GRISP: Guided Recurrent IRI Selection over SPARQL Skeletons

Anonymous ACL submission

Abstract

We present **GRISP** (**G**uided **R**ecurrent **I**RI Selection over **SP**ARQL Skeletons), a novel SPARQL-based question-answering method over knowledge graphs using a fine-tuned small language model (SLM). Given a natural-language question, the method uses the SLM to generate a natural-language SPARQL query skeleton, and then resolves the natural-language placeholders using a backtracking search that retrieves and re-ranks knowledge graph items under knowledge graph constraints. The SLM is jointly trained on skeleton generation and re-ranking data generated from question-query pairs. We evaluate the method on common Wikidata and Freebase benchmarks, and achieve better results than other state-of-the-art fine-tuning-based methods.

1 Introduction

Many large knowledge graphs are available in RDF (the Resource Description Framework), including Freebase (Bollacker et al., 2008), Wikidata (Vrandecic and Krötzsch, 2014), UniProt (Garcia et al., 2019), and OpenStreetMap (Bast et al., 2021). This has driven growing interest in knowledge graph question-answering (KGQA) methods that generate queries in SPARQL, the standard query language for RDF data. At the same time, the recent emergence of agentic capabilities in large language models enabled the first zero-shot approaches—no fine-tuning or in-context examples—for SPARQL-based KGQA. Systems like GRASP (Walter and Bast, 2025) and SPINACH (Liu et al., 2024) interactively explore and query the knowledge graphs in a human-like fashion, yielding state-of-the-art results on some of the KGQA benchmarks. However, they are typically slow and rely on large, expensive-to-run models. Fine-tuning the underlying model is also not feasible without significant hardware resources. If question-query pairs are available, methods based on fine-tuning small language models (SLM) can outperform these agentic zero-shot systems in generation quality and speed, while needing less hardware; see Walter and Bast (2025) for an overview. In this work, we present GRISP, a SPARQL-based KGQA method that follows a generate-then-retrieve approach. GRISP uses a fine-tuned SLM to generate natural-language SPARQL skeletons, in which natural-language labels are used instead of concrete knowledge graph identifiers (IRIs), and then resolves these placeholders using a backtracking search combined with a retrieve-and-re-rank procedure that uses the same SLM for re-ranking.

Let us look at GRISP answering the question *"Where was the director of Pulp Fiction born?"* over Wikidata. GRISP processes it in two stages.

Stage 1: Skeleton generation. GRISP uses beam search over the fine-tuned SLM to generate multiple SPARQL *skeletons* for the question, then discards duplicates and unparseable outputs and keeps the top candidates. A skeleton is a SPARQL query in which every IRI is replaced by a natural-language placeholder, written as `PH:"..."`. For our example, the top-ranked skeleton might be

```
SELECT ?place WHERE {
  PH:"Pulp Fiction" PH:"director" ?d .
  ?d PH:"place of birth" ?place .
}
```

Using multiple skeletons makes the method robust against variations in surface form (e.g., a competing skeleton might use the inverse relation `?d PH:"directs" PH:"Pulp Fiction"`), and provides fallbacks if a skeleton cannot be fully resolved.

Stage 2: Placeholder resolution. For each skeleton, GRISP resolves placeholders left to right. At every placeholder, it performs a *retrieval-and-re-rank* step: it queries the appropriate KG index (entity or property, depending on the syntactic position) with the placeholder label, collects the top

alternatives, and has the SLM re-rank them. The top-ranked alternative replaces the placeholder.

In our example, `PH:"Pulp Fiction"` is resolved against the entity index, yielding candidates such as the 1994 film (wd:Q104123) and the unrelated common-noun item *pulp fiction* (wd:Q351718). The SLM re-ranks them and selects the film. The resolved prefix is now

```
SELECT ?place WHERE {
  wd:Q104123 PH:"director" ?d .
  ?d PH:"place of birth" ?place .
}
```

KG guidance and constraining. Before retrieving matches for the next placeholder, GRISP uses the resolved prefix to constrain the candidate set: it rewrites the triple patterns connected to the placeholder into a single-column SPARQL query, executes it, and restricts the index search to the returned IRIs. For `PH:"director"`, this means executing

```
SELECT DISTINCT ?p WHERE {
  wd:Q104123 ?p ?d .
}
```

so that only properties actually attached to wd:Q104123 are returned, e.g., wdt:P57 (director), wdt:P162 (producer), wdt:P58 (screenwriter). The re-ranker picks wdt:P57. The same mechanism applies one step later: the constraining query for `PH:"place of birth"` now chains through the resolved director,

```
SELECT DISTINCT ?p WHERE {
  wd:Q104123 wdt:P57 ?d .
  ?d ?p ?place .
}
```

restricting the retrieval to properties of that specific person; the re-ranker selects wdt:P19.

Backtracking. If re-ranking places *None* on top, or the constrained retrieval returns no candidates, GRISP backtracks to the previous placeholder, drops the alternative that led there, and re-ranks the remaining alternatives on the stack. For instance, had the re-ranker initially selected wd:Q351718 instead of the film, the next KG-constrained retrieval for `PH:"director"` would have returned no plausible properties, *None* would have moved to the top, and GRISP would have backtracked to pick the film.

Validation and final answer. Once all placeholders are resolved, GRISP executes the SPARQL query and checks that it is *valid*, i.e., that it parses, executes, and returns a non-empty result; otherwise it backtracks one step. In our example, the query yields Quentin Tarantino (wd:Q3772) and his birthplace Knoxville (wd:Q185582). If a skeleton's resolution tree is exhausted, GRISP moves on to the next skeleton from Stage 1, and returns the first one that yields a valid result.

Core Contributions

1. We present a novel SPARQL-based knowledge graph question-answering method for arbitrary knowledge graphs, called GRISP. The method is centered around fine-tuning an SLM to generate SPARQL skeletons and then resolve the placeholders to knowledge graph items. The main novelty lies in the placeholder resolution, which performs a backtracking search with a retrieve-and-re-rank operation under knowledge graph constraints for each placeholder.

2. We evaluate the method on multiple benchmarks for the Freebase and Wikidata knowledge graphs, achieving better results than state-of-the-art methods in a comparable setting, while being much faster than agentic zero-shot methods.

3. We perform extensive component studies and ablations to investigate their effect on the performance of GRISP. We also contrast GRISP with a recent state-of-the-art agentic KGQA method.

4. We release our resources open-source under a permissive license: a demo at arr-grisp.hopto.org, model checkpoints, benchmarks, and predictions at arr-grisp.hopto.org/grisp, and our evaluation at arr-grisp.hopto.org/evaluate. Code will be released upon acceptance, as the repository is difficult to anonymize.

2 Related Work

KGQA methods can broadly be grouped into two families: those that explore the knowledge graph directly to produce an answer, and those that translate the question into a structured query via semantic parsing. Methods in the first family, e.g., Think-on-Graph (Sun et al., 2024), typically walk the KG to find an answer directly, without producing a query. In this work we focus on the second family. Within semantic parsing, two sub-categories are particularly relevant for GRISP: fine-tuning-based methods that train a (typically small) language model on

question-query pairs, and agentic zero-shot methods in which a large language model iteratively interacts with the KG to draft a query.

Fine-tuning-based methods include ChatKBQA (Luo et al., 2024), WikiSP (Xu et al., 2023), DecAF (Yu et al., 2023), and RGR-KBQA (Feng and He, 2025). We compare against ChatKBQA and WikiSP: ChatKBQA achieves state-of-the-art results on Freebase benchmarks and operates in the same generate-then-retrieve setting as GRISP; WikiSP is one of the few methods that focuses on the larger and more popular Wikidata knowledge graph—most others, including ChatKBQA, still evaluate only on Freebase despite it being read-only since 2015 and discontinued in 2016. We describe both in more detail in the following.

ChatKBQA (Luo et al., 2024) is a generate-then-retrieve method. It generates S-expression skeletons and later converts them to SPARQL after retrieval for execution, whereas GRISP directly generates SPARQL skeletons. After skeleton generation, ChatKBQA retrieves candidate entities via a lexical lookup using FACC1 entity mentions, and resolves properties directly from the skeleton placeholder labels. If direct resolution fails, it falls back to searching within the 2-hop neighborhood of retrieved entities and re-ranking candidates using SimCSE (Gao et al., 2021). The two methods differ mainly in the second stage: where ChatKBQA does a one-shot lookup for entities and label-based resolution for properties, GRISP iteratively resolves all placeholders via search-and-re-rank, which enables knowledge graph constraints and backtracking to recover from wrong choices.

WikiSP (Xu et al., 2023) is a retrieve-then-generate method: it first retrieves entities using ReFinED (Ayoola et al., 2022) fine-tuned for Wikidata, and then generates a SPARQL query using the retrieved entities. For properties, WikiSP generates natural-language property labels in the SPARQL output and resolves them to Wikidata property IDs via an exact label match, falling back to the Wikidata search API if that fails. Both ChatKBQA and WikiSP are closely tailored to their respective knowledge graphs and even to specific benchmarks, which limits generality. For example, ChatKBQA uses Freebase-specific FACC1 mentions for entity linking, a tuned S-expression parser per benchmark, and many hard-coded Freebase-specific SPARQL queries and optimizations in its code. WikiSP uses a Wikidata-tuned entity linker and the Wikidata search API. In contrast, we design *and* implement GRISP to be knowledge-graph agnostic. Only the underlying search indices need to be configured per knowledge graph—specifying which properties the KG uses for labels, aliases, and additional metadata—but they are decoupled from the GRISP method itself.

Table 1: F_1-score of GRISP with Qwen2.5 7B and related work on Freebase and Wikidata benchmarks. Time is the average runtime of GRISP per question.

Dataset	**Methods**			**Time**
Freebase	ChatKBQA		GRISP	
CWQ	70.3[1]		75.1	5.4s
WQSP	74.0[1]		76.0	3.9s
Wikidata	WikiSP	WikiSP[2]	GRISP	
QALD-7	43.6[3]	38.9	47.8	7.0s
WWQ	71.9[3]	70.2	79.5	4.0s
WWQ_{dev}	76.5[1]	74.8	85.3	3.5s
SPINACH	7.1[4]	14.7[5]	29.3[5]	14s
WDQL[6]	-	16.0	57.6	9.0s

[1] Evaluated with our metric on their original predictions
[2] Reproduced WikiSP with Qwen2.5 7B and our metric
[3] Taken from Xu et al. (2023)
[4] Taken from Liu et al. (2024); uses gold entities
[5] Trained on WDQL because SPINACH has no train set
[6] Evaluated on a random subset of 1000 samples

On the agentic zero-shot side, the most relevant method for our work is GRASP (Walter and Bast, 2025), which we compare against in our evaluation. GRASP is knowledge-graph agnostic and uses an off-the-shelf LLM (smaller models perform significantly worse, see Walter and Bast, 2025) that interacts with the knowledge graph autonomously, interleaving reasoning steps with function calls to search, list, and execute SPARQL queries until it commits to a final query. Its main advantages over fine-tuning-based methods like GRISP are the much larger underlying model and the function-call loop, which lets it try to adjust its query as many times as needed. The trade-offs are much higher latency and the inability to benefit from question-query training data when it is available. We reuse GRASP's search indices for GRISP, see Section 3.

3 Method

Section 1 and Section 2 sketch the overall working principle of GRISP and how it differs from related semantic-parsing and agentic methods. In the following, we describe the individual mechanisms behind GRISP—skeleton generation with

beam search, retrieve-and-re-rank, backtracking search, and KG guidance—in more detail. Our two-stage generate-then-retrieve design is motivated by the observation that SLMs can perform semantic parsing reliably when IRIs are either known up-front or replaced by natural-language placeholders (Banerjee et al., 2022; Luo et al., 2024; Feng and He, 2025). The resulting skeleton anchors stage two—the placeholder resolution that constitutes GRISP's main novelty—by providing the structure that the mechanisms described below build on.

Skeleton Generation with Beam Search. Beam search has been proven to boost KGQA performance in the past, both for semantic parsing and knowledge graph traversal (Atif et al., 2023; Luo et al., 2024; Sun et al., 2024). We also employ beam search with beam width 8 and sampling during the skeleton generation phase to generate multiple SPARQL skeleton candidates using the SLM. We filter out duplicate skeletons and those that fail to parse. From the remaining skeletons, we take the top-3 to the placeholder resolution stage.

Retrieve and Re-rank. At each natural-language placeholder to be resolved, we perform an index search with the placeholder label, then re-rank the top-10 alternatives using the SLM and select the top-ranked alternative as replacement. Re-ranking is performed list-wise: all alternatives are presented to the SLM together and scored jointly in a single forward pass. Concretely, we assign a single-token identifier to each alternative and sort the alternatives according to the respective next-token logit scores of the identifiers, similar to FIRST (Gangi Reddy et al., 2024). Depending on the position of the placeholder in the query, we either search an entity index (containing subjects and objects) or a property index, using the search indices provided by GRASP. If the position inference fails, e.g., if a placeholder occurs in a `FILTER` expression, we search both indices and re-rank the combined set of alternatives. We always add a *None* alternative to the search results before re-ranking in case no alternative matches. After re-ranking, all alternatives ranked above *None* are kept on a stack for possible backtracking, while those ranked below *None* are discarded. The top-ranked alternative is then popped from the stack and replaces the placeholder. When backtracking to a previous state, we re-rank the remaining alternatives in its stack rather than reusing the original order. Re-ranking costs only a single forward pass, and the original ordering below the top position may be unreliable since training supervises only that position (see the Training Data paragraph in Section 3).

Table 2: EM and F_1-score of GRISP with Qwen2.5 7B on more Wikidata benchmarks. Time is the average runtime per question.

Benchmark	EM	F_1	Time
QALD-10	46.9	47.6	5.8s
LC-QuAD 2.0[1]	73.0	74.2	4.6s
SimpleQuestions[1]	90.6	90.8	2.3s
QAWiki[2]	34.1	47.3	6.1s

[1] Evaluated on a random subset of 2000 samples
[2] Trained on WDQL since QAWiki has no train set

Backtracking Search. The placeholder resolution stage can be viewed as a depth-first search over the tree of placeholder resolutions, with each node being a partially resolved skeleton and each edge a choice of re-ranked alternative. To allow GRISP to recover from entering a path within this tree that leads to a dead end or to an invalid SPARQL query, we allow going back to previous states. For example, this happens if GRISP re-ranks the *None* alternative to the top position.

Knowledge Graph Guidance. To constrain the set of possible replacements for the current placeholder, we derive a single-column SPARQL query from the triple patterns connected to the current placeholder in the already resolved prefix of the SPARQL skeleton, execute it, and restrict the corresponding index search to the returned candidate IRIs. For example, this allows GRISP to respect constraints such as "Only search within properties known to occur for female politicians born in Canada." This context-sensitive guidance reduces false positives during placeholder resolution and often reveals empty or invalid query paths before the skeleton is fully resolved. As a final instance of the same mechanism, once all placeholders are resolved, we execute the fully resolved SPARQL query against the target knowledge graph and backtrack if it fails to execute or returns an empty result. Note that in its current implementation, KG-constrained search is skipped for placeholders occurring inside advanced SPARQL constructs like `UNION`; in those cases we fall back to an unconstrained search.

Training Sample

One or more natural-language questions.

1. Find the german federal state capitals
2. Can you list all capitals of federal states in Germany?

…

Target SPARQL query.

```
SELECT ?capital WHERE {
  ?state wdt:P31 wd:Q1221156 .
  ?state wdt:P36 ?capital .
}
```

Training Data for Skeleton Generation

System Prompt.
Your task is to generate SPARQL query skeleton over Wikidata for answering user questions …

Randomly chosen training sample question.
Can you list all capitals of federal states in Germany?

Target SPARQL query with natural-language placeholders. Placeholder labels are determined from the search indices and randomly set to the corresponding IRI's main label or an alias.

```
SELECT ?capital WHERE {
  ?state PH:"instance of" PH:"German federal state" .
  ?state PH:"capital city" ?capital .
}
```

Training Data for List-wise Re-ranking

System prompt.
Your task is to select the best fitting Wikidata item for replacing a natural-language placeholder in a SPARQL skeleton …

Partially resolved SPARQL query and information for already resolved placeholders.

```
SELECT ?capital WHERE {
  ?state wdt:P31 wd:Q1221156 .
  ?state PH:"capital city" ?capital .
}
```

instance of (wdt:P31), federated state of Germany (wd:Q1221156)

*Alternatives to choose from, as returned by an index search using the placeholder label, plus the **None** alternative. Augmentations are applied as follows: the number and order of alternatives is chosen randomly, and their disambiguation metadata as well as the target alternative are dropped randomly.*
Alternatives for replacing placeholder PH:"capital city":
A) capital of (wd:P1376): country, state, or other administrative division of which the municipality is the governmental seat
B) capital (wd:P31): city of governmental seat of the country
…
K) None of the above

Single-token target identifier corresponding to the target alternative.
Selection: B

Figure 1: Training data synthesis for GRISP's two stages using an exemplary training sample. *Top left*: Training sample consisting of a SPARQL query and one or more corresponding natural language questions. *Bottom left*: Skeleton generation training data built from the training sample by sampling a question and replacing the IRIs in the SPARQL query with their placeholders. *Right*: Re-ranking training data built from the training sample by randomly choosing a target IRI, replacing it and all following IRIs in the SPARQL query with their placeholders, and searching the knowledge graph indices with the target placeholder label.

4 Evaluation

Benchmarks. We evaluate GRISP and compare against ChatKBQA on WebQuestionsSP (WQSP) (Yih et al., 2016) and ComplexWebQuestions (CWQ) (Talmor and Berant, 2018) for Freebase, and against WikiSP on QALD-7 (Usbeck et al., 2017), WikiWebQuestions (WWQ) (Xu et al., 2023), SPINACH (Liu et al., 2024), and Wikidata Query Logs (WDQL) (Walter and Bast, 2026) for Wikidata. We further evaluate GRISP on QALD-10 (Usbeck et al., 2024), LC-QuAD 2.0 (Dubey et al., 2019), SimpleQuestions (Diefenbach et al., 2017), and QAWiki (Moya Loustaunau and Hogan, 2025). See Appendix A for benchmark sizes.

Metric. We follow the evaluation protocol of Walter and Bast (2025) and evaluate all models and related work using a row-wise F_1-score averaged over samples. We also report the Exact Match (EM) score averaged over samples, which is defined as 1 iff the F_1-score is 1, and 0 otherwise.

Training Data. Given a training sample in the form of a question-query pair, we generate training data for skeleton generation by using its question and replacing all IRIs in the SPARQL query with a natural-language placeholder, which is randomly set to the IRI's main label or an alias. The definition of main label and alias is determined by the search index and varies between knowledge graphs, e.g., for Wikidata it is `rdfs:label` and `skos:altLabel`, respectively. For list-wise re-ranking we randomly choose an IRI from the skeleton, then search in the index corresponding to the IRI's position, using the IRI's natural-language placeholder, to get the alternatives to re-rank for this IRI. To make the re-ranker more robust, we perform the following data augmentations for this stage: (1) we randomly vary the search top-k, which determines the number of alternatives to re-rank; (2) we randomly drop the additional metadata shown for each alternative for disambiguation, which again is determined by the search index (e.g., `schema:description` for Wikidata); (3) we randomly drop the target alternative (the one matching the IRI) to make the model learn when to choose *None*; (4) we randomly shuffle the alternatives to avoid position bias. The SLM is jointly trained on both types of data using standard next-token cross-entropy loss for skeleton generation, and cross-entropy restricted to the identifier tokens corresponding to available alternatives for list-wise re-ranking. The latter differs from FIRST, which uses a learning-to-rank loss for re-ranking, because we only have the target alternative available for supervision, and not a full ordering of all alterna-

Table 3: GRISP (fine-tuned Qwen2.5 7B) vs. GRASP (zero-shot) with Qwen2.5 72B and GPT-4.1. Each benchmark shows F_1-score and average runtime per question. Bold and underline mark the best and second-best per column. Note that we only evaluate on samples where all methods provide predictions (GRASP evaluates on random subsets of at most 200 samples). Consequently, scores and runtimes for GRISP might be different from those reported above, and scores and runtimes for GRASP might be different than reported in Walter and Bast (2025).

System	CWQ		WQSP		QALD-7		QALD-10		WWQ		SPINACH	
GRASP $_{\text{Qwen2.5 72B}}$	30.5[2]	56s	40.9	45s	<u>73.5</u>[2]	37s	<u>61.7</u>	47s	68.7	46s	27.6[2]	136s
GRASP $_{\text{GPT-4.1}}$	<u>44.5</u>	<u>37s</u>	<u>52.3</u>	<u>30s</u>	**79.8**	<u>17s</u>	**72.7**	<u>22s</u>	<u>75.3</u>	<u>21s</u>	**40.8**	<u>44s</u>
GRISP $_{\text{Qwen2.5 7B}}$	**77.0**	**5.3s**	**73.3**	**4.2s**	46.7	**7.2s**	44.5	**6.4s**	**78.8**	**3.7s**	<u>29.3</u>[1]	**14s**

[1] Trained on WDQL since SPINACH has no train set
[2] Not reported in Walter and Bast (2025) but original predictions are available

tives. Note that the data generation procedure is run for each sample and epoch. This way we present different training data given the same sample to the model in each epoch, which further mitigates overfitting. Data generation can happen either online (during training) or offline (pre-computed for multiple epochs in advance). The offline option significantly speeds up training and helps with reproducibility. See Figure 1 for an example.

Training Hyperparameters. We use the Qwen2.5 model family (Yang et al., 2024), which provides a wide range of instruction-tuned models from 0.5B to 7B parameters. All results in this paper use the 7B variant unless stated otherwise. We train all models using LoRA (Hu et al., 2022) with $r = 32$ and $\alpha = 32$ on all linear layers for 4 epochs with a base learning rate of $1e^{-4}$, learning-rate warmup and cosine decay, a batch size of 8, and early stopping. We train models up to sizes of 1.5B on a single NVIDIA RTX4090 GPU (24 GB VRAM), and larger models up to 7B on a single NVIDIA H200 GPU (140 GB VRAM). Training times range from a few minutes on QALD-7 to just under one day on WDQL.

Inference Hyperparameters. We use a temperature of 0.4, a top-p value of 0.9, a beam width of 8, and sampling for skeleton generation. We then keep the top-3 generated skeletons for the placeholder resolution stage. For placeholder resolution, we set the search top-k to 10, and enable re-ranking, knowledge graph guidance, and backtracking as described in Section 3. All models are run on a single NVIDIA RTX4090 GPU with 24 GB of VRAM.

Results. Our main results are shown in Table 1, and additional results on Wikidata in Table 2. GRISP outperforms ChatKBQA and WikiSP on all benchmarks. On Wikidata, the gap to WikiSP is most pronounced, e.g., over 8 percentage points (p.p.) on WWQ_{dev} and over 40 p.p. on WDQL.

Both ChatKBQA (LLaMA2 7B/13B) and WikiSP (LLaMA 7B) use base models similar in size to the Qwen2.5 7B model used by GRISP. For a fairer comparison on the same base model, we reproduce one of them using Qwen2.5 7B and choose WikiSP since it uses the older-generation base model and focuses on the more relevant Wikidata knowledge graph. The results (WikiSP[2] in Table 1) are similar to or slightly below the original, but GRISP still consistently outperforms them.

We also observe that GRISP's average runtime per question grows on harder benchmarks where F_1 drops. On more challenging benchmarks like SPINACH, GRISP generates more involved skeletons, performs more retrieve-and-re-rank steps, and backtracks more often. Average runtimes range from 2.3s to 14s depending on benchmark complexity, with most of the time spent on index searches and the final SPARQL query validation. From interacting with GRISP ourselves, we find that it produces sensible SPARQL queries for simple questions ("Where does Barack Obama live") almost always, for medium-difficulty questions ("Albert Einstein's grandchildren, their parents, and their jobs") most of the time, and for hard questions ("All British monarchs") sometimes.

Comparison to GRASP. We find GRISP to be much faster than agentic methods like GRASP when both are run end-to-end on Wikidata. Table 3 compares F_1-scores and runtimes directly. On benchmarks whose query patterns can be learned from sufficient training data, GRISP clearly outperforms GRASP: on CWQ, WQSP, and WWQ, GRASP with GPT-4.1 reaches 44.5, 52.3, and 75.3 compared to GRISP's 77.0, 73.3, and 78.8. When little training data is available, zero-shot agentic

Table 4: Component studies on WWQ_{dev}: EM and F_1-score of GRISP for varying beam width n (Table 4a), search top-k (Table 4b), and model size (Table 4c). All sub-tables use Qwen2.5 7B except for the size study. Bold and underline mark the best and second-best per metric within each sub-table.

(a) Beam width n.

n	EM	F_1
16	**83.4**	<u>85.2</u>
8	**83.4**	**85.3**
4	83.0	84.8
2	82.5	84.4
1	78.8	80.5

(b) Search top-k.

top-k	EM	F_1
15	**83.4**	**85.3**
10	**83.4**	**85.3**
5	83.2	85.1
3	82.7	84.7
1	79.6	81.7

(c) Model size.

Size	EM	F_1
7B	**83.4**	**85.3**
3B	<u>79.9</u>	<u>81.6</u>
1.5B	79.4	80.8
0.5B	72.8	74.8

methods take the upper hand: on QALD-7 (100 samples) and QALD-10 (412 samples), GRASP with GPT-4.1 reaches 79.8 and 72.7 F_1 versus GRISP's 46.7 and 44.5. SPINACH is hard for all systems, but a stronger agentic setup (GRASP with GPT-4.1) reaches 40.8, while GRISP only reaches 29.3 despite being trained on the larger WDQL dataset. This suggests that the more flexible exploration and iterative execution feedback of agentic methods, which are unavailable to GRISP's generate-then-retrieve design, help in this regime. Even there, GRISP can still be a good choice depending on the desired tradeoff between answer quality, inference speed, and the ability to run on a single consumer GPU.

Component and Ablation Studies. We first perform studies on two important inference hyperparameters, the beam width for skeleton generation in Table 4a and the top-k value for search in Table 4b. As expected, performance improves with increasing parameter values, but quickly hits diminishing returns: a beam width larger than 8 or a search top-k larger than 10 do not lead to further improvement. We therefore use these by default.

When altering the size of GRISP's underlying SLM, we get consistent performance improvements with larger models, as shown in Table 4c. The improvements are significant, both from 0.5B to 1.5B and from 3B to 7B. But even the 0.5B model reaches the Qwen-based WikiSP score of 74.8 on WWQ_{dev}, which the 1.5B model clearly surpasses.

Next, we check the effect of removing core mechanisms from GRISP in Table 5. All mechanisms add value, but the largest drop is caused by removing re-ranking, followed by beam search, knowledge graph guidance, constrained search, and backtracking. Here, *w/o constrained search* disables using the SPARQL skeleton to constrain the index searches, while *w/o KG guidance* disables both constrained search and the final SPARQL query validation. We conjecture that the latter two mechanisms would have a greater impact on a knowledge graph with a more complex schema than Wikidata, or on more challenging questions.

We also report two stripped-down GRISP configurations that put it on a more even footing with WikiSP. *All disabled* turns off beam search, re-ranking, constrained search, backtracking, and the final query validation, leaving plain index search; *re-ranking only* additionally re-enables the SLM-based re-ranking, which in this setting acts as a learned entity and property selector over the search results, and is the closest analog in GRISP to what the fine-tuned ReFinED linker does for entities in WikiSP. With all mechanisms off, GRISP reaches only 70.4 F_1 on WWQ_{dev}, below the WikiSP[2] reproduction's 74.8; adding re-ranking lifts GRISP to 79.1, already surpassing it. The GRISP configuration closest to WikiSP in terms of capabilities likely sits somewhere between these two, which fits the experimental results.

Finally, we compare three alternative training setups against the default GRISP setup. *Skeleton* trains only skeleton generation and uses the off-the-shelf Qwen2.5 7B model for re-ranking at inference. *Skeleton w/o re-ranking* also trains only skeleton generation, but skips re-ranking entirely at inference. *Skeleton + re-ranking* trains the two tasks separately, yielding two models that are then used in turn at inference. The default setup outperforms all three, suggesting that joint training of both tasks enables important transfer between them. We also find that training the re-ranking task explicitly is important, since using an untrained Qwen2.5 7B model for re-ranking performs the same as not doing re-ranking at all.

GRISP's behavior and failure modes. To better understand where errors come from in GRISP's two-stage generate-then-retrieve framework, we run two additional studies that are not part of Table 5. First, we measure an *oracle-skeleton* F_1, where we feed the gold SPARQL skeleton directly to the placeholder resolution stage. On WWQ_{dev}, oracle F_1 reaches 98.6 against the default GRISP's 84.4, so 14.2 points of the gap come from skeleton generation and only 1.4 from placeholder resolution. On QALD-10, the split is much more pronounced: oracle F_1 is 94.6 against 46.5, meaning skeleton generation accounts for 48.1 points of loss and placeholder resolution for only 5.4. This is expected given QALD-10's harder queries, and pinpoints the skeleton generator as the main bottleneck. Second, we look at backtracking frequency. On WWQ_{dev}, GRISP backtracks 0.39 times per sample on average, concentrated on 35 of 454 samples (about 5 backtracks each among those that backtrack at all). On QALD-10, the rate jumps to 2.70 per sample, with 125 of 394 samples backtracking (about 8.5 each), reflecting the harder multi-hop queries and showing that backtracking is doing meaningful work in this regime.

Table 5: Ablation and training setup study: EM, F_1-score, and average runtime per question of GRISP with Qwen2.5 7B on WWQ_{dev}. Each mechanism ablation only removes that particular mechanism. Rows within each block are ordered by F_1-score.

Variant	**EM**	**F_1**	**Time**
GRISP	82.4	84.4	5.7s
Mechanism ablations			
w/o backtracking	81.9	83.9	5.7s
w/o constrained search	81.2	83.2	4.3s
w/o KG guidance	79.2	80.7	3.8s
w/o beam search	76.6	79.0	4.1s
w/o re-ranking	72.8	78.6	4.5s
Minimal configurations			
re-ranking only	77.5	79.1	3.0s
all disabled	65.4	70.4	2.8s
Training and inference setups			
skeleton + re-ranking	79.7	82.2	5.3s
skeleton	69.6	75.9	4.8s
skeleton w/o re-ranking	70.1	75.8	4.3s

5 Conclusion

We present GRISP, a novel KGQA method in a generate-then-retrieve setting based on fine-tuning an SLM on the tasks of SPARQL skeleton generation and re-ranking of knowledge graph items. The main novelty of GRISP is the placeholder resolution stage, which uses backtracking search to replace natural-language placeholders in SPARQL skeletons with IRIs, using a retrieve-and-re-rank operation under knowledge graph constraints for each placeholder. Our ablations confirm that it is indeed this placeholder resolution stage that sets GRISP apart from other methods. We evaluate GRISP against state-of-the-art methods that operate in a comparable setting and achieve consistent improvements on benchmarks for Freebase and Wikidata. Notably, GRISP is much faster than current agentic methods but can match and even surpass their performance given sufficient training data, all while running on a single consumer GPU.

GRISP stops as soon as it finds a valid query with a non-empty result. We consider it an interesting direction for future work to add a plausibility check for the result. For example, if the result for the question *All British Monarchs* does not contain Queen Elizabeth II, it is clearly wrong. Performing such a plausibility check is straightforward with an LLM, but it is not clear how to do so with an SLM. Moreover, to make the result of such a check reasonably actionable, there would have to be a way to enhance the pool of skeletons if none leads to a plausible result.

AI Use Statement. We did not use AI for core parts of the work. In particular, we did not use AI to implement code for GRISP or to write paper content from scratch. We did use AI coding assistants in an interactive manner to help us with writing code for non-critical parts of this work, such as the demo webapp at arr-grisp.hopto.org. We also used AI to improve and polish our writing, and to rephrase existing paragraphs. We carefully checked AI-generated writing in all cases.

Limitations

The main limitation of GRISP is that it relies on the availability of question-query training pairs, which limits applicability to knowledge graphs for which such data exists. In low-data regimes or on benchmarks with harder, more diverse queries, GRISP's performance falls behind state-of-the-art agentic methods, as we see on QALD-7, QALD-10, and SPINACH. Our diagnostics suggest that skeleton generation becomes the dominant bottleneck in these regimes, while placeholder resolution stays closer to its ceiling. All reported results are from a single training run per configuration due to compute constraints.

As for potential risks, GRISP can in principle retrieve any information contained in the underlying knowledge graph, but its generated SPARQL queries and their execution results should not be blindly trusted. It is also naturally exposed to biases in the benchmark training data.

References

Farah Atif, Ola El Khatib, and Djellel Difallah. 2023. BeamQA: Multi-hop Knowledge Graph Question Answering with Sequence-to-Sequence Prediction and Beam Search. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, pages 781–790, New York, NY, USA. Association for Computing Machinery.

Tom Ayoola, Shubhi Tyagi, Joseph Fisher, Christos Christodoulopoulos, and Andrea Pierleoni. 2022. ReFinED: An Efficient Zero-shot-capable Approach to End-to-End Entity Linking. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 209–220, Hybrid: Seattle, Washington + Online. Association for Computational Linguistics.

Debayan Banerjee, Pranav Ajit Nair, Jivat Neet Kaur, Ricardo Usbeck, and Chris Biemann. 2022. Modern Baselines for SPARQL Semantic Parsing. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, pages 2260–2265, New York, NY, USA. Association for Computing Machinery.

Hannah Bast, Patrick Brosi, Johannes Kalmbach, and Axel Lehmann. 2021. An Efficient RDF Converter and SPARQL Endpoint for the Complete OpenStreetMap Data. In *SIGSPATIAL/GIS*, pages 536–539. ACM.

Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250. ACM.

Dennis Diefenbach, Thomas Pellissier Tanon, Kamal Deep Singh, and Pierre Maret. 2017. Question Answering Benchmarks for Wikidata. In *ISWC (Posters, Demos & Industry Tracks)*, volume 1963 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia. In *ISWC (2)*, volume 11779 of *Lecture Notes in Computer Science*, pages 69–78. Springer.

Tengfei Feng and Liang He. 2025. RGR-KBQA: Generating Logical Forms for Question Answering Using Knowledge-Graph-Enhanced Large Language Model. In *COLING*, pages 3057–3070. Association for Computational Linguistics.

Revanth Gangi Reddy, JaeHyeok Doo, Yifei Xu, Md Arafat Sultan, Deevya Swain, Avirup Sil, and Heng Ji. 2024. FIRST: Faster Improved Listwise Reranking with Single Token Decoding. In *EMNLP*, pages 8642–8652. Association for Computational Linguistics.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Leyla Garcia, Jerven Bolleman, Sebastien Gehant, Nicole Redaschi, Maria Martin, Alex Bateman, Michele Magrane, Sandra Orchard, Shriya Raj, Shadab Ahmad, Emanuele Alpi, Emily Bowler, Ramona Britto, Borisas Bursteinas, Hema Bye-A-Jee, Tunca Dogan, Penelope Garmiri, George Georghiou, Leonardo Gonzales, and 85 others. 2019. FAIR adoption, assessment and challenges at UniProt. *Scientific Data*, 6(1):175.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Shicheng Liu, Sina J. Semnani, Harold Triedman, Jialiang Xu, Isaac Dan Zhao, and Monica S. Lam. 2024. SPINACH: SPARQL-Based Information Navigation for Challenging Real-World Questions. In *EMNLP (Findings)*, pages 15977–16001. Association for Computational Linguistics.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Anh Tuan Luu. 2024. ChatKBQA: A Generate-then-Retrieve Framework for Knowledge Base Question

Answering with Fine-tuned Large Language Models. In *ACL (Findings)*, pages 2039–2056. Association for Computational Linguistics.

Alberto Moya Loustaunau and Aidan Hogan. 2025. QAWiki v1: Knowledge Graph Question Answering (KGQA) / SPARQL Query Generation Dataset for Wikidata. Zenodo.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. In *ICLR*. OpenReview.net.

Alon Talmor and Jonathan Berant. 2018. The Web as a Knowledge-Base for Answering Complex Questions. In *NAACL-HLT*, pages 641–651. Association for Computational Linguistics.

Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 2017. 7th Open Challenge on Question Answering over Linked Data (QALD-7). In *SemWebEval@ESWC*, volume 769 of *Communications in Computer and Information Science*, pages 59–69. Springer.

Ricardo Usbeck, Xi Yan, Aleksandr Perevalov, Longquan Jiang, Julius Schulz, Angelie Kraft, Cedric Möller, Junbo Huang, Jan Reineke, Axel-Cyrille Ngonga Ngomo, Muhammad Saleem, and Andreas Both. 2024. QALD-10 – The 10th challenge on question answering over linked data: Shifting from DBpedia to Wikidata as a KG for KGQA. *Semantic Web*, 15(6):2193–2207.

Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Sebastian Walter and Hannah Bast. 2025. GRASP: Generic Reasoning And SPARQL Generation Across Knowledge Graphs. In *ISWC (1)*, volume 16140 of *Lecture Notes in Computer Science*, pages 271–289. Springer.

Sebastian Walter and Hannah Bast. 2026. The Wikidata Query Logs Dataset. *Preprint*, arXiv:2602.14594.

Silei Xu, Shicheng Liu, Theo Culhane, Elizaveta Pertseva, Meng-Hsi Wu, Sina J. Semnani, and Monica S. Lam. 2023. Fine-tuned LLMs Know More, Hallucinate Less with Few-Shot Sequence-to-Sequence Semantic Parsing over Wikidata. In *EMNLP*, pages 5778–5791. Association for Computational Linguistics.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *ACL (2)*. The Association for Computer Linguistics.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2023. DecAF: Joint Decoding of Answers and Logical Forms for Question Answering over Knowledge Bases. In *ICLR*. OpenReview.net.

A Benchmark Split Sizes

Table 6: Train, dev, and test split sizes for all benchmarks used in this paper. A dash indicates the split does not exist. All benchmarks use English questions, but the corresponding SPARQL queries may query for data in different languages (e.g., WDQL SPARQL queries cover a wide range of languages).

Benchmark	Train	Dev	Test
Freebase			
CWQ	27,639	3,519	3,531
WQSP	3,453	-	1,639
Wikidata			
QALD-7	100	-	50
WWQ[1]	2,431	454	1,431
SPINACH	-	155	165
WDQL	82,661	10,333	10,333
QALD-10	412	-	394
LC-QuAD 2.0	24,124	-	6,028
SimpleQuestions	19,481	2,821	5,622
QAWiki	-	-	518

[1] WWQ_{dev} in the main results refers to the dev split