Autofocus Retrieval: An Effective Pipeline for Multi-Hop Question Answering With Semi-Structured Knowledge

Anonymous authors
Paper under double-blind review

Abstract

In many real-world settings, machine learning models and interactive systems have access to both structured knowledge, e.g., knowledge graphs or tables, and unstructured content, e.g., natural language documents. Yet, most rely on either. Semi-Structured Knowledge Bases (SKBs) bridge this gap by linking unstructured content to nodes within structured data. In this work, we present Autofocus-Retriever (AF-Retriever), a modular framework for SKB-based, multi-hop question answering. It combines structural and textual retrieval through novel integration steps and optimizations, achieving the best zero- and one-shot results across all three STaRK QA benchmarks, which span diverse domains and evaluation metrics. AF-Retriever's average first-hit rate surpasses the second-best method by 32.1%. Its performance is driven by (1) leveraging exchangeable large language models (LLMs) to extract entity attributes and relational constraints for both parsing and reranking the top-k answers, (2) vector similarity search for ranking both extracted entities and final answers, (3) a novel incremental scope expansion procedure that prepares for the reranking on a configurable amount of suitable candidates that fulfill the given constraints the most, and (4) a hybrid retrieval strategy that reduces error susceptibility. In summary, while constantly adjusting the focus like an optical autofocus, AF-Retriever delivers a configurable amount of answer candidates in four constraint-driven retrieval steps, which are then supplemented and ranked through four additional processing steps. An ablation study and a detailed error analysis, including a comparison of three different LLM reranking strategies, provide component-level insights that are valuable for advancing the model and for enabling researchers and users to adapt, optimize, or extend its parts. The source code is publicly available at [URL will be placed here].

1 Introduction

Large Language Models (LLMs) have rapidly evolved in recent years, demonstrating impressive performance on a range of complex tasks. As a result, they are increasingly integrated into everyday workflows, scientific research, and decision-making processes. However, despite their capabilities, LLMs remain prone to significant limitations. They can produce inaccurate or biased outputs, fail to recognize gaps in their own knowledge, and generate biased or fabricated ("hallucinated") content that is difficult to detect (Bender et al., 2021; Feldman et al., 2023). Their effectiveness in reasoning-based tasks is also the subject of ongoing scrutiny (Kambhampati, 2024). To mitigate these shortcomings, Retrieval-Augmented Generation (RAG) systems have been introduced. These systems enhance LLMs by retrieving relevant external information before generating responses, improving factual accuracy and robustness. Beyond RAG, many contemporary AI systems, including those for question answering (QA), typically leverage either unstructured content like natural language documents or structured data, such as knowledge graphs and relational tables. The latter provides precise, queryable information, while the former offers broad, context-rich knowledge that is retrieved via dense embedding-based search. However, in many real-world domains, both types of information are available and valuable.

This has led to the development of Semi-Structured Knowledge Bases (SKBs), which combine the advantages of both by linking unstructured documents to nodes in knowledge graphs. SKBs thus enable new methods

of knowledge access and inference by bridging the gap between purely symbolic and contextual information. Recent work by Wu et al. (2024b) exemplifies the potential of SKBs. They constructed SKBs across three real-world domains and assembled the STaRK benchmarks of corresponding multi-hop QA datasets. These datasets reflect realistic scenarios where answering a question requires both relational reasoning over structured elements and contextual interpretation of unstructured texts. In response, several specialized and hybrid approaches have been proposed to leverage SKBs for complex QA tasks. However, many existing methods focus on specific techniques in isolation, overlooking opportunities to integrate complementary ideas.

Our system, Autofocus-Retriever (AF-Retriever), outperforms state-of-the-art (SOA) zero- and one-shot methods on all STaRK datasets by synthesizing multiple existing and novel strategies into a cohesive framework that constantly adjusts the focus of symbol and target candidates. Only one preprint reports even better results for one of the benchmark datasets. But it relies on domain-specific training sets for fine-tuning in contrast to our method. The proposed solution includes the following steps, which are described in section 3 in more detail. (i) *Target type prediction:* The first simple, yet effective step identifies which entity type is suitable as an answer and constitutes the first layer of focus. (ii) *Formalization as Cypher queries:* It further takes advantage of the inherent ability of common LLMs to interpret natural language questions and translate them into Cypher (Francis et al., 2018) queries, a popular language for querying property graphs. These queries include an updated target node label and several relational triplets with constants, variables, and their attributes to encode multi-hop questions. Our extended analysis confirms that even smaller, open base models are able to produce Cypher queries off the cuff. (iii) *Querying:* Our analysis shows that including and omitting relational facts in node embeddings can increase or decrease their entity retrieval utility depending on its application. To retrieve semantically relevant entities as constants to be considered, we employ Vector Similarity Search (VSS) without relational data embedded. (iv) *Entity retrieval:* Building on extracted relational triplets and symbol candidates, a variant of graph-based path search prefilters answer candidates by their relational information. This search, implemented as a sequence of set intersections, narrows down answer candidate sets to only those that meet extracted conditions. (v) *Scope expansion:* Because several nodes are eligible for a constant (e.g., "University of Miami", "Miami University", or "Miami Dade College" for the search string "Miami Uni"), AF-Retriever, incrementally increases the number of candidates for constants again so that the relational constraints remain fulfilled and the number of false positives stays small, but still a predefined number of answers is returned. This novel approach dynamically balances the specificity and sensitivity of entity retrieval, a challenge in itself. (vi) *Merging and sorting:* By merging candidate sets from both the aforementioned neuro-symbolic and a more naive vector-based retrieval, AF-Retriever mitigates potential errors introduced by individual retrieval steps. Our parameterized analysis demonstrates that, while the purely vector-based retrieval performs much worse than SOA methods, the infusion of a few of its answers into the candidate pool can notably increase the overall performance. (vii) *Reranking:* Finally, with a focus on detailed unstructured information, AF-Retriever leverages LLMs to rerank the top-k candidates by either scoring or mutually comparing their relevance and coherence with the question. To identify the verifiable most suitable reranking strategy for SKB-based question answering, this work contributes a theoretical and experimental comparison of pointwise, pairwise, and listwise reranking with LLMs.

The remainder of this paper is organized as follows: Section 2 surveys prior work related to question answering over Semi-Structured Knowledge Bases. Section 3 outlines the proposed methodology in detail. Section 4 reports on our experimental evaluation including a comparison with SOA systems, ablation study, and additional analyses. Finally, Section 5 concludes the paper and discusses directions for future research.

2 Related work

Recent surveys have outlined the synergy between Large Language Models (LLMs) and knowledge graphs (KGs), emphasizing both the opportunities and challenges that arise when they are integrated (Agrawal et al., 2024; Pan et al., 2024). Pan & et al. (2023) further highlight the ongoing difficulties in integrating structured reasoning with the generative capabilities of LLMs, pointing to key research directions for improving factual accuracy, context retention, and reasoning over hybrid knowledge sources.

Retrieval across textual and relational knowledge Our work falls within the domain of retrieval-based, multi-hop question answering (QA) over semi-structured knowledge, where both structured (relational) and unstructured (textual) data must be effectively combined. Wu et al. (2024b) introduced the concept of a Semi-Structured Knowledge Base (SKB), linking knowledge graph (KG) nodes with associated free-text documents. Their STaRK benchmark suite evaluates methods designed to operate over such hybrid corpora and includes experiments with the following six baseline techniques. Among these retrieval methods, one of the simplest is *Vector Similarity Search (VSS)*, which embeds a query and candidate entities, each represented by concatenated textual and relational information, into a shared space for cosine similarity computation. *Multi-Vector Similarity Search* extends this by separating and embedding text and structure separately, enabling a finer-grained comparison. *Dense Passage Retriever* (Karpukhin et al., 2020) trains query and passage encoders using QA pairs to optimize retrieval accuracy. *QA-GNN* (Yasunaga et al., 2021) tackles the problem by constructing a joint graph consisting of QA elements and KG nodes. It performs relevance scoring using LLM-generated embeddings, followed by message passing over the graph to facilitate joint reasoning. *VSS+Reranker* (Chia et al., 2024; Zhuang et al., 2024) uses a language model to assign a relevance score to each of the top-k VSS results, refining the initial retrieval with semantic ranking grounded in an SKB context. This corresponds to steps 7 and 8 with a pointwise reranker and $\alpha = 0$ in our pipeline (see below). *AVATAR* (Wu et al., 2024a) represents an agent-based approach, in which a comparator module generates contrastive prompts for optimizing LLM behavior during tool usage. Its performance study includes *ReAct* (Yao et al., 2023), which blends reasoning with next action decision in an interleaved LLM prompt, and Reflexion (Shinn et al., 2023), which employs episodic memory to refine agent decisions across tasks. *HybGRAG*, a complementary line of work presented by Lee et al. (2025), combines multiple retrievers with a critic module to refine answer generation across both structured and textual data. *4StepFocus* (Boer et al., 2024) augments VSS+Reranker by two preprocessing steps: Triplet generation for extraction of relational data by an LLM and substitution of constants and variables in those triplets to narrow down answer candidates. AF-Retriever shares this aspect, but it additionally follows a hybrid retrieval approach, features different LLM reranking strategies, and its greedy-like, VSS-based candidate search is more advanced than the string similarity search of 4StepFocus. Xia et al. (2025) propose *KAR*, a query expansion method that integrates relational constraints from the KG into the retrieval pipeline and performs listwise LLM reranking concatenating n responses with filtered triplets as the final expansion appended to the original query.

Retrieval with integrated domain- and task-specific fine-tuning of employed LLMs Further methods fine-tune LLMs on training sets from the same distribution as test sets before conducting the retrieval procedure. At the cost of LLM interchangeability and applicability when suitable training sets are missing, this increases the potential of higher accuracy. MoR (**M**ixture **of** Structural-and-Textual **R**etrieval) (Lei et al., 2025) and mFAR (**m**ulti-**F**ield **A**daptive **R**etrieval) (Li et al., 2024) address semi-structured retrieval by decomposing documents into multiple fields, each indexed independently. *MoR* frames retrieval in terms of three stages: planning, reasoning, and organizing, bridging graph traversal and text matching through path-based scoring. Thereby, data are decomposed into fields and each field is scored separately against the query using lexical- and vector-based scorers. The reranking component adaptively predicts the importance of a field by conditioning on the document query. *mFAR* (Li et al., 2024) also adaptively weights fields based on the query and generates the entire planning graph in one shot, avoiding repeated LLM calls. In contrast, *PASemiQA* (Yang et al., 2025) repeatedly consults an agent which decides which action to take next, then performs the action to retrieve relevant information from the SKB that is used to determine the next action. In contrast to our incremental scope-expanding approach, if no node for an entity is found via keyword matching, a fixed number of VSS-based candidates is taken in each case.

Property graph databases such as Neo4j and JanusGraph have gained widespread adoption in both academia and industry over the past decade. Cypher, a prominent query language for such systems, allows expressive querying of node and relationship patterns using constructs like MATCH, WHERE, and RETURN (Francis et al., 2018). Most recently, *GraphRAFT* (Clemedtson & Shi, 2025) was released in a preprint including evaluation on two STaRK test sets. It combines different fine-tuned LLMs with a Cypher engine. The first LLM identifies entities in an user query which are inserted in three different templates for one- and two-hop queries with up to two triplets. Another fine-tuned LLM selects the most suited queries, before they are executed to generate a joint subgraph. A third LLM ranks entities within this subgraph. The authors claim

remarkable results on the synthetically generated PRIME dataset but also acknowledge the dependency on data-specific fine-tuning, while our approach leverages the ability of base LLMs to generate Cypher queries from natural language without fine-tuning. Additionally, without adaptation, AF-Retriever is more flexible as it does not build on a fixed query template with a restricted number of triplets and supports more syntax.

Appendix A.4 lists these works including the base LLMs they employed, Section 4 compares our results of AF-Retriever against them on the STaRK benchmark suite.

Additional perspectives Beyond retrieval-focused QA systems, several recent efforts have tackled related challenges such as mitigating hallucinations or enhancing interpretability. Guan et al. (2024) proposed a knowledge graph-based retrofitting method to reduce LLM hallucinations using automatically generated triplets. Wang et al. (2024) use embedding-based techniques to enable QA across multiple documents. Vedula & Parthasarathy (2021) present FACE-KEG, a KG-transformer model for fact verification with an emphasis on explainability. Kundu & Nguyen (2024), as well as Shahi (2024), explore approaches for identifying and structuring false claims within a KG framework for fact checking. Similarly, Shakeel & Jain (2021) take a pipeline approach combining KG embeddings and traditional NLP for factual verification. Knowledge Solver (Feng et al., 2023) builds whole subgraphs first, allowing the LLM to select entities in subsequent path-finding steps, starting from a randomly selected term in the question. Mountantonakis & Tzitzikas (2023) provide a semi-automatic web application that assists users in comparing facts generated by ChatGPT with those in KGs by using multiple RDF KGs for enriching ChatGPT responses. While it has a few parallels to our work, the web interface requires manual interpretation.

Contributions of this work In addition to the integration and optimization of several neuro-symbolic core concepts (VSS, text2cypher parsing, triplet-based graph search, and LLM reranking), this work introduces multiple novel approaches to SKB-based question answering, such as: an incremental procedure to dynamically balance specificity and sensitivity, the introduction of a two-model ensemble strategy, and different LLM reranking approaches to SKB-based question answering. Our modular pipeline, AF-Retriever, outperforms SOA methods. An extensive ablation study and error analysis provides valuable insights for future work.

3 Autofocus-Retriever

3.1 Problem definition and notations

A Semi-Structured Knowledge Base $SKB = (\mathbb{V}, \mathbb{E}, \mathbb{D})$ (Wu et al., 2024b) consists of a knowledge graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ and associated text documents $\mathbb{D} = \bigcup_{v \in \mathbb{V}} D_v$, where $\mathbb{V}$ is a set of graph nodes and $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ a set of their connecting edges, which may be directed or undirected, and weighted or unweighted. Given an SKB, a query q in natural language, and a set of candidates for query answers $\mathbb{C} \subseteq \mathbb{V}$, our question answering scenario expects a list of (up to 20) most fitting candidates to q in descending order.

Before we explain the individual steps of our approach in more detail, we introduce the following notions: Let $\mathbb{V}.\text{types} = \bigcup_{v \in \mathbb{V}} \mathit{type}(v)$ be the set of all existing node types in SKB, where $\mathit{type}(v)$ represents the node type (e.g., "protein") of any node $v \in \mathbb{V}$. Let $y.\text{type} \in \mathbb{V}.\text{types}$ be a predicted node type of the target variable y, and $\mathbb{C}(y.\mathit{type}) \subseteq \mathbb{C}$ the set of all candidate nodes y with $\mathit{type}(y) = y.\text{type}$. $\mathit{NEIGHBORS}(v, e.\text{type}, \mathbb{E}, +)$ returns the union of all nodes u for which an edge of type $e.\text{type} \in \mathbb{E}.\text{types}$ exists that points from u to v. $\mathit{NEIGHBORS}(v, e.\text{type}, \mathbb{E}, -)$ denotes the union of all nodes u for which an edge of type $e.\text{type} \in \mathbb{E}.\text{types}$ exists that points in the other direction – from v to u. A distinction is often made between variables and constants. While constants typically represent a single, fixed value or specific entity, variables act as placeholders for one or more unknown values or entities. In this work, we refer to constants as a small, finite set of candidate nodes that we incrementally extend for entities with a name or title attribute that can be used for a database scan. We define variables as entities without a searchable name or title. We use the term symbols for sets of both constants and variables. This approach acknowledges the potential for ambiguity in certain contexts. For instance, the constant "Mexico" could semantically refer to either the country or the city (i.e., Mexico City). Let S be a dictionary, assigning a sorted array of candidate nodes $\boldsymbol{s}(x)$ to each symbol x. $S[:l]$ denotes a clone of S, where each assigned array $\boldsymbol{s}(x)$ is replaced by a set of its first l nodes.

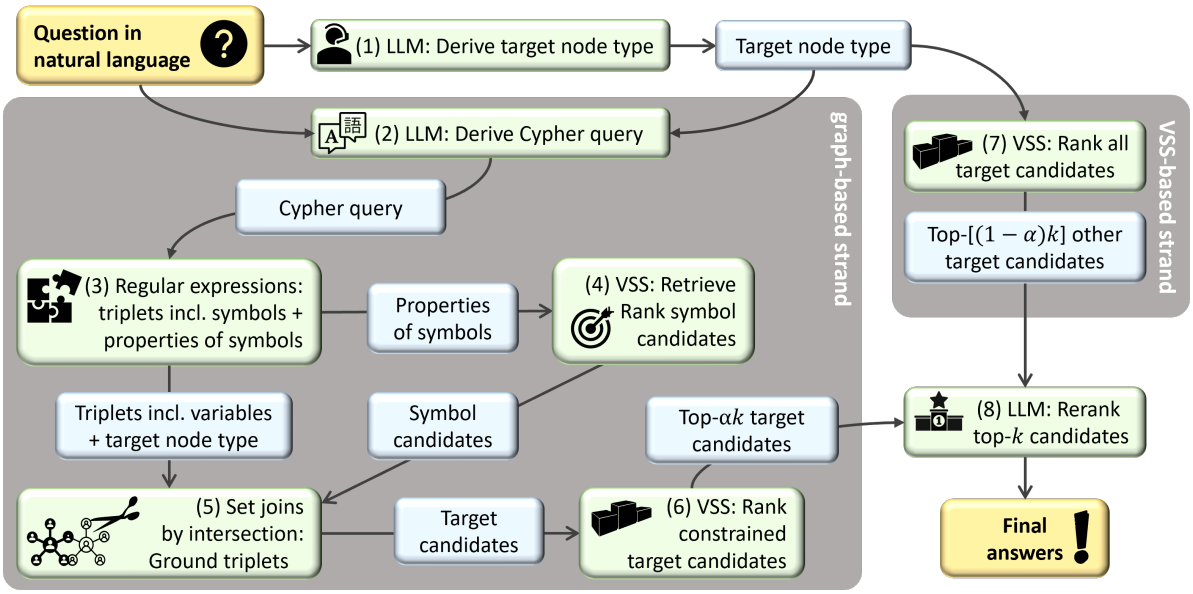

Figure 1: Overview of Autofocus-Retriever's processing from the user query q to the final answers. The green boxes represent its eight steps, the blue boxes represent the objects passed between them.

Algorithm 1 AF-Retriever

Input:
a Semi-Structured Knowledge Base $SKB = (\mathbb{V}, \mathbb{E}, \mathbb{D})$, a query q in natural language, candidates for query answers $\mathbb{C} \subseteq \mathbb{V}$, number of candidates to rank and return k, an upper bound for the number of candidates per constant to consider l_{max}, the graph-based retrieval weight $\alpha \in [0, 1]$

Output: A list of k answers to q in descending order

y.type $\leftarrow$ DERIVE_TARGET_TYPE($q, \mathbb{V}$.types) ▷ step 1
$q'_{\text{cypher}} \leftarrow$ DERIVE_CYPHER($q, \mathbb{V}$.types, $\mathbb{E}$.types, y.type) ▷ step 2
y_{cypher}.type, $\mathbb{T}, \mathbb{S}_{raw} \leftarrow$ REGEX($q'_{\text{cypher}}, \mathbb{V}$.types, $\mathbb{E}$.types) ▷ step 3
$\mathbb{S} \leftarrow$ RETRIEVE_SYMBOL_CANDIDATES($\mathbb{S}_{raw}, \mathbb{V}, \mathbb{D}, l_{max}$) ▷ step 4
$\mathbb{C}_{\text{cypher}} \leftarrow \emptyset$ ▷ start of step 5
$l \leftarrow 1$
while $|\mathbb{C}_{\text{cypher}}| < k$ & $l < l_{max}$ **do**
  $\mathbb{C}_{\text{cypher}} \leftarrow$ GROUND_TRIPLETS($\mathbb{T}, S[:l], y_{\text{cypher}}$.type, $\mathbb{E}$) ▷ Algorithm 2
  $l \leftarrow l^{1.5} + 0.5$
end while ▷ end of step 5
$\boldsymbol{y}_{\text{cypher}} \leftarrow$ VSS($q, \mathbb{C}_{\text{cypher}}, \alpha k, SKB$) ▷ step 6
$\boldsymbol{y}_{\text{vss}} \leftarrow$ VSS($q, \mathbb{C}(y.\text{type}) \backslash \mathbb{C}_{\text{cypher}}, k - |\boldsymbol{y}_{\text{cypher}}|, SKB$) ▷ step 7
$\boldsymbol{y} \leftarrow \boldsymbol{y}_{\text{cypher}} + \boldsymbol{y}_{\text{vss}}$ ▷ concatenate arrays
$\boldsymbol{y} \leftarrow$ RERANK($\boldsymbol{y}, SKB$) ▷ step 8
return $\boldsymbol{y}$

k and l_{max} are hyperparameters, which put a ceiling on the entity search. $\alpha \in [0, 1]$ weighs between two retrieval strategies. These hyperparameters are described below in more detail.

3.2 Pipeline of AF-Retriever

Figure 1 provides a high-level overview of the key components of our framework and the flow of intermediates at its interfaces. Algorithm 1 formalizes the eight steps of AF-Retriever. Each of these steps calls one of the functions explained below with a running example.

Step 1) Target type prediction: As the first step, line 1 of Algorithm 1, DERIVE_TARGET_TYPE(q, $\mathbb{V}$.types) is called, which lets an LLM predict the node type of any answer to the question q. In essence, the LLM is prompted: "An instance of which of the given entity types $\mathbb{V}$.types could correctly answer the query q?" The full templates of all LLM prompts used in our implementation are provided in Appendix A.2. This filter step constitutes the first layer of focus and is considered in both strands of the hybrid framework.

Example input (above the line) and output (below the line) of DERIVE_TARGET_TYPE:
q = "Did any University from Miami produce research in 2015 in molecular biology?" $\mathbb{V}$.types = ["paper","author","institution","field of study"] ——— y.type = "paper"

Step 2) Cypher query extraction: Subsequently, in line 2, DERIVE_CYPHER($q, \mathbb{V}$.types, $\mathbb{E}$.types, y.type) uses an LLM to extract queryable information from q. In summary, the LLM is prompted: "Return a Cypher query derived from the given query q, ensuring that the following restrictions are met: [...]" As central part of these restrictions, the prompt includes the available node and edge types $\mathbb{V}$.types and $\mathbb{E}$.types (called "labels" in Cypher) as well as y.type from the previous step. This inclusion is important to separate the relational information that can be queried in an SKB from non-relational entity features included in q that might be only available in the text documents $\mathbb{D}$ but not in the knowledge graph $\mathcal{G}$. Our extended analysis in section 4 confirms that even smaller, open base models are able to produce Cypher queries making instructions for a custom machine-readable format redundant.

Output of DERIVE_CYPHER($q, \mathbb{V}$.types, $\mathbb{E}$.types, y.type) for the running example:
q'_{cypher} = *"MATCH (i:institution {name: 'Miami uni'})<-[:employed_at]-(a:author)-[:wrote]->(p:paper)* *p-[:has_field_of_study]->(f:field_of_study {name: 'molecular biology'})* *WHERE p.publication_year = 2015* *RETURN p.title".*

Step 3) Parsing with regular expressions: Instead of executing q'_{cypher} in a Cypher-driven database directly, AF-Retriever parses q'_{cypher} into a structured format via nested regular expressions in line 3. This detour enables more controlled downstream processing and supports exchangeable LLMs that usually cannot create vector search-based Cypher queries. REGEX($q'_{\text{cypher}}, \mathbb{V}$.types, $\mathbb{E}$.types) extracts the target variable's node type y_{cypher}.type $\in \mathbb{V}$.types, a set of relational triplets $\mathbb{T}$, and node attributes $\mathbb{S}_{raw}$ from q'_{cypher}, based on Cypher keywords like "MATCH", "WHERE", "RETURN", "CONTAINS", and other standardized syntax. Since the LLM may violate the restriction of returning a variable of type y.type, y_{cypher}.type does not necessarily equal y.type from step 1. Each triplet $\tau \in \mathbb{T}$ is a tuple (h, e, t) of an edge type $e \in \mathbb{E}$.types and two symbols. A symbol is either categorized as a constant if a name or title attribute is given, and as a variable otherwise. Each attribute in S_{raw} is a tuple of a symbol ID, its node type, an attribute name $p \in \mathbb{V}$.attr, and an attribute value. REGEX can filter out invalid or unsupported components from q'_{cypher}.

Output of REGEX($q'_{\text{cypher}}, \mathbb{V}$.types, $\mathbb{E}$.types) for the running example:
y_{cypher}.type = "paper" $\mathbb{T}$ = {(a, "employed at", i), (a, "wrote", p), (p, "has field of study", f)} S_{raw} ={(a, "type", "author"), (p, "type", "paper"), (f, "type", "field_of_study"), (i, "type", "institution"), (p, "publication year", =2015), (f, "name", "molecular biology"), (i, "name", "Miami uni")}

Step 4) Symbol candidates retrieval: RETRIEVE_SYMBOL_CANDIDATES($\mathbb{S}_{raw}, \mathbb{V}, \mathbb{D}, l_{max}$) in line 4 then retrieves l_{max} candidates for each constant and filters an unlimited number of candidates for variables with at least one attribute. The "title" and "name" attributes of each constant are used as their individual search strings, which will be used later. Every attribute that is neither "title" nor "name" is applied as a hard filter for candidates for each symbol. For example, "year $\geq$ 2012" rules out all papers published before 2012. If a filter cannot be applied because an attribute name or value is unavailable in the SKB and it belongs to a constant, then this attribute is appended to its search string. Finally, RETRIEVE_SYMBOL_CANDIDATES sorts each constant's candidates by their vector cosine similarity to the embedding of the constant's search string, then cuts the top l_{max} out. To illustrate why more than one candidate for a constant may be necessary, consider the simple example of searching for a paper by "J. Smith" published in the journal "Nature". Ambiguous references like this can yield multiple plausible, yet contextually inaccurate candidates for both constants. For instance, the first hit for "J. Smith" might have no "Nature" publication, while the second or third hit does. Regarding the entity and relation types, we implement two contrary variants to customize restrictiveness: a) Strict mode: Apply a hard filter on nodes and edges by their predicted entity and relation types, b) lenient mode: Accept any annotated node or edge type with a high vector similarity for higher error tolerance against incorrect entity and relation type classifications.

Output of RETRIEVE_SYMBOL_CANDIDATES for the symbol i of the running example, $l_{max} = 3$:
$\boldsymbol{s}(i)$ =["University of Miami", "Miami University", "Miami Dade College"]

5) Triplets grounding: The purpose of lines 5–10, given the output of the previous steps, is to identify at least k answers (candidates for the target variable y), while keeping the total number of candidates for the target variable and other symbols low to maintain precision. The objective is to focus on k answers, because this hyperparameter corresponds to the predefined number of candidates that are handled by a reranker in the last step. To achieve this balanced scope, the triplets are first grounded with the first hit for each constant ($l = 1$). Then, the focus is expanded by gradually incrementing the number of candidates l per constant until either k answers are found or the limit l_{max} is reached. To reduce the number of steps and thus computation, l is incremented exponentially. In the strict mode, candidates are also filtered by edge types. See Algorithm 2 in Appendix A.3 for details on GROUND_TRIPLETS($\mathbb{T}, S[:\ l], y_{\text{cypher}}.\text{type}, \mathbb{E}$), which returns the candidate set for y depending on l in each iteration. In summary, each triplet (h, e, t) refines the candidate sets for each symbol by propagating constraints bidirectionally via set intersections. This loop continues until convergence (no further eliminations), and the final target candidate set $\mathbb{C}_{\text{cypher}}$ is returned.

Output of GROUND_TRIPLETS($\mathbb{T}, S[:\ l], y_{\text{cypher}}.\text{type}, \mathbb{E}$) for the running example:
Output for $l = 1$: $\mathbb{C}_{\text{cypher}} = \emptyset$ **Output for $l = 2$:** $\mathbb{C}_{\text{cypher}}$ =[{type:"paper", title: "RNA Transcription", year:2015, ...}, {type:"paper", title: "Review on Ribosomes", year:2015, ...}, ...]

Steps 6 and 7) Vector similarity search: Steps 6 and 7 both rank nodes based on their vector cosine similarity to the search string. In both cases, our VSS routine accepts a search string, a candidate set, and a parameter that specifies the number of candidates to return.

Step 6 applies VSS to the question q, the candidates $\mathbb{C}_{cypher}$, and round(αk) as the number of candidates to return in line 11. To increase the chance of hits in a two-model ensemble fashion, step 7 retrieves round($(1 - \alpha)k$) additional top candidates retrieved independently of the Cypher-derived constraints, by calling VSS for the entire candidate set $\mathbb{C}$ in line 12. If y.type is a valid node type, step 7 only considers nodes of this type.

Example output:

Step 6) VSS
$\boldsymbol{y}_{\text{cypher}}$ =[{type:"paper", title: "Review on Ribosomes", year:2015, ...},
{type:"paper", title: "RNA Transcription", year:2015, ...}, ...]

Step 7) VSS
$\boldsymbol{y}_{\text{vss}} = \mathbb{C}_{\text{cypher}}$ =[{type:"paper", title: "Biodiversity in Miami", year:2015, ...},
{type:"paper", title: "RNA Transcription", year:2015, ...}, ...]

Step 8) LLM Reranking: Finally, after combining $\boldsymbol{y}_{cypher}$ and $\boldsymbol{y}_{vss}$ into a unified vector $\boldsymbol{y}$ (line 13), the top-k answers are reranked (line 14). Although reranking does not alter the set of top-k answers, it adjusts their order to increase the number of first hits based on the complete information available for the top candidates. We implement three LLM-based reranking options: pointwise, pairwise, and listwise. The *listwise* reranker sends a single prompt containing all top-k candidates, along with their node attributes, annotated documents, 1-hop relations, and 2-hop relations where the relation type to the candidate is 1-1 or n-1. The LLM is then instructed to order the candidates according to how well they answer q. This straightforward approach requires only one prompt with $O(k)$ tokens, but it demands an LLM with a sufficient large context window and memory to handle all k descriptions simultaneously. The *pairwise* reranker repeatedly prompts the descriptions of only two candidates to an LLM at once and asks which candidate answers the question better. Assuming transitivity, $O(k{*}\log k)$ tokens need to be sent, but a smaller LLM context window is required because less information has to be processed at once. The *pointwise* reranker, as employed by Wu et al. (2024b), presumes that LLMs can produce comparable confidence scores for individual candidates. Our extended experiments show that while the pointwise alternative improves ranking quality over VSS, it performs worse than the two differential (pairwise and listwise) variants on all datasets.

Our implementation of all rerankers provides for three scenarios: 1) In the default case, the node description in every prompt includes all 1-hop and the selected 2-hop relations described above. 2) If the context window is exceeded, the candidate's edges (relations) that are not incident to any retrieved non-target entity candidate are removed from the prompt. 3) If the prompt still exceeds the context limit, all relations are omitted entirely. In practice, even for the listwise reranker, the context window is only rarely exceeded (see Appendix A.6.7).

See Table 1 for a theoretical comparison of the reranking approaches, Appendix A.6.7 for an experimental comparison, and Appendix A.2 for the applied prompts.

Output of RERANK($\boldsymbol{y}$, SKB) for the running example:

$\boldsymbol{y}$ =[{type:"paper", title: "RNA Transcription", year:2015, ..., source: $\boldsymbol{y}_{cypher}$},
{type:"paper", title: "Review on Ribosomes", year:2015, ..., source: $\boldsymbol{y}_{cypher} + \boldsymbol{y}_{vss}$}, ...]

Table 1: Alternative LLM reranking approaches implemented in AF-Retriever for k candidates.

	main characteristic	expected #prompts	expected #tokens
pointwise reranker	one prompt per item assigning a confidence score between 0 and 1 to each item	k	$O(k)$
listwise reranker	one prompt including all entities requesting a sorted list of them	1	$O(k)$
pairwise reranker	binary insertion sort using one prompt for each pairwise comparison during sorting	$k * \log(k)$	$O(k * \log(k))$

4 Evaluation

Datasets We evaluate AF-Retriever on the three STaRK QA benchmark datasets (Wu et al., 2024b): PRIME, MAG, and AMAZON, each paired with a semi-structured knowledge base (SKB). Each STaRK benchmark provides a set of synthetic training, validation, and test queries, as well as a smaller set of human-generated questions. These benchmarks simulate realistic scenarios by combining relational and textual knowledge with diverse, natural-sounding question formats. The SKBs vary substantially: *PRIME (medical domain)* has the richest relational structure, with diverse node and edge types, and a strong focus on relations. *AMAZON (product recommendation)* includes fewer node types but a large proportion of unstructured textual data (i.e., reviews). *MAG (academic paper search)* strikes a balance, featuring multi-hop relational queries as well as textual attributes like paper abstracts. Each SKB contains hundreds of thousands to millions of nodes and edges. In PRIME, every node is a potential answer. In MAG and AMAZON, only all nodes of specific types (papers and products, respectively) qualify. For additional details on the SKBs and datasets, we refer readers to Appendix A.1 and Wu et al. (2024b).

Experiment settings We report standard retrieval metrics averaged over test queries: *hit@m*: Mean of a binary indicator of whether at least one correct item is in the top-m results. *recall@m*: Mean proportion of relevant items among the top-m. *mrr*: **M**ean **r**eciprocal of the **r**ank at which the first relevant answer appears, with m being the maximum of every rank. We use GPT OSS 120B as the LLM backbone, and OpenAI's text-embedding-3-small model for vector-based semantic search of AF-Retriever. We set the maximum number of candidates per symbol $l_{max} = 100$, and, in compliance with the LLM reranker of Wu et al. (2024b), $k = 20$. The hyperparameter α is set to $2/3$, which prefers the graph-based retrieval over the mainly vector-based strand of Algorithm 1, but still includes a reasonable amount of the most vector-similar answers of the predicted node type. An analysis on the STaRK validation and test sets, presented

Table 2: Performance of benchmark methods and AF-Retriever averaged over the three synthetic and human-generated STaRK benchmark test sets. See Section A.4 and Appendix A.4 for full method names, references, and used base LLMs. Results above the horizontal line belong to zero- and one-shot methods, while those below are based on data-specific fine-tuning. A dash indicates that no result is reported for one or more datasets for the concerned metric. The best results are marked in bold, the second-best are underlined. "*" denotes results that were only published for the 10%-version of the synthetically generated test sets.

STaRK test sets	synthetically generated			human-generated		
metric (in percent)	hit@1	hit@5	mrr	hit@1	hit@5	mrr
BM25	27.8	46.9	36.7	-	-	-
VSS (Ada-002)	27.0	47.9	36.8	28.5	46.8	38.3
VSS (Multi-Ada)	27.0	49.7	37.3	-	-	-
DPR	10.1	35.0	21.3	7.6	19.4	14.1
VSS + Reranker*	34.7	55.5	43.7	39.9	55.5	46.4
QA-GNN	16.1	36.8	27.2	-	-	-
ToG	-	-	-	-	-	-
AvaTaR	37.6	55.2	45.5	41.6	56.9	48.5
4StepFocus*	46.9	63.3	54.6	<u>52.9</u>	61.5	50.3
KAR	45.0	62.5	53.0	52.6	<u>63.9</u>	<u>57.6</u>
HybGRAG	-	-	-	-	-	-
ReAct	29.5	48.7	38.1	31.6	48.4	40.3
Reflexion	32.6	51.5	41.6	31.5	45.5	39.8
AF-Retriever	**62.0**	**76.7**	**68.4**	**55.8**	**66.4**	**60.7**
PASemiQA	39.6	-	45.6	-	-	-
mFAR	43.7	67.5	54.5	-	-	-
MoR	<u>48.9</u>	<u>71.0</u>	<u>58.8</u>	-	-	-
GraphRAFT	-	-	-	-	-	-

Table 3: Performance of benchmark methods and AF-Retriever on the synthetically generated test sets. Results above the horizontal line belong to zero- or one-shot methods, while those below are based on fine-tuned LLMs. "*" denotes results that were only published for the 10%-version of the synthetically generated test sets.

STaRK test set	PRIME			MAG			AMAZON		
metric (in percent)	hit@1	hit@5	mrr	hit@1	hit@5	mrr	hit@1	hit@5	mrr
BM25	12.8	27.9	19.8	25.8	45.2	34.9	44.9	67.4	55.3
VSS (Ada-002)	12.6	31.5	21.4	29.1	49.6	38.6	39.2	62.7	50.4
VSS (Multi-Ada)	15.1	33.6	23.5	25.9	50.4	36.9	40.1	65.0	51.6
DPR	4.5	21.8	12.4	10.5	35.2	21.3	15.3	47.9	30.2
VSS+[pointw.]Reranker*	18.3	37.3	26.6	40.9	58.2	49.0	44.8	71.2	55.7
QA-GNN	8.8	21.4	14.7	12.9	39.0	29.1	26.6	50.0	37.8
ToG	6.1	15.7	10.2	13.2	16.2	14.2	-	-	-
AvaTaR	18.4	36.7	26.7	44.4	59.7	51.2	49.9	69.2	58.7
4StepFocus*	39.3	53.2	45.8	53.8	69.2	61.4	47.6	67.6	56.5
KAR	30.4	49.3	39.2	50.5	65.4	57.5	54.2	68.7	61.3
HybGRAG	28.6	41.4	34.5	65.4	75.3	69.8	-	-	-
ReAct	15.3	32.0	22.8	31.1	49.5	39.2	42.1	64.6	52.3
Reflexion	14.3	35.0	24.8	40.7	54.4	47.1	42.8	65.0	52.9
AF-Retriever (ours)	46.2	63.7	54.0	**78.6**	**91.4**	**84.0**	**61.2**	**75.2**	**67.3**
PASemiQA	29.7	-	31.0	43.2	-	50.2	45.9	-	55.7
mFAR	40.9	62.8	51.2	49.0	69.6	58.2	41.2	70.0	54.2
MoR	36.4	60.0	46.9	58.2	78.3	67.1	52.2	74.6	62.2
GraphRAFT	**63.7**	**75.4**	**69.0**	71.1	85.3	76.9	-	-	-

in Appendix A.6.6, suggests that the results presented here can still be further improved slightly, if α was optimized on a validation set. For step 8, we chose the pairwise reranker with binary insertion sort.

Main test results Table 2, compares the average performance of AF-Retriever with the reported results of other baseline and SOA methods (including both peer-reviewed publications and preprints) on the three synthetically- and human-generated test sets. The results on the synthetic test sets are broken down further for each dataset in Table 3. Due to the small size of the human-generated test sets (81-98 compared to 1642-2801 QA pairs) and the limited availability of published results of other methods, especially those of fine-tuning-based methods, we present our detailed results on these sets in Appendix A.5 only. The Methods are categorized as either zero- or one-shot methods, which do not require domain-specific training data, or as systems that rely on fine-tuning. AF-Retriever achieves state-of-the-art performance across all three synthetic STaRK benchmarks, outperforming all other zero- and one-shot baselines in terms of hit@1, hit@5, and mrr. For instance, with regard to hit@1, AF-Retriever outperforms the second-best method (KAR) on the AMAZON dataset by 12.9%. On PRIME, it surpasses the runner-up (4StepFocus) by 17.6%, and on MAG it beats the second-best method (HybGRAG) by 20.2%.

Taking methods into account that require training on individual datasets, GraphRAFT is the only model that outperforms AF-Retriever on the PRIME QA set, achieving a hit@1 rate that is 39.7% higher. This finding suggests that interpreting highly specialized medical entities, characterized by numerous and diverse node and edge types, is particularly challenging for out-of-the-box LLMs. Our extended analysis across multiple LLMs further supports this hypothesis: for PRIME, retrieval quality is stronger influenced by the specific LLM used, whereas on other datasets, several language models perform similarly. Moreover, GraphRAFT's hit@1 rate drops from 44.2% to 14.7% on the PRIME validation set when fine-tuning is omitted (Clemedtson & Shi, 2025). On the MAG test set, AF-Retriever also surpasses GraphRAFT across all evaluation metrics, although the latter consists of fine-tuned components.

Table 4: Ablation study of AF-Retriever. Target node type prediction is omitted for AMAZON and MAG because all questions target the same type. While "step 1 + 7" omits the graph-based retrieval, "steps 1 to 7" describes the hybrid variant with $\alpha = 2/3$. For "steps 1 to 6", in between, the metrics are calculated based on the chronological retrieval order because the answers are actually not yet ordered in another way.

dataset	PRIME				MAG				AMAZON			
metric in %	h@1	h@20	r@20	mrr	h@1	h@20	r@20	mrr	h@1	h@20	r@20	mrr
steps 1 + 7	15.1	48.9	34.1	24.0	32.2	68.3	44.4	42.3	42.1	**81.3**	**35.7**	53.0
steps 1 to 5	23.7	46.8	33.4	29.6	24.0	72.0	44.7	37.1	5.7	19.1	9.8	8.8
steps 1 to 6	30.5	52.4	37.6	36.5	51.7	80.6	51.5	61.8	15.0	28.1	14.6	18.6
steps 1 to 7	33.8	**70.9**	**51.2**	42.5	59.5	**94.3**	**61.4**	71.3	35.6	80.5	35.5	45.1
steps 1 to 8	**46.2**	**70.9**	**51.2**	**54.0**	**78.6**	**94.3**	**61.4**	**84.0**	**61.2**	80.5	35.5	**67.3**

Ablation study Our ablation results are summarized in Table 4. The first row covers the purely VSS-based retrieval with target type prediction if applicable. The next two rows represent the graph-based, neuro-symbolic retrieval, without and with VSS-based ranking of target candidates. The steps 1 to 5 are summarized here, because they are interlocked and only deliver target candidates in combination. Aspects of the individual steps' performance are addressed in the further analysis in the next but one paragraph. The last two lines show the combination of both retrieval strands without and with LLM-based reranking, representing the complete hybrid method.

This study confirms the contribution of each step, because recall@20 steadily increases from step 5 to 7 until the LLM reranker further improves hit@1 and hit@5. Contrary to the trend, step 7 has more hits than steps 2 to 6 on AMAZON regarding hit@20 in 0.2% of the test pairs, and even better than their combination with the non-optimized setting of $\alpha = 2/3$. Table 14 in Appendix reveals that while the specificity of step 5 is 64% and 97% on PRIME and MAG, respectively, if at least one target candidate is returned, it is only 44% for AMAZON, probably due to the concentration on unstructured knowledge. Besides this exception, the hybrid ensemble (steps 1 *to* 7) performs considerably better than either retrieval strategy (step 1 + 7 vs. steps 1 to 6). Across all datasets, although performance can vary noticeably at the edge cases, typically favoring the graph-based strand of the framework, choosing α between 0.5 and 0.8 consistently yields near-optimal results and demonstrates robustness to the precise value. See Figure 2 and Figure 3 in Appendix A.6.6 for details. Finally, the reranker substantially increases the number of first hits and the reciprocal ranks without exception, although hit@1 is still much lower than hit@20, its upper bound.

Further analysis of individual steps This subsection highlights the effect of different design choices for each step, summarizing only the major findings. Comprehensive results with detailed tables are provided in Appendix A.6.

Target node type prediction (steps 1 and 2, Appendix A.6.1): Since all answer candidates of MAG and AMAZON are of the same node type, we only validate the target type prediction on PRIME. Of eight LLMs, only one produced any invalid answers in our experiment, i.e., more than one node type. In 5 out of 280 cases (1.3%), the prediction (y.type) of GPT OSS 120B and Gemma 3 27B, the best performing LLMs on this task is incorrect, confusing disease with effect/phenotype for instance. These errors increase after step 2 and 3, where the target type y_{cypher}.type is parsed from the predicted Cypher query. Only GPT-5 mini and GPT OSS 120B produce syntactically correct answers for each test question and predict 97.8% to 98.2% of the target types correctly.

Strict vs. lenient parsing mode (step 3, Appendix A.6.2): Lenient parsing, which does not filter symbol candidates by their predicted node type, yields the worst results across all datasets. On MAG, hit@20 decreases from 95.6% to 86.1%, while the delta is smaller than 2% on the other datasets. More surprisingly, considering only node types, but ignoring edge types, leads to better performance on PRIME than considering both, although the SKB contains contradictory edges, e.g., "upregulates" and "downregulates". It appears that VSS compensates for this laxity in practice, where the types are included in embedded node descriptions.

Inclusion vs. exclusion of relations in text embeddings of nodes (steps 4, 6, and 7, Appendix A.6.4): We observe that including relational information (up to two hops) in node descriptions before embedding them makes a difference, especially for step 7, which uses no preselection of candidates. Inclusion is crucial for direct retrieval tasks because questions with relational parts cannot be answered without it. However, as our experiments confirm, this additional information can be misleading when used to search for non-target entity candidates (step 3), which reduces performance on PRIME and MAG.

Omitting ambiguous node types (step 2, Appendix A.6.5): Missing links or entities pose a major challenge when working with knowledge graphs. For instance, in the AMAZON SKB, products are linked to categories in a one-to-one relationship. However, considering other possible taxonomies, one product could be assigned to many categories in different subjective ways. While removing "field of study" for MAG from the list of node types in the LLM prompt of step 2 lifts hit@20 from 88% to 96%, hiding the node type "category" for AMAZON does not confirm our assumption and slightly reduces the performance.

Stopping criterion for graph-based retrieval (step 5, Appendix A.6.3): While on PRIME, the overall pipeline reaches its peak for hit@20 at $l_{max} = 100$, the optimal value for l_{max} is 10 on MAG and 1 on AMAZON.

Comparison of reranking strategies (step 8, Appendix A.6.7): All three LLM-based reranking strategies (pointwise, pairwise, and listwise) substantially improve the ranking on the validation sets (see Table 18). The pointwise approach consistently delivers the worst results in terms of first hits and mrr. The LLM's context window is largely sufficient even for listwise reranking in our experiments. Whether listwise or pairwise reranking performs better depends on the dataset, with the listwise approach requiring about 60 ($3k$) times less prompts and six to eight times less input tokens (see Table 17). This is especially noteworthy, because AF-Retriever's running time is heavily dominated by the LLM latency and reranker choice (see Appendix A.6.8).

5 Conclusion and Future Work

We introduced AF-Retriever, a modular and traceable framework for multi-hop question answering over Structured Knowledge Bases (SKBs). It interlinks several neural and symbolic components into a traceable workflow of established and novel approaches. These approaches include target type prediction, vector-semantic search with and without embedded relational knowledge, text-to-Cypher translation for standardized parsing, triplet-based graph exploration, and LLM-driven reranking. AF-Retriever is characterized by an incremental expansion strategy for the scope of constant-like entity candidates in its graph-based retrieval. This strategy balances sensitivity and specificity by searching for k answers that satisfy all relational constraints while minimizing false positives. The system robustly handles both structured relations and unstructured textual information by sequentially coupling these components and incorporating a hybrid retrieval strategy.

Across diverse domains (biomedical, academic paper search, and product recommendation), AF-Retriever demonstrates strong empirical performance. It outperforms state-of-the-art results on the STaRK benchmarks regarding several metrics. Its design is modular, and, due to grounded triplets, AF-Retriever's answers are traceable. Modularity and traceability are two crucial features for deploying LLM-based QA systems in real-world scenarios. An extensive ablation study and evaluation of different compatible design choices highlight the contribution of each component. Notably, AF-Retriever can operate in a zero-shot setting, requiring neither task-specific training data nor fine-tuning, which makes it adaptable across domains.

Future work may explore the potential of fine-tuning individual components, such as Cypher generation and reranking, to achieve greater gains when training data for specific steps are available. Additionally, extending the framework to support expressive logical reasoning would increase its applicability. In summary, AF-Retriever and the modular evaluation of this work take an important step toward reliable and transparent high-precision question answering grounded in semi-structured knowledge.

References

Garima Agrawal, Tharindu Kumarage, Zeyad Alghamdi, and Huan Liu. Can knowledge graphs reduce hallucinations in llms?: A survey. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 3947–3960, 2024.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In Proc. of FaccT 2021, pp. 610–623, 2021.

Derian Boer, Fabian Koch, and Stefan Kramer. Harnessing the power of semi-structured knowledge and llms with triplet-based prefiltering for question answering. In International Joint Conference on Learning and Reasoning, pp. 283–292. Springer, 2024.

Y. K. Chia, P. Hong, L. Bing, and S. Poria. InstructEval: Towards holistic evaluation of instruction-tuned large language models. In Proc. of the First Workshop on the Scaling Behavior of Large Language Models (SCALE-LLM 2024), pp. 35–64, 2024.

Alfred Clemedtson and Borun Shi. Graphraft: Retrieval augmented fine-tuning for knowledge graphs on graph databases. arXiv preprint arXiv:2504.05478, 2025.

Philip Feldman, James R Foulds, and Shimei Pan. Trapping llm hallucinations using tagged context prompts. arXiv preprint arXiv:2306.06085, 2023.

Chao Feng, Xinyu Zhang, and Zichu Fei. Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs. arXiv preprint arXiv:2309.03118, 2023.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In Proceedings of the 2018 international conference on management of data, pp. 1433–1445, 2018.

X. Guan, Y. Liu, H. Lin, Y. Lu, B. He, X. Han, and L. Sun. Mitigating large language model hallucinations via autonomous knowledge graph-based retrofitting. In Proc. of AAAI 2024, pp. 18126–18134, 2024.

Subbarao Kambhampati. Can large language models reason and plan? Annals of the New York Academy of Sciences, 1534(1):15–18, 2024.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In EMNLP (1), pp. 6769–6781, 2020.

A. Kundu and U. T. Nguyen. Automated fact checking using a knowledge graph-based model. In Prof. of ICAIIC 2024, pp. 709–716, 2024.

Meng-Chieh Lee, Qi Zhu, Costas Mavromatis, Zhen Han, Soji Adeshina, Vassilis N Ioannidis, Huzefa Rangwala, and Christos Faloutsos. Hybgrag: Hybrid retrieval-augmented generation on textual and relational knowledge bases. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 879–893, 2025.

Yongjia Lei, Haoyu Han, Ryan A Rossi, Franck Dernoncourt, Nedim Lipka, Mahantesh M Halappanavar, Jiliang Tang, and Yu Wang. Mixture of structural-and-textual retrieval over text-rich graph knowledge bases. arXiv preprint arXiv:2502.20317, 2025.

Millicent Li, Tongfei Chen, Benjamin Van Durme, and Patrick Xia. Multi-field adaptive retrieval. arXiv preprint arXiv:2410.20056, 2024.

Michalis Mountantonakis and Yannis Tzitzikas. Using multiple RDF knowledge graphs for enriching ChatGPT responses. In Prof. of the ECML PKDD 2023 Demo Track, pp. 324–329. Springer, 2023.

Jeff Z. Pan and et al. Large language models and knowledge graphs: Opportunities and challenges. TGDK, 1(1):2:1–2:38, 2023. doi: 10.4230/TGDK.1.1.2.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. IEEE Trans. Knowl. Data Eng., 36(7):3580–3599, 2024. doi: 10.1109/TKDE.2024.3352100.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at TREC-3. British Library Research and Development Department, 1995.

G. K. Shahi. Fakekg: A knowledge graph of fake claims for improving automated fact-checking (student abstract). Proceedings of the AAAI Conference on Artificial Intelligence, 37(13):16320–16321, Jul. 2024. doi: 10.1609/aaai.v37i13.27020.

Danish Shakeel and Nitin Jain. Fake news detection and fact verification using knowledge graphs and machine learning. ResearchGate preprint, 10, 2021.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems, 36: 8634–8652, 2023.

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

Nikhita Vedula and Srinivasan Parthasarathy. FACE-KEG: fact checking explained using knowledge graphs. In Prof. of ACM WSDM 2021, pp. 526–534. ACM, 2021.

Y. Wang, N. Lipka, R. A. Rossi, A. Siu, R. Zhang, and T. Derr. Knowledge graph prompting for multi-document question answering. Proc. of AAAI 2024, (17):19206–19214, Mar. 2024.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis Ioannidis, Karthik Subbian, Jure Leskovec, and James Y Zou. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. Advances in Neural Information Processing Systems (NeurIPS), 37:25981–26010, 2024a.

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), Advances in Neural Information Processing Systems (NeurIPS 2024), volume 37, pp. 127129–127153. Curran Associates, Inc., 2024b.

Yu Xia, Junda Wu, Sungchul Kim, Tong Yu, Ryan A. Rossi, Haoliang Wang, and Julian McAuley. Knowledge-aware query expansion with large language models for textual and relational retrieval. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 4275–4286. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.216.

Hansi Yang, Qi Zhang, Wei Jiang, and Jianguo Li. Pasemiqa: Plan-assisted agent for question answering on semi-structured data with text and relational information. arXiv preprint arXiv:2502.21087, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In International Conference on Learning Representations (ICLR), 2023.

M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In Proc. of the NAACL 2021, pp. 535–546, 2021.

H. Zhuang, Z. Qin, K. Hui, J. Wu, L. Yan, X. Wang, and M. Bendersky. Beyond yes and no: Improving zero-shot llm rankers via scoring fine-grained relevance labels. In Proc. of the NAACL 2024, pp. 358–370, 2024.

A Appendix

A.1 STaRK SKB Properties

Table 5: Properties of the STaRK SKBs by Wu et al. (2024b)

	#entity types	#relation types	avg. degree	#entities	#relations	#tokens
AMAZON	4	4	18.2	1,035,542	9,443,802	592,067,882
MAG	4	4	43.5	1,872,968	39,802,116	212,602,571
PRIME	10	18	125.2	129,375	8,100,498	31,844,769

A.2 Used Prompts

Prompt of DERIVE_TARGET_TYPE:
Given several instances of these types: {candidate_types}. An instance of which type could correctly answer the query: {question}

Return nothing but the type of which the instance must be of.

Prompt of DERIVE_CYPHER:
Generate a Cypher query based on the given query Q. Please follow the restrictions precisely!

- Simple Syntax: Use a very basic and short Cypher syntax.
- Content Accuracy: Omit any information that cannot be exactly captured with one of the given, available node labels, or available keywords.
- No Quantifications: Avoid using quantifications.
- No Negations: Skip negated facts, avoid using "NOT" or "<>".
- No "OR": Do not use "OR".
- Available Keywords: Restrict yourself to the available keywords: MATCH, WHERE, RETURN, AND, CONTAINS.
- Date Format: Format dates as YYYY-MM-DD.

Given Information:

- Query Q: {query}
- Available Node Labels: {nodes_to_consider}
- Available Relationship Labels: {edges_to_consider}

Example: MATCH (d:disease)-[:is_effect/phenotype_of_disease]->(e:effect/phenotype)
MATCH (e)-[:protein/gene_is_associated_with_effect/phenotype]->(g:gene/protein)
WHERE g.name = "IGF1"
RETURN d.title

Prompt of pointwise reranker:

Examine if a {node_type} satisfies a given query and assign a score from 0.0 to 1.0. If the {node_type} does not satisfy the query, the score should be 0.0. If there exists explicit and strong evidence supporting that {node_type} satisfies the query, the score should be 1.0. If partial evidence or weak evidence exists, the score should be between 0.0 and 1.0. Here is the query: {query} Here is the information about the {node_type} Please score the {node_type} based on how well it satisfies the query. [...]

Prompt of listwise Reranker:

The rows of the following list consist of an ID number, a type and a corresponding descriptive text: {possible_answers} Sort this list in descending order according to how well the elements can be considered as answers to the following query: {query} Please make absolutely sure that the element which satisfies the query best is the first element in your order. Return ONLY the corresponding ID numbers separated by commas in the asked order.'

Prompt of pairwise Reranker:

The following two elements consist of an ID number, a type and a corresponding descriptive text: {node1_id}, {node_type_1}, {doc_info_1}. {node2_id}, {node_type_2}, {doc_info_2}. Please find out which of the elements satisfy the following query better: {query} Return ONLY the corresponding ID number which corresponds to the element that satisfies the given query best. Nothing else.

A.3 Triplet Grounding Algorithm

Algorithm 2 GROUND_TRIPLETS

Input:
a set of triplets $\mathbb{T}$, a dictionary $\mathbb{S}$ that assigns a set of candidate nodes $\mathbb{S}(x)$ to each symbol x, a target variable y, a set of edges $\mathbb{E}$

Output: A set of candidate nodes for y

$\mathbb{S}_{clone} \leftarrow \mathbb{S}$
for all $\tau = (h, e, t) \in \mathbb{T}$ **do**
  $\mathbb{N} \leftarrow \emptyset.$
  for all $h^{(j)} \in \mathbb{S}(h)$ **do**
    $\mathbb{N} \leftarrow \mathbb{N} \cup \text{NEIGHBORS}(h^{(j)}, e, \mathbb{E}, -)$
  end for
  $\mathbb{S}(h) \leftarrow \mathbb{S}(h) \cap N$
  $\mathbb{N} \leftarrow \emptyset.$
  for all $t^{(j)} \in \boldsymbol{s}(t)$ **do**
    $\mathbb{N} \leftarrow \mathbb{N} \cup \text{NEIGHBORS}(t^{(j)}, e, \mathbb{E}, +)$
  end for
  $\mathbb{S}(t) \leftarrow \mathbb{S}(t) \cap \mathbb{N}$
end for
if $\mathbb{S}_{clone} = \mathbb{S}$ **then**
  return $\mathbb{S}(y)$
else
  GROUND_TRIPLETS(T, S, y, E)
end if

A.4 Base LLMs of Related Work

Table 6: Categorized benchmark methods developed for SKB-based multi-hop question answering and their LLMs employed. Section 4 contrasts their performance on the STaRK benchmark sets.

retrieval category	method title	base LLM
textual	BM25: Best Matching 25 (Robertson et al., 1995)	-
	VSS (Ada-002): Vector Similarity Search	-
	VSS (Multi-ada-002): Multi-Vector Similarity Search	-
	DPR: Dense Passage Retriever (DPR) (Karpukhin et al., 2020)	RoBERTa
	VSS + Reranker: Vector Similarity Search + LMM Reranker(Chia et al., 2024; Zhuang et al., 2024)	GPT4
structural	QA-Graph Neural Networks (Yasunaga et al., 2021)	-
	ToG: Think-on-Graph (Sun et al., 2024)	-
hybrid	AvaTaR (Wu et al., 2024a)	Claude 3 Sonnet
	4StepFocus (Boer et al., 2024)	GPT4o
	KAR: Knowledge-Aware Retrieval (Xia et al., 2025)	GPT4o
	HybGRAG: Hybrid Retrieval-Augmented Generation on Textual and Relational Knowledge Bases (Lee et al., 2025)	Claude 3 Opus
	ReAct (Yao et al., 2023)	Claude 3 Opus
	Reflexion (Shinn et al., 2023)	Claude 3 Opus
	AF-Retriever (ours)	GPT OSS (120B)
hybrid with fine-tuned base LLM(s)	PASemiQA: Plan-Assisted Question Answering with Semi-structured data (Yang et al., 2025)	fine-tuned LLaMa2 (7B) + GPT4
	mFAR: Multi-Field Adaptive Retrieval (Li et al., 2024)	custom
	MoR: Mixture of Structural-and-Textual Retrieval (Lei et al., 2025)	fine-tuned LLaMa 3.2 (3B)
	GraphRAFT: Retrieval Augmented Fine-Tuning for Knowledge Graphs on Graph Databases	fine-tuned gemma2-9b-text2cypher +OpenAI gpt-4o-mini +LLaMa-3.1 (8B)

A.5 Results on Human-generated Test Sets

Table 7: Performance of baseline, SOA methods, and AF-Retriever on the human-generated STaRK benchmark test sets. Methods for which we did not find a reporting for these datasets are omitted.

dataset	PRIME			MAG			AMAZON		
metric (in percent)	hit@1	hit@5	mrr	hit@1	hit@5	mrr	hit@1	hit@5	mrr
VSS (Ada-002)	17.4	34.7	26.4	28.6	41.7	35.8	39.5	64.2	52.6
VSS (Multi-ada-002)	24.5	39.8	33.0	23.8	41.7	31.4	46.9	72.8	58.7
DPR	2.0	9.2	7.0	4.7	9.5	7.9	16.0	39.5	27.2
VSS + Reranker	28.4	48.6	36.2	36.9	45.2	40.3	54.3	72.8	62.7
AvaTaR	33.0	51.4	41.0	33.3	42.9	38.6	58.3	**76.5**	65.9
4StepFocus	50.5	65.5	57.9	47.6	51.2	49.2	60.5	67.9	64.3
KAR	45.0	60.6	51.9	51.2	58.3	54.5	**61.7**	72.8	**66.3**
ReAct	21.7	33.3	28.2	27.3	40.0	33.9	45.6	71.7	58.8
Reflexion	16.5	33.0	24.0	28.6	39.3	36.5	49.4	64.2	59.0
AF-Retriever (ours)	**57.1**	**69.4**	**62.7**	**52.4**	**60.7**	**55.9**	58.0	69.1	63.5

A.6 Results of Extended Analysis

All experiments of this extended modular analysis were conducted on the same 10% of the QA pairs draws from the validation sets. To avoid unnecessary computation, all metrics were measured before reranking in the following until Appendix A.6.7. Unless otherwise specified, the following parameters were used: $k = 20$, $\alpha = 2/3$, $l_{max} = 100$, LLM: GPT OSS 120B with medium reasoning effort, embeddings with relational information for step 7, and without relational information for steps 4 and 6.

A.6.1 LLM choice for target type prediction and Cypher formalization

Table 8: Properties of different LLM's evaluated in the rest of this subsection.

model name	manufacturer	open weights	# parameters	context window
GPT-5 mini	OpenAI	proprietary	not disclosed	400,000
GPT OSS 120B	OpenAI	yes	117B (5.1 activated)	131.072
LLaMa 4 Scout Instruct	Meta	yes	109B (17B activated)	327.680
Gemma 3 27B	Google	yes	27B	131.072
Mistral Small 3.2 Instruct	Mistral AI	yes	24B	128.000
GPT OSS 20B	OpenAI	yes	21B (3.6 activated)	131.072
Qwen 3 14.8B	Alibaba Cloud	yes	15B	40.960
Phi-4	Microsoft	yes	14B	16.384

Table 9: Relative number of invalid (unparseable syntax, e.g., more than one answer, or non-existent node type) and incorrect target type predictions after step 1 (direct instruction to return most likely target type) and step 2 (parsed from Cypher query) on PRIME for different open-weight LLMs. Most confused answers are "effect/phenotype" and "disease" as well as "pathway" and "biological process". MAG and AMAZON are omitted, because all answer candidates are of the same node type.

	step 1		step 2	
	invalid answers	wrong target type	invalid answers	wrong target type
GPT-5 mini	**0.0**	3.1	**0.0**	1.8
GPT OSS 120B	**0.0**	**1.8**	**0.0**	2.2
LLaMa 4 Scout Instruct	3.6	6.3	0.4	4.0
Gemma 3 27B	**0.0**	**1.8**	0.9	6.2
Mistral Small 3.2 Instruct	**0.0**	3.1	1.3	7.6
GPT OSS 20B	**0.0**	2.2	4.9	**1.3**
Qwen 3 14B	0.9	3.1	2.2	4.0
Phi-4	**0.0**	3.6	2.7	6.7

Table 10: Relative number of invalid and incorrect target type predictions after step 1 (direct instruction to return most likely target type) and step 2 (parsed from Cypher query) on PRIME for different "reasoning effort" values. Evaluated on GPT OSS 120B LLM.

	step 1		step 2	
	invalid answers	wrong target type	invalid answers	wrong target type
none	**0.0**	2.7	**0.0**	1.8
low	**0.0**	**1.8**	**0.0**	**0.9**
medium	**0.0**	**1.8**	**0.0**	2.2
high	**0.0**	2.2	4.9	1.3

Table 11: Performance of AF-Retriever with different LLMs.

	PRIME		MAG		AMAZON	
metrics (in percent)	hit@20	recall@20	hit@20	recall@20	hit@20	recall@20
GPT-5 mini	70.5	49.7	95.1	57.9	**77.3**	**32.6**
GPT OSS 120B	**71.9**	**52.2**	**95.9**	**58.4**	75.3	31.9
LLaMa 4 Scout Instruct	58.0	41.6	85.7	51.2	72.1	30.5
Gemma 3 27B	65.2	45.1	89.5	54.2	75.3	31.2
Mistral Small 3.2 Instruct	59.8	42.4	93.6	57.2	74.7	31.3
GPT OSS 20B	69.2	49.3	94.7	57.6	**77.3**	32.2
Qwen 3 14B	59.8	41.7	92.1	56.3	75.3	31.5
Phi-4	67.4	48.8	95.1	57.6	73.4	30.5

Table 12: Performance of AF-Retriever with different "reasoning effort" hyperparameters of the LLM (GPT OSS 120B).

	PRIME		MAG		AMAZON	
metrics (in percent)	hit@20	recall@20	hit@20	recall@20	hit@20	recall@20
none	68.3	49.3	**95.9**	58.3	76.0	32.0
low	68.3	49.2	95.1	57.8	76.6	32.0
medium	**71.9**	**52.2**	**95.9**	**58.4**	75.3	31.9
high	70.5	50.1	95.5	58.0	**77.3**	**32.6**

A.6.2 Strict vs. lenient parsing mode

Table 13: Effect of lenient parsing mode (no hard filter by node and edge types) vs strict parsing mode (ruling out all triplets with invalid node or edge types) and their hybrids. Concerns step 2.

	PRIME		MAG		AMAZON	
	hit@20	recall@20	hit@20	recall@20	hit@20	recall@20
NO filter by node and edge types	69.2	50.6	86.1	52.0	74.0	31.5
Filter triplets by relation types	70.5	51.7	**95.9**	**58.4**	**76.0**	**32.1**
Filter constants by node types	**71.9**	**52.2**	**95.9**	**58.4**	75.3	31.9
Filter by both labels	70.1	50.9	**95.9**	**58.4**	75.3	31.9

A.6.3 Constant candidate scope expansion

Table 14: This table summarizes how AF-Retriever performs under varying $l_{\max}$ values. If steps 1–3 yield fewer than one relevant triplet linked to the target entity, steps 4–6 are skipped, as triplet-based retrieval cannot be performed. The first column reports the proportion of QA pairs processed in steps 4 and 5. The next two columns present the average precision (fraction of true positives among all answer candidates) and average specificity (fraction of true positives among all ground-truth answers) for those QA pairs. Smaller $l_{\max}$ values typically result in fewer grounded answers. The following column indicates the number of QA pairs for which at least one answer candidate was retrieved. This number provides an upper bound for hit@m for step 6 ($\forall m$) shown in the next column. The final two columns display the overall performance of AF-Retriever for different $l_{\max}$ values, including supplemented answers from step 7. All metrics represent average relative scores, expressed as percentages. The three vertical sections refer to the PRIME, MAG, and AMAZON validation sets.

	steps 1-5			steps 1-6			steps 1-8	
l_{max}	processed	precision	specificity	>0 retrievals	hit@20	recall@20	hit@20	recall@20
0	0.0	-	-	-	-	-	46.4	33.0
1	87.9	**24.8**	48.6	55.4	**71.8**	**51.8**	69.2	49.4
2	87.9	23.0	54.1	64.3	67.4	50.7	71.0	51.2
3	87.9	21.6	56.1	67.4	68.2	50.6	71.0	51.2
10	87.9	18.4	59.0	73.7	67.3	49.5	71.0	51.6
33	87.9	12.9	61.4	79.0	64.4	46.9	71.4	51.8
100	87.9	7.0	**64.3**	82.6	64.9	47.4	**71.9**	**52.2**
333	87.9	6.5	**64.3**	**83.0**	64.5	47.2	**71.9**	**52.2**
0	0.0	-	-	-	-	-	72.6	44.9
1	85.0	**30.4**	92.9	83.5	93.7	53.5	95.9	58.2
2	85.0	**15.5**	94.2	83.5	95.5	53.9	95.5	58.0
3	85.0	11.4	89.9	82.4	92.0	55.3	94.9	**60.8**
10	85.0	9.0	96.0	84.2	96.0	55.1	**96.2**	58.6
33	85.0	8.8	96.4	84.6	96.4	55.2	95.9	58.4
100	85.0	8.8	**96.8**	**85.0**	**96.5**	**55.4**	95.9	58.4
333	85.0	8.8	**96.8**	**85.0**	**96.5**	**55.4**	95.9	58.4
0	0.0	-	-	-	-	-	74.7	30.5
1	64.9	**5.7**	35.3	42.9	**60.6**	**26.7**	**77.9**	**32.6**
2	64.9	4.1	36.6	47.4	53.4	24.9	76.0	32.1
3	64.9	4.9	39.0	52.6	55.6	24.3	77.3	32.3
10	64.9	3.8	42.0	55.2	55.3	23.6	77.3	32.3
33	64.9	3.9	43.5	60.4	50.5	22.5	75.3	31.9
100	64.9	3.8	**44.3**	**62.3**	50.0	22.1	75.3	31.9
333	64.9	3.8	**44.3**	**62.3**	50.0	22.1	75.3	31.9

A.6.4 Embeddings

Table 15: Differential analysis of adding or leaving out relational information in textual node descriptions before embedding each SKB node for VSS for step 4 (all symbol candidates), step 6 (graph-based retrieval), and step 7 (VSS-based retrieval). The relational information includes direct neighbors and 2-hop paths where the relation type to the embedded entity is 1-1 or n-1.

	relations included in							
	step 4	step 6	step 7	hit@1	hit@5	h@20	r@20	mrr
PRIME	yes	yes	yes	28.1	48.7	67.4	49.0	36.9
	yes	yes	no	26.3	46.0	67.0	48.8	35.5
	yes	no	yes	29.0	49.6	68.3	49.5	38.1
	yes	no	no	27.2	46.9	67.4	48.8	36.6
	no	yes	yes	37.9	55.4	71.0	51.7	44.9
	no	yes	no	37.1	53.6	70.1	51.3	44.1
	no	no	yes	**38.4**	**56.2**	**71.9**	**52.2**	**45.8**
	no	no	no	37.5	54.5	70.5	51.3	45.0
MAG	yes	yes	yes	61.3	86.5	95.5	58.2	72.4
	yes	yes	no	**62.0**	86.5	94.7	57.9	73.0
	yes	no	yes	60.9	86.5	94.7	57.8	72.5
	yes	no	no	61.7	86.5	94.0	57.6	73.1
	no	yes	yes	61.3	**87.2**	**96.6**	**58.7**	72.7
	no	yes	no	**62.0**	**87.2**	95.9	58.4	73.3
	no	no	yes	60.9	**87.2**	95.9	58.4	72.8
	no	no	no	61.7	**87.2**	95.1	58.1	**73.4**
AMAZON	yes	yes	yes	**24.7**	42.9	76.6	33.2	**33.6**
	yes	yes	no	24.0	41.6	**77.9**	**33.7**	33.4
	yes	no	yes	24.0	**43.5**	76.0	32.6	33.1
	yes	no	no	23.4	42.2	77.3	33.0	32.9
	no	yes	yes	23.4	42.2	76.0	32.6	32.7
	no	yes	no	22.7	40.9	77.3	33.0	32.5
	no	no	yes	24.0	42.9	75.3	31.9	32.8
	no	no	no	23.4	41.6	76.6	32.4	32.5

A.6.5 Omitting ambiguous node types

Table 16: Effect of concealing specific node types and incident edge types in the LLM prompt for receiving the Cypher query (step 2).

	concealed types	h@20	r@20
MAG	$\emptyset$	88.0	52.2
	"field of study"	**95.9**	**58.4**
AMAZON	$\emptyset$	75.3	31.9
	"category"	**77.9**	**33.4**

A.6.6 Weighting of both parallel retrieval strategies

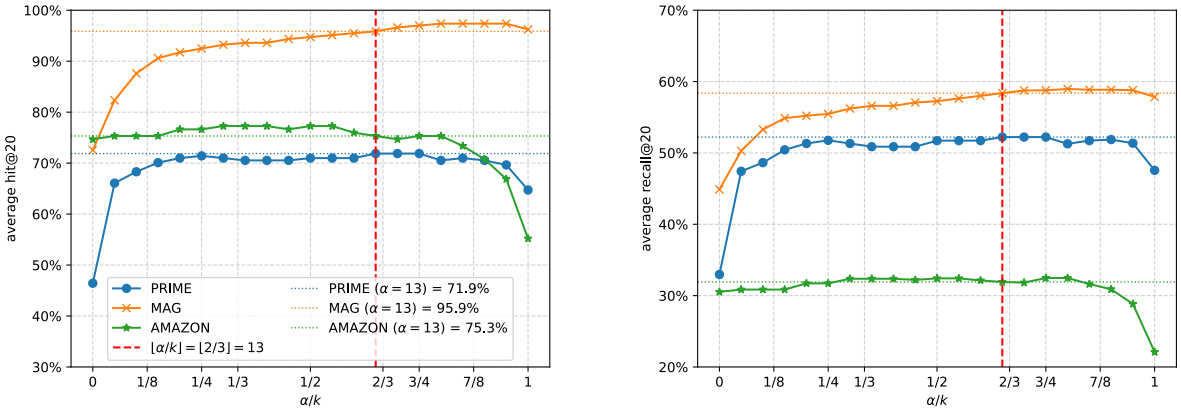

Figure 2: Hit@20 and recall@20 of AF-Retriever depending on α on 10% of the validation sets

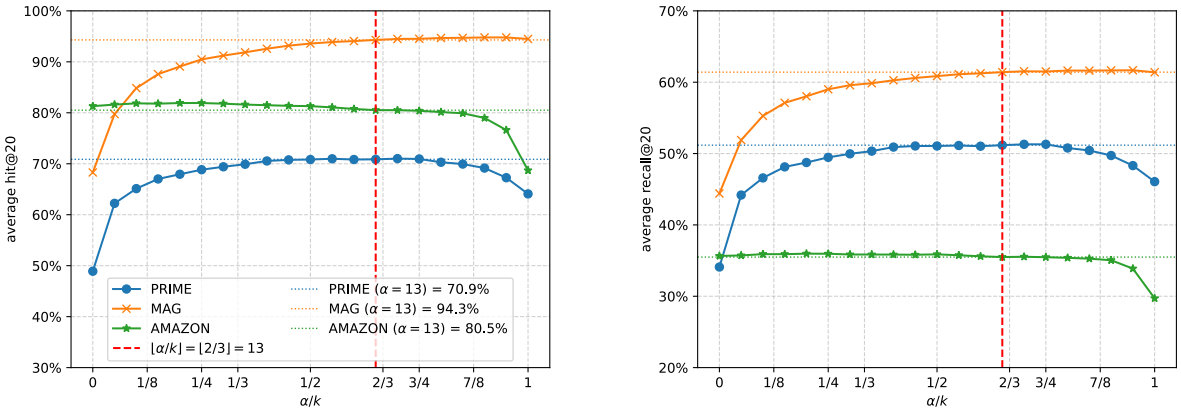

Figure 3: Hit@20 and recall@20 of AF-Retriever depending on α on the full test sets

A.6.7 Results reranker

Table 17: Occurrences of context window overflows, average number of prompts, and average sum of sent and received tokens for different rerankers. Incident (a) of a test QA pair is one if at least one prompt with node descriptions including complete 1-hop relations, and 2-hop relations where the relation type to the candidate is 1-1 or n-1, exceeds the reranker LLM's context window. Incident (b) of a test pair is one if at least one prompt with node descriptions including only those relations to retrieved non-target candidates also exceeds the limit. $k = 20$.

dataset	reranker	incident (a)	incident (b)	avg. number of prompts	avg. input tokens	avg. output tokens
PRIME	pointwise	0.0	0.0	k	52,244	5,741
	listwise	5.4	0.9	1	35,265	2,256
	pairwise	0.0	0.0	$3.0k$	276,357	37,122
MAG	pointwise	0.0	0.0	k	23,900	5,608
	listwise	1.1	0.0	1	16,563	2,005
	pairwise	0.0	0.0	$3.0k$	125,402	34,076
AMAZON	pointwise	0.0	0.0	k	25,211	4,752
	listwise	0.0	0.0	1	20,803	2,743
	pairwise	0.0	0.0	$3.1k$	137,379	2,4856

Table 18: Performance of AF-Retriever with different reranking strategies.

reranker	LLM	PRIME			MAG			AMAZON		
		hit@1	hit@5	mrr	hit@1	hit@5	mrr	hit@1	hit@5	mrr
pointwise	GPT OSS 20b	35.3	64.7	47.7	59.0	85.0	70.5	47.4	68.2	55.6
	GPT OSS 120b	41.1	64.3	51.7	55.3	81.2	67.3	48.1	68.2	56.9
listwise	GPT OSS 20b	46.0	**65.6**	53.7	75.2	92.1	82.2	50.6	63.0	55.9
	GPT OSS 120b	47.8	62.5	54.4	**83.1**	**93.6**	**87.5**	47.4	64.9	55.5
pairwise	GPT OSS 20b	47.8	**65.6**	55.4	73.3	89.8	80.3	55.2	70.8	61.6
	GPT OSS 120b	**49.1**	64.7	**55.7**	71.8	89.5	79.6	**60.4**	**72.1**	**65.2**

A.6.8 Runtimes

Table 19: Average, median, and maximum runtimes of AF-Retriever's pipeline steps on the PRIME validation set, measured on an EPYC 9454 Genoa (48 Core) CPU and 2 A100 80 GB GPUs for the LLM GPT OSS 120b via the vLLM inference engine. In the main configuration, we set $\alpha = 2/3, k = 20, l_{max} = 3$, and used the listwise reranker with low reasoning effort. Rows below the sum line report similar performant but slower variants: $l_{max} = 100$ in step 5, and other rereanker variants with good reranking performance in step 8. Embeddings of all SKB nodes were precomputed.

step	median	average	maximum
1	0.31	0.37	5.02
2	2.23	2.41	6.28
3	0.00	0.00	0.00
4	0.21	0.14	1.58
5	0.03	0.30	13.39
6	0.03	0.07	0.69
7	0.04	0.06	0.61
8	7.94	19.27	134.08
sum	10,79	22.62	161.65
5 ($l_{max} = 100$)	0.12	0.78	44.57
8 (pairwise reranker)	85.98	93.23	751.07
8 (listwise reranker, medium reasoning effort)	17.31	28.19	173.18
8 (pairwise reranker, medium reasoning effort)	244.98	267.69	1199.54

As expected, the regular expression-based step (step 3) is computationally negligible. Triplet grounding (step 5) is typically fast, but can take several seconds in complex cases, reinforced with large l_{max}. The embedding-based steps complete in less than 5ms on average, having node embeddings preprocessed. The embedding-based step 4 has outliers up to 1.58s due to latency of the OpenAI's api server during our experiments. Overall runtime is dominated by LLM latency: with the listwise reranker, steps 1, 2, and 8 each require one LLM call, with runtimes ranging from a maximum of 5.02 s for the short prompt in step 1 to more than two minutes for the long, top-k prompt in step 8 with exceptional long reasoning time. The pairwise reranker is, on average, without parallelization, 4.6x slower than the listwise version (9.3x times with increased reasoning effort).