

SCAIR: Schema-Conditioned Agentic Iterative Reasoning for Enterprise Knowledge Graphs

Anonymous ACL submission

Abstract

Knowledge Graph-based Retrieval-Augmented Generation (KG-RAG) enables natural language interaction with structured enterprise knowledge, yet existing agentic approaches that perform well on public benchmarks often fail to generalize to real-world enterprise Knowledge Graphs (KGs), which are dense, schema-driven, and operationally constrained. To address these limitations, we propose SCAIR (Schema-Conditioned Agentic Iterative Reasoning), a training-free framework that integrates structured planning with controlled iterative reasoning by injecting schema-conditioned structural priors and enforcing schema-aware traversal during multi-hop reasoning. Experiments on an enterprise-oriented benchmark constructed from a real-world Configuration Management DataBase (CMDB) demonstrate that SCAIR substantially improves performance over existing KG-RAG methods. Crucially, our study highlights that reliable enterprise graph reasoning cannot rely on generic agentic designs; instead, it must explicitly incorporate the target domain's structural and operational constraints into the reasoning process. We demonstrate that by aligning agent design with business logic, substantial performance gains can be achieved without the need for costly model retraining.

1 Introduction

In the era of Industry 4.0, enterprises generate vast amounts of heterogeneous data distributed across isolated systems, ranging from structured sensor logs to unstructured maintenance reports (Lasi et al., 2014; Frank et al., 2019). Knowledge Graphs (KGs), as a practical solution for integrating such data, provide a flexible structure that explicitly models entities and their dependencies within a unified semantic network (Hogan et al., 2021; Pan et al., 2024). This structural representation is particularly valuable for complex industrial applica-

tions (Liu et al., 2023; Listl et al., 2024; Grangel-González et al., 2020). For example, in enterprise IT and manufacturing environments, Configuration Management Databases (CMDBs) are usually modeled as KGs to represent machines, components, production lines, and their operational states (Schmidt et al., 2025b). Such KGs support business-critical queries, including dependency analysis, fault diagnosis, and component replacement, where reasoning over interconnected assets is essential for reliable decision-making.

To democratize access to structured enterprise knowledge, KG-based Retrieval-Augmented Generation (KG-RAG) has emerged as an effective solution (Peng et al., 2024; Zhu et al., 2025; Zhang et al., 2025). By enabling users to interact with enterprise KGs through natural language, KG-RAG retrieves grounded and explainable evidence to answer complex queries. More recently, agentic KG-RAG approaches typically follow one of two paradigms: *plan-and-execute*, which learns explicit reasoning plans or subgraph selection strategies (He et al., 2024; Luo et al., 2024), and *ReAct-style iterative exploration*, which dynamically interleaves reasoning and graph traversal (Sun et al., 2024; Chen et al., 2024; Zhou et al., 2025b). Both paradigms have demonstrated strong performance on public KGQA benchmarks.

To evaluate the deployment readiness of these paradigms, we construct an enterprise-oriented KGQA benchmark derived from a real-world manufacturing CMDB. Unlike the sparse and open-domain graphs used in existing benchmarks, this dataset captures the dense connectivity, strict schema constraints, and operational dependencies characteristic of industrial environments. Experiments on this benchmark reveal that existing agentic KG-RAG methods fail to generalize under these conditions. Public benchmarks can partially obscure these weaknesses, both because their graphs are structurally simpler and because entity and re-

lation names often overlap with LLM pretraining corpora, which may inflate reported performance. In contrast, the enterprise setting exposes systematic failure modes: ReAct-style exploration suffers from uncontrolled search expansion and semantic drift in dense subgraphs, while plan-and-execute methods rely heavily on distribution-specific training and struggle to generalize across query patterns.

Motivated by these findings, we introduce **SCAIR** (Schema-Conditioned Agentic Iterative Reasoning), a hybrid agentic framework that unifies structured planning with controlled iterative reasoning. SCAIR injects schema-conditioned structural priors, enforces schema-aware traversal during multi-hop reasoning, and controls topic entity propagation to balance exploration and exploitation during search. Notably, SCAIR outperforms all evaluated baselines on the enterprise benchmark without requiring any task-specific training.

The key lesson from this study is that generic agentic designs optimized on simplified public benchmarks often fail to generalize to real business use cases. Effective enterprise graph reasoning must explicitly account for the structural and operational characteristics of the target domain and reflect these constraints in the agent design. With a clear understanding of the business scenario and systematic failure analysis of existing solutions, substantial performance gains can be achieved without costly model retraining.

2 Preliminaries

2.1 Knowledge Graph Question Answering

KGs represent structured knowledge as a graph of entities connected by typed relations. Formally, a KG is defined as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$, where \mathcal{E} denotes the set of entities, \mathcal{R} the set of relation types, and $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ the set of triples. Each triple $(e_h, r, e_t) \in \mathcal{T}$ encodes an atomic fact, stating that a relation $r \in \mathcal{R}$ holds between a head entity $e_h \in \mathcal{E}$ and a tail entity $e_t \in \mathcal{E}$. Knowledge Graph Question Answering (KGQA) aims to answer natural language questions by reasoning over relevant facts in \mathcal{G} .

2.2 Existing Approaches

KGQA has been addressed through two main paradigms. **Semantic parsing** approaches translate natural language questions into executable logical forms, such as SPARQL queries, which can then

be executed over the KG to obtain answers (Berant and Liang, 2014; Berant et al., 2013; Reddy et al., 2014). In contrast, **information retrieval-based** methods (i.e., KG-RAG) retrieve and rank candidate entities or subgraphs using distributed representations, often leveraging graph neural networks to encode structural information, and then generate answers conditioned on the retrieved evidence (Bordes et al., 2015; Dong et al., 2015). More recently, the field has converged toward **agentic reasoning frameworks**, in which an LLM operates as a decision-making agent that plans, interacts with the KG, and reasons over retrieved evidence (Luo et al., 2024; He et al., 2024; Sun et al., 2024; Chen et al., 2024).

Within this agentic paradigm, approaches such as Reasoning on Graphs (RoG) (Luo et al., 2024) and G-Retriever (He et al., 2024) follow a **Plan-and-Execute** design (Wang et al., 2023). The agent first constructs an explicit reasoning plan or identifies a target subgraph, and subsequently executes this plan through constrained graph retrieval before generating an answer. These methods are primarily training-based, as their planning or retrieval components are learned from supervision to align reasoning plans or subgraph selection with downstream question-answering objectives. Think-on-Graph (ToG) (Sun et al., 2024) and Plan-on-Graph (PoG) (Chen et al., 2024) adhere more closely to the **ReAct**-style framework (Yao et al., 2023), interleaving reasoning steps with iterative graph exploration and allowing the agent to adapt traversal decisions based on intermediate observations. This class of methods is typically training-free, relying on the LLM’s inference-time reasoning capabilities rather than task-specific parameter optimization.

3 An Enterprise-Oriented KGQA Benchmark

In this section, we first examine why widely used KGQA benchmarks fail to capture the requirements of industrial applications. We then introduce an enterprise-oriented benchmark constructed from a real-world Configuration Management Database (CMDB), designed to evaluate agentic KG-RAG systems under realistic operational constraints.

3.1 Limitations of Existing KGQA Benchmarks

KGQA benchmarks such as *WebQSP* (Yih et al., 2016) and *Complex Web Questions (CWQ)* (Talmor

and Berant, 2018) are built on open-domain KGs (e.g., Freebase (Bollacker et al., 2008)) and are designed to evaluate compositional reasoning over relatively sparse graph structures. While influential, these benchmarks rely on assumptions that do not align with enterprise use cases.

First, they assume that each question maps to a single, well-defined executable query that fully captures the user’s intent. In industrial settings, however, questions are often underspecified and constraint-driven, and multiple reasoning paths may be valid depending on the operational context. Second, benchmark questions are largely *retrospective*, treating the KG as an encyclopedia for fact retrieval. Enterprise questions are typically *prospective* and problem-oriented, involving implicit business logic such as operational status, compatibility, or conditional replacement, which is encoded structurally in the KG rather than explicitly stated in text. Finally, semantic leakage from LLM pretraining is difficult to avoid in public benchmarks, as entity and relation names often overlap with pretraining data. This makes it unclear whether strong performance reflects genuine graph reasoning or implicit memorization.

These differences expose a clear gap between existing KGQA benchmarks and the requirements of real-world industrial applications.

3.2 Enterprise KGQA Benchmark

To address the gap between public benchmarks and real industrial requirements, we construct an enterprise KGQA benchmark grounded in a real-world manufacturing CMDB. The benchmark consists of (i) a KG derived from the CMDB (CMDB-KG), and (ii) business-oriented question–answer pairs.

CMDB-KG. CMDB-KG is constructed from a real-world manufacturing CMDB (Schmidt et al., 2025a) that integrates heterogeneous enterprise data, including production lines, machines, components, manufacturers, and operational attributes. The graph contains 116,369 triples. Construction details are provided in Appendix A.1.

Example 1 (CMDB-KG). *Figure 1 shows a representative fragment of CMDB-KG. A production line (e.g., Line W509-6) is linked to multiple machines via hasMachine, and each machine connects to its installed components through hasComponent. Machines and components are annotated with operational status (e.g., working, idle, broken), and components may additionally be connected*

by similarTo relations to indicate functional interchangeability.

Question-Answer Pair Generation. We begin by collecting representative business-oriented information needs from the enterprise setting and abstracting them into a set of structured query templates aligned with the CMDB schema. These templates are then instantiated automatically over the CMDB-KG in a scalable manner, generating executable question–answer pairs grounded in real enterprise data. The resulting benchmark comprises 9 representative query types and a total of 19,080 questions. Detailed specifications are provided in the Appendix A.2.

Example 2 (Business-Oriented Question). *Consider the question: “Which working components can replace broken components installed on machines in production line W509-6?” Unlike public benchmark questions, this query cannot be expressed as a single well-defined compositional logical form. It requires identifying broken components in W509-6, retrieving functionally equivalent components via similarTo, and enforcing operational constraints. In particular, valid replacements must be both working and installed on idle machines; otherwise, removing them would disrupt other production lines. These implicit business constraints must be respected during reasoning.*

Evaluation Protocol. We follow the evaluation protocol of Zhou et al. (2025a,c). Let \mathcal{P}_q and \mathcal{A}_q denote the predicted and ground-truth answer sets for question q , respectively. *Accuracy* measures exact set match ($\mathcal{P}_q = \mathcal{A}_q$). *Hits@Any* measures whether at least one correct answer is retrieved ($\mathcal{P}_q \cap \mathcal{A}_q \neq \emptyset$). *Precision* and *Recall* quantify answer correctness and completeness based on set overlap, and *F1* is their harmonic mean.

3.3 Limitations of Existing Agentic KG-RAG Paradigms

Through qualitative analysis of model outputs and reasoning traces on the enterprise benchmark, we observe systematic limitations in both dominant agentic KG-RAG paradigms: plan-and-execute and ReAct-style iterative exploration.

Limitations of plan-and-execute approaches. Plan-and-execute methods mitigate uncontrolled exploration by learning explicit reasoning plans or subgraph selection strategies. However, they rely on substantial training and hyperparameter

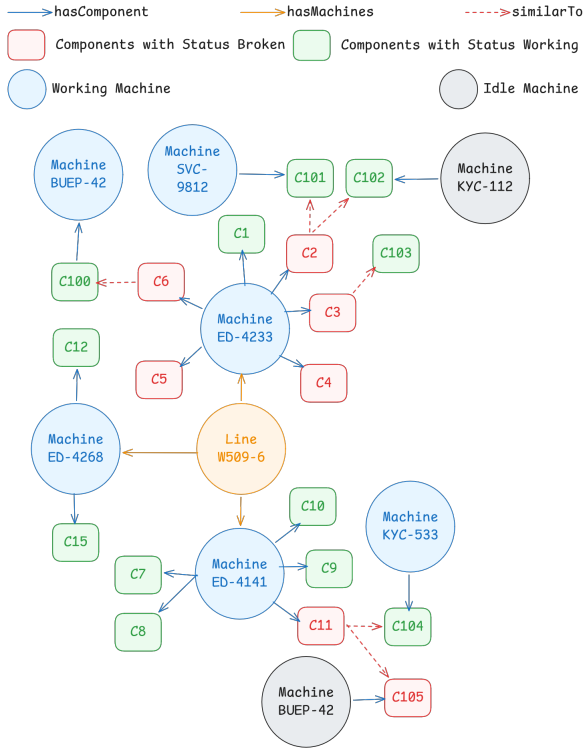


Figure 1: Representative fragment of CMDB-KG.

tuning to perform reliably, which is particularly challenging when adapting large models in enterprise environments. In practice, effective training requires sufficient coverage of the underlying query and relation distributions; otherwise, learned planning patterns tend to overfit to training-specific structures and fail to generalize to unseen schemas or reasoning compositions. This dependence on distribution-specific training limits robustness under evolving enterprise KGs.

Limitations of ReAct-style exploration. ReAct-based methods rely on iterative, relevance-driven traversal. In dense enterprise graphs, this often leads to *search explosion*, as models expand through high-degree attribute nodes (e.g., identifiers or status values), rapidly increasing the search space. Moreover, traversal guided primarily by semantic similarity frequently results in *schema-agnostic reasoning*, producing paths that are semantically plausible but structurally invalid under enterprise schema constraints. As reasoning depth increases, these issues compound into *semantic drift*, where the agent deviates from the original query intent due to locally relevant yet globally irrelevant paths.

4 SCAIR: Schema-Conditioned Agentic Iterative Reasoning

To address the enterprise-specific limitations identified in Section 3.3, we propose **SCAIR**, a novel KG-RAG method built on a **training-free hybrid agentic paradigm** (Figure 2). Unlike existing approaches that adopt either a plan-and-execute strategy or a ReAct-style iterative exploration (Section 2.2), SCAIR integrates structured planning with controlled iterative reasoning within a unified framework.

SCAIR is guided by three design principles: (i) schema-conditioned planning to provide structural priors, (ii) schema-aware iterative reasoning to ensure valid and focused multi-hop traversal, and (iii) controlled topic entity propagation to balance exploration and exploitation during search.

Schema-Conditioned Planning. Before iterative reasoning (i.e. the ReAct loop), SCAIR performs a lightweight planning stage to introduce structural priors. Specifically, we generate schema-consistent relation paths based on entity types and relation definitions to restrict traversal to valid relation compositions and avoid expansion through irrelevant high-degree nodes. In parallel, the input question is decomposed into depth-aligned subquestions that specify the information required at each reasoning hop. Unlike strict plan-and-execute systems, this stage does not commit to a single fixed plan, but instead provides structural guidance that constrains subsequent exploration.

Schema-Aware Iterative Reasoning. The core of SCAIR follows an iterative agentic loop. At each depth, candidate relations connected to the current topic entities are retrieved, filtered using schema constraints, and scored conditioned on the corresponding subquestion. The selected relations are then expanded to candidate entities, which are further scored and pruned based on the accumulated reasoning context. Traversal is thus guided jointly by semantic relevance and schema validity, preventing structurally invalid paths and reducing search explosion in dense enterprise graphs. The process terminates once sufficient evidence is collected or the maximum reasoning depth is reached.

Exploitation-Exploration Tradeoff. SCAIR balances exploration and exploitation through controlled topic entity propagation. Rather than replacing the topic entity set at each depth, the algorithm maintains a union of previously discovered entities

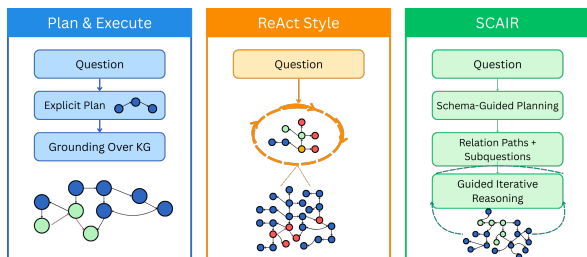


Figure 2: Overview of Plan-and-Execute, ReAct and SCAIR.

and newly expanded ones. This preserves earlier reasoning anchors (exploitation) while enabling deeper traversal into new regions of the graph (exploration). In contrast, ReAct-style methods typically update the working state using only newly expanded entities, which can prematurely discard earlier topic entities and limit the opportunity to explore alternative relations connected to them.

5 Experiments and Results

5.1 Experimental Settings

All methods are evaluated on the enterprise CMDB benchmark. We compare SCAIR with representative agentic KG-RAG paradigms: plan-and-execute methods (RoG, G-Retriever) and ReAct-style iterative methods (ToG, PoG), covering both training-based and training-free strategies. Baseline model configurations and hyperparameters follow the settings reported in the original papers to ensure fair comparison. Detailed configurations are provided in Table 3 in the Appendix D. Training-based models are fine-tuned on two NVIDIA A100 GPUs, while training-free methods are evaluated via OpenAI API.

5.2 Results

Overall Performance. Table 1 reports the performance of baseline KG-RAG methods and SCAIR on the CMDB benchmark. Overall, baseline performance remains limited across all metrics. Among the baselines, training-based methods perform better than training-free approaches. In particular, G-Retriever achieves the strongest aggregate results among baselines (25.27 Accuracy, 24.28 F1), while RoG attains the highest Hits@Any (38.49) but with lower precision and recall. Training-free methods (ToG and PoG) consistently underperform across metrics. Importantly, SCAIR outperforms all baselines across evaluation metrics. These results indicate that while existing KG-RAG paradigms strug-

gle under enterprise KG conditions, aligning reasoning with schema structure, as done in SCAIR, leads to consistent and significant gains.

Performance Across Query Types. To better understand the source of the performance gains, we analyze accuracy across query types (Figure 3). SCAIR consistently outperforms training-free ReAct-style methods (ToG and PoG) across nearly all categories, with especially large improvements on multi-hop (2p, 3p), intersection (2i), and complex queries. These categories are particularly sensitive to uncontrolled traversal and semantic drift, where ReAct-based exploration often fails to maintain structural validity.

Compared to training-based approaches (RoG and G-Retriever), SCAIR achieves competitive or superior performance across most query types. While G-Retriever performs strongly on simpler path queries (e.g., 1p), its accuracy drops substantially on complex and constraint-heavy queries. This pattern suggests that training-based models tend to internalize reasoning templates prevalent in the training distribution, but struggle to generalize. In contrast, SCAIR maintains high accuracy even in the most challenging *complex* category, indicating stronger robustness to distribution shifts and structurally diverse reasoning patterns. The same trends are reflected in Hits@Any and F1 scores (see Appendix E).

Overall, the improvements are distributed across query types rather than concentrated on isolated patterns, suggesting that the gains arise from more reliable control over traversal and reasoning structure rather than from stronger language modeling or memorization.

Ablation Study We analyze the impact of structural control by selectively disabling schema-aware relation filtering and entity-centric constraints in the proposed agentic KG-RAG framework. As shown in Figure 4, removing these controls leads to a consistent degradation in performance across query categories. Overall, the results indicate that explicit structural constraints play a central role in the robustness of the proposed approach.

6 Discussion

6.1 Implications for Industrial Deployment

Our results challenge the implicit assumption in KGQA literature that improved language modeling automatically translates to deployment readi-

Method	Accuracy	Hits@Any	F1	Precision	Recall
RoG	20.80	38.49	19.09	23.32	20.37
G-Retriever	25.27	36.60	24.28	28.25	24.95
ToG	16.04	20.26	14.65	15.62	15.07
PoG	16.17	21.41	14.49	15.50	14.60
SCAIR	35.14	47.56	31.72	36.78	32.60

Table 1: Overall performance comparison.

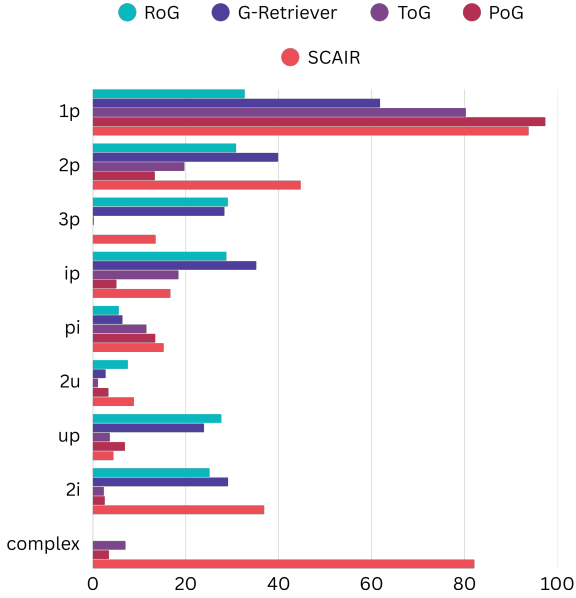


Figure 3: Accuracy comparison across query types. Results for other evaluation metrics are reported in the Appendix E.

ness. In enterprise settings, the "correctness" of an answer is bound by implicit operational constraints (e.g., component availability or compatibility) rather than just semantic relevance. We identify three critical principles for deploying KG-RAG in such dense, schema-governed environments:

1. Structural Validity Must Gate Semantic Relevance. In dense CMDBs, semantic similarity is a noisy proxy for utility. High-degree attribute nodes (e.g., "Status: Broken") often act as "semantic supernodes," causing ReAct-style agents to drift into operationally irrelevant subgraphs. Effective retrieval must therefore be *structure-first*: schema constraints should prune the search space *before* semantic scoring occurs, preventing the "hallucinated validity" observed in standard baselines.

2. Traversal Control is the Primary Bottleneck. Comparing LLaMA-2 and Qwen-2.5 backbones (Appendix E.3) reveals that stronger parametric knowledge improves answer generation but fails to prevent search explosion. The failure mode is architectural, not parametric. For practitioners, this

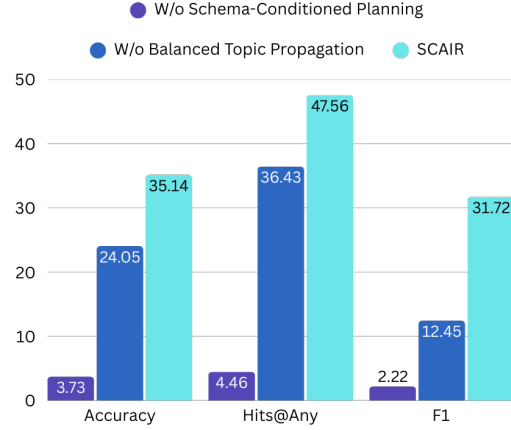


Figure 4: Component Impact on Overall Performance.

implies that investing in lightweight, schema-aware planning yields higher reliability gains than simply scaling the inference backbone.

3. Inference-Time Adaptation Outperforms Re-training. Plan-and-execute methods often overfit to specific query templates seen during training, making them brittle to the frequent schema evolutions typical of enterprise IT. Training-free frameworks like SCAIR, which inject structural priors at inference time, offer a more maintainable deployment strategy. They allow the reasoning engine to adapt to new business rules or schema updates without the cost of continuous supervised fine-tuning.

7 Conclusion and Future Work

This work exposes the fragility of current agentic KG-RAG paradigms when applied to the dense, constraint-heavy reality of industrial KGs. By introducing a realistic CMDB benchmark, we demonstrate that dominant failure modes stem from a misalignment between open-domain retrieval heuristics and enterprise operational logic.

Our proposed framework, SCAIR, mitigates these issues by enforcing structural alignment within the agentic loop. However, the path to fully autonomous enterprise agents requires further evolution. Future research should move beyond prompt-based heuristics by encoding business rules (e.g., disruption risk, valid replacement) as explicit schema annotations or typed constraints. While inference-time control is effective, designing training objectives that reward structural validity rather than only final answer accuracy remains a promising direction for next-generation graph reasoning models.

498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553

References

Erik Arakelyan, Pasquale Minervini, Daniel Daza, Michael Cochez, and Isabelle Augenstein. 2023. Adapting neural link predictors for data-efficient complex query answering. In *NeurIPS*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544. ACL.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *ACL (1)*, pages 1415–1425. The Association for Computer Linguistics.

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

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

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. *Advances in Neural Information Processing Systems*, 37:37665–37691.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *ACL (1)*, pages 260–269. The Association for Computer Linguistics.

Alejandro Germán Frank, Lucas Santos Dalenogare, and Néstor Fabián Ayala. 2019. Industry 4.0 technologies: Implementation patterns in manufacturing companies. *International journal of production economics*, 210:15–26.

Irlán Grangel-González, Felix Lösch, and Anees ul Mehdi. 2020. Knowledge graphs for efficient integration and access of manufacturing data. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 93–100. IEEE.

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. In *NeurIPS*.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. 2021. Knowledge graphs. *ACM Computing Surveys (Csur)*, 54(4):1–37.

Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. 2014. Industry 4.0. *Business & information systems engineering*, 6(4):239–242.

Franz Georg Listl, Daniel Dittler, Gary Hildebrandt, Valentin Stegmaier, Nasser Jazdi, and Michael Weyrich. 2024. Knowledge graphs in the digital twin: A systematic literature review about the combination of semantic technologies and simulation in industrial automation. *IEEE Access*. 554
555
556
557
558
559

Yushan Liu, Bailan He, Marcel Hildebrandt, Maximilian Buchner, Daniela Inzko, Roger Wernert, Emanuel Weigel, Dagmar Beyer, Martin Berbalk, and Volker Tresp. 2023. A knowledge graph perspective on supply chain resilience. In *D2R2*, volume 3401 of *CEUR Workshop Proceedings*. CEUR-WS.org. 560
561
562
563
564
565

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. 566
567
568
569
570
571

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jipu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599. 572
573
574
575
576

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921*. 577
578
579
580

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Trans. Assoc. Comput. Linguistics*, 2:377–392. 581
582
583
584

Wilma Johanna Schmidt, Irlán Grangel-González, Tobias Huschle, Lena Wagner, Evgeny Kharlamov, and Adrian Paschke. 2025a. Llm-supported mapping generation for semantic manufacturing treasure hunting. In *ESWC (2)*, volume 15719 of *Lecture Notes in Computer Science*, pages 84–101. Springer. 585
586
587
588
589
590

Wilma Johanna Schmidt, Irlan Grangel-González, Tobias Huschle, Lena Wagner, Evgeny Kharlamov, and Adrian Paschke. 2025b. Myam: Llm-supported mapping generation for semantic manufacturing retrieval. In *European Semantic Web Conference*, pages 135–140. Springer. 591
592
593
594
595
596

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *ICLR*. OpenReview.net. 597
598
599
600
601
602

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *NAACL-HLT*, pages 641–651. Association for Computational Linguistics. 603
604
605
606

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *ACL* 607
608
609
610

611 (1), pages 2609–2634. Association for Computational
612 Linguistics.

613 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak
614 Shafraan, Karthik R. Narasimhan, and Yuan Cao. 2023.
615 React: Synergizing reasoning and acting in language
616 models. In *ICLR*. OpenReview.net.

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

622 Qinggang Zhang, Shengyuan Chen, Yuanchen Bei,
623 Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong,
624 Hao Chen, Yi Chang, and Xiao Huang. 2025. A
625 survey of graph retrieval-augmented generation for
626 customized large language models. *arXiv preprint*
627 *arXiv:2501.13958*.

628 Dongzhuoran Zhou, Yuqicheng Zhu, Yuan He, Jiaoyan
629 Chen, Evgeny Kharlamov, and Steffen Staab. 2025a.
630 Evaluating knowledge graph based retrieval aug-
631 mented generation methods under knowledge incom-
632 pleteness. *arXiv preprint arXiv:2504.05163*.

633 Dongzhuoran Zhou, Yuqicheng Zhu, Xi Xia Wang,
634 Hongkuan Zhou, Jiaoyan Chen, Steffen Staab, Yuan
635 He, and Evgeny Kharlamov. 2025b. Gr-agent: Adap-
636 tive graph reasoning agent under incomplete knowl-
637 edge. *arXiv preprint arXiv:2512.14766*.

638 Dongzhuoran Zhou, Yuqicheng Zhu, Xi Xia Wang,
639 Hongkuan Zhou, Yuan He, Jiaoyan Chen, Steffen
640 Staab, and Evgeny Kharlamov. 2025c. What breaks
641 knowledge graph based rag? empirical insights
642 into reasoning under incomplete knowledge. *CoRR*,
643 abs/2508.08344.

644 Xiangrong Zhu, Yuexiang Xie, Yi Liu, Yaliang Li,
645 and Wei Hu. 2025. Knowledge graph-guided
646 retrieval augmented generation. *arXiv preprint*
647 *arXiv:2502.06864*.

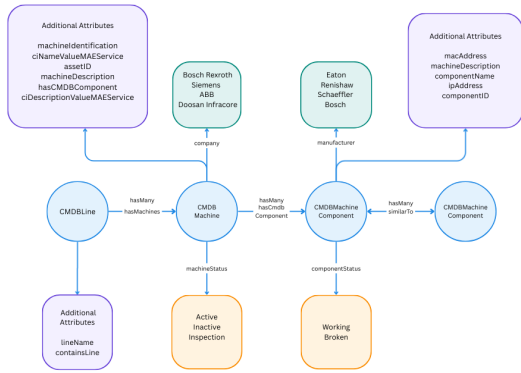


Figure 5: High-level schema overview of the enterprise CMDB KG. The figure illustrates the core entity types (lines, machines, components), their relations, and selected operational attributes introduced to support enterprise reasoning tasks.

A Benchmark construction details

A.1 CMDB-KG Construction

KG Enhancement. To support realistic reasoning patterns, we extend the original CMDB KG with a small set of auxiliary relations commonly required in operational scenarios, such as component status, manufacturer information, and functional similarity between components. These extensions do not alter the core semantics of the CMDB but enable queries related to diagnostics, filtering, and component replacement that are central to enterprise use cases. (Figure 5) illustrates the high-level schema structure of the CMDB KG which supports complex diagnostic and replacement queries.

A.2 Question-Answer Pair Construction.

Rather than collecting natural language questions from crowd workers or logs, we construct the benchmark using a template-based question generation framework grounded in the CMDB schema. Each question template corresponds to a predefined reasoning pattern (illustrated in Figure 7), including multi-hop projection, logical conjunction, disjunction, and domain-specific compositional reasoning. The benchmark covers nine query categories: single-hop projection (1p), multi-hop projection (2p, 3p), intersection (2i), path-intersection hybrids (ip, pi), union (2u, up), and domain-specific complex queries involving operational constraints.

Templates are defined over the CMDB schema and specify both the underlying graph traversal pattern and a natural language realization. During

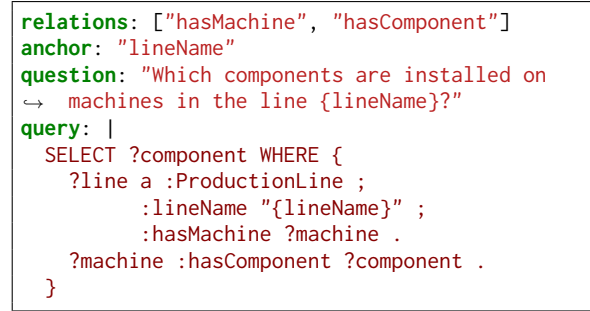


Figure 6: Example of a template-based question specification and its executable graph query used for benchmark construction.

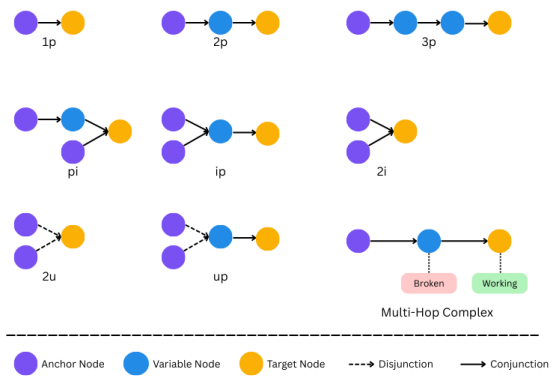


Figure 7: Illustrative examples of structured reasoning templates used in this work. The visualization style and template taxonomy are inspired by (Arakelyan et al., 2023)

instantiation, template placeholders are grounded with concrete entities from the CMDB KG, and the resulting queries are executed against the KG to obtain gold-standard answers. An example of a template specification and its corresponding executable query is shown in Figure 6.

In addition to standard compositional patterns, the benchmark includes domain-specific multi-hop queries that reflect real industrial scenarios. For example, replacement queries require identifying functionally similar components that satisfy operational constraints, such as being currently functional while the original component is broken. These queries combine topological traversal with attribute-based filtering and implicit business logic, making them particularly challenging for KG-RAG systems.

As shown in Table 2, the benchmark contains 19,080 questions in total. While single-hop (1p) queries dominate in absolute number due to large entity coverage, the benchmark includes 7,080 com-

Table 2: Distribution of query types and instantiated question instances.

Type	#Temp.	#Inst.
1p	1	12,000
2p	22	1,690
3p	14	930
2i	27	1,497
ip	14	590
pi	18	1,202
2u	12	516
up	12	577
complex	1	78
Complex (excl. 1p)	–	7,080
Total	–	19,080

positional queries (excluding 1p), spanning multi-hop, intersection, union, and domain-specific complex patterns. This distribution reflects realistic enterprise workloads, where simple lookups are frequent but structurally complex queries are critical for diagnostic and operational tasks.

B Case Study of Existing Methods

We report representative failure cases observed for ReAct style ToG on the enterprise benchmark. Each case includes (i) a real benchmark question, (ii) the intended KG reasoning pattern, and (iii) a concrete failure symptom consistent with the failure modes.

B.1 Search Explosion

Question (complex). *Which working components can replace broken components installed on machines in production line W509-6?*

Intended KG reasoning. Line W509-6 $\xrightarrow{\text{hasMachine}}$ machines $\xrightarrow{\text{hasComponent}}$ components $\xrightarrow{\text{componentStatus}}$ filter broken $\xrightarrow{\text{similarTo}}$ candidates $\xrightarrow{\text{componentStatus}}$ filter working. Additionally, candidates installed on idle machines are preferred to avoid disrupting other lines.

Observed failure symptom. ReAct-style exploration frequently expands via high-degree attribute relations before committing to similarTo. A typical trajectory is:

- Step 1:** retrieve machines for W509-6 (hasMachine)
- Step 2:** retrieve components (hasComponent)
- Step 3:** expand via manufacturer or componentStatus (hub attribute)
- Step 4:** retrieve many unrelated “working” components sharing the same attribute value

This produces large candidate sets dominated by generic attribute matches (e.g., many components linked to working), exhausting the beam budget and preventing exploration along the intended similarTo relation.

Example incorrect output. Predicted answer sets often contain many unrelated working components (high Hits@Any but low precision), e.g., {P-E11-26838, P-E11-27317, P-E11-31641, ...}, where the returned components share an attribute hub (e.g., working) but are not valid replacements for the broken components in W509-6.

B.2 Schema-Agnostic Reasoning

Question (3p-style). *Which IP addresses are assigned to components installed on machines in production line W509-6?*

Intended KG reasoning. ProductionLine $\xrightarrow{\text{hasMachine}}$ Machine $\xrightarrow{\text{hasComponent}}$ Component $\xrightarrow{\text{ipAddress}}$ IP.

Observed incorrect traversal. Instead of expanding toward ipAddress, the agent may select a semantically plausible relation such as manufacturer:

Machine $\xrightarrow{\text{manufacturer}}$ BoschX $\xrightarrow{\text{manufacturer}^{-1}}$ Machine.

Because network configuration is often associated with vendors in real-world settings, this transition appears relevant at the language level. However, it expands through a high-degree manufacturer node and retrieves machines unrelated to the queried production line.

B.3 Semantic Drift Under Increasing Reasoning Depth

Question (3p-style). *Which production lines contain machines that use components manufactured by Siemens?*

Intended KG reasoning. Manufacturer(Siemens) $\xrightarrow{\text{manufacturer}^{-1}}$ Component $\xrightarrow{\text{hasComponent}^{-1}}$ Machine $\xrightarrow{\text{hasMachine}^{-1}}$ ProductionLine.

Observed drift pattern. After reaching Component nodes, the agent may select a semantically plausible but task-irrelevant relation such as similarTo:

Component $\xrightarrow{\text{similarTo}}$ Component $\xrightarrow{\text{hasComponent}^{-1}}$ Machine.

Because similarity is often associated with compatibility, this transition appears reasonable at the language level. However, it expands the search to components not manufactured by Siemens, thereby violating the original constraint.

Effect. Although each hop is schema-valid, the reasoning progressively drifts from the manufacturer constraint. As depth increases, locally plausible transitions accumulate, leading to production lines unrelated to Siemens-manufactured components. This illustrates semantic drift in multi-hop enterprise reasoning.

Summary. These cases instantiate the failure modes discussed in Section 3.3: uncontrolled expansion via high-degree hubs, schema-agnostic relation selection, and progressive semantic drift under increasing reasoning depth.

C More Details of SCAIR

Algorithm 1 provides the full pseudocode of SCAIR. In the following sections, we also detail implementation parameters, prompting strategy, and search control mechanisms used in our experiments.

Search Control Mechanisms. To stabilize traversal in dense graphs, we apply three controls: (i) *subsampling* when relation expansion yields more than 20 entity candidates, (ii) *cycle prevention* by tracking visited (entity, relation) expansions and discarding repeats, and (iii) a *half-stop* policy: if no valid entity remains at depth t , we skip expansion and continue with the previous topic state at depth $t+1$.

C.1 Reasoning Parameters and Execution Setup

SCAIR is implemented using GPT-4.1-mini via API-based inference. We adopt a multi-turn chat format with explicit reasoning instructions.

Decoding parameters are set to temperature = 0.4 during exploration steps (relation and entity scoring) to allow controlled diversity, and temperature = 0.0 during reasoning and final answer generation to ensure deterministic outputs.

The maximum reasoning depth is set to $d = 4$, and beam width to $w = 6$. When entity expansion yields more than 20 candidates, we apply subsampling and retain the top 8 candidates per relation (pre-pruning), before beam pruning.

All questions are processed sequentially due to the multi-step API interaction. API retries are enabled to mitigate transient failures.

The choice of beam width $w = 6$ reflects a trade-off between search coverage and stability in dense enterprise graphs. Smaller beam widths restrict exploration and reduce recall, while larger values increase noise and semantic drift. Empirical sensitivity analysis over beam width is provided in Appendix E.2, where we show that $w = 6$ yields the best balance between F1 and computational efficiency.

C.2 Prompting Design

SCAIR operates entirely at inference time using structured prompts that guide schema-conditioned planning, iterative traversal, and answer verification. To ensure reproducibility, we provide the core prompt templates used in each stage below.

Relation Path Generation. Given the enterprise KG schema and the input question, the model first generates valid relation paths that are consistent with the ontology. These paths restrict traversal to schema-valid compositions and prevent structurally invalid expansions.

Relation Path Generation Prompt

You are a reasoning assistant over an industrial Knowledge Graph.

Given a question and the KG schema below, generate a valid relation path required to answer the question. Use only relations present in the schema. Assume the topic entity is already known.

KG Structure: CMDBLLine -hasMachines-> CMDBMachine -hasCmdbComponent-> CMDBMachineComponent -similarTo-> CMDBMachineComponent

Relations: - CMDBLLine: lineName, hasMachines - CMDBMachine: machineIdentification, machineDescription, hasCmdbComponent, company, machineStatus - CMDBMachineComponent: componentId, componentName, macAddress, ipAddress, manufacturer, componentStatus, similarTo

Output a numbered list of relation paths.

Q: <Question>

Subquestion Decomposition. Conditioned on the generated relation path, the model decomposes

854 the original question into depth-aligned subques-
855 tions. Each subquestion corresponds to a single
856 hop in the relation path.

Subquestion Decomposition Prompt

You are given: - A question - A relation path
- The topic entity
Decompose the question into small subques-
tions, each corresponding to one relation in
the path.
Rules: - Each subquestion must be self-
contained. - Highlight the required relation
using curly braces. - Follow the order of
the relation path exactly. - Output only a
numbered list.
Q: <Question> Relation Path: <Path> Topic
Entity: <Entity>

857

858 **Relation Scoring.** At each depth, candidate rela-
859 tions connected to the current topic entities are
860 scored conditioned on the current subquestion.
861 Schema-consistent relations are prioritized.

Relation Scoring Prompt

Given: - The question - The current sub-
question - Candidate relations - The relation
path
Assign scores to relations such that: - Re-
lations highlighted in curly braces receive
the highest score. - Scores must sum to 1. -
Only consider the current subquestion (no
future hops).
Q: <Question> Sub Question: <Subques-
tion> Topic Entity: <Entity> Candidate Re-
lations: <Relations> Relation Path: <Path>

862

863 **Entity Scoring.** For each selected relation, candi-
864 date entities are scored based on their usefulness for
865 answering the current subquestion. Binary scoring
866 (0 or 1) is used to reduce ambiguity.

Entity Scoring Prompt

You are given: - The question - The current
subquestion - The current relation - Candi-
date entities - The relation path
Score each entity strictly for this hop: - 1 if
critical for the subquestion. - 0 otherwise. -
Do not hallucinate entities.
Q: <Question> Sub Question: <Subques-
tion> Relation: <Relation> Entities: <Enti-
ties> Relation Path: <Path>

867

868 **Sufficiency Evaluation.** Before answer gener-
869 ation, the model evaluates whether the retrieved
870 triples are sufficient to answer the question.

Sufficiency Check Prompt

Given a question and retrieved knowledge
triples, determine whether the information
is sufficient to answer.
Output: - Yes followed by the final answer
in curly braces, or - No with a brief expla-
nation.
Q: <Question> Knowledge Triples:
<Triples>

871

872 **Answer Generation.** If sufficient evidence is
873 available, the model generates the final answer
874 grounded strictly in the retrieved triples.

Answer Generation Prompt

Given the question and retrieved knowl-
edge triples, generate the final answer. The
answer entities must be wrapped in curly
braces. Do not hallucinate additional enti-
ties.
Q: <Question> Knowledge Triples:
<Triples>

D Baseline Implementation Details

875

876 All baseline implementations follow the original
877 architectural designs and training procedures de-
878 scribed in their respective publications. We adopt
879 the reported hyperparameters directly (see Table 3).
880 If a parameter is unspecified, we use default values
881 from the official codebases. 882

883 **Hardware.** Training-based baselines were fine-
884 tuned on two NVIDIA A100 GPUs (80GB).
885 Inference-only methods were executed sequentially
886 via API due to their multi-step reasoning structure.

Table 3: Key configuration settings for baseline models.

Model	Key Settings
RoG	Backbone: LLaMA-2-7B-Chat (LoRA $r = 8$); Epochs: 3; Batch: 2; LR: $2e^{-5}$; Optimizer: AdamW; Cosine schedule; bf16
G-Retriever	Backbone: LLaMA-2-7B; GNN layers: 4; Hidden: 1024; Epochs: 10 (early stop=2); LR: $1e^{-5}$; fp16
ToG	Model: GPT-4.1-mini; Max depth: 3; Beam: 3; Entity subsampling: 20→5; Temp: 0.4 (explore), 0.0 (reason)
PoG	Model: GPT-4.1-mini; Planning depth: 4; Relation pruning enabled; Temp: 0.3 (planning/reasoning)

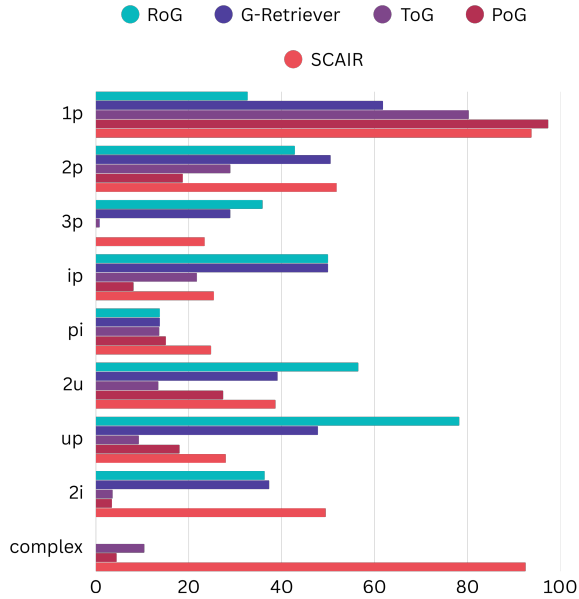


Figure 8: Hits@Any per query type for all evaluated methods.

All implementations use PyTorch with mixed precision where applicable.

E More Experimental Results

E.1 Hits@Any and F1 Across Query Types

This appendix reports Hits@Any and F1-score across query types for all evaluated methods. These results complement the accuracy-based analysis presented in the main text (Section 5.2) and provide additional insight into partial answer overlap and retrieval robustness.

Consistent with the accuracy trends discussed in the main body, SCAIR maintains strong performance across multi-hop, intersection, and complex query categories. The improvements are reflected not only in exact-match accuracy but also in overlap-based metrics, indicating more reliable retrieval and reasoning behavior across structurally diverse query types.

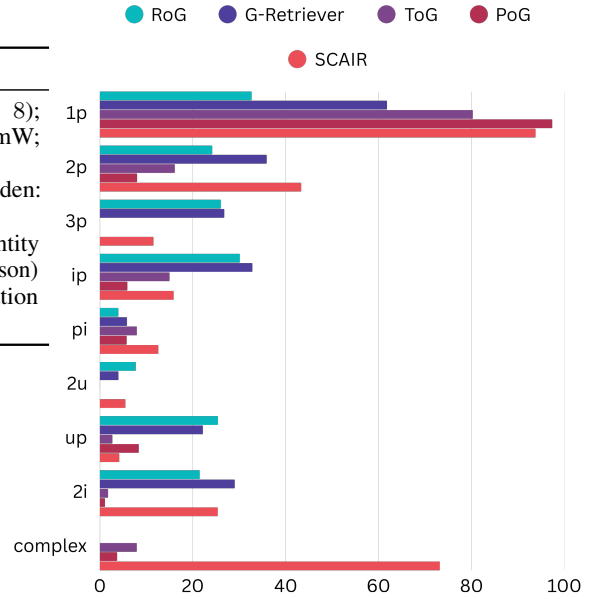


Figure 9: F1-score per query type for all evaluated methods.

E.2 Sensitivity to Beam Width

To analyze the robustness of SCAIR with respect to search hyperparameters, we evaluate performance under varying beam width w . We vary $w \in \{3, 4, 6\}$ while keeping the maximum depth fixed at $d = 4$.

As shown in Figure 10, increasing the beam width from 3 to 6 consistently improves Accuracy, Hits@Any, and F1. The improvement from $w = 3$ to $w = 4$ is moderate, while the gain from $w = 4$ to $w = 6$ is more pronounced, particularly for F1 and Hits@Any. This indicates that broader exploration improves coverage of relevant relations and entities in dense enterprise graphs.

However, the gains diminish as w increases, suggesting that simply expanding the beam cannot indefinitely improve performance. Larger beams introduce more structurally irrelevant candidates, increasing computational cost without proportional accuracy gains.

We therefore adopt $w = 6$ as a balanced setting that provides strong performance while maintaining controlled exploration.

E.3 Effect of Stronger Training-Based Backbones

To investigate whether stronger language backbones can resolve the structural limitations observed in Plan&Execute approaches, we replace the original LLaMA backbone in RoG with Qwen

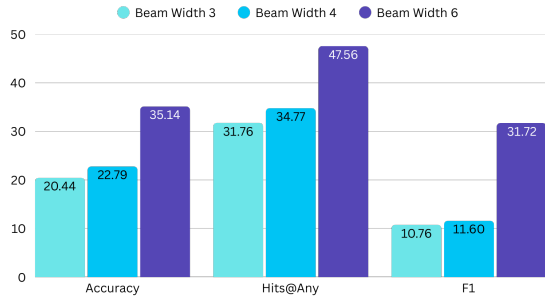


Figure 10: Performance of SCAIR under different beam widths.

and retrain under identical settings.

In a controlled setting where gold-standard intermediate triples are provided, Qwen demonstrates stronger local reasoning ability than LLaMA, correctly answering complex compositional queries that LLaMA fails to resolve. This suggests that Qwen has improved conditional reasoning capacity when the relevant evidence is explicitly given.

However, when evaluated in the full end-to-end KG-RAG pipeline, replacing LLaMA with Qwen does not yield meaningful improvements in overall benchmark performance (Figure 11). Performance on multi-hop and complex query types remains limited.

This discrepancy indicates that the primary bottleneck is not answer generation or conditional reasoning given correct evidence, but rather the structural search process that retrieves the evidence. Stronger LLM backbones improve reasoning over provided triples, but they do not mitigate uncontrolled traversal, schema-agnostic exploration, or search explosion in dense enterprise graphs.

These results reinforce our central claim: structural control over graph traversal is more critical than backbone strength for enterprise KG-RAG robustness.

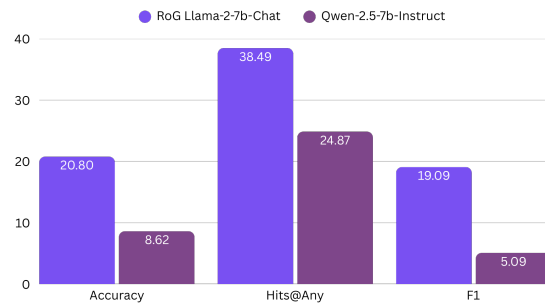


Figure 11: Comparison of RoG with LLaMA and Qwen backbones.

Algorithm 1 Schema-Aware and Question-Guided Method

Require: Question Q , initial topic entities E_0 , beam width w , maximum depth d , KG \mathcal{G}

Ensure: Answer set \mathcal{P}_Q

```
1:  $\mathcal{P} \leftarrow \text{GENERATERELATIONPATHS}(Q, \mathcal{G})$   $\triangleright$ 
   schema-guided relation paths
2:  $\{q_1, q_2, \dots, q_d\} \leftarrow$ 
    $\text{DECOMPOSEQUESTION}(Q, \mathcal{P})$   $\triangleright$ 
   depth-aligned subquestions
3:  $E_0 \leftarrow$  given topic entities
4: for  $t = 0$  to  $d - 1$  do
5:   Relation Search:
6:    $\mathcal{R}_t \leftarrow \text{RETRIEVERELATIONS}(E_t, \mathcal{G})$ 
7:    $\mathcal{R}_t \leftarrow \text{FILTERRELATIONS}(\mathcal{R}_t, q_{t+1}, \mathcal{P})$ 
8:   Score  $\mathcal{R}_t$  using  $\text{LLM}(Q, q_{t+1}, E_t)$ 
9:   Refine scores using relation paths  $\mathcal{P}$ 
10:   $\mathcal{R}_t \leftarrow \text{TOPK}(\mathcal{R}_t, w)$ 
11:  Entity Search:
12:   $E'_t \leftarrow \emptyset$ 
13:  for all  $r \in \mathcal{R}_t$  do
14:     $C \leftarrow \text{RETRIEVEENTITIES}(E_t, r, \mathcal{G})$ 
15:    Score  $C$  using  $\text{LLM}(Q, q_{t+1}, r, C, \mathcal{P})$ 
16:    Update search history with  $(E_t, r, C)$ 
17:     $E'_t \leftarrow E'_t \cup C$ 
18:  end for
19:  if  $E'_t = \emptyset$  then
20:    Half-stop and skip to next depth
21:  end if
22:   $E'_t \leftarrow \text{TOPK}(E'_t, w)$ 
23:  Reasoning Check:
24:  if  $\text{SUFFICIENTEVIDENCE}(Q, E'_t)$  then
25:    return  $\text{GENERATEANSWER}(Q, E'_t)$ 
26:  end if
27:   $E_{t+1} \leftarrow E_t \cup E'_t$   $\triangleright$  balanced topic entity
   propagation
28: end for
29: return  $\text{BESTCANDIDATE}(Q, E_d)$   $\triangleright$  final
   half-stop
```
