowledge graph question answering (KGQA) aims to identify the entities and relations in a knowledge graph needed to answer a given question. Recent approaches use large language models (LLMs) to generate textual relation paths, but this introduces a discrete vocabulary bottleneck because the relations generated by an LLM do not necessarily match the relation vocabulary of the KG. We propose grounded latent reasoning (LATENTRAG), a framework that performs intermediate reasoning in continuous latent space and directly grounds each latent reasoning step in the KG. Given a question, an LLM first generates a sequence of continuous latent thoughts without decoding intermediate reasoning steps into discrete tokens. Each latent thought is then decoded into scores over the KG's actual relations, which guide entity propagation through the graph and produce grounded multi-hop reasoning paths. The retrieved paths are subsequently provided to a downstream reasoning module to generate the final answer. This design allows the LLM to reason without committing to textual relation names while ensuring that each graph reasoning step follows an actual KG relation. Experiments on different datasets show the effectiveness of the proposed method.

1 INTRODUCTION

Knowledge graph question answering (KGQA) aims to answer natural-language questions by reasoning over a structured knowledge graph (KG), which represents facts as $(\text{subject}, \text{relation}, \text{object})$ triples. Given a question and a topic entity, a KGQA system typically identifies relations and entities to answer the question (He et al., 2021; Shi et al., 2021; Zhang et al., 2022). Because the reasoning process is grounded in an external and structured graph, the relations selected at each hop determine which part of the graph is explored and, ultimately, which entities can be reached. An incorrect relation at any hop can therefore lead the reasoning process away from the correct answer, making accurate relation selection a central challenge in KGQA.

Large language models (LLMs) have recently been used to guide this process because of their strong language understanding and reasoning capabilities (Wei et al., 2022; Sun et al., 2024; Mavromatis & Karypis, 2024). A common approach is to formulate KG reasoning as a planning problem, where an LLM generates a sequence of relations in natural language, such as `born_in` $\rightarrow$ `city_of`, and the generated relation path is subsequently matched against the KG to retrieve and verify candidate paths (Luo et al., 2024). However, this formulation introduces a fundamental mismatch between the LLM and the KG. The LLM generates text from its own token vocabulary, whereas the KG defines relations using a separate, domain-specific vocabulary determined by its schema and construction process. These two vocabularies are not necessarily aligned. For example, an LLM may correctly infer that a question requires the relation "place of birth," but generate `place_of_birth` when the KG uses `born_in`, or generate `location.born` when the KG uses `people.person.place_of_birth`. The resulting retrieval failure does not necessarily indicate incorrect reasoning. Instead, it arises because the LLM must commit to a particular textual relation before its output can be matched to the KG's actual relation vocabulary. Thus, although textual planning provides an intuitive interface between LLM reasoning and KG traversal, it imposes a discrete vocabulary bottleneck between the two.

An alternative line of research explores latent reasoning, where an LLM performs intermediate reasoning in its continuous hidden representation space rather than explicitly decoding every intermediate

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.

step into a natural-language token (Hao et al., 2024; Yue et al., 2025; Deng et al., 2026; Shen et al., 2025; Xu et al., 2025; Geiping et al., 2025). Instead of generating a token and then converting that token back into a continuous representation for subsequent computation, the model can feed its hidden representation directly into the next reasoning step. This allows intermediate reasoning to remain in a continuous space without requiring the model to commit to a particular discrete symbol at every step. As a result, latent reasoning can preserve information that may be difficult to express through a single token and can avoid unnecessary dependence on the model's discrete vocabulary. This property makes latent reasoning particularly appealing for KGQA, where the vocabulary used to express an intermediate reasoning step may not coincide with the relation vocabulary defined by the KG.

This motivates us to avoid asking an LLM to explicitly generate a textual relation path. Instead, we let the LLM perform latent reasoning over the question and decode each latent reasoning step directly into the KG's actual relation space. Based on this idea, we propose grounded latent reasoning (LATENTRAG), which connects continuous latent reasoning with the concrete structure of a KG. LATENTRAG consists of three stages. First, an LLM backbone generates a sequence of continuous latent thoughts autoregressively, with each thought conditioned on the preceding latent thoughts through hidden-state feedback. Second, each latent thought is decoded into scores over the KG's actual relations, restricting the reasoning process to relations that are available and reachable from the current entities. These relation scores are then used to propagate an entity distribution through the KG, producing grounded candidate paths across multiple hops. Finally, the retrieved paths, together with the original question, are provided to a downstream reasoning module that generates the final answer. By decoding latent thoughts directly into the KG's relation space, LATENTRAG avoids requiring the LLM to generate an exact textual name for each relation while keeping every reasoning step grounded in the actual graph. Experiments on multiple KGQA benchmarks demonstrate the effectiveness of LATENTRAG.

We make the following contributions:

- We are the first to combine latent reasoning with KGQA, replacing explicit textual relation-path generation with continuous latent reasoning and demonstrating its effectiveness on real-world KGQA benchmarks.
- We propose grounded latent reasoning (LATENTRAG), which decodes each latent reasoning step directly into the KG's actual relation space, allowing continuous reasoning while maintaining explicit grounding in the graph structure.
- We conduct extensive experiments on multiple KGQA benchmarks, demonstrating that LATENTRAG achieves strong performance across datasets and consistently benefits from grounded latent reasoning.

2 PRELIMINARIES AND PROBLEM DEFINITION

A knowledge graph (KG) $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ consists of a set of entities $\mathcal{E}$, a set of relations $\mathcal{R}$, and a set of triples $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$. Each triple (e, r, e') indicates that entity e is connected to entity e' by relation r. Given a natural-language question q and a topic entity $e_0 \in \mathcal{E}$, KGQA aims to identify an answer entity or a set of answer entities that can be reached from e_0 by traversing the KG. We consider multi-hop KGQA, where answering a question may require following several relations in sequence. For example, a question may require first following a `born_in` relation from a person to a city and then a `country` relation from that city to a country. More generally, a reasoning path can be written as $e_0 \xrightarrow{r_1} e_1 \xrightarrow{r_2} \cdots \xrightarrow{r_T} e_T$, where each $(e_{t-1}, r_t, e_t) \in \mathcal{T}$ and e_T is a candidate answer. The main challenge is to select the appropriate relation at each hop while keeping the resulting traversal grounded in the actual graph.

2.1 LATENT REASONING IN LANGUAGE MODELS

Large language models typically perform reasoning through autoregressive token generation. Let f_θ denote a causal decoder-only language model and let h_t denote its hidden representation at an intermediate position. In standard autoregressive generation, the model converts this representation

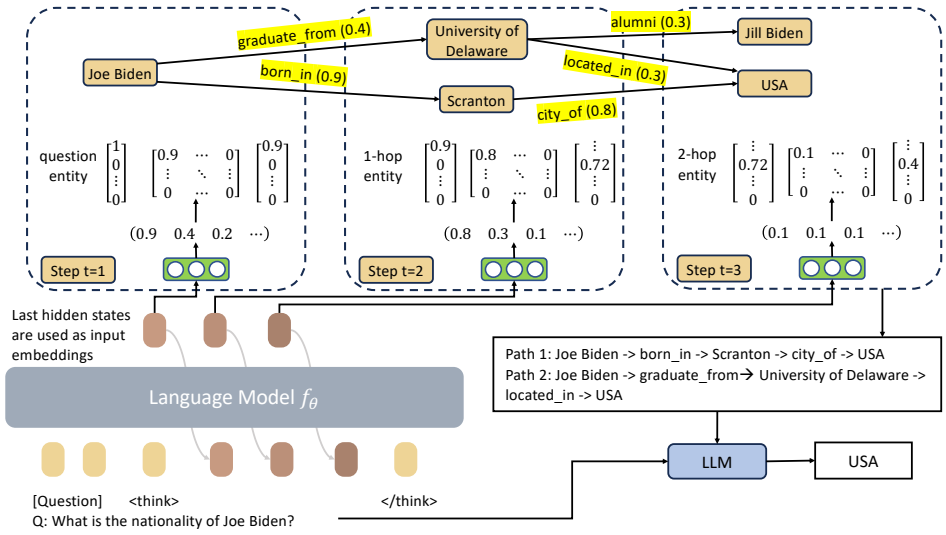

Figure 1: Overview of LATENTRAG.

into a distribution over its vocabulary and samples or selects the next token:

$$x_{t+1} \sim \mathrm{softmax}(Wh_t), \qquad h_{t+1} = f_\theta\big(\mathrm{Embed}(x_{t+1})\big), \tag{1}$$

where W is the output projection and $\mathrm{Embed}(\cdot)$ maps a discrete token back into the model's embedding space. Thus, an intermediate reasoning step is normally expressed as a discrete token before the model continues to the next step. Latent reasoning removes this discrete bottleneck from intermediate computation. Instead of decoding the current hidden representation into a token and feeding that token back into the model, the model directly feeds its continuous representation into the next reasoning step (Hao et al., 2024): $h_{t+1} = f_\theta(h_t)$. The model can consequently perform a sequence of intermediate computations entirely in its continuous representation space. These latent states do not need to correspond to individual words or tokens, allowing the model to carry out intermediate reasoning without committing to a particular textual representation at every step. When combining LLM text reasoning with latent reasoning, people usually use two learned embeddings, $\mathbf{e}_{\texttt{<think>}}$ and $\mathbf{e}_{\texttt{</think>}} \in \mathbb{R}^d$, to mark the beginning and end of a latent reasoning span (Hao et al., 2024). These embeddings delimit the portion of the computation that is performed in latent space and distinguish it from the surrounding textual context. A single latent reasoning step may also contain multiple hidden-state transitions. Given an initial representation h_0, we define $h_i = f_\theta(h_{i-1})$, $i = 1, \ldots, c$, where c denotes the number of latent transitions used for one reasoning step. The final representation h_c is passed to the next component of the model, while the intermediate states provide additional computation in latent space.

3 METHODS: GROUNDED LATENT REASONING

Existing LLM-based KGQA methods often ask an LLM to generate a textual relation path, but the generated relations may not match the relation vocabulary of the KG, or the generated path may not exist in the graph at all. This creates a mismatch between the LLM's reasoning and the KG structure, even when the LLM has identified the correct reasoning direction. To address this problem, we introduce LATENTRAG, which replaces explicit textual path generation with continuous latent reasoning. Instead of generating a relation name at each hop, the LLM generates a sequence of continuous latent "thoughts," with each thought representing the reasoning state for one hop. Each latent thought is then decoded directly into scores over the KG's actual relations, restricted to those reachable from the current entity state. These grounded relation scores are used to traverse the KG and update the entity distribution until the answer entities are reached. In this way, LATENTRAG allows the LLM to reason in continuous latent space while grounding every reasoning step in the actual structure of the KG (Figure 1).

3.1 LLM LATENT REASONING

Following the latent reasoning paradigm (Hao et al., 2024), we allow the LLM to perform intermediate reasoning directly in its continuous hidden-state space rather than generating explicit text tokens. Given a question q, we first feed it to the LLM and append a special latent-thought token $\mathbf{e}_{\texttt{<think>}}$ to indicate the beginning of latent reasoning:

$$H^{(q)} = f_\theta(q), \qquad h_0 = f_\theta\left(\mathbf{e}_{\texttt{<think>}} \mid q\right). \tag{2}$$

Starting from h_0, the model generates a sequence of latent thoughts by feeding the previous hidden state directly back into the model:

$$h_t = f_\theta(h_{t-1} \mid q), \qquad t = 1, \ldots, T. \tag{3}$$

Here, $h_t \in \mathbb{R}^d$ represents the t-th latent thought. Unlike conventional autoregressive generation, h_t is not projected onto the vocabulary and converted into a discrete token before the next reasoning step. Instead, it is directly used as the input representation for producing h_{t+1}. The model can therefore perform a sequence of intermediate reasoning steps entirely in continuous space:

$$q \to \mathbf{e}_{\texttt{<think>}} \to h_1 \to h_2 \to \cdots \to h_T \to \mathbf{e}_{\texttt{</think>}}. \tag{4}$$

The latent states are not required to correspond to human-readable words or predefined relation names. They serve as continuous intermediate representations of the model's reasoning, which are subsequently decoded into KG-specific relations by the grounding module described in Section 3.2. We use a learned end-of-thought embedding $\mathbf{e}_{\texttt{</think>}}$ to terminate the latent reasoning span after the final latent thought.

3.2 DECODING A LATENT THOUGHT INTO GRAPH STRUCTURE

Once a latent thought is obtained, it is grounded in the knowledge graph by decoding it into relation scores and using these scores to guide graph traversal. At each reasoning step, the latent thought is mapped to a score for each relation in the KG. These relation scores define a transition matrix over the actual edges of the graph, which is then used to propagate the current entity distribution by one hop (Zhao et al., 2024). After T reasoning steps, the entity distributions from different depths are combined to obtain the final answer distribution. The entire reasoning process is differentiable and trained end to end.

Relation Score Calculation. Let h_{t-1} denote the latent thought at reasoning step t. We first decode this latent representation into scores over the KG relation set $\mathcal{R}$:

$$s_t = \sigma\left(Q_{\text{rel}} h_{t-1} + b_{\text{rel}}\right) \in [0,1]^{|\mathcal{R}|}, \tag{5}$$

where $\sigma(\cdot)$ is the element-wise sigmoid function. Thus, $s_t[r]$ measures how strongly relation r is associated with the current reasoning step. We use independent sigmoid scores rather than a softmax because multiple relations may be relevant at the same step. The relation scores do not themselves define new edges in the graph. Instead, they are used to assign weights to the edges that already exist in the KG.

Step-by-Step Graph Reasoning. We next convert the relation scores into a transition matrix over the KG. Let $\mathcal{E}$ denote the entity set, and let M encode the relations associated with the edges of the KG. The transition matrix $W_t \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{E}|}$ is defined as

$$(W_t)_{ij} = \begin{cases} s_t[r], & \text{if } (e_i, r, e_j) \in \mathcal{T}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

In other words, an existing KG edge from e_i to e_j receives the score of its corresponding relation, while pairs of entities that are not connected by an edge receive zero transition weight. This construction explicitly grounds the reasoning process in the observed KG structure. The model can therefore learn which existing relations to follow. Let $p_{t-1} \in \mathbb{R}^{|\mathcal{E}|}$ denote the entity distribution after step $t-1$. Starting from the question entity, the distribution is propagated through the transition matrix:

$$p_t = p_{t-1} W_t, \tag{7}$$

where p_0 is initialized as a one-hot vector at the question entity. The resulting p_t assigns scores to entities reachable through the relations selected at step t. Repeating this process allows the model

to explore multi-hop paths in the KG while maintaining a soft distribution over candidate entities. Since the transition matrix is sparse, the matrix multiplication can be computed efficiently, making the propagation process computationally inexpensive.

Combining Reasoning Steps. After T reasoning steps, we use the question embedding q to compute a weight distribution β over the steps, then take the weighted sum of the per-step entity scores to obtain the final entity distribution (Zhao et al., 2024):

$$\beta = \mathrm{softmax}\left(\mathrm{MLP}(q)\right), \qquad \hat{p} = \sum_{t=1}^{T} \beta_t p_t. \tag{8}$$

The resulting distribution $\hat{p}$ is used to score candidate answer entities. All components, including latent reasoning, relation decoding, graph propagation, and hop aggregation, are jointly optimized toward the final answer objective.

Training Loss. The model is trained end to end using standard backpropagation. Since latent reasoning, relation decoding, graph propagation, and hop aggregation are differentiable, the final answer loss can propagate gradients through the entire reasoning process and into the latent thoughts generated by the language model. Let $y \in \{0,1\}^{|\mathcal{E}|}$ denote the multi-hot vector of gold answer entities, and let $\hat{p}$ denote the final entity scores from Eq. 8. We optimize a weighted mean-squared error with learnable weights w:

$$\mathcal{L}_{\mathrm{ent}} = \frac{\sum_{e \in \mathcal{E}} w_e \left(\hat{p}_e - y_e\right)^2}{\sum_{e \in \mathcal{E}} w_e}, \tag{9}$$

When intermediate relation annotations are available, we additionally supervise the relation scores at the corresponding reasoning steps. Let $y_{\mathrm{rel}}^{(t)} \in \{0,1\}^{|\mathcal{R}|}$ denote the multi-hot vector of gold relations for step t. We use binary cross-entropy to encourage the model to assign high scores to gold relations:

$$\mathcal{L}_{\mathrm{rel}} = \frac{1}{T_{\mathrm{sup}}} \sum_{t=1}^{T_{\mathrm{sup}}} \mathrm{BCE}\left(s_t, y_{\mathrm{rel}}^{(t)}\right), \tag{10}$$

where $T_{\mathrm{sup}} \leq T$ is the number of reasoning steps for which relation supervision is available. The overall training objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{ent}} + \lambda \mathcal{L}_{\mathrm{rel}}, \tag{11}$$

where λ controls the contribution of the auxiliary relation supervision. For questions without an annotated relation at a particular step, that step is excluded from $\mathcal{L}_{\mathrm{rel}}$.

3.3 Reasoning Path Generation

After T reasoning steps, the model produces the final answer distribution $\hat{p}$, together with the relation scores $s_1, \ldots, s_T$ and the intermediate entity distributions $p_1, \ldots, p_T$. We use these outputs to reconstruct explicit reasoning paths for downstream reasoning and analysis (Zhao et al., 2024). Path reconstruction is a deterministic post-processing procedure and introduces no additional learned parameters or gradients. Starting from the topic entity, we traverse actual triples in the knowledge graph according to the relation scores predicted at each reasoning step and serialize the resulting walks as explicit paths.

Initialization. Let e_0 denote the topic entity associated with the question. We initialize the path set with the zero-hop path containing only e_0: $\mathcal{P}_0 = \{(e_0)\}$. At reasoning step τ, we use the predicted relation scores s_τ to determine which relations can be used to extend the current paths.

Extraction. For $\tau = 1, \ldots, \hat{t}$, each partial path is extended by following every KG triple $(e, r, e') \in \mathcal{T}$. The relation and its score are appended to the path. A resulting path of length $\hat{t}$ therefore has the form

$$e_0 \xrightarrow{r_1} e_1 \xrightarrow{r_2} \cdots \xrightarrow{r_{\hat{t}}} e_{\hat{t}},$$

where every edge $(e_{\tau-1}, r_\tau, e_\tau)$ is an observed triple in $\mathcal{T}$. For a path $\pi = (e_0, r_1, e_1, \ldots, r_{\hat{t}}, e_{\hat{t}})$, we define its score as the mean of the relation scores along the path:

$$S(\pi) = \frac{1}{\hat{t}} \sum_{\tau=1}^{\hat{t}} s_\tau[r_\tau]. \tag{12}$$

Reranking. We next select candidate endpoint entities according to the entity distribution at the chosen depth. Let $\mathcal{E}_K$ denote the top-K entities ranked by $p_{\hat{t}}$. We retain only extracted paths whose endpoints belong to $\mathcal{E}_K$. For each candidate entity, its paths are ranked according to $S(\pi)$, and the top N paths are retained:

$$\mathcal{P}_{K,N} = \bigcup_{e \in \mathcal{E}_K} \mathrm{TopN}\left(\{\pi : \mathrm{end}(\pi) = e\}, S(\pi)\right). \tag{13}$$

The resulting set $\mathcal{P}_{K,N}$ provides explicit symbolic representations of the model's reasoning over the KG, and will be used to generate the final answers.

3.4 Reasoning: Answering from Retrieved Paths

The retrieved path set $\mathcal{P}_{K,N}$ is provided to a separate reasoning module together with the original question q to generate the final answers (Luo et al., 2024; Zhao et al., 2024). Each path is serialized as a structured sequence, and the resulting paths are incorporated into an instruction prompt. The reasoning module is an LLM with parameters θ_{read}, independent of the backbone model f_θ used for latent reasoning and path retrieval. Given the question and retrieved paths, it generates the final answer as ordinary text:

$$a \sim P_{\theta_{\mathrm{read}}}\left(a \mid q, \mathcal{P}_{K,N}\right). \tag{14}$$

Its role is simply to interpret the retrieved symbolic paths in the context of the question and generate a natural-language answer. This separates graph-grounded path retrieval from answer generation. The proposed model determines which KG paths to retrieve, while the reasoning module uses these paths to produce the final textual response.

4 Experiments

4.1 Experiment Settings

Datasets. We evaluate the reasoning ability of LatentRAG on two widely used KGQA benchmarks, WebQuestionSP (WebQSP) (Yih et al., 2016) and Complex WebQuestions (CWQ) (Talmor & Berant, 2018). Both datasets contain questions that require reasoning over paths of up to four hops. Freebase (Bollacker et al., 2008) serves as the background knowledge graph for both datasets, containing approximately 88 million entities, 20,000 relations, and 126 million triples.

Baselines. We compare LatentRAG with 21 existing KGQA systems from five categories: (1) embedding-based methods, (2) retrieval-augmented methods, (3) semantic parsing methods, (4) LLMs without external knowledge graphs, and (5) LLMs augmented with knowledge graphs. We use LLaMA2-Chat-7B as the backbone, following RoG Luo et al. (2024).

Evaluation Metrics. Following prior work (Luo et al., 2024; Mavromatis & Karypis, 2024), we report Hits@1 and F1. Hits@1 measures the fraction of questions for which the top-ranked predicted answer is correct. F1 measures the balance between precision and recall over the complete gold answer set, which may contain multiple entities.

4.2 Main Results

Main Results. Main Results. Table 1 compares our method with representative KGQA baselines across embedding-based, retrieval-based, semantic-parsing, LLM, and LLM+KG approaches. LatentRAG achieves 88.1 Hits@1 and 74.9 F1 on WebQSP, outperforming all listed baselines on both metrics. Compared with GNN-RAG, our method improves Hits@1 by 7.5 points and F1 by 3.6 points, demonstrating the effectiveness of our grounded latent reasoning framework for KGQA.

Table 1: Performance comparison with different baselines on the two KGQA datasets. The results are from Luo et al. (2024) and Mavromatis & Karypis (2024).

Type	Method	WebQSP		CWQ	
		Hits@1	F1	Hits@1	F1
Embedding	KV-Mem (Miller et al., 2016)	46.7	34.5	18.4	15.7
	EmbedKGQA (Saxena et al., 2020)	66.6	–	45.9	–
	NSM (He et al., 2021)	68.7	62.8	47.6	42.4
	TransferNet (Shi et al., 2021)	71.4	–	48.6	–
	KGT5 (Saxena et al., 2022)	56.1	–	36.5	–
	BiNet Liu et al. (2022)	69.2	64.1	48.3	46.2
Retrieval	GraftNet (Sun et al., 2018)	66.4	60.4	36.8	32.7
	PullNet (Sun et al., 2019)	68.1	–	45.9	–
	SR+NSM (Zhang et al., 2022)	68.9	64.1	50.2	47.1
	SR+NSM+E2E (Zhang et al., 2022)	69.5	64.1	49.3	46.3
LLMs	Flan-T5-xl (Chung et al., 2022)	31.0	–	14.7	–
	Alpaca-7B (Taori et al., 2023)	51.8	–	27.4	–
	LLaMA2-Chat-7B (Touvron et al., 2023)	64.4	–	34.6	–
	ChatGPT (OpenAI, 2022)	66.8	–	39.9	–
	ChatGPT+CoT (OpenAI, 2022; Wei et al., 2022)	75.6	–	48.9	–
LLMs+KGs	KD-CoT (Wang et al., 2023)	68.6	52.5	55.7	–
	UniKGQA (Jiang et al., 2023)	77.2	72.2	51.2	49.1
	kG-COT (Zhao et al., 2024)	84.9	–	62.3	–
	RoG (Luo et al., 2024)	80.0	70.8	57.8	56.2
	GNN-RAG (Mavromatis & Karypis, 2024)	80.6	71.3	61.7	**59.4**
	LATENTRAG (Grounded Latent Reasoning, ours)	**88.1**	**74.9**	**63.4**	56.8

On CWQ, LATENTRAG achieves the highest Hits@1 of 63.4 among all listed methods, surpassing GNN-RAG by 1.7 points and kG-COT by 1.1 points. It also achieves an F1 of 56.8, exceeding RoG by 0.6 points and UniKGQA by 7.7 points. Taken together, these results demonstrate that LATENTRAG consistently delivers strong answer prediction performance across both WebQSP and CWQ, particularly in Hits@1, while maintaining competitive F1 performance.

4.3 ABLATION STUDY

Table 2: Performance with and without the final reasoning step.

Method	WebQSP		CWQ	
	Hits@1	F1	Hits@1	F1
w/o REASONING	65.97	50.15	35.00	25.38
w/ REASONING	**88.08**	**74.85**	**63.38**	**56.76**
Gain	**+22.11**	**+24.70**	**+28.38**	**+31.38**

Effectiveness of the Final Reasoning Model. Table 2 evaluates the contribution of the final REASONING step by comparing the full model with a variant that directly answers from the retrieved paths. Removing REASONING leads to substantial performance degradation on both datasets. On WebQSP, Hits@1 and F1 decrease from 88.08 to 65.97 and from 74.85 to 50.15, respectively, corresponding to gains of 22.11 and 24.70 points from the final reasoning step. The effect is even more pronounced on CWQ, where REASONING improves Hits@1 from 35.00 to 63.38 and F1 from 25.38 to 56.76, yielding gains of 28.38 and 31.38 points. These results show that retrieving relevant paths alone is insufficient for reliable answer prediction. The final REASONING step plays a critical role in interpreting and integrating the retrieved paths to identify the answer, with particularly large benefits on the more challenging CWQ dataset.

Latent embeddings per reasoning hop. Table 3 studies the effect of the number of latent embeddings used per hop. On WebQSP, performance is consistently strong across all settings, with $c = 2$ achieving

Table 3: Latent embeddings per hop (c) vs. performance on WebQSP and CWQ.

Total latent emb.	WebQSP		CWQ	
c	Hit@1 (%)	F1 (%)	Hit@1 (%)	F1 (%)
1	87.96	74.41	**63.47**	**56.76**
2	**88.08**	**74.85**	63.19	56.08
3	88.02	74.81	63.16	56.15
4	87.90	74.46	63.13	56.22

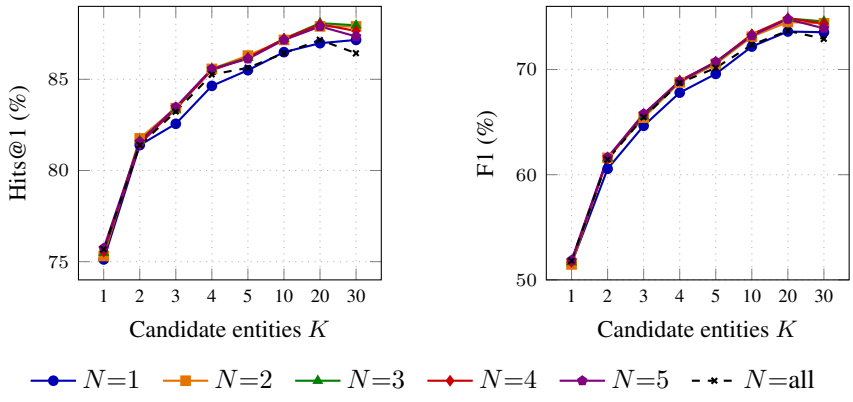

Figure 2: Ablation over the number of candidate entities K and the number of paths kept per entity N on WebQSP. Left: Hits@1. Right: F1.

the best Hits@1 and F1 scores of 88.08% and 74.85%, respectively. Increasing c beyond 2 provides no additional benefit, with $c = 3$ yielding nearly identical results and $c = 4$ performing slightly worse. On CWQ, $c = 1$ achieves the best performance, with Hits@1 and F1 scores of 63.47% and 56.76%, respectively. The differences across settings are small, with less than 0.4 points in Hits@1 and 0.7 points in F1. Overall, the results suggest that a small number of latent embeddings per hop is sufficient for effective latent reasoning.

Ablation over the number of candidate entities K and the number of paths kept per entity N. Figure 2 shows the effect of K and N on WebQSP. Performance improves steadily as K increases up to $K = 20$, while $K = 30$ brings no further gain and slightly lowers results for $N \geq 3$, suggesting that additional candidate entities introduce irrelevant information. Varying N has a much smaller effect than varying K, and moderate values ($N = 2$–5) generally outperform keeping all paths. The best Hits@1 is obtained with $K = 20$, $N = 3$ (88.08), while $N = 4$ reaches a nearly identical Hits@1 of 88.02 and the highest F1 of 74.87. We therefore use $K = 20$ and $N = 3$ as the default setting, which gives strong performance with a compact set of reasoning paths. Full results are reported in Table 5 in the appendix.

Case Studies. We further illustrate two retrieval cases from WebQSP in Table 4. Both RoG (Luo et al., 2024) and our method use the same LLM backbone, allowing us to examine how the retrieved paths affect the final answers. In the first case, RoG retrieves paths connecting Vietnam to tour operators and neighboring countries, which leads the LLM to generate incorrect answers. In contrast, our method retrieves direct paths from Vietnam to Vietnamese dong through relevant currency relations, enabling the LLM to answer correctly. In the second case, RoG retrieves paths from St. Louis Park to several entities through the generic `containedby` relation, including Minnesota, but fails to retrieve a path identifying Hennepin County. Our method retrieves direct paths connecting St. Louis Park to Hennepin County, allowing the LLM to produce the correct answer. These cases demonstrate that our retrieval strategy can identify more relevant reasoning paths and provide the LLM with the evidence needed to answer KGQA questions correctly.

Table 4: Case studies of retrieval on WebQSP. Both systems use the same LLM backbone.

Q: "What currency should you take to Vietnam?"	**A:** Vietnamese dong

KG-RAG (RoG)	`Vietnam → travel.tour_operator.travel_destinations → Adventures by Disney → travel.travel_destination.tour_operators → Cambodia`
	`Vietnam → travel.tour_operator.travel_destinations → Adventures by Disney → travel.travel_destination.tour_operators → Laos`
	LLM answer: Cambodia; Laos ✗

Ours	`Vietnam → location.country.currency_used → Vietnamese dong`
	`Vietnam → finance.currency.countries_used_reverse → Vietnamese dong`
	LLM answer: Vietnamese dong ✓

Q: "What county is St. Louis Park in?"	**A:** Hennepin County

KG-RAG (RoG)	`St. Louis Park → location.location.containedby → University of Phoenix-Minneapolis/St Paul Campus`
	`St. Louis Park → location.location.containedby → United States of America`
	`St. Louis Park → location.location.containedby → Minnesota`
	LLM answer: Minnesota ✗

Ours	`St. Louis Park → location.hud_county_place.county → Hennepin County`
	`St. Louis Park → location.location.containedby → Hennepin County`
	`St. Louis Park → location.location.containedby → United States of America`
	`St. Louis Park → location.location.containedby → Minnesota`
	LLM answer: Hennepin County ✓

5 Related Work

LLM-based reasoning over knowledge graphs. Recent KGQA methods increasingly use LLMs to guide reasoning over knowledge graphs (Sun et al., 2024; Mavromatis & Karypis, 2024; Luo et al., 2025; Wang et al., 2023; Jiang et al., 2023). RoG (Luo et al., 2024) generates textual relation paths and retrieves matching paths from the KG before final answer prediction. While effective, this approach requires generated relation names to match the KG vocabulary, introducing a discrete textual bottleneck between LLM reasoning and KG traversal. KG-CoT (Zhao et al., 2024) instead performs differentiable entity propagation through the KG's actual edges, providing explicit grounding without textual relation generation. However, its hop-level reasoning is independently derived from a fixed question representation and does not model sequential dependence between reasoning steps. Our approach combines the advantages of these two directions by performing sequential reasoning in continuous latent space while directly decoding each latent step into the KG's relation space.

Continuous latent reasoning. Coconut (Hao et al., 2024) and HRPO (Yue et al., 2025) explore latent reasoning by keeping intermediate reasoning states in continuous representation space rather than explicitly decoding them into text (Deng et al., 2026; Shen et al., 2025; Xu et al., 2025; Geiping et al., 2025). These methods demonstrate the potential of continuous latent reasoning for open-domain tasks, but their intermediate reasoning is not grounded in an external structured resource. In contrast, LatentRAG brings continuous latent reasoning to KGQA by directly decoding latent reasoning steps into the KG's actual relations, enabling sequential reasoning while maintaining explicit grounding in the graph. More broadly, our work builds on prior KGQA research using structured knowledge graphs as external sources of information (Ji et al., 2023; Yih et al., 2016; Talmor & Berant, 2018), while focusing specifically on the integration of latent reasoning and KG-grounded multi-hop reasoning.

6 Conclusion

In this work, we presented Grounded Latent Reasoning (LatentRAG) for knowledge graph question answering. Different from traditional methods, LatentRAG performs intermediate reasoning in continuous latent space and directly decodes each latent thought into scores over the KG's actual relation space. The resulting relation scores are used to propagate entity distributions through the graph, keeping the reasoning process grounded in the KG while avoiding the need to generate exact textual relation names. Experiments on multiple KGQA benchmarks demonstrate that LatentRAG effectively combines the flexibility of latent reasoning with the explicit grounding provided by the KG structure, leading to strong and consistent performance across datasets.

AI USE DISCLOSURE

We used generative AI tools to improve the clarity, readability, and grammar of the manuscript. No generative AI tools were used for research ideation, methodology design, experimentation, data analysis, or interpretation of results. All AI-assisted text was reviewed and revised by the authors, who take full responsibility for the final content of this work.

REFERENCES

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1247–1250, 2008.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022. URL https://arxiv.org/abs/2210.11416.

Jingcheng Deng, Zihao Wei, Liang Pang, Junhong Wu, Shicheng Xu, Zenghao Duan, and Huawei Shen. Latent-GRPO: Group relative policy optimization for latent reasoning. arXiv preprint arXiv:2604.27998, 2026.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. arXiv preprint arXiv:2502.05171, 2025.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. arXiv preprint arXiv:2412.06769, 2024.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining (WSDM), pp. 553–561, 2021.

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

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12), 2023.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph, 2023. URL https://arxiv.org/abs/2212.00959.

Lihui Liu, Boxin Du, Jiejun Xu, Yinglong Xia, and Hanghang Tong. Joint knowledge graph completion and question answering. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '22, pp. 1098–1108, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539289. URL https://doi.org/10.1145/3534678.3539289.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In International Conference on Learning Representations (ICLR), 2020.

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 (ICLR), 2024.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Yuan-Fang Li, Chen Gong, and Shirui Pan. Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models, 2025. URL https://arxiv.org/abs/2410.13080.

Costas Mavromatis and George Karypis. GNN-RAG: Graph neural retrieval for large language model reasoning. arXiv preprint arXiv:2405.20139, 2024.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1400–1409, 2016.

OpenAI. Introducing ChatGPT. https://openai.com/blog/chatgpt, 2022.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 4498–4507, 2020.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. Sequence-to-sequence knowledge graph completion and question answering. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 2814–2828, 2022.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. arXiv preprint arXiv:2502.21074, 2025.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. TransferNet: An effective and transparent framework for multi-hop question answering over relation graph. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 4149–4158, 2021.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. Open domain question answering using early fusion of knowledge bases and text. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 4231–4242, 2018.

Haitian Sun, Tania Bedrax-Weiss, and William Cohen. PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. 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. 2380–2390, 2019.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In International Conference on Learning Representations (ICLR), 2024.

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 (NAACL), 2018.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. Knowledge-driven CoT: Exploring faithful reasoning in LLMs for knowledge-intensive question answering. arXiv preprint arXiv:2308.13259, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Advances in Neural Information Processing Systems (NeurIPS), 2022.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. arXiv preprint arXiv:2502.12134, 2025.

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 (ACL), 2016.

Zhenrui Yue, Bowen Jin, Huimin Zeng, Honglei Zhuang, Zhen Qin, Jinsung Yoon, Lanyu Shang, Jiawei Han, and Dong Wang. Hybrid latent reasoning via reinforcement learning. arXiv preprint arXiv:2505.18454, 2025.

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 (ACL), pp. 5773–5784, 2022.

Ruilin Zhao, Feng Zhao, Long Wang, Xianzhi Wang, and Guandong Xu. KG-CoT: Chain-of-thought prompting of large language models over knowledge graphs for knowledge-aware question answering. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24), 2024.

K	Paths kept per entity (N)					
	1	2	3	4	5	All
1	75.12 / 51.56	75.31 / 51.47	75.49 / 51.69	75.55 / 51.70	75.74 / 51.88	75.68 / 51.82
2	81.39 / 60.56	81.76 / 61.60	81.63 / 61.57	81.45 / 61.50	81.57 / 61.65	81.39 / 61.44
3	82.56 / 64.65	83.42 / 65.42	83.35 / 65.56	83.42 / 65.73	83.48 / 65.80	83.23 / 65.46
4	84.64 / 67.80	85.57 / 68.78	85.57 / 68.89	85.50 / 68.94	85.57 / 68.86	85.26 / 68.73
5	85.50 / 69.58	86.30 / 70.53	86.18 / 70.65	86.18 / 70.70	86.12 / 70.74	85.63 / 70.15
10	86.49 / 72.18	87.16 / 73.12	87.22 / 73.29	87.22 / 73.35	87.16 / 73.21	86.43 / 72.35
20	86.98 / 73.61	87.90 / 74.51	**88.08** / 74.85	88.02 / **74.87**	87.90 / 74.78	87.16 / 73.66
30	87.16 / 73.54	87.90 / 74.39	87.96 / 74.57	87.65 / 74.36	87.35 / 73.91	86.43 / 72.91

Table 5: Ablation over the number of candidate entities K and the number of paths kept per entity N on WebQSP. Each cell reports Hits@1 / F1 (%).

A Training Hyperparameters

All experiments use `meta-llama/Llama-2-7b-hf` (Touvron et al., 2023) as the backbone. We apply LoRA (Hu et al., 2021) with rank 16, α=32, and dropout 0.05. We use RAdam (Liu et al., 2020) with a learning rate of 10^{-3} for newly added modules and 2×10^{-4} for the LoRA parameters, with weight decay 10^{-5}. The batch size is 32, and models are trained for 60 epochs with linear warmup over the first 10% of training steps. We clip gradients to a value of 0.5 with a maximum norm of 2. The training objective combines a weighted squared error on the final entity distribution with an auxiliary binary cross-entropy loss for per-hop relation prediction, with the auxiliary loss weighted by 1.0.

Effect of K and N. Table 5 examines how the number of candidate entities K and the number of paths retained per entity N affect the retrace-and-answer step. Increasing K provides substantial gains initially, with performance improving from 75.49 Hit@1 at K=1 to 86.18 at K=5 for N=3, and reaching 88.08 at K=20. Increasing K to 30 provides no further benefit. In contrast, the choice of N has a smaller effect. Retaining a moderate number of paths generally performs better than keeping all paths, while retaining only one path is consistently less effective. At K=20, N=3 achieves the best Hit@1 of 88.08, while N=4 gives a nearly identical Hit@1 of 88.02 and the highest F1 of 74.87. We therefore use K=20 and N=3 as the headline setting.

A.1 Time Efficiency

Table 6 compares the inference time of our proposed method with RoG on WebQSP test questions. Our method requires only 0.049 s per question, compared with 0.498 s for RoG, resulting in approximately a $10.2\times$ reduction in inference time.

Table 6: Inference time comparison on WebQSP test questions.

Method	Sec./Question
RoG baseline	0.498s
Proposed method	0.049s

B Prompts

Latent reasoning. For the latent reasoning stage of LatentRAG, we use the following prompt:

```
[INST] <<SYS>>
<</SYS>>
Please generate a valid relation path that can be helpful for
    answering the following question: {question} [/INST] [<think>] [
    h_1] ... [h_T] [</think>]
```

Here <think> and </think> are learned embedding vectors rather than vocabulary tokens, and each h_t is the backbone's last hidden state from the previous step, fed back as the next input embedding.

Answer generation. To produce the final answer, we select the top-K candidate entities by confidence score and keep the N highest-scoring reasoning paths for each. We verbalize these paths as `entity -> relation -> entity` chains, one per line. The reader LLM then receives the following prompt:

```
[INST] <<SYS>>
<</SYS>>
Based on the reasoning paths, please answer the given question. Please
     keep the answer as simple as possible and return all the possible
     answers as a list.

Reasoning Paths:
{path_1}
{path_2}
...
{path_M}

Question:
{question} [/INST]
```

C THE EFFECTIVENESS OF LATENT REASONING

Table 7: Effect of latent reasoning.

	WebQSP Hit@1 / F1	CWQ Hit@1 / F1
Without latent reasoning	85.2 / 72.3	62.7 / 53.3
With latent reasoning	88.1 / 74.9	63.4 / 56.8

The results show that latent reasoning consistently improves performance on both datasets. The consistent gains across both datasets indicate that latent reasoning provides useful intermediate representations for KGQA beyond the baseline without latent reasoning.