HiFiRepoQA: DAG-Structured Multi-Agent Framework for Repository-Level Q&A

Anonymous ACL submission

Abstract

Understanding and using large software repositories is increasingly difficult as modern projects grow in scale, span multiple programming languages, and evolve rapidly. Developers frequently raise repository-level questions that require integrating information scattered across code files, documentations, and configurations. Existing approaches struggle with challenges such as linear reasoning strategies that cannot model partially ordered knowledge dependencies and weak handling of noisy context. To address these challenges, we propose **HiFiRepoQA**, a repository-level question answering framework based on structured task decomposition and controlled information acquisition. It models repository-level questions in a directed acyclic graph (DAG), enabling parallel execution of independent goals while isolating irrelevant context. To facilitate realistic evaluation, we further construct **HiFiRepoQA Bench**, a high-fidelity benchmark with 526 real user questions collected from GitHub repositories. Experimental results show its advantages over open-source and commercial baselines. We further evaluate its usefulness by submitting generated answers to GitHub, where 9 questions receive positive feedback or are marked as closed due to our response.

1 Introduction

With the rapid growth of large-scale software systems, modern projects often span millions of lines across multiple languages and modules (Zhang et al., 2023; Fan et al., 2023). They evolve into complex, interconnected systems that are difficult for developers, contributors, and users to comprehend. This complexity hinders onboarding, maintenance, and collaborative contribution, underscoring a pressing need for intelligent tooling to support repository-level information acquisition (Guo et al., 2025; De Martino et al., 2025).

On the other hand, Question Answering (Q&A) systems have long been a cornerstone of artificial intelligence, enabling users to obtain precise information through natural language interaction. In general domains, such as open-domain knowledge or customer support, Q&A technologies significantly enhance information access efficiency and user experience by retrieving and synthesizing facts from structured or unstructured corpora.

However, when applied to the specialized context of software repository Q&A, general paradigms face profound and distinctive challenges. First, real-world developer questions are highly specialized and often require knowledge beyond the general scope of pretrained large language models (LLMs), making direct LLM application prone to errors or hallucinations. Second, these questions are inherently unstructured: they are frequently ambiguous, combine natural language with code snippets or error traces, and conceal multiple intertwined intents, which makes accurate question decomposition exceptionally difficult. Finally, the information needed to answer a question is typically scattered across numerous files and modules, posing a fundamental retrieval dilemma-knowing what to retrieve, when to retrieve, and how to integrate fragmented evidence into a correct answer (Jimenez et al., 2024).

Inspired by hierarchical human cognitive decision-making mechanisms, we propose HiFiRepoQA, a DAG-specialized multi-agent framework that decomposes complex repository-level questions into a directed acyclic graph (DAG) of interdependent subtasks and performs coordinated planning, retrieval, localized analysis, and verification-driven answer generation. By explicitly modeling subtask dependencies, DAG enables a structured execution of repository-level reasoning. As illustrated in Figure 1, the DAG enables independent subtasks to be executed in parallel, isolating different reasoning paths and avoiding unnecessary context mixing. This parallelism not only improves efficiency but

also confines redundant and irrelevant context to local execution branches, thereby enhancing overall robustness. Furthermore, by explicitly modeling dependencies between subtasks, the DAG ensures that when intermediate conclusions are aggregated, each is based solely on its relevant evidence, minimizing contextual interference during answer synthesis and leading to more accurate and interpretable final answers.

To better capture real-world repository usage scenarios, we construct a high-quality repository question answering benchmark `HiFiRepoQA Bench` strictly sourced from real repository scenarios and authentic user questions. Specifically, our benchmark consists of 526 user questions collected from GitHub, spanning 170 repositories across 10 programming languages. Experimental results show that our approach improves LLM-as-Judge scores by 8%-15% and rubric based answer correctness (RAC) by 28%-67% over vanilla LLMs and retrieval-based baselines, and yields additional gains of 1%-4% (LLM-as-Judge) and 2%-10% (RAC) even compared to commercial systems such as Cline and Cursor. We further evaluate the usefulness of `HiFiRepoQA` by submitting generated answers to GitHub, where 9 questions receive positive feedback or are marked as closed due to our response.

This paper makes the following contributions:

- `HiFiRepoQA`, a multi-agent framework that decomposes complex repository-level questions into a directed acyclic graph of interdependent subtasks and performs coordinated planning, retrieval, localized analysis, and verification-driven answer generation.
- `HiFiRepoQA Bench`, a benchmark for repository Q&A with 526 questions and answers from real-world software development context.
- Rubric-based Answer Correctness, an evaluation metric for repository Q&A that explicitly measures whether a predicted answer covers the required content points specified by scoring rubrics extracted from reference answers.
- Extensive experiments, demonstrating superior performance over open-source and commercial baselines.
- Open public code and benchmark, facilitating further research in this task. `https://github.com/HiFiRepoQA/HiFiRepoQA`

2 Related Work

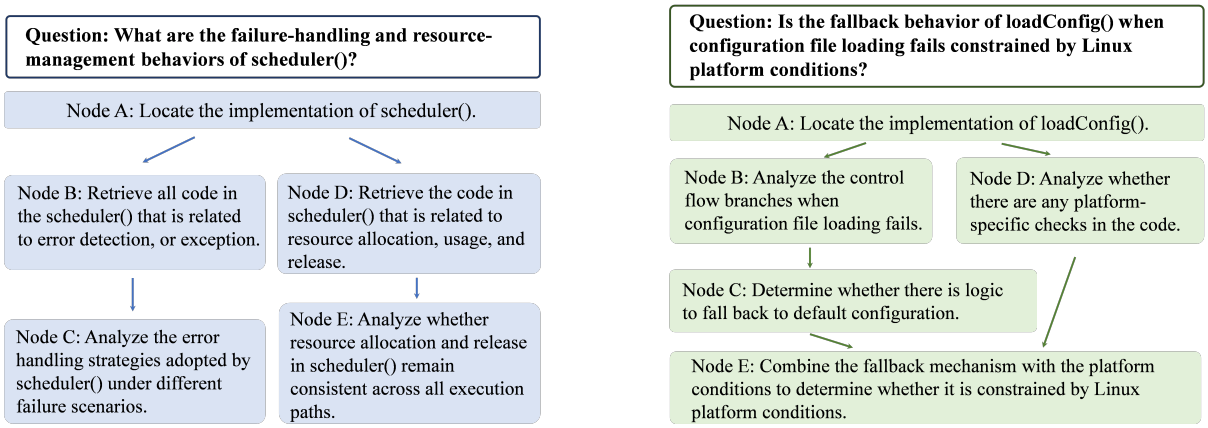
Code Question Answering. Several code question-answering datasets have been proposed in recent years, providing valuable resources for related research (Table 1). ProCQA (Li et al., 2024) offers a large-scale collection of snippet-level code Q&A pairs, primarily targeting local code understanding, but it does not cover more complex repository-level Q&A scenarios. CodeRepoQA (Hu et al., 2025) collects over 580,000 GitHub issues from 70 repositories across ten widely-used programming languages. Although large-scale, it may include non-Q&A content, such as feature requests or bug-fix discussions. CoReQA (Chen et al., 2025) improves readability and structural consistency by rewriting questions; SpyderCodeQA (Strich et al., 2024) is constructed through manual annotation by human experts at the repository level; SWE-QA (Peng et al., 2025) relies on templated question formulations and LLM-based generation; yet, these processes may partially reduce the natural ambiguity and complexity inherent in real-world developer inquiries.

Dataset	Size	Repo Level	High Fidelity	Quality Curated
ProCQA	500w	✗	✓	✗
CodeQA	19w	✗	✗	✗
CoReQA	1k5	✓	✗	✓
CodeRepoQA	58w	✓	✓	✗
SpyderCodeQA	325	✓	✗	✓
SWE-QA	576	✓	✗	✓
`HiFiRepoQA Bench`	526	✓	✓	✓

Table 1: Comparison Between `HiFiRepoQA Bench` and Existing Code QA Benchmarks

For snippet-level Code Q&A, models such as CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2020), and instruction-tuned LLMs have shown promising results. At the repository level, RAG-based (Chen et al., 2025) methods and commercial systems such as Cline (Cline, 2025) and Cursor (Cursor, 2025) can answer cross-module questions, but are not specifically designed for repository-level code question answering.

LLM-Based Agents for Program Comprehension. Program comprehension helps users understand existing codebases and has been extensively studied in tasks such as code summarization (Sharma et al., 2021). Recent work has explored LLM-based agents for code summarization, leveraging tool use and iterative reasoning to handle large code contexts (Yang et al., 2025b; Luo et al.,



(a) DAG Example 1.

(b) DAG Example 2.

Figure 1: DAG-based question decomposition with dependency-aware context isolation.

2024; Lomshakov et al., 2024). Unlike summarization, which produces general descriptions of code artifacts, repository Q&A requires addressing semantically specific, user-posed queries by retrieving and integrating dispersed evidence across multiple functions, classes, or files. This setting needs repository-scale retrieval as well as multi-hop reasoning and user-intention alignment, yet existing LLM-based program comprehension frameworks have seen limited exploration in this context.

3 Approach

This section presents `HiFiRepoQA`, a multi-agent framework designed for repository-level code question answering. Unlike previous methods that rely on monolithic reasoning or linear agent pipelines, our approach explicitly decomposes complex questions into a directed acyclic graph (DAG) of subtasks, enabling fine-grained isolation of information, controlled context construction, and more reliable multi-step reasoning. As shown in Figure 2, `HiFiRepoQA` consists of four types of specialized agents: (1) a **Planning Agent** that decomposes questions into a DAG; (2) **Retrieval Agents** that construct and query a repository-level knowledge base; (3) **Analysis Agents** that assess evidence sufficiency and perform localized reasoning over retrieved information; and (4) a **Synthesis Agent** that integrates verified evidence and produces the final answer.

3.1 Planning Agent

The Planning Agent aims to decompose complex repository-level code questions into structured subtasks, which are then delegated to specialized agents. The Planning Agent constructs a DAG, allowing subtasks to be executed according to their dependencies. This formulation enables specialized agents to focus on the local context of each subtask, avoiding redundant contextual coupling and helping to mitigate error propagation across reasoning steps.

3.1.1 DAG Definition

Formally, given a repository-level question Q, the Planning Agent constructs an execution plan represented as a DAG,

$$\mathcal{P} = (V, E), \tag{1}$$

where each vertex $v \in V$ represents a planning node associated with a well-defined local goal, and $E \subseteq V \times V$ is a set of directed dependency edges. An edge $(v_i \rightarrow v_j) \in E$ indicates that the execution of node v_j depends on the intermediate result produced by node v_i.

Each planning node is defined as a tuple

$$v = \big(\tau(v),\ \gamma(v),\ \mathcal{C}(v)\big), \tag{2}$$

where $\tau(v) \in \{\textsc{Retrieval}, \textsc{Analysis}\}$ denotes the node type, indicating whether node v performs repository information acquisition (as shown in Section 3.2) or localized semantic reasoning (as shown in Section 3.3); $\gamma(v)$ specifies the concrete retrieval or analysis goal associated with node v, and $\mathcal{C}(v)$ represents the execution context composed of multiple information sources provided for achieving this goal.

Formally, the execution context associated with node v is defined as

$$\mathcal{C}(v) = \mathcal{O}_{\text{pred}}(v) \cup \mathcal{I}(v), \tag{3}$$

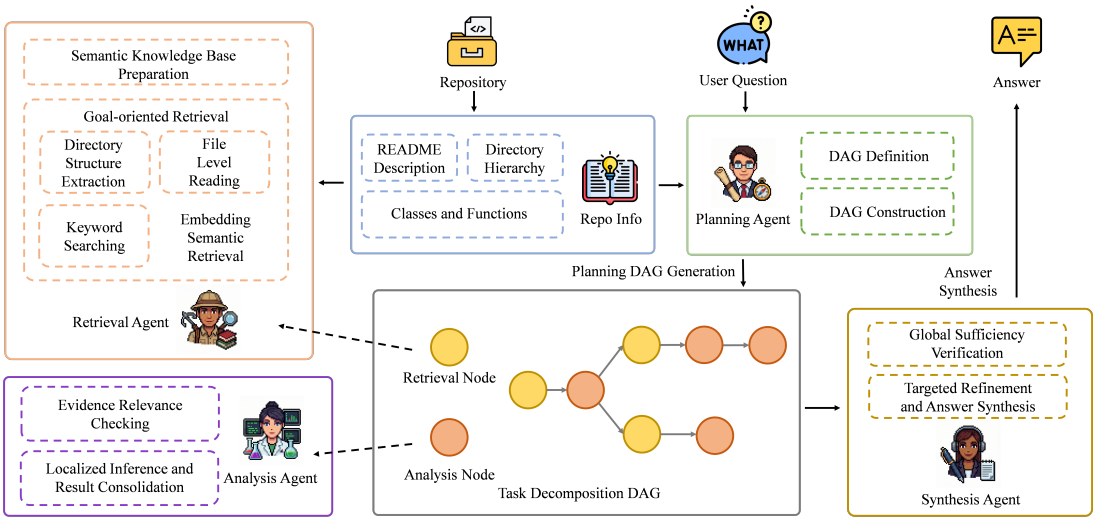


Figure 2: Overview of HiFiRepoQA.

where $\mathcal{O}_{\text{pred}}(v)$ denotes the set of intermediate outputs produced by all predecessor nodes of v, and $\mathcal{I}(v)$ represents the subset of original question information explicitly assigned to node v during planning.

$$\mathcal{I}(v) \subseteq Q, \quad Q = \left(q_{\text{text}}, q_{\text{code}}, q_{\text{output}}, q_{\text{error}}\right). \quad (4)$$

where Q denotes the heterogeneous information sources in the repository-level question, in which q_{text} is the natural language descriptions, q_{code} is the code snippets, q_{output} is the runtime outputs from successful executions, and q_{error} is the error messages or exception traces from failed executions.

This selective assignment prevents irrelevant or misleading information from propagating into downstream reasoning while reducing the input context length for individual agents. For example, nodes concerned with failure diagnosis are usually provided with error traces or runtime outputs, whereas nodes focusing on semantic or functional understanding primarily rely on natural language descriptions and code snippets.

3.1.2 DAG Construction

Building on the above formal definition, the Planning Agent constructs a DAG for a specific question using schema-guided prompting. The produced DAG will serve as the global execution blueprint for multi-agent collaboration.

To provide the LLM with a global view of the repository for DAG planning, the Planning Agent organizes the LLM input with complementary contextual signals, including the question title and body, high-level repository structural and descriptive information (e.g., directory hierarchy and README descriptions).

To ensure structured and controllable planning, the prompt enforces a strict output schema. The model is required to return a JSON-formatted DAG, where each node explicitly specifies its node type $\tau(v)$, local goal $\gamma(v)$, the original question subset $\mathcal{I}(v)$, and dependency relations between nodes.

This schema-constrained prompting guides the model to directly generate a valid DAG rather than a free-form decomposition. In practice, the generated output is automatically validated against the predefined schema. If the output violates the schema constraints, the Planning Agent re-prompts the model until a valid DAG is obtained or the maximum number of attempts is reached.

We also introduce two prompt strategies as follows. The first is feasibility gating, which first determines whether the current question information is sufficient to directly answer, and if so, output an empty plan to avoid over planning. The second is goal conditional atomic planning. It forces each node corresponding to a single goal with a concrete deliverable, and requires dependencies to be explicitly encoded, prohibiting cycles, and encouraging shallow, parallel-friendly graph structures. Detailed prompt for Planning Agent is shown in Appendix A.6.

3.2 Retrieval Agent

The Retrieval Agent is responsible for providing precise and goal aligned repository-level evidence for downstream analysis. Before performing retrieval, the Retrieval Agent first checks whether a

knowledge base has been constructed for the target repository. If not, an offline knowledge base preparation process is automatically triggered before the retrieval begins.

3.2.1 Semantic Knowledge Base Preparation

To align the representation spaces of natural language and code, we construct a knowledge base by summarizing function- and class-level code units and encoding the summaries into vector representations. Specifically, we first perform unified parsing of files in the repository using Tree-sitter (Tree-sitter Contributors, 2024), and treat extracted functions and classes as the basic code units.

Each code unit is represented as

$$c_i = (\text{meta}_i,\ \text{code}_i,\ \text{desc}_i,\ \text{locate}_i), \quad (5)$$

where meta_i denotes metadata such as the unit name, file path, source-span coordinates, and programming language; code_i contains the raw source code of the function or class; desc_i includes associated descriptive information, such as docstrings and parameter specifications; locate_i records the precise location of the code unit, including the file path and the corresponding line number span.

After obtaining the units, we further construct a high-level semantic summary for each c_i and obtain its vector representation via an embedding model. Specifically, we input the meta information, code snippets, and descriptive information into a summarization model, and then send the resulting summary to an embedding:

$$s_i = \text{LLM\_summary}(\text{meta}_i,\ \text{code}_i,\ \text{desc}_i), \quad (6)$$

$$\mathbf{e}_i = \text{Embed}(s_i). \quad (7)$$

The embedding $\mathbf{e}_i$, the meta_i and the locate_i together form a fine-grained semantic representation of the repository.

In addition, we store repository-level structural information in the knowledge base, including the directory tree of folders and files, as well as repository-level global context consisting of README files.

3.2.2 Goal-oriented Retrieval

`HiFiRepoQA` first constructs a set of specialized retrieval tools to support different retrieval needs. These retrieval tools are designed to cover complementary levels of repository understanding, ranging from global structure to fine-grained semantic evidence. Specifically, directory-level and file-level tools provide coarse-grained structural and contextual views of the repository, while keyword-based and embedding-based retrieval support precise localization and semantic matching of relevant code units. Concretely, these tools include:

Directory Structure Extraction, extracting the repository's directory hierarchy to expose its global organization;

File Level Reading, reading code by file path to obtain full context;

Keyword Searching, locating functions, classes, or variables via exact keyword matching;

Embedding Semantic Retrieval, retrieving related implementations using function- or class-level semantic vectors.

For a retrieval node v_i, the Retrieval Agent operates on a localized context consisting of the node goal $\gamma(v)$, the associated goal-conditioned subset of heterogeneous question components $\mathcal{I}(v)$ assigned by the Planning Agent, and the outputs from preceding agents $\mathcal{O}_{\text{pred}}(v)$.

Based on the information it performs a multi-round, tool-augmented retrieval process that iteratively gathers sufficient repository evidence to fulfill the node's goal. Multi-round retrieval refers to an iterative retrieval-verification process. In each round, the agent performs a lightweight coverage check over the retrieved evidence to assess whether it sufficiently supports the fulfillment of the current goal. If not, the agent resets the retrieval tools and the retrieval focus to initiate another retrieval round. This process continues until the accumulated evidence satisfies the node goal or a predefined maximum number of retrieval rounds is reached.

For the candidate code snippets retrieved through the Embedding Semantic Retrieval tool, we further introduce a re-ranking stage with a cross-encoder embedding model (Wang et al., 2021). This two-stage retrieval pipeline enables `HiFiRepoQA` to filter out noisy candidates and prioritize evidence that is most relevant to the node's goal.

3.3 Analysis Agent

The Analysis Agent conducts goal-driven analysis over the repository-level evidence available at the current node.

The Analysis Agent first assesses the relevance of the evidence with respect to the goal of the current node $\gamma(v)$. Specifically, it evaluates whether the evidence explicitly refers to goal-related identifiers (e.g., function or class names), or whether the functionality or behavioral semantics implied

by the evidence align with the intent expressed by the current goal. Evidence deemed irrelevant to $\gamma(v)$ is discarded to prevent noise accumulation and spurious correlations.

The agent further consolidates the analysis results and associated evidence into a structured intermediate output, which encodes the local conclusions and an evidence list inherited from the $\mathcal{O}(v_i)$ when available. This output can be directly consumed by subsequent nodes in the DAG or used as input to the final answer generation stage when the current node has no successors.

3.4 Synthesis Agent

The Synthesis Agent aims to generate the final answer by integrating all related evidence and intermediate conclusions provided by the retrieval and analysis agents.

Specifically, the Synthesis Agent is activated after all agents at the terminal nodes of the DAG have completed their processing. These terminal nodes correspond to Retrieval or Analysis Agents that have no successors in DAG, and thus represent the final reasoning states associated with their respective sub-problems. Upon activation, the Synthesis Agent aggregates the structured outputs produced by these agents and jointly considers them with the original question and repository-level structural information from the knowledge base, including directory hierarchies and global context consisting of README files. Based on this consolidated representation, the agent evaluates whether the collected evidence and intermediate conclusions are sufficient to support a final answer.

If not, the agent will actively trigger a new Retrieval-Analysis round. It traces these deficiencies back to the corresponding sub-problems or repository regions that remain under-explored or weakly justified, and selectively triggers an additional round. In this round, retrieval and analysis are guided by the missing or weakly supported information identified during verification, forming a controlled closed-loop refinement process.

4 Experiment Design

To comprehensively evaluate the performance of HiFiRepoQA, we propose the following three research questions (RQs):

RQ1 (Effectiveness and Efficiency) How effective and efficient is HiFiRepoQA in answering repository-level code questions?

RQ2 (Ablation Study) How does each component of HiFiRepoQA contribute to its overall performance?

RQ3 (Usefulness Evaluation) How effective is HiFiRepoQA in real-world settings as judged by human evaluation and user feedback?

HiFiRepoQA Bench. To address the absence of a high-quality benchmark for realistic repository-level Q&A as discussed in Section 2, we construct a new benchmark HiFiRepoQA Bench consisting of 526 repository-level Q&A pairs collected from 70 actively maintained open-source GitHub repositories. The dataset covers ten mainstream programming languages, including Python, Java, C, C++, JavaScript, TypeScript, Go, C#, Rust, and PHP. The detailed data collection and curation process is described in Appendix A.1.

Baselines. To evaluate the effectiveness of HiFiRepoQA, we compare it with representative baselines spanning direct inference, retrieval-augmented methods, prompting-based reasoning, and commercial programming assistants. Specifically, we include **Vanilla LLM**, which prompts a LLM to answer questions without external support; **RAG** (Lewis et al., 2020), which retrieves relevant code or documentation as additional context; **Sliding-Window RAG (SW-RAG)** (Gao et al., 2023), which further enhances retrieval by traversing repository files with overlapping windows to capture local and cross-file context; and **Chain-of-Thought (CoT)** (Wei et al., 2022), which guides explicit step-by-step reasoning via prompting. In addition, we adopt **Cline** (Cline, 2025)[1] and **Cursor** (Cursor, 2025)[2], which are mature commercial AI-assisted programming systems that have been widely adopted in real-world software development and have millions of active users. Note that Cursor does not support customized LLM APIs and the evaluation is only conducted in terms of its original LLM (Claude-4.5-Sonnet).

Evaluation Metrics. Following existing studies (Strich et al., 2024)(Peng et al., 2025), we adopt LLM-as-Judge, using GPT-4o-mini (OpenAI, 2024) as the judging model, to score each answer along five dimensions: *Correctness (Cor.), Completeness (Com.), Relevance (Rel.), Clarity (Cla.), and Reasoning quality (Rea.)*. We further report the average score (*Avg.*) across these five dimensions

[1] 1.6M active users and 56.6k GitHub stars as of January 2026. https://cline.bot/enterprise.

[2] 64% of Fortune 500 companies have adopted Cursor as of January 2026. https://cursor.com/enterprise.

to provide an overall assessment.

We additionally design a *Rubric-based Answer Correctness (RAC)* metric to more accurately measure the correctness of generated answers. RAC conducts fine-grained evaluation of answers by checking whether predicted answers meet the scoring rules extracted from correct answers. Details are shown in Appendix A.2.

All experiments are conducted under identical experimental settings across different backbone LLMs to ensure fair and comparable results. Detailed experimental configurations are provided in Appendix A.3.

5 Results and Analysis

5.1 Effectiveness and Efficiency (RQ1)

Tables 2–4 summarize the performance of HiFiRepoQA and baseline methods across three backbone LLMs. Results on additional backbone models exhibit consistent trends and are provided in Appendix A.5. Overall, HiFiRepoQA consistently achieves the best performance on both LLM-as-Judge metrics and RAC, demonstrating strong effectiveness and robustness for repository-level question answering.

Compared with vanilla LLMs, prompt-based reasoning (CoT), and retrieval-based baselines (RAG and SW-RAG), HiFiRepoQA consistently achieves substantial performance gains across all backbone models. It improves the average LLM-as-Judge score by 8%-15%, while yielding substantially larger improvements on RAC, ranging from 28% to 67%.

Even when compared with mature commercial AI-assisted programming systems such as Cline and Cursor, HiFiRepoQA remains competitive across most settings. While the margins on the averaged LLM-as-Judge scores are relatively small, ranging from 1% to 4%, HiFiRepoQA consistently achieves higher RAC scores, ranging from 2% to 10%, indicating more accurate answers with fewer hallucinations. This highlights that HiFiRepoQA's design is better aligned with repository-level Q&A, rather than general-purpose code interaction.

We compare representative methods under the same backbone model (Claude-4.5-Sonnet) and report the average time per question in Table 5. While HiFiRepoQA introduces additional inference cost due to explicit question decomposition and node-level retrieval and verification, its inference time (103.9s) remains within a practical range for repository-level Q&A. This overhead primarily stems from multi-agent coordination involving multiple LLM invocations and retrieval steps to progressively identify goal-relevant repository evidence. The embedding construction in HiFiRepoQA is a one-time offline cost amortized over multiple queries, whereas commercial systems such as Cline and Cursor can also have such repository processing costs. Moreover, strategies such as parallel execution of independent DAG nodes or early stopping when sufficient evidence is obtained can further reduce the time. Overall, HiFiRepoQA achieves a reasonable balance between effectiveness and efficiency.

Method	LLM-as-Judge Metrics						RAC
	Cor.	Com.	Rel.	Cla.	Rea.	Avg.	
Vanilla	7.41	7.36	9.35	8.16	7.52	7.96	4.95
↑ Δ *Ours*	21%	25%	3%	7%	21%	15%	50%
CoT	7.48	7.51	9.28	8.34	7.78	8.08	5.32
↑ Δ *Ours*	20%	22%	4%	4%	17%	13%	40%
RAG	7.88	8.05	8.92	8.21	7.91	8.19	5.58
↑ Δ *Ours*	14%	14%	8%	6%	15%	11%	33%
SW-RAG	8.03	8.23	9.36	8.35	8.12	8.42	5.75
↑ Δ *Ours*	12%	11%	3%	4%	12%	8%	29%
Cline	8.55	8.62	9.58	8.60	8.71	8.81	6.85
↑ Δ *Ours*	5%	6%	1%	1%	4%	4%	9%
HiFiRepoQA	**8.98**	**9.17**	**9.66**	**8.71**	**9.09**	**9.12**	**7.44**

Table 2: Comparative performance of HiFiRepoQA and baselines on *GPT-4o*. (RQ1)

Method	LLM-as-Judge Metrics						RAC
	Cor.	Com.	Rel.	Cla.	Rea.	Avg.	
Vanilla	7.50	7.12	9.46	7.93	7.51	7.90	4.09
↑ Δ *Ours*	12%	19%	3%	3%	18%	10%	67%
CoT	7.54	7.19	9.42	7.98	7.60	7.95	4.55
↑ Δ *Ours*	11%	18%	4%	2%	16%	10%	50%
RAG	7.66	7.25	8.76	7.69	7.62	7.80	5.05
↑ Δ *Ours*	10%	17%	11%	6%	16%	12%	35%
SW-RAG	7.76	7.39	8.93	7.97	7.69	7.95	5.11
↑ Δ *Ours*	8%	14%	9%	2%	15%	10%	34%
Cline	8.18	8.02	9.37	8.02	8.35	8.39	6.21
↑ Δ *Ours*	3%	5%	4%	1%	6%	4%	10%
HiFiRepoQA	**8.40**	**8.45**	**9.75**	**8.13**	**8.83**	**8.71**	**6.84**

Table 3: Comparative performance of HiFiRepoQA and baselines on *Qwen3-32B*. (RQ1)

5.2 Ablation Study (RQ2)

Table 6 reports the ablation results of HiFiRepoQA, evaluating the contribution of its major components. Overall, both structured planning and retrieval contribute substantially to the final perfor-

Method	LLM-as-Judge Metrics						RAC
	Cor.	Com.	Rel.	Cla.	Rea.	Avg.	
Vanilla	7.60	7.55	9.12	8.15	7.80	8.04	5.76
↑ Δ *Ours*	20%	26%	5%	9%	18%	15%	36%
CoT	7.62	7.68	9.07	8.25	7.98	8.12	6.05
↑ Δ *Ours*	20%	23%	6%	8%	15%	14%	29%
RAG	8.05	8.10	8.85	8.20	8.15	8.27	6.13
↑ Δ *Ours*	14%	17%	9%	8%	13%	12%	28%
SW-RAG	8.11	8.18	9.27	8.36	8.22	8.43	6.09
↑ Δ *Ours*	13%	16%	4%	6%	12%	10%	28%
Cline	8.65	8.73	9.79	8.84	8.81	8.96	7.35
↑ Δ *Ours*	6%	9%	-2%	1%	5%	3%	6%
Cursor	9.09	9.26	9.63	8.74	9.17	9.18	7.65
↑ Δ *Ours*	1%	2%	-0.1%	2%	0.4%	1%	2%
HiFiRepoQA	**9.14**	**9.48**	**9.62**	**8.89**	**9.21**	**9.27**	**7.82**

Table 4: Comparative performance of HiFiRepoQA and baselines on *Claude-4.5-Sonnet*. (RQ1)

Method	Avg. Time
Vanilla LLM	8.4s
CoT	17.8s
RAG	9.4s
SW_RAG	9.8s
Cline	64.1s
Cursor‡	59.3s
HiFiRepoQA (Summarize&Embedding)	2m 46s
HiFiRepoQA (Inference)	103.9s

Table 5: Comparative time cost using Claude-4.5-Sonnet on HiFiRepoQA and baselines (RQ1). ‡Cursor does not support customizable APIs (although it claims using Claude-4.5-Sonnet); its latency may differ from other methods.

mance. Specifically, replacing the DAG-based planner with linear planning results in a substantial performance decline (11% RAC reduction), suggesting that explicitly modeling subtask dependencies is critical for structured repository-level reasoning.

5.3 Usefulness Evaluation (RQ3)

We evaluate the practical usefulness of HiFiRepoQA in real-world open-source scenarios. We collect 41 unanswered GitHub questions from 7 active repositories and use HiFiRepoQA to generate complete answers. A human evaluation is first conducted to assess the quality of the generated responses in terms of correctness, informativeness, and fluency on a 5-point Likert scale. On average, HiFiRepoQA achieves scores of 4.27, 4.69, and 4.82 on these three dimensions respectively, indicating that the generated answers are accurate, informative, and well-written.

Module	LLM-as-Judge Metrics						RAC
	Cor.	Com.	Rel.	Cla.	Rea.	Avg.	
Ours	8.98	9.17	9.66	8.71	9.09	9.12	7.44
w/o Planning	8.15	8.06	9.31	8.44	7.98	8.38	5.85
w/o DAG	8.42	8.62	9.39	8.58	8.66	8.73	6.63
w/o Retrieval	7.57	7.58	9.33	8.26	7.84	8.11	5.36
w/o S-Retrieval†	8.29	8.55	9.38	8.54	8.61	8.67	6.22

† S-Retrieval denotes the embedding-based semantic retrieval tool within the retrieval agent.

Table 6: Contribution of different modules. (RQ2)

Repository	IDs
numpy/numPy	#28950
PaddlePaddle/Paddle	#76327
PaddlePaddle/PaddleOCR	#17150, #17176, #17275, #17276, #17377
collabora/WhisperLive	#406
open-compass/VLMEvalKit	#1367

Table 7: Information of resolved questions. (RQ3)

To further assess real-world acceptability, we submit manually verified answers generated by HiFiRepoQA using neutral accounts with anonymized usernames. This evaluation protocol allows us to assess the practical acceptability of the generated answers while avoiding disruption to the open-source community. Among all submitted answers, 9 questions receive positive feedback or are marked as closed due to our response, as summarized in Table 7. These results demonstrate that HiFiRepoQA can provide practically useful answers that are accepted by real users in real-world open-source development environments.

Detailed data collection procedures and evaluation protocols are provided in Appendix A.4.

6 Conclusion

Repository-level question answering is essential, yet remains a significant challenge due to the scale of modern codebases, the heterogeneity of information sources, and the partially ordered nature of repository knowledge. This paper proposes HiFiRepoQA, a multi-agent repository question answering framework that explicitly models subtask dependencies through DAG-based planning and supports targeted retrieval and localized reasoning over large repositories. Together, the proposed approach and the accompanying HiFiRepoQA bench provide a foundation for the next generation of repository Q&A systems.

Limitations

By adopting DAG-based planning and multi-role agent collaboration, HiFiRepoQA improves controllability and robustness for repository-level code question answering. However, this design introduces additional time overhead. As shown in Table 5, when using Claude-4.5-Sonnet, HiFiRepoQA's inference time is longer than that of single-model or lightweight retrieval-based approaches. This overhead primarily arises from multi-agent coordination, repeated LLM invocations, and multi-round retrieval to progressively narrow down and identify goal-relevant repository evidence. Future work may reduce latency through engineering strategies such as parallel execution, early stopping, lightweight scheduling, and caching mechanisms. Furthermore, deploying locally hosted models instead of remote closed-source models could further mitigate network-related delays.

In addition, HiFiRepoQA Bench emphasizes question authenticity, answer verifiability, and strict filtering during its construction, resulting in a relatively conservative dataset scale. While this design improves evaluation stability and the reliability of experimental conclusions, future work may further extend the benchmark to larger-scale or more open-ended question collections.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Anthropic. 2025. Introducing claude sonnet 4.5.

Jialiang Chen, Kaifa Zhao, Jie Liu, Chao Peng, Jierui Liu, Hang Zhu, Pengfei Gao, Ping Yang, and Shuiguang Deng. 2025. Coreqa: uncovering potentials of language models in code repository question answering. *arXiv preprint arXiv:2501.03447*.

Cline. 2025. Cline: Autonomous coding agent right in your ide. https://github.com/cline/cline. Accessed: 2026-01-03.

Cursor. 2025. Cursor: Ai-assisted code editor. https://cursor.com/. Accessed: 2025-12-31.

Vincenzo De Martino, Joel Castaño, Fabio Palomba, Xavier Franch, and Silverio Martínez-Fernández. 2025. A methodological framework for llm-based mining of software repositories. *arXiv preprint arXiv:2508.02233*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53. IEEE.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1).

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, and 1 others. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*.

Jiale Guo, Suizhi Huang, Mei Li, Dong Huang, Xingsheng Chen, Regina Zhang, Zhijiang Guo, Han Yu, Siu-Ming Yiu, Pietro Lio, and 1 others. 2025. A comprehensive survey on benchmarks and solutions in software engineering of llm-empowered agentic system. *arXiv preprint arXiv:2510.09721*.

Ruida Hu, Chao Peng, Jingyi Ren, Bo Jiang, Xiangxin Meng, Qinyun Wu, Pengfei Gao, Xinchen Wang, and Cuiyun Gao. 2025. Understanding large language model performance in software engineering: A large-scale question answering benchmark. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, page 3025–3029, New York, NY, USA. Association for Computing Machinery.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rock-

täschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Zehan Li, Jianfei Zhang, Chuantao Yin, Yuanxin Ouyang, and Wenge Rong. 2024. Procqa: a large-scale community-based programming question answering dataset for code search. *arXiv preprint arXiv:2403.16702*.

Vadim Lomshakov, Andrey Podivilov, Sergey Savin, Oleg Baryshnikov, Alena Lisevych, and Sergey Nikolenko. 2024. Proconsul: Project context for code summarization with llms. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 866–880.

Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, and 1 others. 2024. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. *arXiv preprint arXiv:2402.16667*.

OpenAI. 2024. Gpt-4o-mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accessed: 2026-01-04.

Weihan Peng, Yuling Shi, Yuhang Wang, Xinyun Zhang, Beijun Shen, and Xiaodong Gu. 2025. Swe-qa: Can language models answer repository-level code questions? *arXiv preprint arXiv:2509.14635*.

Tushar Sharma, Maria Kechagia, Stefanos Georgiou, Rohit Tiwari, Indira Vats, Hadi Moazen, and Federica Sarro. 2021. A survey on machine learning techniques for source code analysis. *arXiv preprint arXiv:2110.09610*.

Jan Strich, Florian Schneider, Irina Nikishina, and Chris Biemann. 2024. On improving repository-level code qa for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 209–244.

Tree-sitter Contributors. 2024. Tree-sitter: An incremental parsing system. https://github.com/tree-sitter/tree-sitter. Accessed: 2026-01-03.

Yanmeng Wang, Jun Bai, Ye Wang, Jianfei Zhang, Wenge Rong, Zongcheng Ji, Shaojun Wang, and Jing Xiao. 2021. Enhancing dual-encoders with question and answer cross-embeddings for answer retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2306–2315.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Dayu Yang, Antoine Simoulin, Xin Qian, Xiaoyi Liu, Yuwei Cao, Zhaopu Teng, and Grey Yang. 2025b. Docagent: A multi-agent system for automated code documentation generation. *CoRR*, abs/2504.08725.

Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. 2023. A survey on large language models for software engineering. *arXiv preprint arXiv:2312.15223*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

A Appendix

A.1 Details of Dataset Contribution

During the construction of the HiFiRepoQA Bench We select a total of 70 open-source repositories from GitHub, covering ten mainstream programming languages: Python, Java, C, JavaScript, TypeScript, Go, C++, C#, Rust and PHP. To guarantee the quality of the benchmark, we retain repositories with at least 500 stars and with the most recent commit later than July 2024, confirming they are still under active maintenance. At the same time, to ensure the legality and compliance of data, the repositories need to adopt loose open source licenses such as MIT, Apache-2.0, Unlicensed, or Creative Commons.

After repository selection, we automatically crawl real-world data from each repository, restricting our collection to entries created after January 1, 2025. To ensure overall data quality, we further apply a strict multi-stage filtering procedure.

We leverage metadata (e.g., tags and labels) to exclude candidate data items primarily related to feature requests or bug fixing. For cases whose type cannot be reliably determined from tags alone, we employ GPT-4o (Hurst et al., 2024) to classify whether a candidate data item represents a genuine user question, rather than a bug report or a feature request.

To ensure the overall quality of the benchmark, we retain only questions that have a clear and verifiable answer.

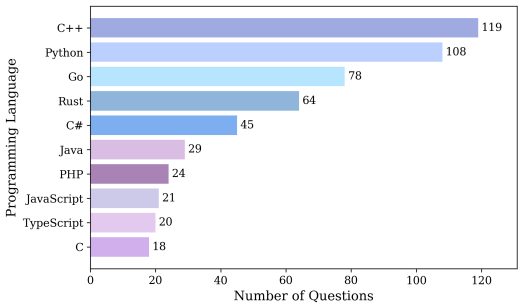


Figure 3: The Language Distribution of HiFiRepoQA Bench

Specifically, we first use GPT-4o-mini(OpenAI, 2024) to automatically determine whether there is a clear and verifiable answer. If it exists,the model extracts the recognized correct answer. We further conduct manual verification on a randomly sampled 20% of the retained questions, and observe high agreement with the automatic filtering results. The language distribution of the benchmark is illustrated in Figure 3.

A.2 Details of Rubric-based Answer Correctness (RAC)

This appendix provides the detailed design of the Rubric-based Answer Correctness (RAC) metric.

During the offline stage, for each question, we use `gpt-4o-mini` (OpenAI, 2024) to automatically construct corresponding scoring rules. During dataset construction, we ensure that each question is associated with a clear and verifiable answer. Here we use `gpt-4o-mini` to break the reference answer into a set of atomized scoring points.

Formally, we denote the rubric as

$$\mathcal{R} = \{r_1, r_2, \ldots, r_k\}, \tag{8}$$

where each r_i represents an atomic scoring point. Each scoring point r_i belongs only to one of the following two categories: MUST-HAVE, used to characterize the key information that must be met for predicting answers; MUST NOT HAVE, used to characterize the constraints that predicted answers must not violate. To reflect their relative importance, each scoring point r_i is assigned a discrete weight $w_i \in \{1, 2, 3\}$, corresponding to its importance in correctly answering the question. Each question typically contains 3-8 scoring points, all of which are designed to be single, verifiable, and have minimal ambiguity, ensuring consistency and stability in the evaluation results when facing semantically equivalent but differently expressed correct answers.

In the online evaluation stage, we use `gpt-4o-mini` to check whether the predicted answers meet the above scoring criteria one by one, and calculate the final score based on this. Specifically, the scoring of RAC is defined as follows:

$$\text{Score} = 10 \times \frac{\sum_i w_i \, s_i}{\sum_i w_i}, \tag{9}$$

where $w_i \in \{1, 2, 3\}$ denotes the importance weight of the i-th scoring point, and $s_i \in \{0, 1\}$ indicates whether the predicted answer satisfies that scoring point. The final score is normalized to the range $[0, 10]$.

A.3 Experimental Settings.

In the retrieval stage, we use Qwen3-Embedding-8B (Zhang et al., 2025) as the text embedding model to encode and perform similarity retrieval on repository related texts; On this basis, Qwen3-Reranker-8B is used to rerank the candidate results to improve the relevance and ranking quality of the retrieval results.

In the inference stage, we evaluate `HiFiRepoQA` on various mainstream LLMs to test its applicability and stability under different model sizes and architectures. Specifically, the experiment covers Qwen3-32B (Yang et al., 2025a), Claude-4.5-Sonnet (Anthropic, 2025), DeepSeek-R1 (DeepSeek-AI, 2025), GPT-OSS-20B (Agarwal et al., 2025), and GPT-4o (Hurst et al., 2024). To reduce the impact of randomness and ensure reproducibility, the generation temperature is fixed to 0. Except for the differences in the models themselves, all other experimental settings are kept consistent to ensure fair and comparable results.

A.4 Details of Usefulness Evaluation (RQ3)

A.4.1 Data Collection

We collect unanswered GitHub Questions from 7 active repositories, including NumPy, Paddle, WhisperLive, PaddleOCR, VLMEvalKit, PaddleX, and trae-agent. The selected posts were created between March 23, 2025 and December 19, 2025. After filtering out duplicated, ill-formed, or already resolved posts, we obtain a total of 41 unanswered questions.

A.4.2 Human Evaluation Protocol

We conduct a human evaluation along three dimensions: correctness, informativeness, and fluency,

each rated on a 5-point Likert scale. Correctness measures factual accuracy and the absence of hallucinations; Informativeness evaluates whether the answer provides sufficient information to address the question; Fluency assesses clarity and readability.

To reduce subjective bias, we design three identical questionnaires for each Q&A pair, ensuring that each pair is independently evaluated by three different participants. Each questionnaire consists of four parts:

(1) Question description;

(2) The answer generated by `HiFiRepoQA`;

(3) Relevant repository content to assist participants in understanding the question;

(4) Descriptions of the three evaluation dimensions, based on which participants rate the Q&A pair. We recruit five experts with over five years of software development experience as evaluators, none of whom are the authors of this paper. Before the formal evaluation, the evaluators are provided with task instructions and example questionnaires to ensure a clear understanding of the evaluation procedure and scoring criteria. Ultimately, each question is scored by three evaluators, and the final result is obtained by averaging score of each dimension.

A.4.3 GitHub Submission Protocol

To avoid disrupting the open-source community, we only submit answers that are manually verified to be correct and helpful. All answers are posted using newly registered neutral accounts with anonymized usernames. If a question receives positive feedback from the question asker or repository maintainers, or is marked as closed or resolved after our response, we consider it successfully resolved.

A.5 RQ1 results on Additional Backbone Models

This appendix reports the complete RQ1 results on additional backbone models. All settings, evaluation metrics, and compared methods are identical to those in the main paper. The results in Table 8-9 exhibit consistent relative performance trends across different backbones, supporting the same conclusions as those reported in Section 5.1.

Method	LLM-as-Judge Metrics						RAC
	Cor.	Com.	Rel.	Cla.	Rea.	Avg.	
Vanilla	7.74	7.96	9.01	8.12	7.94	8.15	5.17
↑ Δ *Ours*	13%	10%	9%	5%	13%	10%	38%
CoT	7.79	8.04	8.95	8.18	8.08	8.21	5.58
↑ Δ *Ours*	13%	9%	10%	5%	11%	9%	28%
RAG	8.23	8.15	8.89	8.13	8.26	8.33	5.85
↑ Δ *Ours*	7%	7%	10%	5%	9%	8%	22%
SW-RAG	8.35	8.24	9.05	8.28	8.34	8.45	5.92
↑ Δ *Ours*	5%	6%	9%	3%	8%	6%	20%
Cline	8.52	8.48	9.60	8.36	8.65	8.72	6.55
↑ Δ *Ours*	3%	3%	2%	2%	4%	3%	9%
HiFiRepoQA	**8.78**	**8.75**	**9.82**	**8.55**	**8.98**	**8.98**	**7.12**

Table 8: Comparative performance of `HiFiRepoQA` and baselines on *DeepSeek-R1*. (RQ1)

Method	LLM-as-Judge Metrics						RAC
	Cor.	Com.	Rel.	Cla.	Rea.	Avg.	
Vanilla	7.46	7.19	9.19	7.83	7.41	7.82	5.05
↑ Δ *Ours*	15%	18%	6%	6%	19%	12%	38%
CoT	7.52	7.26	9.17	7.88	7.53	7.87	5.36
↑ Δ *Ours*	14%	17%	7%	5%	17%	12%	30%
RAG	7.75	7.36	9.23	7.93	7.58	7.97	5.63
↑ Δ *Ours*	10%	16%	6%	5%	16%	10%	24%
SW-RAG	8.18	8.02	9.37	8.02	8.35	8.39	6.21
↑ Δ *Ours*	5%	6%	4%	4%	5%	5%	12%
Cline	8.29	8.31	9.52	8.18	8.33	8.53	6.32
↑ Δ *Ours*	3%	3%	3%	2%	6%	3%	10%
HiFiRepoQA	**8.55**	**8.52**	**9.77**	**8.31**	**8.79**	**8.79**	**6.96**

Table 9: Comparative performance of `HiFiRepoQA` and baselines on *GPT-OSS-20B*. (RQ1)

A.6 Prompt of Planning Agent

You are an expert in Code Repo QA. Below is a question from the {repo} repository. You need to determine if you can answer it directly. If you need additional information, please create a plan to acquire it. If you can answer the question directly, just give an empty plan array. Based on the GitHub question below, plan a short Directed Acyclic Graph (DAG) of subtasks (Keep the number of steps as few as necessary.).
Question Title: {title}
Question Description: {body}
Typed Question Inputs (heterogeneous modeling, pre-planning):
{typed_inputs}
The question contains heterogeneous information sources:

- q_text: Natural language descriptions (title, body, comments)
- q_code: Code snippets provided in the question
- q_output: Runtime outputs from successful executions
- q_error: Error messages or exception traces from failed executions

Repository Tree and Knowledge Base Context:
{kb_context}
Documentation (Markdown) Snippets:
{md_context}
Requirements:

- Output a DAG of steps with explicit dependencies where applicable.
- Each step must be atomic and outcome-oriented.
- Each step must include: id, type, title, goal, input_types, and optional depends_on.
- type must be either "Retrieval" (for repository information acquisition) or "Analysis" (for localized semantic reasoning).
- input_types is a required array drawn from ["q_text", "q_code", "q_output", "q_error"] indicating which heterogeneous information sources from the question this step should receive. Only assign relevant information types to each step (e.g., error diagnosis steps should include q_error, semantic understanding steps primarily use q_text and q_code).
- depends_on is an array of step ids this step depends on; omit or [] if none.
- Avoid cycles; prefer shallow depth and parallelizable branches when possible.
- You can only READ repository content; do NOT write/modify files.

Return JSON array:
{"id": "step-1", "type": "Retrieval", "goal": "Identify candidate files", "input_types": ["q_text"], "depends_on": []},
{"id": "step-2", "type": "Retrieval", "goal": "Collect semantic matches", "input_types": ["q_text"], "depends_on": ["step-1"]},
{"id": "step-3", "type": "Retrieval", "goal": "Find exact references", "input_types": ["q_code"], "depends_on": ["step-1"]},
{"id": "step-4", "type": "Analysis", "goal": "Diagnose root cause", "input_types": ["q_error", "q_code"], "depends_on": ["step-2", "step-3"]}

A.7 Prompt of Retrieval Agent

You are a repo Q&A retrieval agent. Decide what to retrieve for the current step.
Current Step:

- Title: {step_title}
- Goal: {step_goal}
- Assigned Input Types: {input_types_str}

Question Title: {title}
Question Description: {body}
Typed Question Inputs (assigned):
{typed_inputs}
Prior Context (JSON): {prev_ctx}
Return JSON ONLY with the following fields:
{
"previous_useful_info": "previous useful info",
"needs_embedding_search": true,
"embedding_search_terms": ["term1", "term2"],
"needs_keyword_search": false,
"keyword_search_terms": ["term1", "term2"],
"needs_file_analysis": true,
"target_files": ["path/a", "path/b"],
"reasoning": "why these searches/files",
"step_title": "{step_title}",
"step_goal": "{step_goal}"
}

A.8 Prompt of Analysis Agent

You are a repo Q&A analysis agent. Using the current step, question, prior context and the retrieved snippets, draft a concise intermediate answer.
Current Step: {step_title} → {step_goal}
Question Title: {title}
Question Description: {body}

Prior Context: {prev_ctx}
Retrieved Snippets: {evid_snips}

Before answering, assess whether each retrieved snippet is relevant to the goal of the current step, either by explicitly referencing goal-related identifiers (e.g., function or class names), or by describing behavior or semantics that align with the intent of the goal. If a snippet is not relevant, do not rely on it and instead reason based on your own understanding, without being distracted by irrelevant information.

Return JSON:
{ "previous_useful_info": "previous useful info",
"answer": "short answer",
"reasoning": ["step1", "step2"],
"evidence": [{"file": "path", "snippet": "...", "type": "code|doc|file"}]}

A.9 Prompt of Synthesis Agent

A.9.1 Evidence Verification

You are an expert software engineer and open-source maintainer.
Your task is to determine whether the currently collected context is sufficient to produce a reliable final answer to the GitHub question.
Repository Name
{repo}
Question Details
Title: {title}
Description: {body}
Repository Structure
Directory tree and global documentation (e.g., README):
{repo_structure}
Context
{context}
Task
Based on the question, the context, and repository structure, decide whether the context is sufficient to support a final answer.
Output Format
Return only the following JSON:
{ "whether_sufficient": "True | False",
"goal": ""
}
Rules:

- If "whether_sufficient" is "True", leave "goal" empty.
- If "whether_sufficient" is "False", "goal" must briefly state the parts of the current question that remain unresolved.

A.9.2 Final Answer Generation

You are an expert software engineer and open-source maintainer. Your task is to carefully analyze a GitHub question and provide a comprehensive and well-reasoned answer.
Repository Name
{repo}
Question Details
Title: {title}
Description: {body}
Additional Context
Below is optional background information from the repository (may include code snippets, documentation, or config files). Use it only if relevant to your reasoning:
{context}
Your Task

1. Read and understand the question carefully.
2. Use the provided context to support your reasoning (only if relevant). If none is relevant, use your own knowledge to answer the question.
3. Produce a **clear, technically accurate, and actionable** answer to help the question author.
4. Support your answer with evidence (code or documentation snippets).
5. Explain your reasoning logically.

Output Format
Return your response **only** in the following JSON structure (no extra commentary):
{
"answer": "Your complete and detailed solution or explanation.",
"evidence": [
{
"file": "path/to/file",
"snippet": "relevant code or doc excerpt",
"type": "code | doc | config"
}
],
"reasoning": "Step-by-step reasoning that led to

your conclusion."
}
Quality Requirements

- Be accurate, concise, and technically grounded.
- Use evidence from the repo context when possible.

A.10 Use of AI Assistants

AI-assisted tools were used in a limited manner to help improve grammatical correctness and writing clarity. The authors are fully responsible for the content of the paper, including all ideas, methods, experiments, and conclusions.