Agent Retrieval on Textual and Relational Knowledge Bases

Stanford CS224N {Custom} Project

Shiyu Zhao
Department of Computer Science
Stanford University
shiyuz@stanford.edu

Abstract

Semi-structured knowledge bases with textual and relational information are pervasive in real-world scenarios, which poses complex information retrieval challenges. The task of semi-structured retrieval is to retrieve node entities from the knowledge bases given natural-sounding queries, which involve complex textual properties and diverse relational information between node entities. However, previous works have primarily studied textual and relational retrieval as separate topics. Moreover, the existing techniques using large language models (LLMs) lack the capability to flexibly handle the textual and relational aspects, which can lead to a large discrepancy in retrieval accuracy. To this end, we devise a novel approach called Agent Retriever, which generates Python programs with access to knowlege base APIs and is designated to flexibly interact and reason over the semi-structured knowlege bases. Notably, Agent Retriever iteratively conducts contrastive reasoning to improve the program in the generation process. To conduct a systematic evaluation, we construct a large-scale benchmark with three retrieval datasets that simulate real-world semi-structured retrieval scenarios. Specifically, through extensive experiments, we show that Agent Retriever achieves excellent performance on all three benchmark datasets, outperforming the baselines with an average of over 5% on the retrieval metrics.

1 Key Information to include

- External collaborator and mentor: Advised by Prof. Jure Leskovec and Shirley Wu
- Sharing project: A continuation of previous independent research project.

2 Introduction

Natural-language queries are the primary forms in human society for acquiring information, involving diverse knowledge in the real world Hirschman and Gaizauskas (2001); Kaufmann and Bernstein (2010); Jamil (2017). For example, users on e-commerce web search can express complex information needs by combining free-form elements or constraints, such as "*Can you help me find a push-along tricycle from Radio Flyer that's both fun and safe for my kid?*". Here, the underlying knowledge can be represented in semi-structured formats Oguz et al. (2022); Wang et al. (2020); Ryu et al. (2014), referred to as semi-structured knowledge bases (SKBs), which integrate unstructured data, such as textual descriptions and natural language expressions (*e.g.,* descriptions about a tricycle), with structured data, like entity interactions on knowledge database (*e.g.,* a tricycle with "belongs to" relation with brand Radio Flyer). This allows semi-structured knowledge bases to represent comprehensive knowledge in specific applications, making them indispensable in domains such as e-commerce He and McAuley (2016), social media Mansmann et al. (2014), and biomedicine Chandak et al. (2023); Hu et al. (2020). In this work, we focus on semi-structured knowledge bases (SKBs) with texts and relation information, which are two prevalent modalities on the existing knowledge bases. In this context, information retrieval serves as a predominant step by identifying relevant data

	Example query	Title of ground truth node(s)
STAR-AMAZON	*Looking for durable Dart World brand dart flights that resist easy tearing. Any recommendations?*	<Amazon Standard Flights> <Dart World Broken Glass Flight> (12 more)
	What are recommended scuba diving weights for experienced divers that would fit well with my Gorilla PRO XL waterproof bag?	<Sea Pearls Vinyl Coated Lace Thru Weight>
STAR-MAG	*Search publications by Hao-Sheng Zeng on non-Markovian dynamics.*	<Distribution of non-Markovian intervals...> <Comparison between non-Markovian...>
	What are some nanofluid heat transfer research papers published by scholars from Philadelphia University?	<A Numerical Study on Convection Around A Suqare Cylinder using AL2O3-H2O Nanofluid>
STAR-PRIME	*Could you provide a list of investigational drugs that interact with genes or proteins active in the epididymal region?*	<(S)-3-phenyllactic Acid>, <Anisomycin>, <Puromycin>
	Search for diseases without known treatments and induce pruritus in pregnant women, potentially associated with Autoimmune.	<Intrahepatic Cholestasis>
	Please find pathways involving the POLR3D gene within nucleoplasm.	<RNA Polymerase III Chain Elongation>
	Which gene or protein associated with lichen amyloidosis can bind interleukin-31 to activate the PI3K/AKT and MAPK pathways?	<OSMR>, <IL31RA>

Table 1: Example queries from STAR, which involve semi-structured information (the relational and textual aspects are highlighted).

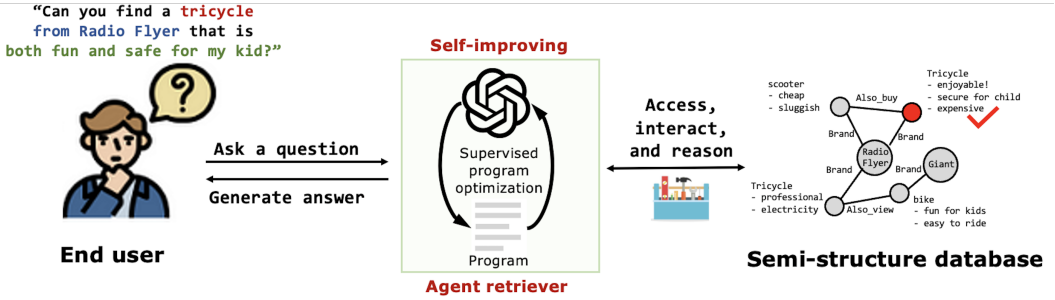


Figure 1: Agent Retriever overview: For an input query, the LLM agent generates a Python program using a given API to access, interact with, and reason over a semi-structured database. With the ground truth as feedback, the agent iteratively improves itself.

from the semi-structured knowledge bases, such as specific tricycles that match the request, which are then processed to generate accurate answers to user queries Singhal (2001). The retrieval accuracy on semi-structured knowledge bases is crucial for enhancing user experience, supporting informed decision-making, and preventing hallucination.

Limitations of Existing Works. However, previous works and benchmarks have primarily focused on either purely textual queries on unstructured knowledge Izacard and Grave (2021); Guu et al. (2020); Lee et al. (2019) or purely relational queries on structured knowledge Berant et al. (2013); Yih et al. (2015); Yang et al. (2018b), which are inadequate to study the complexities of retrieval tasks involving semi-structured knowledge bases. Recently, large language models (LLMs) have demonstrated significant potential in information retrieval tasks Guu et al. (2020); Lewis et al. (2021); Shi et al. (2023). Nevertheless, it remains an open question how to effectively apply LLMs to the specific challenge of retrieval from semi-structured knowledge bases (SKBs). It is also important to note that existing works focus mainly on general knowledge, e.g., from Wikipedia. In fact, knowledge may also come from private sources, necessitating information retrieval systems adaptable to private semi-structured knowledge bases. Therefore, there is a gap in our understanding of how to apply LLM-based retrieval systems to handle the complex interplay between textual and relational requirements in queries that optionally involve private knowledge.

Our contribution: . To address this gap, our goal is to develop a benchmark for LLM retrieval on SKBs and propose a novel LLM-based retriever called Agent Retriever. As illustrated in table1, we present a large-scale Semi-structure retrieval benchmark on Textual and Relational knowledge bases (STAR). We propose a novel method Agent Retriever, as shown in figure1, that can effectively access, interact, and reason over SKBs with constant self-improving. With STAR, we delve deeper into retrieval tasks on SKBs, evaluate the capability of current retrieval systems, and demonstrates the superiority of Agent Retriever over three datasets. We point out the key features of STAR as follows: 1) **Natural-sounding queries on semi-structured knowledge:** The queries in our benchmark are crafted to incorporate rich relational information and complex textual properties. Additionally, these queries closely mirror the types of questions users would naturally ask in real-life scenarios.

2) **Context-specific reasoning in diverse domains:** Covering areas on product recommendation, academic paper search, and precision medicine inquiry, the queries entail reasoning capabilities including the ability to infer customer interests, understand specialized field descriptions, and deduce relationships involving multiple subjects mentioned within the query.

The main contributions of our method Agent Retrieverare as follows: 1) **Agent with tool kits for flexible semi-structure retrieval:** Equipped with APIs and interfaces to different databases, Agent Retriever largely improves LLM's flexibility and accuracy on retrieval from data sources of different structures. 2) **Supervised agent optimization:** With groundtruth distribution as supervision signal, Agent Retriever is able to dynamically adjust and improve the retrieval programs iteratively.

We perform extensive experiments on each benchmark dataset for our method Agent Retriever and the prevailing retrieval systems. We highlight Agent Retriever's capability to handle textual and relational requirements on large-scale SKBs that involve million-scale entities or relations. **We constructed the benchmark and build Agent Retriever all from scratch.**

3 Related Work

Structured, Unstructured, and Semi-structured QA Datasets. Structured QA datasets utilize explicit relational structures in knowledge bases or tabular formats. WebQuestions Bordes et al. (2014) and SimpleQuestions Bordes et al. (2015) challenge models to interpret and extract data from knowledge graphs, focusing on entity and relation-specific queries. In the tabular domain, WikiSQL Zhong et al. (2017) and Spider Yu et al. (2018) test the conversion of natural language inquiries into SQL queries. These datasets are instrumental in advancing structured data retrieval but may not fully encapsulate the multifaceted nature of queries that blend structured and unstructured data. Unstructured QA datasets focus on answer retrieval from textual sources. SQuAD Rajpurkar et al. (2016) exemplifies single-document retrieval, emphasizing contextual comprehension. Multi-document retrieval datasets like HotpotQA Yang et al. (2018a) and TriviaQA Joshi et al. (2017) introduce complexity by requiring reasoning across documents or linking trivia questions to evidence documents. These datasets test retrieval and reasoning skills in text-rich environments but often do not address the intricacies of relational reasoning across diverse data types. Semi-structured QA datasets, like WikiTableQuestions Pasupat and Liang (2015) and TabFact Chen et al. (2020a), integrate both tabular and textual information, presenting a more nuanced data retrieval challenge. HybridQA Chen et al. (2020b) and TabMCQ Jauhar et al. (2016) extend this integration, requiring models to perform validation and answer multifaceted questions using combined data types. However, the focus often remains skewed toward tabular understanding, signaling a need for more balanced datasets that offer deep relational and textual data integration for advancing semi-structured data retrieval capabilities.

LLMs for Information Retrieval: The integration of LLMs in information retrieval has been a subject of extensive research, leading to the development of several innovative models. Retrieval-Augmented Generation (RAG) employs a retrieve-and-generate approach, using a non-parametric memory to augment the generation process of a Transformer-based language model Lewis et al. (2021). REALM (Retrieval-Augmented Language Model) introduces a retriever model that is jointly learned with the language model, enabling it to dynamically retrieve relevant documents Guu et al. (2020). REPLUG freezes LLM and trains the retriever with the signals from LLM Shi et al. (2023). G-Retriever introduces a retrieval mechanism within the LLM framework to enhance its ability to source and integrate external information, facilitating a more informed and contextually relevant generation process He et al. (2024). For other retrieval methods, QA-GNN combines the retrieval capabilities of a language model with graph neural networks to represent relational data in question-answering tasks Yasunaga et al. (2021). Dense Passage Retrieval (DSP) focuses on improving the retrieval phase by using dense vector representations of text Khattab et al. (2023). While these models represent significant advancements in the field of information retrieval using LLMs, these models cannot simultaneously handle textual and relational information, struggling with complex information in that requires further reasoning. Furthermore, most of the previous methods involve extensive training, which are non-scalable to large-scale knowledge bases and not easy to be adapted to private knowledge bases. These limitations signal the need for a retrievers with expressive and flexible retrieval capabilities over data of mixed structures.

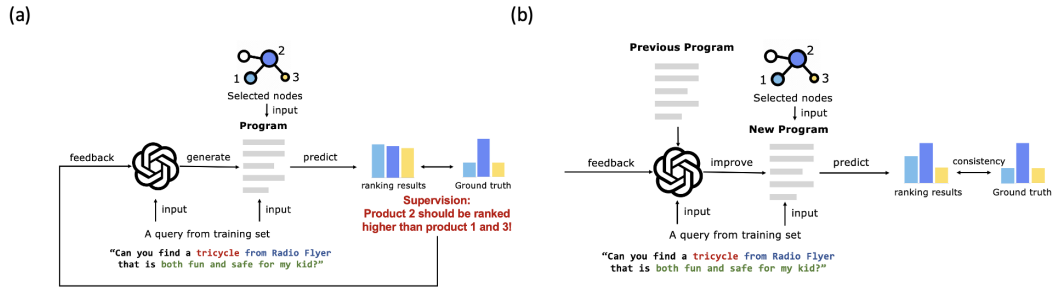


Figure 2: Iterative self-improvement in Agent Retriever workflow: During the training phase, upon receiving a query, the model is tasked with generating and executing a program to retrieve from SKBs. The predicted ranking results, the ground truth results, and previous programs will be fed back to the agent. This feedback is intricately analyzed and utilized by the agent to refine and optimize the program, aiming for enhanced performance in successive iterations.

4 Approach

4.1 Problem definition

This work addresses semi-structured question answering retrieval, integrating relational and textual data extraction from databases $\mathcal{D} = \{R, T\}$, with R and T representing relational and textual data, respectively. The objective is to construct a model f that, given a query $Q = \{Q_r, Q_t\}$, maps it to an answer A. The answer A are nodes in $\mathcal{D}$ satisfying both relational Q_r and textual Q_t query parts. The task is defined as finding $f : \{Q_r, Q_t\} \times \mathcal{D} \rightarrow A$, effectively merging R and T to generate A. The input of the model is semi-structure knowledge base and a query, and the output should be a set of predicted answers.

4.2 Methodology

Our methodological framework features an Agent Retriever that interfaces with semi-structured knowledge bases through a bespoke suite of APIs. These APIs empower the Agent Retriever to construct Python programs that are composed of custom-defined functions, API calls, and logical code structures. As shown in fig 3, upon receiving a natural language query during training, the model is prompted to generate a Python program tailored to the query's requirements. This program is then executed to query the semi-structured knowledge base, facilitating the retrieval and synthesis of relevant information. The retrieval outcomes are subsequently juxtaposed with ground truth data, and the program is iteratively refined through a feedback loop. This iterative process continues until a pre-defined convergence criterion is met or the predictions align with the ground truth. For testing, several developed Python programs are preserved in their final state. A program is selected based on its applicability to a given query, and its execution yields the retrieval results. **We have developed and implemented all methodologies from scratch.**

API tool kits for flexible semi-structure retrieval. The Agent Retriever framework encompasses a comprehensive set of APIs, each group specifically designed to maximize the retrieval capabilities within a certain domain. The knowledge base APIs allow the agent to pinpoint critical details in hard requirements by querying node attributes, such as product names and brands, and by understanding the relationships between different nodes—for example, products that customers also viewed. This set of APIs is essential for reasoning over relational data within the knowledge base. On the other hand, the embedding system APIs are crucial for processing textual content, providing tools to embed textual descriptions and compute similarities or exact match scores between strings, thereby evaluating textual relevance to user queries. The LLM APIs work in conjunction with these by leveraging large language models to perform tasks such as summarizing texts, verifying requirements within a query context, computing relevance scores, and extracting specific information from blocks of text. This division into distinct API groups grants Agent Retriever a flexible and dynamic approach to handling the intricate interplay of textual and relational data, enabling it to adjust to varied data structures and query intricacies with improved accuracy and adaptability. Each API group contributes its unique strengths to the retrieval process, ensuring that Agent Retriever is not just a static retrieval

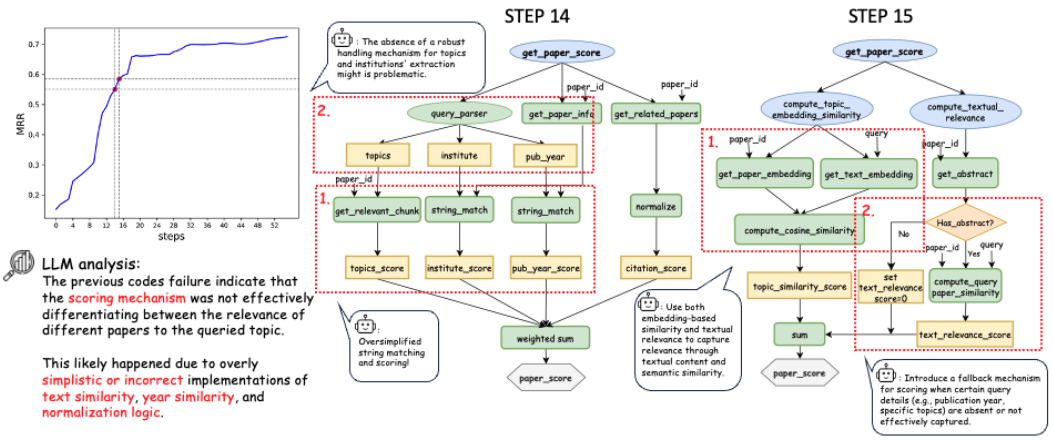


Figure 3: Illustration of supervised agent optimization on STAR-MAG: Curve on the left up shows how the retrieval MRR improves during optimization. Specifically, at step 14, after the agent got feedback, it analyzed and drew two conclusions: 1. string matching score function is oversimplified 2. textual extraction and similarity calculation is not robust. Both problems are addressed and improved in the program at step 15.

tool, but a sophisticated agent capable of intelligent and flexible interaction with semi-structured knowledge. The whole list of APIs provided can be referred at fig 9 at Appendix.

Supervised agent optimization The process initiates with the agent classifying a given query into one of k predefined categories, each corresponding to a unique information retrieval challenge within the knowledge base. For each classification, the agent constructs a Python script utilizing a designated API toolkit. This script is tasked with executing the retrieval function, culminating in a predicted ranking of node entities relevant to the query. These preliminary predictions are subsequently calibrated against a ground truth dataset, generating a differential signal that serves as a feedback loop for programmatic refinement. Iteratively, the agent assimilates insights from this feedback, alongside historical script performances, to iteratively evolve its code generation strategy. This results in the agent enhancing its accuracy in terms of correspondence with ground truth rankings, as demonstrated by the empirical improvements in MRR across sequential optimization steps depicted in fig 3. As shown in the figure, during the transition from Step 14 to Step 15, the agent harnesses feedback to ascertain two crucial programmatic deficiencies: an oversimplified string matching score function and a lack of robustness in textual extraction and similarity computation. The agent then implements enhancements to address these shortcomings, optimizing the underlying algorithmic components for improved retrieval efficacy.

Upon completion of the training phase, wherein the agent iterates through this optimization cycle, the most performative script for each query category is identified based on its validation set metrics. In the testing phase, the agent leverages this optimized script, now frozen to prevent further modification, to execute retrieval tasks. The chosen script for each category is thus adept at handling the corresponding semi-structured queries, showcasing the adaptability and precision of the retrieval system trained through this rigorous supervised optimization methodology.

5 Experiments

5.1 Data

We develop three large-scale retrieval datasets on three semi-structure knowledge bases (SKB) for our task. We will briefly introduce the SKBs with both relational and textual information; the retrieval datasets with natural-sounding queries fusing both textual and relational requirements; and highlight our novel dataset construction pipeline, which simulates user queries involving textual and relational knowledge and automatically generates the ground truth answers in the retrieval datasets.

Semi-structured Knowledge Base. We construct three SKBs from the relational structure between entities and the textual information associated with a subset of the entities: Amazon SKB, MAG SKB, and Prime SKB. A detailed demonstration is at Appendix fig 6. We include Table 3 for the

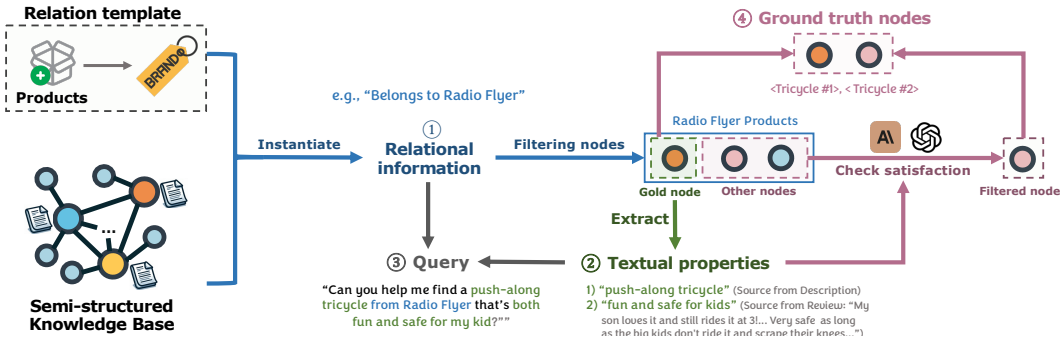


Figure 4: The process of constructing semi-structured retrieval datasets involves four main steps: 1) Sample Relational Requirement: Based on relational templates, sample a relational requirement on a SKB. 2) Extract Textual Properties: From a node that meets the relational requirement, extract relevant textual properties. 3) Combine Information: Merge the relational information and textual properties to form a natural-sounding query. 4) Construct Ground Truth Nodes: Check if nodes satisfy the textual properties using multiple language models to establish ground truth nodes.

comprehensive public sources we used to construct the SKBs, the statistics of the relational structure in Table 4 and detailed introduction of each SKB in section A.2.3 at the appendix.

Retrieval Dataset on Semi-structured Knowledge Bases. Utilizing our knowledge bases, we develop three novel retrieval datasets: STARK-AMAZON, STARK-MAG, and STARK-PRIME. Specifically, the task is to retrieve node entities over the SKBs given any input queries. To highlight, the queries feature the interplay and fusion between relational and textual knowledge. Moreover, to elevate their applicability in practical scenarios, these queries are designed to mimic real-life query patterns in terms of the natural-sounding property and flexible formats. Each query may have single answer or multiple answers, formulated as a subset of entities from the knowledge base. In Table 1, we demonstrate example queries from these benchmarks. We detail our retrieval benchmarks' scale in Table 5 and introduce the specifics of each retrieval dataset at A.3.1 in appendix. Besides, we also conduct extensive analysis of dataset on the query and answer length, query diversity, and ratio of relational *vs.* textual information. The analysis is included at A.3.2.

5.2 Dataset Construction

In this section, we outline an efficient, automated pipeline for constructing retrieval datasets, integrating relational and textual information to generate accurate ground truth answers. Our methodology, illustrated in Figure 4, comprises four distinct steps:

- **Sampling Relational Requirements:** We initiate the process by selecting a relation template (e.g., "(a product) belongs to <brand>") and instantiate it to derive specific relational requirements, which in turn yield candidate entities. This step is vital for diversity and is detailed with 28 different templates for the Prime SKB in Appendix A.4, addressing various analytical and practical needs.
- **Extracting Textual Properties:** Next, we choose one candidate entity as the gold answer and extract relevant textual information tailored to different user roles, such as customers or researchers. We leverage GPT-3.5-turbo-16k and Claude-v2 for extracting these properties from the STARK-AMAZON and other datasets, respectively.
- **Combining Textual and Relational Information:** We then synthesize this information into queries using two LLMs in a bid to eliminate LLM biases and enhance query diversity. Specifically, the first-stage integration is guided by various criteria, such as ensuring a natural-sounding query, and the second stage instructs the LLM to enrich the context and rephrase the language, thereby posing a more demanding reasoning challenge in comprehending the requirements of the query.
- **Filtering Additional Answers:** The final step involves verifying additional candidate entities against the textual requirements using three Claude models. Only candidates that pass the verification across all models are included in the final ground truth answer set. During this process, we also evaluate the accuracy of the gold nodes that pass the verification criteria. The results indicate accuracy rates of 86.6%, 98.9%, and 92.3% for STARK-AMAZON, STARK-MAG, and STARK-PRIME, respectively, demonstrating the effectiveness of our filtering approach in maintaining high-quality ground truth answers.

Our pipeline's effectiveness is evidenced by its adaptability to various SKBs and the high quality of the ground truth answers. The complete prompts used in each step are documented in Appendix A.6. Dataset segmentation into training, validation, and testing subsets is systematically detailed, with specific strategies employed for each dataset as described in Table 5.

5.3 Evaluation metrics and baselines

To evaluate the performance of our retrieval models, we employ a comprehensive set of metrics alongside baseline methods for a robust comparative analysis.

Hit@k. We use Hit@k for $k = 1$ and $k = 5$ to assess whether the top-k results contain the correct answer, evaluating both the model's immediate accuracy and its relevance in the first five results.

Recall@k. With Recall@k for $k = 20$, we measure the proportion of relevant answers in the top-k results, offering a view of the model's comprehensive retrieval performance.

Mean Reciprocal Rank (MRR). The MRR metric calculates the average inverse rank of the first relevant answer, gauging the model's ability to prioritize the foremost correct response.

To benchmark our models, we contrast them with several baselines: Vector Similarity Search (VSS) utilizes `text-embedding-ada-002` to measure cosine similarity between queries and entities. Multi-Vector Similarity Search (Multi-VSS) expands on this by using multiple vectors for entity representation. The Dense Retriever model refines the search using dual encoders and contrastive learning with hard negatives. Retrieval-augmented generation (RAG-FT) combines pretrained embeddings with fine-tuned generation capabilities. QAGNN improves upon this by integrating entity graphs for enriched semantic-relational retrieval. Enhancing VSS, the VSS + LLM Reranker employs models like GPT-4-turbo and Claude-v2 for sophisticated reranking.

5.4 Experimental details

For dataset construction, we use GPT-3.5-turbo-16k and Claude-v2 for extracting textual properties, Claude-v2 and GPT-4-Turbo for combining textual and relational information, and Claude-v2 for filtering additional answers. Within Agent Retriever , Claude serves as the code generator. We set $k = 12$ for group categorization, with a batch size of 25 queries per iteration and a total of 100 self-improvement iterations. These iterations are distributed across 8 machines with 128 CPU cores and 8 A100 GPUs, allowing dataset training to complete within 1-2 days.

5.5 Results

	STaR-Amazon				STaR-MAG				STaR-Prime			
	Hit@1	**Hit@5**	**Recall@20**	**MRR**	**Hit@1**	**Hit@5**	**Recall@20**	**MRR**	**Hit@1**	**Hit@5**	**Recall@20**	**MRR**
VSS	0.3916	0.6273	0.5329	0.5035	0.2908	0.4961	0.4836	0.3862	0.1263	0.3149	0.3600	0.2141
Multi-VSS	0.4007	0.6498	0.5512	0.5155	0.2592	0.5043	**0.5080**	0.3694	0.1510	0.3356	0.3805	0.2349
Dense Retriever	0.1529	0.4793	0.4449	0.3020	0.1051	0.3523	0.4211	0.2134	0.0446	0.2185	0.3013	0.1238
RAG-FT	0.2026	0.3037	0.3969	0.2340	0.0361	0.0482	0.0627	0.0442	0.0370	0.0867	0.1359	0.0618
QAGNN	0.2656	0.5001	0.5205	0.3775	0.1288	0.3901	0.4697	0.2912	0.0885	0.2135	0.2963	0.1473
VSS+Claude Reranker	0.4266	0.6746	0.5376	0.5329	0.3202	0.5334	0.4834	0.4129	0.1611	0.3582	0.3598	0.2466
VSS+GPT4 Reranker	0.4479	0.7117	0.5535	0.5569	0.4090	0.5818	0.4860	0.4900	0.1828	0.3728	0.3405	0.2655
Agent Retriever	**0.5014**	**0.7203**	**0.5812**	**0.5861**	**0.4608**	**0.5932**	0.4970	**0.5201**	**0.1941**	**0.3975**	**0.4155**	**0.2876**
Improvement	11.9%	1.2%	5.0%	5.3%	12.7%	2.1%	-2.2%	6.1%	6.2%	6.6%	9.2%	8.3%

Table 2: Main experimental results. Last row shows the relative improvements over the best metric value among the baselines.

The experimental results depicted in Table 2 reveal that the Agent Retriever outperforms all other methods on the STAR datasets. It achieves the highest scores across most metrics, including Hit@1, Hit@5, Recall@20, and MRR, demonstrating a substantial performance leap from existing models. For instance, in the STaR-Amazonset, Agent Retriever's Hit@1 improvement is nearly 12% over VSS+GPT4 Reranker. Overall, Agent Retrieverdemonstrates more than 5% increase over previous SOTA model on most metrics over the datasets. In contrast to models that rely on LLM fine-tuning, such as RAG, the Agent Retriever's superior performance illustrates the advantage of supervised agent optimization. While models on retrieval single structure data like Dense Retriever (for text) and QAGNN (for relational) show lower Hit@1 and Recall@20 scores, indicating a struggle to adapt to data of another structure, the Agent Retriever benefits from using flexible interfaces and APIs to different data sources, enabling it to retrieve accurately from semi-structured knowledge bases. The results also demonstrate the hardness of our dataset and call for future study in this

area. There remains room for optimization, as indicated by the absolute percentages. Hit@1 on the STAR-PRIME subset is only around 19.41%, suggesting that the correct answer is hard to be ranked first. Additionally, the Recall@20 metric not surpassing 60% suggests that relevant answers might be missing from the top rankings, emphasizing the need for further refinement in the comprehensiveness.

6 Analysis

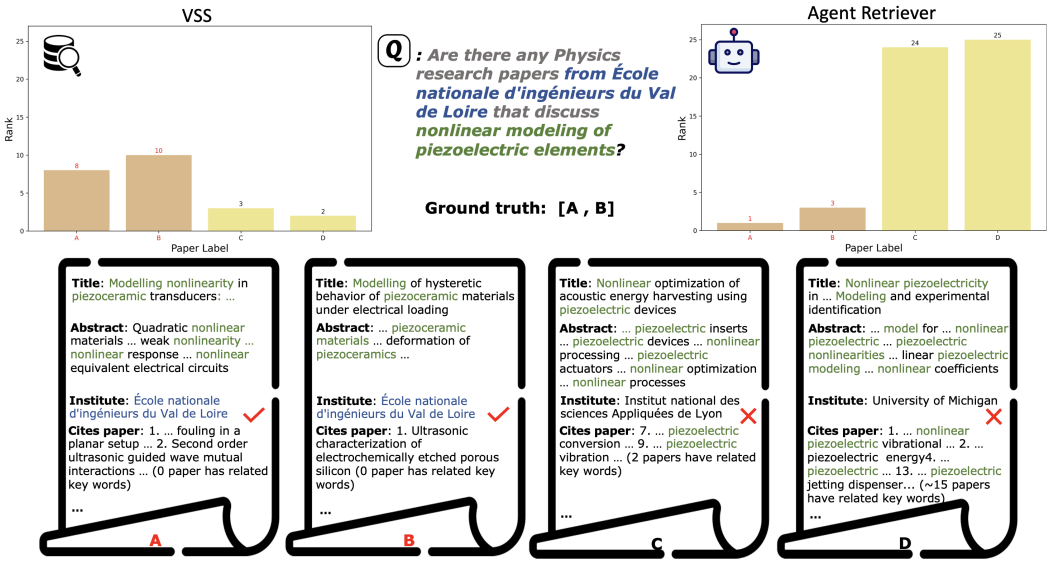


Figure 5: Advantage of using tool kits: VSS mistakenly ranks non-ground truth papers C and D higher due to the repeated key words in the relational information "cites paper." In Agent Retriever, in addition to embedding APIs, it also uses get_related_nodes (see Fig 9 for more APIs) to check the key institute information, correctly prioritizing the ground truth papers A and B.

We mainly analyze the method part in this section, and leave extensive analysis on our self-constructed dataset at Appendix A.3.2. As shown in fig 5, we provide a case study comparing VSS and Agent Retriever to demonstrate the flexibility, the reasoning ability, and the thoughtfulness of our method by using tool kits for retrieval. Specifically, we examine a user query that requests papers from an institution on a particular topic. In this scenario, VSS fails to adequately address the relational requirement due to directly embedding the entire documents without detailed analysis. As a result, papers containing frequently repeated keywords, such as “nonlinear modeling” and “piezoelectric elements”, or citing many relevant papers, are mistakenly assigned high scores. However, the result improves significantly when Agent Retriever is equipped with API tool kits to reason the relationship between the query and directly retrieve from knowledge base by get_related_nodes, making the scores more accurately reflect the papers’ relevance to the query. As methods for single-structure data, such as VSS, may capture some aspects of the query, they often fail to capture comprehensive information from data of different structures. By equipping LLMs with APIs and optimizing LLMs through supervised agent optimization, the model is able to reason and flexibily retrieve accurately from semi-structure knowledge bases.

7 Conclusion

In summary, Agent Retriever represents a pioneering approach in semi-structured knowledge base retrieval, deftly integrating textual and relational data through a novel use of Python-program generation and API interaction. Our contributions include a large-scale, purpose-built benchmark that illustrates Agent Retriever ’s superior performance, reflecting an significant improvement of on retrieval metrics compared to existing baselines. The method’s iterative refinement via supervised optimization exemplifies the potential for LLMs to evolve beyond traditional limitations, paving the way for future advancements in domain-specific retrieval systems. This work sets a new standard for the field and underscores the untapped potential of LLMs in complex information retrieval scenarios.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar. Association for Computational Linguistics.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075.

Payal Chandak, Kexin Huang, and Marinka Zitnik. 2023. Building a knowledge graph to enable precision medicine. *Scientific Data*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020a. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020b. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *ICML*. PMLR.

Ruining He and Julian J. McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. ACM.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering.

Lynette Hirschman and Robert Gaizauskas. 2001. Natural language question answering: the view from here. *natural language engineering*.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2021. Open graph benchmark: Datasets for machine learning on graphs.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *EACL*.

Hasan M. Jamil. 2017. Knowledge rich natural language queries over structured biological databases. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*.

Sujay Kumar Jauhar, Peter D. Turney, and Eduard H. Hovy. 2016. Tabmcq: A dataset of general knowledge tables and multiple-choice questions. *CoRR*, abs/1602.03960.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*.

Esther Kaufmann and Abraham Bernstein. 2010. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *Journal of Web Semantics*.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2023. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *ACL*.

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

Svetlana Mansmann, Nafees Ur Rehman, Andreas Weiler, and Marc H. Scholl. 2014. Discovering OLAP dimensions in semi-structured data. *Inf. Syst.*

Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*. ACM.

Barlas Oguz, Xilun Chen, Vladimir Karpukhin, Stan Peshterliev, Dmytro Okhonko, Michael Sejr Schlichtkrull, Sonal Gupta, Yashar Mehdad, and Scott Yih. 2022. Unik-qa: Unified representations of structured and unstructured knowledge for open-domain question answering. In *ACL Findings*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas. Association for Computational Linguistics.

Pum-Mo Ryu, Myung-Gil Jang, and Hyunki Kim. 2014. Open domain question answering using wikipedia-based knowledge model. *Inf. Process. Manag.*

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2023. REPLUG: retrieval-augmented black-box language models. 2301.12652.

Amit Singhal. 2001. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*

Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Paul Hsu, and Kuansan Wang. 2015. An overview of microsoft academic service (MAS) and applications. In *WWW*.

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quant. Sci. Stud.*

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018a. HotpotQA: A dataset for diverse, explainable multi-hop question answering. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018b. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *EMNLP*.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. Qa-gnn: Reasoning with language models and knowledge graphs for question answering.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. Brussels, Belgium. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Appendix

A.1 Demonstration of Semi-structured Knowledge Bases

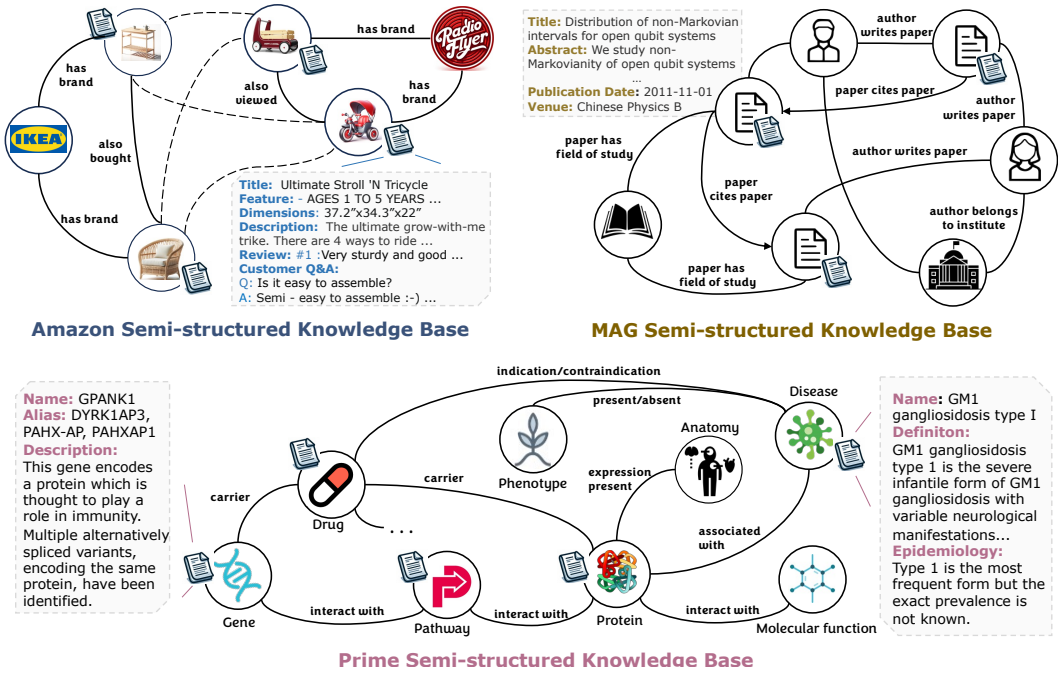


Figure 6: Demonstration of Semi-structured Knowledge Bases, where each knowledge base combines both textual and relational information in a complex way, making the retrieval tasks challenging.

A.2 Benchmark details

A.2.1 Benchmark source

	relational structure	textual information
STARK-AMAZON	Amazon Product Reviews	Amazon Product Reviews Amazon Question and Answer Data
STARK-MAG	ogbn-mag	ogbn-papers100M, Microsoft Academic Graph
STARK-PRIME	PrimeKG	`disease`: Orphanet; `drug`: DrugBank; `pathway`: Reactome; `gene`: Ensembl, NCBI Entrez, Uniprot, UCSC, CPDB

Table 3: Sources of relational structure and textual information of the benchmarks

A.2.2 Statistics of semi-structured knowledge bases

	#entity types	#relation types	avg. degree	#entities	#relations	#tokens
STARK-AMAZON	2	3	3.0	1,032,407	3,886,603	592,067,882
STARK-MAG	4	4	10.6	1,872,968	19,919,698	212,602,571
STARK-PRIME	10	18	62.6	129,375	8,100,498	31,844,769

Table 4: Data statistics of our constructed semi-structured knowledge bases

A.2.3 Details of semi-structure knowledge bases

Amazon Semi-structured Knowledge Base. This knowledge base is derived from the Sports and Outdoors category from Amazon Product Reviews He and McAuley (2016) and Amazon Question

and Answer Data McAuley et al. (2015). The knowledge base features two entity types: `product` and `brand`, and three types of relational information: `also_bought, also_viewed` between `product` entities, and `has_brand` relation between `product` and `brand` entities. In total, it comprises around 1.03M entities (`product` entities: 0.96M, `brand` entities: 0.07M) and 3.9M relations (`also_bought`: 1.7M, `also_viewed`: 1.3M, `has_a_brand`: 0.9M). We obtain the textual information by combining the meta data from Amazon Product Reviews with the customer Q&A records from Amazon Question and Answer Data. This provides a rich amount of textual data, including product titles, descriptions, prices, customer reviews, and related customer Q&A for each `product`. For `brand` entities, we extract brand titles as the textual attribute. Especially, Amazon SKB features an extensive textual data, which is largely contributed from customer reviews and Q&A. We also observe a long-tail distribution of text data on each product by number of tokens.

MAG Semi-structured Knowledge Base. This knowledge base is constructed from obgn-MAG Hu et al. (2021), obgn-papers100M Hu et al. (2021), and Microsoft Academic Graph (version 2019-03-22) Wang et al. (2020); Sinha et al. (2015). The knowledge base contains around 1.9M entities under four entity types (`author`: 1.1M, `paper`: 0.7M, `institution`: 9K, `field_of_study`: 0.06M) and 20M relations under four relation types (`author_writes_paper`: 7.9M, `paper_has_field_of_study`: 7.2M, `paper_cites_paper`: 4.9M, `author_affiliated_with_institution`: 1.0M). We construct the knowledge base by selecting papers[1] shared between obgn-MAG and obgn-papers100M. The relational structure comes from the extracted subgraph in obgn-mag based on the selected papers. The textual information such as titles and abstracts is sourced from obgn-papers100M. Additionally, we augment the textual data by integrating details from the Microsoft Academic Graph database, providing extra information like paper venue, author and institution names. This SKB demonstrates a large number of entities and relations associate with `paper` nodes, especially on citation and authorship relation types.

Prime Semi-structured Knowledge Base. We leverage the existing knowledge graph PrimeKG Chandak et al. (2023) which contains ten entity types including `disease, drug, gene/protein`, and eighteen relation types, such as `associated_with, synergistic_interaction, indication, expression_present`. The entity count in our knowledge base is approximately 129K, with around 8M relations. The details on entity numbers and relation tuples are available in Appendix A.4.2. Compared to the Amazon and MAG SKBs, Prime SKB is denser and features a greater variety of relation types. While PrimeKG already provides text information on `disease` and `drug` entities, including drug mechanisms and disease descriptions, we additionally integrate the textual details from multiple databases for `gene/protein` and `pathway` entities such as genomic position, gene activity summary and pathway orthologous event.

A.3 Query dataset statistics

	#queries	#queries w/ multiple answers	avg. #answers	train / val / test
STARK-AMAZON	9,100	7,082	17.99	0.65 / 0.17 / 0.18
STARK-MAG	13,323	6,872	2.78	0.60 / 0.20 / 0.20
STARK-PRIME	11,204	4,188	2.56	0.55 / 0.20 / 0.25

Table 5: Dataset statistics on STARK.

A.3.1 Details of query dataset

STARK-AMAZON. This dataset comprises 9,100 customer-focused queries aimed at product searches, with a notable 68% of these queries yielding multiple answers. The dataset prioritizes customer-oriented criteria, highlighting textual elements such as product quality, functionality, and style. Additionally, it accentuates relational aspects, including brand and product connections (*e.g.,* complementary or substitute items). The queries are framed in detailed, conversation-like formats, enriching the context and enhancing the dataset's relevance to real-world scenarios.

STARK-MAG. This dataset comprises 13,323 paper queries, with roughly half yielding multiple answers. Beyond the single-hop relational requirements, STARK-MAG also emphasizes the fusion between the textual requirements with multi-hop queries. For example, "Are there any papers from

King's College London" highlights the metapath (`institution` → `author` → `paper`) on the relational structure. We designed three single-hop and four multi-hop relational query templates, *cf.* Appendix A.4.1. The textual aspects of the queries diversifies, focusing on different elements such as the paper's topic, methodology, or the advancements it introduces, sourced mainly from the abstracts.

STAR-PRIME. This dataset, comprising 11,204 queries across all ten entity types, incorporates single-hop and multi-hop relational information similar to STAR-MAG. We developed 28 multi-hop query templates, detailed in Appendix A.4.2, to cover various relation types and ensure their practical relevance. For instance, the query "What is the drug that targets the genes or proteins expressed in <anatomy>?" serves applications in precision medicine and pharmacogenomics, aiding researchers and healthcare professionals in identifying drugs that act on genes or proteins associated with specific anatomical areas and enabling more targeted treatments. For entities like `drug`, `disease`, `gene/protein`, and `pathway`, the queries are a hybrid of relational requirements and textual requirements drawn from documents. A notable feature is the deliberate simulation of three distinct roles – medical scientist, doctor, and patient – for certain queries related to drugs and diseases. This is intended to diversify the language used across various user types and to evaluate the robustness of the retrieval system under varied linguistic scenarios. For entities such as `effect/phenotype`, the queries rely solely on relational data due to limited textual information in the knowledge base.

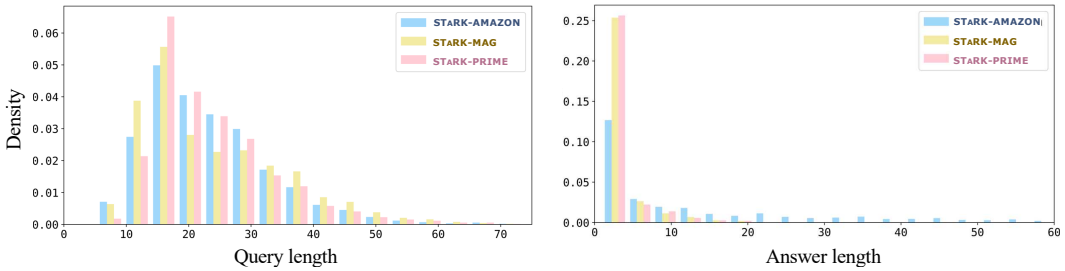
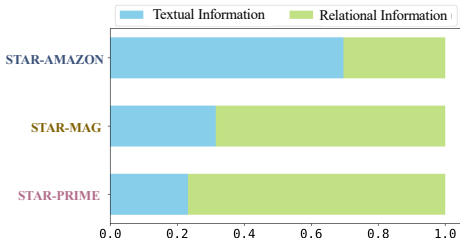


Figure 7: Distribution of query and answer lengths on STAR datasets.

Table 6: Average query diversity on STAR datasets.

	Shannon Entropy	Type-Token Ratio
STAR-AMAZON	10.39	0.179
STAR-MAG	10.25	0.180
STAR-PRIME	9.63	0.143

Figure 8: Average relative composition of relational vs. textual information.



A.3.2 Query dataset analysis

Data Distribution Analysis. Further, to qualitatively understand the characteristics of our benchmark datasets, we study the data distribution in three dimensions:

- **Query and Answer Length**. We analyze the query length distribution (measured in the number of words) and the number of ground truth answers for each query. The query length reflects the amount of context information provided by users, while the number of ground truth answers indicates the level of inclusiveness or ambiguity of queries within the specific SKB. As illustrated in Figure 7, all three datasets exhibit similar query length distributions, with most queries containing around 16 words. Queries with up to 50 words typically involve mentions of other entities, such as product or paper titles, or provide more detailed context, such as specific symptoms descriptions. Interestingly, the answer length distribution for STAR-AMAZON shows a more significant long-tail pattern, with approximately 22% of answers exceeding 30 entities, whereas the answer lengths for STAR-PRIME and STAR-MAG are all within 20 entities. On average, as shown in Table 5, STAR-AMAZON presents an average answer length of 5.32, indicative of the e-commerce recommendation domain

where general customer inquiries frequently result in diverse recommendations. While STARK-PRIME has the smallest average answer length, partially attributed to the smaller size of the Prime SKB compared to the other two SKBs.

- **Query Diversity**. A diverse set of queries pose challenges for broader applicability to meet varying user demands. Specifically, we measure query diversity using Shannon Entropy, which quantifies the uncertainty in the word distribution across all queries, and the Type-Token Ratio (TTR), which calculates the proportion of unique words to the total number of words. Higher values of Shannon Entropy and TTR indicate greater lexical diversity, reflecting a wider range of topics and linguistic expressions in the query set. As shown in Table 6, we observe high values of Shannon Entropy on all of the datasets (for reference, a text with 1000 unique words, each occurring with equal probability, the Shannon Entropy is 9.97). While TTR is more sensitive to the text size, the TTR values consistently demonstrate a steady presence of unique words relative to the large text size.
- **Ratio of Relational *vs.* Textual Information**. A key feature of our benchmark dataset is the composition of textual and relational information. Therefore, it is crucial to understand the proportionality between these two types of information. We calculate the ratio of the length of relational requirements to the length of textual requirements for each query and then average these ratios across each dataset. Intuitively, the ratio serves as an approximation of the relative distribution based on the length of requirements, and does not directly reflect the importance of each type of information in determining the final answers. As shown in Figure 8, the ratios vary across different datasets, highlighting a differing emphasis on textual versus relational information. This distribution variation presents additional challenges for retrieval systems, requiring them to adapt to the dataset characteristic and specific balance of information.

A.4 Types and meta-paths

A.4.1 STARK-MAG

Relational query templates:

metapath	multi-hop query template
(author → paper)	"Can you list the papers authored by <author>?"
(paper → paper)	"Which papers have been cited by the paper <paper>?"
(field_of_study → paper)	"Can you provide a list of papers in the field of <field_of_study>?"
(institution → author → paper)	"What papers have been published by researchers from <institution>?"
(paper → author → paper)	"What papers have been published by researchers that are coauthors of <paper>?"
(paper → author → paper ← field_of_study ← paper)	"Can you find papers that share a coauthor with <paper> and are also in the same field of study?"
(institution → author → paper ← field_of_study)	"Are there any papers associated with <institution> and are in the field of <field_of_study>?"

For example, the metapath (field_of_study $\rightarrow$ paper) requires an initial field_of_study entity to be filled in the corresponding query template. For multi-hop metapaths, the last metapath (institution $\rightarrow$ author $\rightarrow$ paper $\leftarrow$ field_of_study) requires an institution entity and a field_of_study entity to initialize the query.

A.4.2 STARK-PRIME

Number of entities in each type.

```
#disease:            17,080
#gene/protein:       27,671
#molecular_function: 11,169
#drug:                7,957
#pathway:             2,516
#anatomy:            14,035
```

```
#effect/phenotype:   15,311
#biological_process: 28,642
#cellular_component:  4,176
#exposure:              818
```

Relational tuples.

```
[(anatomy, expression absent, gene/protein),
(anatomy, expression present, gene/protein),
(anatomy, parent-child, anatomy),
(biological_process, interacts with, exposure),
(biological_process, interacts with, gene/protein),
(biological_process, parent-child, biological_process),
(cellular_component, interacts with, exposure),
(cellular_component, interacts with, gene/protein),
(cellular_component, parent-child, cellular_component),
(disease, associated with, gene/protein),
(disease, contraindication, drug),
(disease, indication, drug),
(disease, linked to, exposure),
(disease, off-label use, drug),
(disease, parent-child, disease),
(disease, phenotype absent, effect/phenotype),
(disease, phenotype present, effect/phenotype),
(drug, carrier, gene/protein),
(drug, contraindication, disease),
(drug, enzyme, gene/protein),
(drug, indication, disease),
(drug, off-label use, disease),
(drug, side effect, effect/phenotype),
(drug, synergistic interaction, drug),
(drug, target, gene/protein),
(drug, transporter, gene/protein),
(effect/phenotype, associated with, gene/protein),
(effect/phenotype, parent-child, effect/phenotype),
(effect/phenotype, phenotype absent, disease),
(effect/phenotype, phenotype present, disease),
(effect/phenotype, side effect, drug),
(exposure, interacts with, biological_process),
(exposure, interacts with, cellular_component),
(exposure, interacts with, gene/protein),
(exposure, interacts with, molecular_function),
(exposure, linked to, disease),
(exposure, parent-child, exposure),
(gene/protein, associated with, disease),
(gene/protein, associated with, effect/phenotype),
(gene/protein, carrier, drug),
(gene/protein, enzyme, drug),
(gene/protein, expression absent, anatomy),
(gene/protein, expression present, anatomy),
(gene/protein, interacts with, biological_process),
(gene/protein, interacts with, cellular_component),
(gene/protein, interacts with, exposure),
(gene/protein, interacts with, molecular_function),
(gene/protein, interacts with, pathway),
(gene/protein, ppi, gene/protein),
(gene/protein, target, drug),
(gene/protein, transporter, drug),
(molecular_function, interacts with, exposure),
(molecular_function, interacts with, gene/protein),
(molecular_function, parent-child, molecular_function),
```

```
(pathway, interacts with, gene/protein),
(pathway, parent-child, pathway)]
```

Relational query templates.

```
{
(effect/phenotype → [phenotype absent] → disease ← [!indication] ← drug):
    "Find diseases with zero indication drug and are associated with <effect/phenotype>",
(drug → [contraindication] → disease ← [associated with] ← gene/protein):
    "Identify diseases associated with <gene/protein> and are contraindicated with <drug>",
(anatomy → [expression present] → gene/protein ← [expression absent] ← anatomy):
    "What gene or protein is expressed in <anatomy1> while is absent in <anatomy2>?",
(anatomy → [expression absent] → gene/protein ← [expression absent] ← anatomy):
    "What gene/protein is absent in both <anatomy1> and <anatomy2>?",
(drug → [carrier] → gene/protein ← [carrier] ← drug):
    "Which target genes are shared carriers between <drug1> and <drug2>?",
(anatomy → [expression present] → gene/protein → [target] → drug):
    "What is the drug that targets the genes or proteins which are expressed in <anatomy>?",
(drug → [side effect] → effect/phenotype → [side effect] → drug):
    "What drug has common side effects as <drug>?",
(drug → [carrier] → gene/protein → [carrier] → drug):
    "What is the drug that has common gene/protein carrier with <drug>?",
(anatomy → [expression present] → gene/protein → enzyme → drug):
    "What is the drug that some genes or proteins act as an enzyme upon, where the genes or proteins are expressed in <anatomy>?",
(cellular_component → [interacts with] → gene/protein → [carrier] → drug):
    "What is the drug carried by genes or proteins that interact with <cellular_component>?",
(molecular_function → [interacts with] → gene/protein → [target] → drug):
    "What drug targets the genes or proteins that interact with <molecular_function>?",
(effect/phenotype → [side effect] → drug → [synergistic interaction] → drug):
    "What drug has a synergistic interaction with the drug that has <effect/phenotype> as a side effect?",
(disease → [indication] → drug → [contraindication] → disease):
    "What disease is a contraindication for the drugs indicated for <disease>?",
(disease → [parent-child] → disease → [phenotype present] → effect/phenotype):
    "What effect or phenotype is present in the sub type of <disease>?",
(gene/protein → [transporter] → drug → [side effect] → effect/phenotype):
    "What effect or phenotype is a [side effect] of the drug transported by <gene/protein>?",
(drug → [transporter] → gene/protein → [interacts with] → exposure):
    "What exposure may affect <drug>s efficacy by acting on its transporter genes?",
(pathway → [interacts with] → gene/protein → [ppi] → gene/protein):
    "What gene/protein interacts with the gene/protein that related to <pathway>?",
(drug → [synergistic interaction] → drug → [transporter] → gene/protein):
    "What gene or protein transports the drugs that have a synergistic interaction with <drug>?",
(biological_process → [interacts with] → gene/protein → [interacts with] → biological_process):
    "What biological process has the common interactino pattern with gene or proteins as <biological_process>?",
(effect/phenotype → [associated with] → gene/protein → [interacts with] → biological_process):
    "What biological process interacts with the gene/protein associated with <effect/phenotype>?",
(drug → [transporter] → gene/protein → [expression present] → anatomy):
    "What anatomy expressesed by the gene/protein that affect the transporter of <drug>?",
(drug → [target] → gene/protein → [interacts with] → cellular_component):
    "What cellular component interacts with genes or proteins targeted by <drug>?",
(biological_process → [interacts with] → gene/protein → [expression absent] → anatomy):
    "What anatomy does not express the genes or proteins that interacts with <biological_process>?",
(effect/phenotype → [associated with] → gene/protein → [expression absent] → anatomy):
    "What anatomy does not express the genes or proteins associated with <effect/phenotype>?",
(drug → [indication] → disease → [indication] → drug) & (drug → [synergistic interaction] → drug):
    "Find drugs that has a synergistic interaction with <drug> and both are indicated for the same disease.",
(pathway → [interacts with] → gene/protein → [interacts with] → pathway) & (pathway → [parent-child] → pathway):
    "Find pathway that is related with <pathway> and both can [interacts with] the same gene/protein.",
(gene/protein → [associated with] → disease → [associated with] → gene/protein) & (gene/protein → [ppi] → gene/protein):
    "Find gene/protein that can interect with <gene/protein> and both are associated with the same disease.",
(gene/protein → [associated with] → effect/phenotype → [associated with] → gene/protein) & (gene/protein → [ppi] → gene/protein):
    "Find gene/protein that can interect with <gene/protein> and both are associated with the same effect/phenotype."
}
```

where $[\cdot]$ denotes the relation type.

A.5 API tool kits

A.6 Prompts

A.6.1 Extracting textual requirements

Prompt for STaR-Amazon: Textual requirement extraction

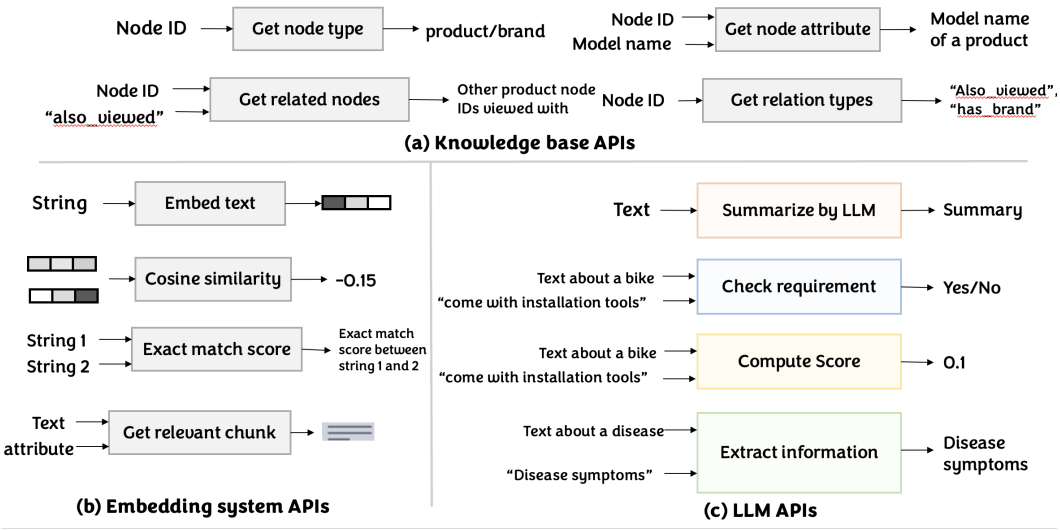


Figure 9: APIs provided to the agent.

```
You are an intelligent assistant that extracts diverse positive
requirements and negative perspectives for an Amazon product. I will
give you the following information:

- product: <product name>
- dimensions: <product dimensions>
- weight: <product weight>
- description: <product description>
- features:
  #1: <feature #1>
  ...
- reviews:
  #1:
    summary: <review summary>
    text: <full review text>
  #2:
    ...
- Q&A:
  #1:
    question: <product-related question>
    answer: <answer to product-related question>
  #2:
    ...

Based on the given product information, you need to (1) identify the
product's generic category, (2) list all of the negative perspectives
and their sources, and (2) extract up to five hard and five soft
requirements relevant to customers' interests along with their sources.
 (1) For example, the product's generic category can be "a chess book"
or "a phone case for iphone 6", do not use the product name directly.
(2) Negative perspectives are those that the product doesn't fulfill,
which come from the negative reviews or Q&A. (3) For the requirements,
you should only focus on the product's advantages and positive
perspects. Hard requirements mean that product must fulfil, such as
size and functionality. Soft requirements are not as strictly defined
but still desirable, such as a product is easy-to-use. For (2) and (3),
 each source is a composite of the key and index (if applicable)
```

```
separated by "-", such as "description", "Q&A-#1". You should provide
the response in a specific format as follows where "item" refers to the
 product's generic category, e.g., "a chess book".

Response format:

{
  "item": <the product's generic category> ,
  "negative": [[<source of negative perspective>, <negative perspective
   description>]],
  "hard": [[<source of hard requirement>, <hard requirement description
  >], ...],
  "soft": [[<source of soft requirement>, <soft requirement description
  >], ...]
}

Here is an example of the response:
{
  "item": "a camping chair",
  "negative": [["reviews-#3", "the chair is not sturdy enough"], ["Q&A
  -#1", "wrong color"]],
  "hard": [["description", "has a breathable mesh back"], ["description
  ", "the arm is adjustable"], ["dimensions", "more than 35 inches long
  "], ["features-#7", "with a arm rest cup holder"], ["Q&A-#4", "need
  to come with a carrying bag"]],
  "soft": [["description", "suitable for outdoors"], ["features-#9", "
  compact and save space"], ["reviews-#6", "light and portable"]]
}

This is the information of the product that you need to write response
for:

Information:
<product_doc>
Response:
```

Prompt for STAR-MAG: Textual requirement extraction

```
You are a helpful assistant that helps me extract one short requirement
 (no more than 10 words) about a paper from the paper information that
researchers might be interested in. The requirement can be about the
paper content, publication date, publication venue, etc. The
requirement should be general and not too specific. I will give you the
 paper information, and you should return a short phrase about the
paper, starting with 'the paper...'. This is the paper information:
<doc_info>
Please only return the short and general requirement without additional
 comments.
```

Prompt for STAR-PRIME: Textual requirement extraction

```
You are a helpful assistant that helps me extract <n_properties> from a
 given <target> information that a <role> may be interested in.
<role_instruction>
```

```
Each property should be no more than 10 words and start with "the <
target>". You should also include the source of each property as
indicated in the paragraph names of the information, e.g., "details.
mayo_symptoms", "details.summary", etc. You should return a list of
properties and their sources following the format:
[["<short_property1>", "<source1>"], ["<short_property2>", "<source2
>"], ...]
This is the information:
<doc_info>
Please provide only the list with <n_properties> in your response.
Response:
```

According to the role assigned to simulate the query content, the `<role_instruction>` as shown below is filled in accordingly.

role	role instruction
Doctor	Doctors typically ask questions aimed at diagnosing and treating. Their questions tend to be direct and practical, focusing on aspects involving side effects, symptoms, and complications etc.
Medical scientist	Medical scientists often ask questions that reflect the complexity and depth of the scientific inquiry in the medical field. Their questions tend to be detailed and specific, focusing on aspects such as: etiology and pathophysiology, genetic factors, association with pathway, protein, or molecular function.
Patient	Patients typically don't know the professional medical terminology. Their questions tend to be straightforward, focusing on practical concerns on the symptons, effects, and inheritance etc., instead of the detailed mechanisms, which may also include more context.

A.6.2 Combining relational and textual requirements

Prompt for STAR-AMAZON: Fuse relational and textual requirements

```
You are an intelligent assistant that generates queries about an Amazon
 item. I will provide you with the item name, requirements, and its
negative customer reviews. Your task is to create a natural-sounding
customer query that leads to the item as the answer, using the
requirements that are non-conflicting with the negative reviews, and
provide the indices of the requirements used. For example:

Information:
- item: a soccer rebounder
- requirements:
#1: needs a heavy-duty 1-inch to 3-inch steel tube frame
#2: should be adjustable for practicing different skills
#3: should be durable
#4: usually be viewed together with <SKLZ Star-Kick Hands Free Solo
Soccer Trainer>
- negative reviews:
#1: it was broken after a few uses

Response:
{
  "index": [1, 2, 4],
```

```
  "query": "Please recommend a soccer rebounder with a steel frame,
  about 2 inches thick, that can adapt to different skill levels. We
  had a blast using the <SKLZ Star-Kick Hands Free Solo Soccer Trainer>
   with my family, and I'm on the lookout for something similar."
}

As the negative review indicates that the soccer rebouncer lacks
durability, your query should only incorporate requirements #1, #2, and
 #4 while excluding #3. A requirement should only be excluded if it
conflicts with negative feedback or is unlikely to align with customers
' interests. For relational requirements about another <product>, do
not directly use "usually bought/viewed together with <product>" in the
 query. You must deduce the item's relationship with <product> into
substitute or complement, and create various user scenarios, such as
the item should be compatible or used with <product> (for complements)
or match in style with <product> (for substitute), to make the queries
sound natural. Except for <product>, you should change the description
but convey similar meanings. The query structure is completely flexible
. Here is the information to generate the requirement indices and a
natural-sounding query:

Information:
<product_req_and_neg_comments>
Response:
```

Prompt for STAR-MAG: Fuse relational and textual requirements

```
You are a helpful assistant that helps me generate a new query by
incorporating an additional requirement into a given query, and form a
coherent and natural-sounding question.
This is the existing query:
<query>
This is the additional requirement:
<additional_textual_requirement>
You should be creative in combining the existing query and requirement,
 and flexible in structuring the new query, adding context as needed.
Please return the new query without additional comments:
```

The prompt of a second-time rewrite by GPT-4 Turbo:

```
You are a helpful assistant that helps make a researcher's query about
a paper more natural-sounding, akin to the language used in ArXiv web
searches. You should change the description but convey similar meanings
. The query structure is completely flexible. The original query:
"<query>"
Please only output the new query without additional comments:
```

Prompt for STAR-PRIME: Fuse relational and textual requirements

```
You are a helpful assistant that helps me generate a natural-sounding
and coherent query as if you were a <role>. The query should be created
 based on a list of requirements for searching <plural_target> in a
database. I will provide you with the requirements in the following
format:
[<requirement1>, <requirement2>, ...]
```

```
You should create the query based solely on the given requirements.
Moreover, you should craft the query from the perspective of a <role>.
<role_instruction>
For example, a query from a <role> could be
"<example_query>"
You can be flexible in structuring the query and adding additional
context. Ensure that the query uses different descriptions than the
original property descriptions while retaining similar meanings. The
query should sound concise and natural. These are the requirements:
<requirements>
Please create the query based on the given requirements and provide
only the query without additional comments. Your response:
```

The prompt of a second-time rewrite by GPT-4 Turbo:

```
You are a helpful assistant that helps me rewrite a query that searches
 for <plural_target> from the perspective of a <role>. You should
maintain the requirements from the original query and the
characteristics of the <role>, while being creative and flexible in
structuring the query. Ensure the revised query is concise and natural-
sounding. Original query: "<query>". Please output only the rewritten
query:
```

A.6.3 Filtering additional answers

Prompt for STAR-AMAZON: Filtering additional answers

Filter products by general category

```
You are an intelligent assistant that identifies whether an Amazon
product belongs to a given category. I will give you the product
information. You should only answer yes / no in the response. For
examples, the product <SKLZ Star-Kick Hands Free Solo Soccer Trainer>
belongs to the category "soccer trainer" and the product <Test your
Opening, Middlegame and Endgame Play - VOLUME 2> belongs to the
category "a chess opening book", while <Baby Girls One-piece Shiny
Athletic Leotard Ballet Tutu with Bow> doesn't belong to category "an
adult tutu".

Information:
- product title: <<product_title>>
- product description: <product_description>

Does the product belong to "<target_category>"? Response (yes/no):
```

Filter products by requirements

```
You are a helpful assistant that helps me check whether an Amazon
product satisfies the given requirements. I will provide you with the
product information, which may include the product description,
features, reviews, and Q&A from customers. Your task is to assess
whether the product meets each requirement based on the provided
information. If there is no information that supports the requirement,
your response for that requirement is "NA". If there is relevant
information that supports the requirement, your response for that
requirement is the information source that fulfills the requirement.
Each information source is a composite of the key and index (if
```

```
applicable), separated by "-", such as "description", "features-#3", "Q
&A-#1", "reviews-#2". If there are multiple sources,

Response:
{
    1: "NA" or [the information sources that satisfy the requirement
    #1],
    2: "NA" or [the information sources that satisfy the requirement
    #2],
    ...
}

Here is the product information:
<product_doc>
The requirements are as follows:
<customer_requirements>

Response:
```

Prompt for STAR-MAG: Filtering additional answers

```
You are a helpful assistant that helps me verify whether a given <
target_node_type> is subject to a requirement. I will provide you with
the <target_node_type> information and the requirement, and you should
return only a 'True' or 'False' value, indicating whether the <
target_node_type> meets the requirement.
This is the <target_node_type> information:
<doc_info>
This is the requirement:
<additional_textual_requirement>
Please return only the boolean value without additional comments:
```

Prompt for STAR-PRIME: Filtering additional answers

```
You are a helpful assistant tasked with verifying whether a given <
target> satisfies each of the provided requirements. I will give you
the requirements in the following format:
{1: <requirement1>, 2: <requirement2>, ...}
When evidence in the <target> information confirms a requirement is met
, cite the source, for example, 'details.mayo_symptoms', 'details.
summary'. If no direct evidence exists, indicate this with 'NA'. The
output in JSON format should be as follows:
{1: 'NA' or <source1>, 2: 'NA' or <source2>, ...}
This is the <target> information:
<doc_info>
These are the requirements:
<requirements>
Please provide only the JSON in your response. Response:"
```