DON'T WEAR TOO MANY HATS: CAPABILITY MODULARIZATION FOR MULTI-DOCUMENT ANALYTICAL QUESTION ANSWERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-document analytical question answering (MDocAQA) requires coordination among different information processing capabilities over evidence distributed across files. In existing methods, individual agent typically needs to perform operations across multiple capabilities. Since single agent often struggles to sustain effective reasoning and reflection throughout long-horizon workflows that require coordinating diverse capabilities, its performance tends to be poor. To address this issue, we propose an MDocAQA framework called Sensemaker. Inspired by the model of expert sensemaking, which represents information analysis as an iterative process containing distinct information processing activities, Sensemaker organizes MDocAQA into three layers (*i.e.*, task layer, capability layer, and tool layer) and explicitly defines five modules responsible for different information processing capabilities in the capability layer. During task execution, the task-layer agent dynamically coordinates these modules, each instantiated as a specialized agent equipped with dedicated tools. This explicit capability modularization reduces the workload on individual agent and makes its intermediate states easier to inspect. Extensive experimental results on various benchmarks show that Sensemaker outperforms representative MDocAQA methods, with up to a 29.71% relative improvement in average answer score over the strongest baseline method.

1 INTRODUCTION

Multi-document analytical question answering (MDocAQA) (Li et al., 2026b) requires intelligent systems to identify, extract, and synthesize evidence across a collection of provided files. Since some evidence is scattered across multiple files and may become identifiable only through cross-file analysis and iterative exploration, MDocAQA requires strong abilities in planning, content understanding, reasoning, reflection, and so on (Wang et al., 2024b; Li et al., 2024).

Recent advances in agents based on large language models (LLMs) (Qin et al., 2024; Wu et al., 2023) offer a feasible strategy for MDocAQA. When given access to the file library, agents can search across files and adapt subsequent actions based on the retrieved information (Wang et al., 2024c; Zhao et al., 2024b; Yang et al., 2025). However, in existing methods, individual agent typically needs to handle multiple information processing capabilities within a single task execution. As illustrated in Figure 1, this design introduces two critical limitations: (a) Agent overload: coordinating the operations required by these capabilities imposes a heavy workload on the agent, increasing the risk of context forgetting, redundant exploration, and evidence misattribution; (b) Process opacity: the execution steps and intermediate outputs associated with these capabilities are entangled within the reasoning trajectory of a single agent, making intermediate states difficult to inspect and errors difficult to trace. These limitations often compromise the effectiveness of the corresponding capabilities, thereby reducing overall task performance.

To address these limitations, this paper proposes Sensemaker, an MDocAQA framework inspired by the model of expert sensemaking (Pirolli & Card, 2005). As revealed in Pirolli & Card (2005), expert sensemaking refers to an iterative process that generates structured evidence and conclusions from substantial information through the coordination among multiple information processing activities. Motivated by this model, Sensemaker organizes MDocAQA into three layers: task layer,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

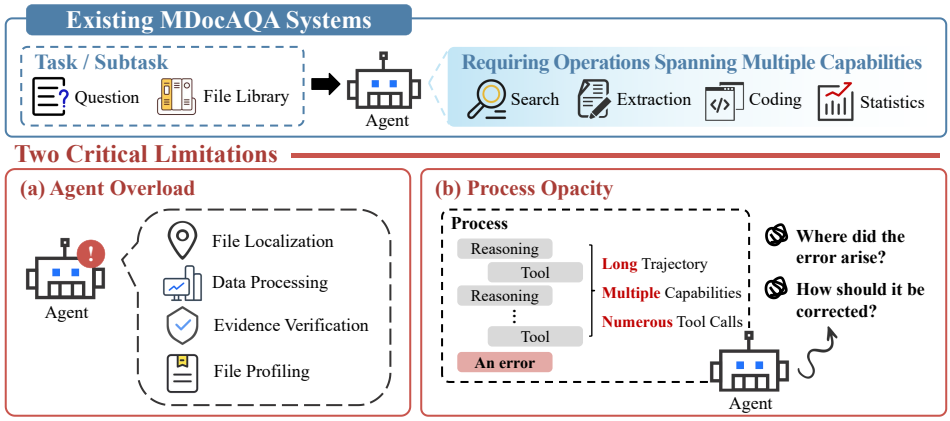


Figure 1: Limitations of existing MDocAQA methods: (a) Requiring a single agent to perform operations across multiple information processing capabilities imposes a heavy workload on the agent; (b) Entangling the execution steps and intermediate outputs of multiple capabilities within one reasoning trajectory makes intermediate states difficult to inspect and errors difficult to trace.

capability layer, and tool layer. Specifically, the task layer contains a single agent responsible for maintaining the global objective and coordinating the overall analysis. The capability layer explicitly defines five modules, namely file localization (FL), file profiling (FP), content extraction (CE), data processing (DP), and evidence verification (EV). Among them, FL identifies candidate files based on the current evidence state; FP provides file descriptions, such as file types, content summaries, and possible roles in the current task; CE extracts fine-grained evidence from the targeted files; DP performs statistical analysis, numerical computation, and data aggregation; EV verifies the validity of the accumulated evidence. The tool layer exposes dedicated tools for well-defined operations. During task execution, the task-layer agent dynamically invokes the five capability modules based on the query and execution history, while a specialized agent within each module uses dedicated tools to fulfill its designated responsibility. By explicitly modeling different information processing capabilities and assigning them to specialized agents, Sensemaker reduces the workload on individual agent and makes intermediate states of the execution trajectory easy to inspect, thereby enabling high-quality information processing throughout the task execution.

The effectiveness of our proposed Sensemaker has been well confirmed by extensive experiments on six MDocAQA benchmarks. When compared with representative methods, Sensemaker improves the answer score averaged across six benchmarks by up to 0.309. Additionally, the evidence sets constructed by Sensemaker cover more essential files and fewer irrelevant files than those constructed by the baseline methods, with maximum gains of 0.704 in file recall and 0.649 in file precision. These results demonstrate that Sensemaker enables effective analytical reasoning over multiple files.

The main contributions are summarized as follows.

- We propose Sensemaker, a multi-agent framework with three layers (*i.e.*, task layer, capability layer, and tool layer). By coordinating global planning, information processing, and tool execution across these layers, Sensemaker can effectively address MDocAQA.
- We design five modules in the capability layer, namely FL, FP, CE, DP, and EV. By assigning different analytical responsibilities to specialized modules and providing each module with dedicated support, Sensemaker alleviates the two issues faced by existing methods.
- We conduct extensive experiments on various MDocAQA benchmarks to show that our Sensemaker consistently outperforms representative baseline methods.

2 Related Work

Depending on whether the solution process is performed by one agent or distributed across multiple agents, existing methods for MDocAQA can be categorized into two types, namely *single-agent framework* and *multi-agent framework*.

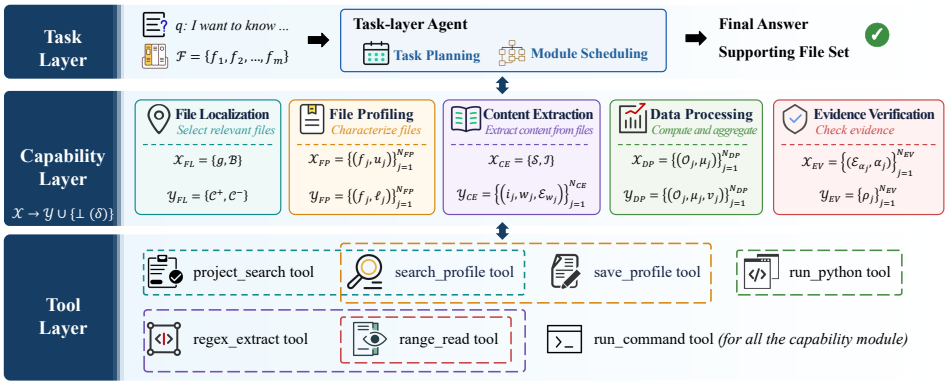


Figure 2: Overview of our proposed Sensemaker, which contains three layers: (a) Task Layer: a central agent tracks the overall objective and coordinates capability modules. (b) Capability Layer: five specialized modules handle distinct information processing capabilities. (c) Tool Layer: dedicated tools execute concrete operations under the control of the agents in capability modules.

Single-Agent Framework. These methods (Saad-Falcon et al., 2024; Yang et al., 2025) rely on one agent to solve MDocAQA. One strategy (Bai et al., 2025; Wang et al., 2024b) places all file contents in the context of the agent for direct reasoning. However, agents often struggle to reason reliably over long contexts (Liu et al., 2024). Standard retrieval-augmented generation (RAG) (Wang et al., 2024c;a) instead supplies the agent with only task-relevant text segments. However, in MDocAQA, some evidence becomes relevant only after cross-document analysis and iterative exploration, limiting the retrieval accuracy of standard RAG. To address this issue, some methods (Yao et al., 2022; Yang et al., 2025) allow the agent to plan retrieval and revise subsequent searches based on intermediate results. Moreover, some methods, such as LongTA (Li et al., 2026a), equip the agent with additional tools to support evidence management and quantitative analysis.

Multi-Agent Framework. These methods (Zhao et al., 2024a; Li et al., 2026b) distribute the solution process for an MDocAQA task across multiple agents. A mainstream strategy, adopted by LongAgent (Zhao et al., 2024a) and Chain-of-Agents (Zhang et al., 2024), partitions the contents of the file library into multiple subsets and assigns each subset to a separate agent. Moreover, some methods use multiple agents to improve specific processing stages of MDocAQA. For instance, MAIN-RAG (Chang et al., 2025) focuses on the retrieval stage by designing predictor and judge agents to improve retrieval accuracy, while HuggingGPT (Shen et al., 2023) focuses on selecting suitable expert models from a model pool for subtasks.

However, existing methods often require a single agent to handle multiple information processing capabilities. This design can overload the agent and entangle the intermediate outputs of different capabilities within one reasoning trajectory, making the process difficult to inspect. In contrast, Sensemaker adopts a "task–capability–tool" architecture and defines five specialized modules in the capability layer. By distributing analytical responsibilities across different modules, Sensemaker reduces the capability workload on individual agent and makes intermediate states easier to inspect.

Besides, some methods (Shen et al., 2023; Zhao et al., 2024a; Li et al., 2026b) tackle complex tasks through task decomposition. However, they face a trade-off between generality and controllability. Allowing LLMs to determine decompositions supports adaptability to diverse tasks, but makes the executions difficult to predict and constrain. Conversely, predefined decomposition strategies provide strong performance, but limit generality. In contrast, Sensemaker mitigates this tension by imposing structure at the capability level. It retains flexible task planning while coordinating predefined capability modules with well-defined responsibilities. This design supports adaptation to different MDocAQA tasks while providing explicit control over intermediate operations.

3 METHODOLOGY

In this section, we introduce the overall framework of Sensemaker. As shown in Figure 2, Sensemaker organizes multi-document analysis into three layers, namely task layer, capability layer, and

tool layer. Formally, given a question q and a file library $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$, Sensemaker needs to produce an answer to q together with the set of files that support this answer. During the solution process of Sensemaker, the task layer decides what to do next and dispatches subtasks to the capability layer, the capability layer performs multiple information processing capabilities through the tool layer, and the returned structured results update the evidence state that the task layer uses for its next decision. In this way, each capability-layer agent needs to possess only one capability during execution, making its operations explicit and inspectable.

3.1 Task Layer

Task layer contains a single agent that is responsible for coordinating the overall analysis. It does not directly read files or invoke tools. Instead, it only interacts with the five capability modules. Specifically, the responsibility of this agent primarily includes the following three items:

- **Task Planning.** This agent plans the future actions according to the question and the current state of the analysis, deciding what the task requires next.
- **Capability module scheduling.** Given the current requirement, this agent invokes appropriate capability modules to advance the analysis process.
- **Final answer generation.** As the only component that sees the whole evidence chain, this agent generates the final answer based on the accumulated evidence.

Because every module returns a structured result rather than a free-form narration, the evidence state stays explicit and the trajectory of the whole analysis remains easy to inspect.

3.2 Capability Layer

Capability layer models five capability modules to undertake different information processing responsibilities required for MDocAQA, namely FL, FP, CE, DP, and EV. Each module contains one specialized agent to perform the corresponding capabilities. Formally, a module $k \in \{\mathrm{FL}, \mathrm{FP}, \mathrm{CE}, \mathrm{DP}, \mathrm{EV}\}$ can be expressed as follows:

$$k : \mathcal{X}_k \to \mathcal{Y}_k \cup \{\bot(\delta)\}, \tag{1}$$

where $\mathcal{X}_k$ denotes the structured input to module k and $\mathcal{Y}_k$ denotes its successful output. Besides, $\bot(\delta)$ denotes a rejection carrying a diagnostic δ, which names the unhandled parts of $\mathcal{X}_k$ that either fall outside the capability scope of k or lack the information required for processing.

File Localization (FL). FL selects relevant files from the library according to the question and the evidence gathered so far. Its input $\mathcal{X}_{\mathrm{FL}}$ and output $\mathcal{Y}_{\mathrm{FL}}$ are instantiated as:

$$\mathcal{X}_{\mathrm{FL}} = \{g, \mathcal{B}\}, \quad \mathcal{Y}_{\mathrm{FL}} = \left\{\mathcal{C}^+, \mathcal{C}^-\right\}, \tag{2}$$

where g denotes the task information and $\mathcal{B} \subseteq \mathcal{F}$ denotes a search boundary that limits the files FL is allowed to consider. Besides, $\mathcal{C}^+, \mathcal{C}^- \subseteq \mathcal{B}$ denote two disjoint sets of candidate files. Here $\mathcal{C}^+$ collects the files with direct evidence of relevance, while $\mathcal{C}^-$ collects the files that look possibly relevant but need further verification. Retaining $\mathcal{C}^-$ is important for multi-document analysis, because a file that seems to be of low relevance at present may turn out to be necessary once more evidence is collected. Consequently, keeping such files as candidates avoids discarding them too early.

File Profiling (FP). FP characterizes specified files in terms of attributes such as file type, content summary, and possible roles in the current task. Its input $\mathcal{X}_{\mathrm{FP}}$ and output $\mathcal{Y}_{\mathrm{FP}}$ are instantiated as:

$$\mathcal{X}_{\mathrm{FP}} = \{(f_j, u_j)\}_{j=1}^{N_{\mathrm{FP}}}, \quad \mathcal{Y}_{\mathrm{FP}} = \{(f_j, \ell_j)\}_{j=1}^{N_{\mathrm{FP}}}, \tag{3}$$

where N_{FP} denotes the number of files to be profiled, u_j denotes the required information for f_j, and ℓ_j denotes the answer to u_j. Besides, every triplet (f_j, u_j, ℓ_j) is stored in a profile repository $\mathcal{D}$ and can be reused by subsequent module invocations within the same run.

Content Extraction (CE). CE extracts fine-grained evidence from the specified files. Its input $\mathcal{X}_{\mathrm{CE}}$ and output $\mathcal{Y}_{\mathrm{CE}}$ are instantiated as:

$$\mathcal{X}_{\mathrm{CE}} = \{\mathcal{S}, \mathcal{I}\}, \quad \mathcal{Y}_{\mathrm{CE}} = \left\{\left(i_j, w_j, \mathcal{E}_{w_j}\right)\right\}_{j=1}^{N_{\mathrm{CE}}}, \quad \mathcal{E}_{w_j} = \left\{\left(s_{w_j}^t, p_{w_j}^t, r_{w_j}^t\right)\right\}_t. \tag{4}$$

Table 1: Descriptions of the tools in Sensemaker. Tool-call syntax is presented in Appendix G.

Tool	Module	Operation
`search_profile`	FL, FP	Retrieving information in $\mathcal{D}$ by keyword, file path, or regular expression
`project_search`	FL	Retrieving candidate files within a bounded search scope by filtering file patterns and matching keywords
`save_profile`	FP	Storing a new file profile or updating an existing one in $\mathcal{D}$
`range_read`	CE, EV	Returning the contents of the requested line ranges in one or more files
`regex_extract`	CE	Extracting matching content and contextual information from specified files using multiple regular-expression patterns
`run_python`	DP	Running Python code and returning the result
`run_command`	*All*	Executing the shell command and returning the result

where N_{CE} denotes the number of items needed to extract, $\mathcal{S} \subseteq \mathcal{F}$ denotes the set of source files, and $\mathcal{I} = \{i_1, \ldots, i_{N_{\mathrm{CE}}}\}$ denotes the items to extract. For i_j with $j \in \{1, 2, \cdots, N_{\mathrm{CE}}\}$, w_j is the extracted information and $\mathcal{E}_{w_j}$ is the set of evidence records supporting w_j. In its t-th record, $r^t_{w_j}$ denotes a raw source fragment supporting w_j, $s^t_{w_j} \in \mathcal{S}$ denotes the file that $r^t_{w_j}$ comes from, and $p^t_{w_j}$ denotes the position of $r^t_{w_j}$ inside $s^t_{w_j}$. This design binds each extracted result to its supporting source fragments, allowing downstream claims built on the result to be re-examined.

Data Processing (DP). DP carries out the quantitative work of the analysis, *e.g.*, statistical summarization and numerical computation. Its input $\mathcal{X}_{\mathrm{DP}}$ and output $\mathcal{Y}_{\mathrm{DP}}$ are instantiated as:

$$\mathcal{X}_{\mathrm{DP}} = \{(\mathcal{O}_j, \mu_j)\}_{j=1}^{N_{\mathrm{DP}}}, \quad \mathcal{Y}_{\mathrm{DP}} = \{(\mathcal{O}_j, \mu_j, v_j)\}_{j=1}^{N_{\mathrm{DP}}}, \tag{5}$$

where N_{DP} denotes the number of tasks, $\mathcal{O}_j$ denotes the source contents to compute over, μ_j denotes the requested data processing on $\mathcal{O}_j$, and v_j denotes the execution result of μ_j on $\mathcal{O}_j$. As v_j is computed rather than directly extracted from files, it is derived evidence instead of a fact stated somewhere in a file, which distinguishes it from the evidence that CE returns.

Evidence Verification (EV). EV checks whether the facts underlying each conclusion are supported by the cited source spans. Its input $\mathcal{X}_{\mathrm{EV}}$ and output $\mathcal{Y}_{\mathrm{EV}}$ are instantiated as:

$$\mathcal{X}_{\mathrm{EV}} = \left\{(\mathcal{E}_{\alpha_j}, \alpha_j)\right\}_{j=1}^{N_{\mathrm{EV}}}, \quad \mathcal{Y}_{\mathrm{EV}} = \{\rho_j\}_{j=1}^{N_{\mathrm{EV}}}, \tag{6}$$

where N_{EV} denotes the number of claims submitted for checking, $\mathcal{E}_{\alpha_j}$ denotes the set of evidence records cited to support the claim α_j, and ρ_j denotes the verification result for α_j. Since the input already specifies where each claim should be checked, EV only judges a claim at the place to which it is attributed, rather than searching for better support or producing new claims.

3.3 Tool Layer

This layer equips the capability modules with various tools that execute concrete information processing operations. All these tools follow two design principles. First, the input and output of every tool are structurally defined, so that each operation and its result remain clear and inspectable. Second, each module can access only the tools required by its own responsibility, which reinforces the boundaries among different capabilities. Detailed descriptions of the tools are presented in Table 1.

4 Experiments

In this section, we present the experimental setup and analyses of our experimental results.

4.1 Experimental Setup

In this part, we introduce the details of the experimental setup. Detailed descriptions of the baseline methods (including their implementations in the experiments) and benchmarks are provided in Appendices B and C, respectively.

Baseline Methods. We compare Sensemaker with seven methods: (a) Direct Long-context Reasoning (DLcR) (Bai et al., 2025), (b) Retrieval-Augmented Generation (RAG) (Lewis et al., 2020),

Table 2: Answer scores on six benchmarks. For each backbone model, the best score on each benchmark is shown in bold. *Avg.* is the unweighted mean over all six benchmarks.

Model	Method	Benchmark						*Avg.*
		LBv2	Loong	xAFS	FRAMES	MuDA	RxA	
Kimi-K2.6	DLcR (Bai et al., 2025)	0.616	0.643	0.303	0.123	0.500	0.087	0.379
	RAG (Lewis et al., 2020)	0.579	0.542	0.336	0.152	0.521	0.093	0.370
	LongAgent (Zhao et al., 2024a)	0.653	0.672	0.370	0.248	0.479	0.113	0.422
	CoA (Zhang et al., 2024)	0.643	0.673	0.342	0.218	0.458	0.177	0.419
	MAIN-RAG (Chang et al., 2025)	0.576	0.548	0.442	0.161	0.530	0.242	0.417
	BELLE (Zhang et al., 2025a)	0.677	0.693	0.506	0.283	0.608	0.237	0.501
	LongTA (Li et al., 2026a)	0.693	0.702	0.485	0.368	0.560	0.284	0.515
	Sensemaker (ours)	**0.731**	**0.757**	**0.621**	**0.606**	**0.705**	**0.588**	**0.668**
GLM-5.2	DLcR (Bai et al., 2025)	0.669	0.692	0.330	0.198	0.551	0.123	0.427
	RAG (Lewis et al., 2020)	0.621	0.603	0.358	0.227	0.491	0.084	0.397
	LongAgent (Zhao et al., 2024a)	0.704	0.722	0.382	0.318	0.539	0.172	0.473
	CoA (Zhang et al., 2024)	0.693	0.712	0.367	0.286	0.518	0.238	0.469
	MAIN-RAG (Chang et al., 2025)	0.632	0.613	0.439	0.243	0.590	0.296	0.469
	BELLE (Zhang et al., 2025a)	0.723	0.732	0.497	0.363	0.651	0.307	0.545
	LongTA (Li et al., 2026a)	0.757	0.747	0.539	0.442	0.620	0.358	0.577
	Sensemaker (ours)	**0.765**	**0.799**	**0.670**	**0.646**	**0.723**	**0.632**	**0.706**
Qwen3.5	DLcR (Bai et al., 2025)	0.627	0.628	0.264	0.133	0.479	0.078	0.368
	RAG (Lewis et al., 2020)	0.589	0.532	0.291	0.163	0.500	0.067	0.357
	LongAgent (Zhao et al., 2024a)	0.659	0.652	0.336	0.238	0.458	0.192	0.422
	CoA (Zhang et al., 2024)	0.648	0.647	0.318	0.222	0.443	0.168	0.408
	MAIN-RAG (Chang et al., 2025)	0.589	0.543	0.409	0.172	0.509	0.253	0.413
	BELLE (Zhang et al., 2025a)	0.683	0.672	0.467	0.277	0.572	0.212	0.480
	LongTA (Li et al., 2026a)	0.699	0.707	0.494	0.352	0.539	0.244	0.506
	Sensemaker (ours)	**0.707**	**0.738**	**0.615**	**0.589**	**0.678**	**0.535**	**0.644**

(c) LongAgent (Zhao et al., 2024a), (d) Chain-of-Agents (CoA) (Zhang et al., 2024), (e) MAIN-RAG (Chang et al., 2025), (f) BELLE (Zhang et al., 2025a), and (g) LongTA (Li et al., 2026a).

Backbone Models. To examine whether the experimental conclusions generalize across backbone models, we instantiate all the methods with three LLMs: (a) Kimi-K2.6 (Moonshot AI, 2026), (b) GLM-5.2 (Z.ai, 2026), and (c) Qwen3.5-397B-A17B (Qwen Team, 2026). For simplicity, we refer to Qwen3.5-397B-A17B as Qwen3.5.

Benchmarks. We evaluate all the employed methods on six benchmarks: (a) the multi-document QA subset of LongBench v2 (LBv2) (Bai et al., 2025), (b) Loong (Wang et al., 2024b), (c) xAFS (Supermemory, 2026), (d) FRAMES (Krishna et al., 2025), (e) MuDABench (MuDA) (Li et al., 2026b), and (f) RxA. Among them, the first five are public MDocAQA benchmarks. To assess the agent performance in a realistic domain with a large-scale file library, we construct RxA, in which both the file library and questions are derived from real-world electronic design automation (EDA) tasks. The file library contains more than 13,000 files, resulting in a substantially larger search space and making evidence construction considerably more challenging than it is in existing benchmarks.

Evaluation Metrics. For all the employed benchmarks, we follow their corresponding papers (Bai et al., 2025; Wang et al., 2024b; Supermemory, 2026; Krishna et al., 2025; Li et al., 2026b) and utilize $Sco.$ (see Appendix D.1 for the specific calculation formulas) for evaluating the performance of the compared methods. We conduct three independent runs for each experimental configuration under the same settings and report the mean of the three results.

4.2 Main Results

In this part, we present the experimental results and detailed analysis to highlight the effectiveness of our proposed Sensemaker. To complement the quantitative evaluation with a qualitative examination of system behavior, we provide a comparative case study in Appendix H.



Figure 3: Answer scores of all the methods under the gold-file setting, where the file library assigned to each question only contains the files required to answer that question.

Table 3: Comparison of file recall (abbreviated as "R") and precision (abbreviated as "P") on four benchmarks under three backbone models. The best score for each metric on each benchmark under each backbone model is shown in bold.

Model	Method	xAFS		FRAMES		MuDA		RxA	
		R	P	R	P	R	P	R	P
Kimi-K2.6	DLcR (Bai et al., 2025)	0.318	0.346	0.148	0.123	0.362	0.338	0.043	0.046
	RAG (Lewis et al., 2020)	0.380	0.385	0.191	0.138	0.412	0.387	0.062	0.060
	LongAgent (Zhao et al., 2024a)	0.684	0.542	0.281	0.313	0.637	0.512	0.109	0.116
	CoA (Zhang et al., 2024)	0.652	0.681	0.259	0.334	0.596	0.548	0.094	0.124
	MAIN-RAG (Chang et al., 2025)	0.603	0.723	0.317	0.476	0.451	0.781	0.159	0.190
	BELLE (Zhang et al., 2025a)	0.668	0.746	0.385	0.431	0.574	0.706	0.236	0.171
	LongTA (Li et al., 2026a)	0.713	0.764	0.410	0.448	0.792	0.752	0.297	0.205
	Sensemaker (ours)	**0.845**	**0.832**	**0.780**	**0.707**	**0.902**	**0.851**	**0.696**	**0.622**
GLM-5.2	DLcR (Bai et al., 2025)	0.322	0.311	0.163	0.127	0.378	0.342	0.049	0.038
	RAG (Lewis et al., 2020)	0.393	0.381	0.199	0.158	0.430	0.385	0.066	0.062
	LongAgent (Zhao et al., 2024a)	0.726	0.581	0.325	0.338	0.692	0.557	0.142	0.127
	CoA (Zhang et al., 2024)	0.693	0.718	0.302	0.356	0.648	0.589	0.124	0.134
	MAIN-RAG (Chang et al., 2025)	0.642	0.852	0.338	0.496	0.497	0.812	0.179	0.198
	BELLE (Zhang et al., 2025a)	0.716	0.781	0.422	0.455	0.629	0.742	0.263	0.180
	LongTA (Li et al., 2026a)	0.762	0.803	0.445	0.475	0.843	0.834	0.331	0.217
	Sensemaker (ours)	**0.890**	**0.875**	**0.818**	**0.729**	**0.937**	**0.894**	**0.753**	**0.687**
Qwen3.5	DLcR (Bai et al., 2025)	0.317	0.331	0.157	0.119	0.371	0.344	0.039	0.039
	RAG (Lewis et al., 2020)	0.384	0.380	0.187	0.147	0.410	0.392	0.065	0.062
	LongAgent (Zhao et al., 2024a)	0.671	0.536	0.271	0.305	0.621	0.506	0.102	0.113
	CoA (Zhang et al., 2024)	0.643	0.672	0.253	0.326	0.587	0.539	0.088	0.122
	MAIN-RAG (Chang et al., 2025)	0.611	0.816	0.310	0.472	0.462	0.773	0.154	0.187
	BELLE (Zhang et al., 2025a)	0.674	0.738	0.374	0.424	0.558	0.697	0.225	0.168
	LongTA (Li et al., 2026a)	0.721	0.827	0.400	0.488	0.779	0.743	0.284	0.202
	Sensemaker (ours)	**0.808**	**0.841**	**0.743**	**0.661**	**0.872**	**0.838**	**0.671**	**0.581**

Sensemaker consistently outperforms all baseline methods. As reported in Table 2, Sensemaker achieves the highest $Sco.$ in all 18 evaluation settings (three backbone models $\times$ six benchmarks). Across the six benchmarks, Sensemaker achieves $Sco.$ of 0.668, 0.706, and 0.644 under Kimi-K2.6, GLM-5.2, and Qwen3.5, respectively, corresponding to maximum improvements of 0.298, 0.309, and 0.287 over the baseline methods. These consistent gains across backbone models and task domains demonstrate the effectiveness of explicit and fine-grained capability modeling of multi-document analysis in Sensemaker.

Sensemaker performs high-quality analysis over the provided evidence. Since every question in the four benchmarks (*i.e.*, xAFS, FRAMES, MuDA, and RxA) is associated with a gold file set containing the files required to answer it, we replace the original file library for each question with

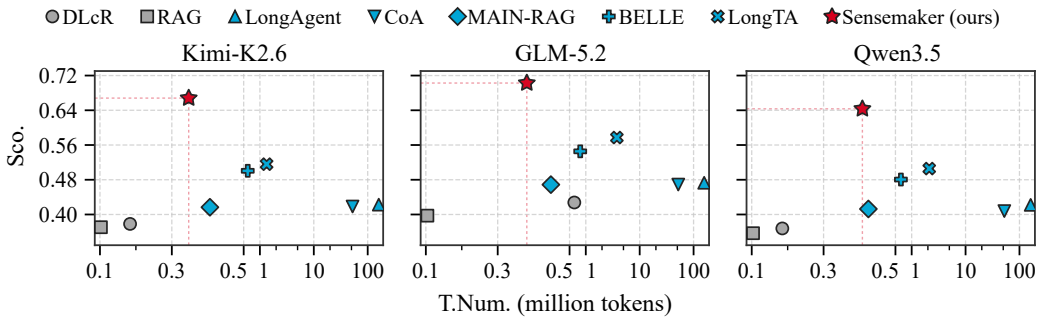


Figure 4: Comparison of reasoning efficiency. Points closer to the upper-left corner indicate higher answer quality with lower token usage. DLcR and RAG are shown in gray because each produces the final answer with a single LLM call, whereas all other methods perform iterative multi-step inference. They are included for completeness but excluded from efficiency comparisons.

the gold file set and report the answer scores $Sco._{(gold)}$ of all the methods in Figure 3. In all 12 settings (three backbone models $\times$ four benchmarks), Sensemaker achieves the highest $Sco._{(gold)}$, with a maximum improvement of 0.405 in the average score over the baseline methods. The performance gaps indicate that the advantage of Sensemaker is not limited to finding relevant files. By explicitly modeling the capabilities required for multi-document analysis and providing them with specific tools, Sensemaker can effectively convert the provided gold files into correct answers.

Sensemaker achieves accurate evidence retrieval. For each method, we compute the file-level recall R and precision P on the four benchmarks that provide gold file sets (see Appendix D.2 for detailed formulas). As reported in Table 3, the evidence sets constructed by Sensemaker cover more relevant files and fewer irrelevant files than those produced by the baseline methods, with maximum gains of 0.704 in R and 0.649 in P. We attribute this advantage to the effectiveness of the capability modules in Sensemaker and the high-quality coordination among them, which progressively refine task understanding of the system and enable targeted retrievals, particularly for evidence whose relevance emerges only through cross-file analysis and iterative exploration.

Sensemaker performs efficient multi-document analysis. For each task, we count all the tokens across agents from task input to final answer generation. Figure 4 plots the average token count across benchmarks, $T.Num.$ (in millions), against the corresponding average answer score. Compared with the baseline methods that perform multi-step inference, Sensemaker achieves higher $Sco.$ with lower $T.Num.$. For instance, when using Kimi-K2.6 as the backbone model, Sensemaker consumes only 17.38% of the tokens used by LongTA while improving the $Sco.$ by 0.153. These results indicate that Sensemaker enables high-quality multi-document analysis at a low inference cost.

4.3 Performance Investigation

We conduct in-depth investigations on Sensemaker to analyze its effectiveness. For the backbone model, we use GLM-5.2 because it achieves the best overall performance among the three models in the main experiments. For the benchmarks, we select MuDA, FRAMES, and RxA to cover different file library sizes. We construct five module-level variants ("Sensemaker w/o dedicated FL, FP, CE, DP, or EV") and three architecture-level variants ("Sensemaker w/o capability instantiation", "Sensemaker w/ general subagent", and "Sensemaker w/o capability layer"). The detailed descriptions of these variants are presented in Appendix E. We report the results in Table 4, with each metric averaged over the three benchmarks.

Explicit capability modeling is important for MDocAQA. The variants progress from no capability modeling (Sensemaker w/o capability layer and Sensemaker w/ general subagent), through prompt-level modeling (Sensemaker w/o capability instantiation) and partial instantiation (five module-level variants), to the complete capability layer (Sensemaker). Table 4 shows that all variants underperform Sensemaker, with performance improving as capability modeling becomes more explicit. Moreover, compared with "Sensemaker w/o capability layer", "Sensemaker w/ general subagent" yields only a small gain in Sco. (0.409 to 0.418), while Sensemaker reaches 0.667, suggesting a greater benefit from capability modeling. The results indicate that specialized, indepen-

Table 4: Ablation results for Sensemaker. The format of the results is "(average result)(signed difference from Sensemaker)". "↑" means that higher values are better, and "↓" means that lower values are better.

Variant	$Sco.\uparrow$	$Sco._{(gold)}\uparrow$	$R\uparrow$	$P\uparrow$	$T.Num.\downarrow$
Sensemaker (ours)	0.667	0.775	0.836	0.770	0.405
w/o dedicated FL	0.548-0.119	0.768-0.007	0.586-0.250	0.735-0.035	0.501+0.096
w/o dedicated FP	0.579-0.088	0.754-0.021	0.774-0.062	0.558-0.212	0.474+0.069
w/o dedicated CE	0.522-0.145	0.739-0.036	0.752-0.084	0.716-0.054	0.726+0.321
w/o dedicated DP	0.492-0.175	0.684-0.091	0.810-0.026	0.748-0.022	0.445+0.040
w/o dedicated EV	0.536-0.131	0.724-0.051	0.790-0.046	0.746-0.024	0.369-0.036
w/o capability instantiation	0.455-0.212	0.655-0.120	0.520-0.316	0.492-0.278	0.820+0.415
w/ general subagent	0.418-0.249	0.597-0.178	0.462-0.374	0.478-0.292	1.133+0.728
w/o capability layer	0.409-0.258	0.603-0.172	0.449-0.387	0.421-0.349	1.040+0.635

dently executable capability modules support reliable, high-quality multi-document analysis through structured coordination, dedicated execution, and explicit intermediate states.

FL primarily improves coverage of the required files. Removing the dedicated FL module decreases R by 0.250 relative to Sensemaker, the largest recall loss among the module-level variants, whereas P and $Sco._{(gold)}$ decline by only 0.035 and 0.007, respectively. The large reduction in R indicates that FL helps the system obtain the required information more completely.

FP helps distinguish relevant files from irrelevant candidates. Without the dedicated FP module, P decreases by 0.212, the largest precision loss among the module-level variants, while R declines by only 0.062. This contrast indicates that the file descriptions and assessments of file roles produced by FP help the system identify files that truly contribute to the task.

CE enables effective and efficient extraction of fine-grained evidence. Removing the dedicated CE module decreases $Sco.$ by 0.145 (the second-largest reduction among the module-level variants) and increases $T.Num.$ by 0.321 (the largest increase among the module-level variants). These results indicate that the dedicated CE module enables effective and efficient extraction of usable evidence from selected files, thereby improving answer quality while reducing reasoning cost.

DP plays a central role in converting available evidence into correct answers. Removing the dedicated DP module causes the largest reductions in both $Sco.$ and $Sco._{(gold)}$ among the module-level variants. In contrast, R and P decrease by only 0.026 and 0.022, respectively. This pattern indicates that DP mainly contributes to downstream evidence processing, including statistical analysis and numerical computation, after identifying relevant files.

EV improves task performance across multiple dimensions. Removing the dedicated EV module slightly reduces reasoning cost (decreasing $T.Num.$ by 0.036) but leads to degradation across all four remaining metrics. The modest overhead reflects the additional reasoning required to assess the validity of the evidence. The resulting gain in task performance outweighs this additional cost.

5 Conclusion

In this work, we identify two critical limitations faced by existing MDocAQA systems, namely agent overload and process opacity. To address these limitations, we propose Sensemaker, which organizes multi-document analysis as interactions among three layers (*i.e.*, the task, capability, and tool layers) and explicitly defines five modules in the capability layer. The task-layer agent maintains global coordination, while five specialized modules perform the corresponding operations with dedicated tools. Under this design, each agent can focus on a clearly defined responsibility, thereby effectively executing the operations associated with its capability. Comprehensive experiments across multiple MDocAQA benchmarks confirm that Sensemaker outperforms various representative MDocAQA methods. More broadly, we believe that the issues observed in existing MDocAQA methods may reflect a broader limitation of agentic systems that coordinate heterogeneous capabilities, motivating the exploration of explicit capability-level modeling as a promising direction for developing reliable agentic systems across domains.

AI USE STATEMENT

In this work, we used generative AI tools to evaluate system outputs on the benchmarks whose evaluation protocols prescribe LLM-based evaluation. We did not use generative AI tools to propose or refine research hypotheses, develop the conceptual framework, design the methodology or experiments, implement the proposed method, generate or process datasets, or analyze or interpret the resulting evaluation scores. The required disclosure tasks concerning theoretical models, mathematical claims and proofs, and translation are not applicable to this work. Additionally, we used generative AI tools for language editing to improve the grammar, clarity, concision, and readability of the manuscript.

We reviewed all AI-assisted work. Every AI-assisted language edit was examined by the authors and retained only after confirming that it preserved the intended technical meaning. For LLM-based evaluation, we followed the evaluation protocol presented in Appendix D.1 and applied the same evaluator procedure consistently to all compared methods within each benchmark. The authors verified the evaluation implementation and score aggregation and independently analyzed and interpreted the resulting scores. All research ideas, technical decisions, experimental conclusions, and scientific claims were developed by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our research focuses solely on scientific and methodological questions and does not involve human-subject or animal studies. Any potentially confidential information contained in the industrial data constructed in this work will be removed or anonymized before public release. We anticipate no significant ethical risks or conflicts of interest associated with this study. We are committed to upholding high standards of scientific integrity and research ethics to ensure the validity and reliability of our findings.

REPRODUCIBILITY STATEMENT

We document the information needed to independently re-implement Sensemaker and reproduce its experimental results throughout the paper and appendices.

For Sensemaker, the efforts that have been made to ensure reproducibility are as follows.

- Section 3 presents the overall architecture and formalizes the task, capability, and tool layers.
- Appendix F specifies the implementation settings, including the reasoning budgets, scheduling rules, concurrency limits, and tool-execution timeouts.
- Appendix G documents the structured invocation syntax, arguments, and example call for each tool.

For the experimental results, the efforts that have been made to ensure reproducibility are as follows.

- Section 4.1 describes the experimental setup, including the baseline methods, backbone models, benchmarks, and evaluation protocols.
- Appendix B provides the implementation details of each baseline method.
- Appendix C presents the descriptions of the adopted benchmarks. For RxA, the benchmark introduced in this work, we document its project scope, file-library composition, question categories, and representative examples.
- Appendix D provides the definitions and calculation formulas of the evaluation metrics in the experiments.
- Appendix E specifies the implementation of all Sensemaker variants used in the performance investigation.

Together, these materials explain how Sensemaker and the baseline methods are implemented, how the experiments are conducted, and how the reported scores are computed. They support independent re-implementation and verification of the reported findings. The complete codebase for

Sensemaker will be made publicly available upon acceptance. We will also make a sanitized version of RxA publicly available, with confidential information from the underlying industrial project removed or anonymized.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

Chia-Yuan Chang, Zhimeng Jiang, Vineeth Rakesh, Menghai Pan, Chin-Chia Michael Yeh, Guanchu Wang, Mingzhi Hu, Zhichao Xu, Yan Zheng, Mahashweta Das, et al. Main-rag: Multi-agent filtering retrieval-augmented generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2607–2622, 2025.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4745–4759, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Chuhan Li, Ziyao Shangguan, Yilun Zhao, Deyuan Li, Yixin Liu, and Arman Cohan. M3sciqa: A multi-modal multi-document scientific qa benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15419–15446, 2024.

Yiyang Li, Zheyuan Zhang, Tianyi Ma, Zehong Wang, Keerthiram Murugesan, Chuxu Zhang, and Yanfang Ye. Longda: Benchmarking llm agents for long-document data analysis. *arXiv preprint arXiv:2601.02598*, 2026a.

Zhanli Li, Yixuan Cao, Lvzhou Luo, and Ping Luo. Navigating large-scale document collections: Mudabench for multi-document analytical qa. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6877–6898, 2026b.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Moonshot AI. Kimi k2.6: Advancing open-source coding. Kimi Technical Blog, 2026. URL `https://www.kimi.com/blog/kimi-k2-6`.

Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of the International Conference on Intelligence Analysis*, 2005.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *Proceedings of the International Conference on Learning Representations*, pp. 9695–9717, 2024.

Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents. Qwen Technical Blog, 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

Jon Saad-Falcon, Joe Barrow, Alexa Siu, Ani Nenkova, Seunghyun Yoon, Ryan A Rossi, and Franck Dernoncourt. Pdftriage: Question answering over long, structured documents. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 153–169, 2024.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Supermemory. xafs: A benchmark for extendable agentic file systems. Hugging Face, 2026. URL https://huggingface.co/datasets/supermemory/xAFS.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11897–11916, 2024a.

Minzheng Wang, Longze Chen, Cheng Fu, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, et al. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 5627–5646, 2024b.

Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 19206–19214, 2024c.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Zukang Yang, Zixuan Zhu, and Jennifer Zhu. Curiousllm: Elevating multi-document question answering with llm-enhanced knowledge graph reasoning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 274–286, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Z.ai. Glm-5.2: Built for long-horizon tasks. Z.ai Technical Blog, 2026. URL https://z.ai/blog/glm-5.2.

Taolin Zhang, Dongyang Li, Qizhou Chen, Chengyu Wang, and Xiaofeng He. Belle: A bi-level multi-agent reasoning framework for multi-hop question answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4184–4202, 2025a.

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

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arık. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

Jun Zhao, Can Zu, Hao Xu, Yi Lu, Wei He, Yiwen Ding, Tao Gui, Qi Zhang, and Xuan-Jing Huang. Longagent: achieving question answering for 128k-token-long documents through multi-agent collaboration. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 16310–16324, 2024a.

Qingfei Zhao, Ruobing Wang, Yukuo Cen, Daren Zha, Shicheng Tan, Yuxiao Dong, and Jie Tang. Longrag: A dual-perspective retrieval-augmented generation paradigm for long-context question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 22600–22632, 2024b.

A Relation to the Expert Sensemaking Model

Sensemaker is inspired by the expert sensemaking model proposed by Pirolli & Card (2005). The model describes information analysis as an iterative process composed of distinct information processing activities and organizes these activities into two major loops. The foraging loop focuses on finding useful information from a large information space and extracting evidence for further analysis. The sensemaking loop focuses on organizing and examining the collected evidence to develop conclusions. Importantly, these activities do not follow a fixed sequence. New evidence can trigger further search, while the current understanding can reveal missing information and cause the analyst to revisit earlier activities.

Sensemaker adopts this functional view of analysis and adapts it to MDocAQA. Specifically, we model five recurring information processing capabilities, namely file localization (FL), file profiling (FP), content extraction (CE), data processing (DP), and evidence verification (EV). Each capability is implemented as an independently executable module with a specialized agent and dedicated tools, while the task layer dynamically coordinates these modules according to the question and the accumulated evidence. The relation between these modules and the activities in the expert sensemaking model is conceptual rather than a direct stage mapping.

Specifically, the first three modules mainly correspond to information acquisition and evidence construction in the foraging loop. FL searches the file library and narrows the candidate set according to the current information need. FP provides compact descriptions of candidate files and supports further relevance assessment before detailed reading. CE then reads targeted files and extracts the fine-grained evidence required by the current analysis. Together, these modules transform a large file library into a smaller set of explicit and traceable evidence.

DP and EV support later analysis over the collected evidence. DP handles computations and aggregation that cannot be obtained by directly reading a single source. Its outputs are therefore derived evidence rather than facts explicitly stated in files. EV focuses on evidentiary reasoning by checking whether analytical claims are supported by their attributed source spans. This separation allows evidence transformation and evidence checking to be performed as explicit and inspectable operations.

Another important connection lies in iterative coordination. In the expert sensemaking model, information acquisition and higher-level analysis repeatedly affect each other. Sensemaker realizes this property through the task layer. After each capability module returns a structured result, the task layer updates the current analysis state and decides which capability is required next. A capability can therefore be invoked multiple times during one task. For example, newly extracted evidence may reveal the need for another file search, while a computed result may require source verification. Sensemaker thus coordinates different capabilities according to the evolving evidence instead of following a predefined pipeline.

Overall, the expert sensemaking model provides two main design principles for Sensemaker. First, complex analysis can be viewed as the coordination of distinct information processing activities. Second, these activities interact iteratively as the available evidence and current understanding evolve. Sensemaker adapts these principles to MDocAQA by explicitly modeling five information processing capabilities and coordinating them through the task layer.

B Introduction to Baseline Methods

This section provides descriptions of the baseline methods and their implementation details in our experiments.

Direct Long-context Reasoning (Bai et al., 2025). This method concatenates the contents of all files in a library into a single context and provides the context directly to the backbone model for answer generation. When the concatenated file context exceeds the maximum input length supported by the backbone model, the file context is truncated from the end to ensure that the resulting input remains within this limit. Regarding the order of file concatenation, we apply a fixed-seed random permutation, following the document-shuffling protocol used in Loong (Wang et al., 2024b). The resulting order is kept identical across all backbone models and repeated runs.

Retrieval-Augmented Generation (Lewis et al., 2020). This method divides each file into overlapping chunks and ranks all chunks according to their semantic similarity to the question. It then provides the top-k chunks, together with the question, to the backbone model for answer generation. In our experiments, we split the content of all the files into chunks of 5,120 tokens with an overlap of 1,000 tokens and retrieve the top-20 chunks for each question. Moreover, we adopt Qwen3-Embedding-8B (Zhang et al., 2025b) as the embedding model and cosine similarity to measure semantic similarity. When the concatenated retrieved context exceeds the maximum input length of the backbone model, we apply the same truncation strategy as in Direct Long-context Reasoning.

LongAgent (Zhao et al., 2024a). This method adopts a divide-and-conquer strategy by partitioning the contents of all files into multiple text chunks and assigning each chunk to a separate member agent. In each interaction round, a leader agent interprets the current progress and broadcasts the next-step instruction to all members. Each member agent reads its assigned chunk and returns clues relevant to the instruction. If the clues provided by different member agents conflict, the leader agent invokes inter-member communication, during which the relevant member agents share their assigned chunks to reach a consensus. In our experiments, following the official implementation (Zhao et al., 2024a), we set the maximum number of interaction rounds to three. Moreover, we set the text chunk size according to the maximum context length supported by the corresponding backbone model.

Chain-of-Agents (Zhang et al., 2024). This method first concatenates the contents of all files in a library and partitions the resulting long context into an ordered sequence of segments, each short enough to be processed by one worker agent. The worker agents process these segments sequentially. Each worker receives the question, its assigned segment, and the communication unit produced by the preceding worker. It then updates the communication unit by incorporating relevant information from its assigned segment and passes the revised unit to the next worker. After the final segment has been processed, a manager agent uses the final communication unit to generate the answer. In our experiments, we set the text chunk size according to the maximum context length supported by the corresponding backbone model.

MAIN-RAG (Chang et al., 2025). This method first uses the same semantic retriever as RAG to retrieve up to 20 candidate chunks for the question. A predictor agent generates a tentative answer from each (question, chunk) pair, forming a (question, chunk, answer) triplet. A judge agent then evaluates whether each triplet provides relevant information for answering the question and assigns it a score based on the difference between the log probabilities of "Yes" and "No". The candidate chunks are ranked by these scores and filtered using an adaptive threshold derived from their score distribution. Finally, a final-predictor agent generates the answer from the retained chunks in ranked order. In our experiments, we split the content of all the files into chunks of 5,120 tokens with an overlap of 1,000 tokens and adopt Qwen3-Embedding-8B as the embedding model.

BELLE (Zhang et al., 2025a). This method first classifies the question into one of four types: Inference, Comparison, Temporal, or Null. It then employs a bi-level multi-agent debate to construct an execution plan by selecting and combining five reasoning operators: chain-of-thought reasoning, single-step retrieval, iterative-step retrieval, sub-question decomposition, and adaptive-step retrieval. At the first level, affirmative and negative debaters propose and challenge different operator combinations. At the second level, a fast debater evaluates the operator selection in the current debate round, while a slow debater integrates the debate history; a judge then determines the final execution plan. A multi-hop QA executor invokes the selected operators step by step and integrates their outputs to produce the final answer. In our experiments, following Zhang et al. (2025a), we adopt file-level BM25 retrieval (returning the top-20 files for each retrieval) and set the maximum number of debate rounds to three.

LongTA (Li et al., 2026a). This method employs a ReAct-style agent (Yao et al., 2022) that iteratively accesses files, retrieves relevant information, records intermediate findings, and executes Python code for data analysis. The framework provides tools for reading selected document units, performing exact keyword search, retrieving relevant text using BM25, maintaining external notes, and executing analytical code. In our experiments, we retain the tool organization and reasoning loop described in Li et al. (2026a), while setting the maximum reasoning budget to 200 steps.

Table 5: Representative problems in RxA.

Type	Problem
Physical Design	What are the minimum and maximum permitted widths for each routing metal layer? What is the minimum same-layer spacing for each routing metal layer, and does the spacing requirement vary with wire width?
Verification Rules	What are the minimum and maximum metal-density limits for each layer, and what dummy-metal-fill requirements apply? What is the maximum allowable antenna ratio for each metal layer, and which layers require particular attention during antenna-violation repair?
Timing Constraints	What input and output delay values are specified by the SDC constraints? Which timing paths are declared as false paths by the SDC constraints?
Implementation Diagnostics	What case-analysis values are specified by the SDC constraints, and to which pins do these values propagate? What is the worst negative slack (WNS) after placement, and which endpoints exhibit the most severe timing violations?

C Introduction to Benchmarks

This section introduces the six benchmarks used in our experiments.

LongBench v2 (Bai et al., 2025). It is a long-context benchmark comprising 503 multiple-choice questions across six task categories. We use only its multi-document QA subset, which contains 125 questions covering academic, legal, financial, government, and news documents. Each question in this subset is associated with multiple documents whose combined context has a median length of 34,000 words.

Loong (Wang et al., 2024b). It is a bilingual multi-document analytical QA benchmark containing 1,600 questions drawn from financial reports, legal cases, and academic papers. The file library associated with each question contains 11 documents on average, with a total context length ranging from 10,000 to 250,000 tokens.

xAFS (Supermemory, 2026). It provides 13 synthetic but realistic file libraries that include emails, Slack exports, meeting transcripts, laboratory notes, contracts, journals, and code reviews. These libraries range in size from 5 to approximately 10,000 files. Based on the information contained in these libraries, xAFS presents 110 questions.

FRAMES (Krishna et al., 2025). It contains 824 human-annotated questions requiring evidence from Wikipedia, covering numerical, tabular, multi-constraint, temporal, and post-processing reasoning. Each question requires evidence from 2 to 15 Wikipedia articles. In our experiments, we collect all evidence articles listed by the benchmark (2,507 articles) and compile them into a file library for all the questions. Agents must rely on the information in this library to answer each question.

MuDABench (Li et al., 2026b). It is a financial analytical QA benchmark containing 332 questions and 589 documents in total. The documents include annual reports, corporate announcements, and environmental, social, and governance reports from publicly listed companies in China and the United States. Together, these documents span more than 80,000 pages. Each question is associated with a file library containing 5 to 38 documents from this collection.

RxA. It is an MDocAQA benchmark derived from a real digital integrated circuit project. Its file library covers logic synthesis, place-and-route, physical verification, parasitic extraction, and static timing analysis. The library contains more than 13,000 heterogeneous files, including flow scripts,

design constraints, intermediate artifacts, logs, and reports. Some logs and reports contain hundreds of thousands of lines.

EDA engineers provided 30 MDocAQA questions based on problems encountered in their actual work. These questions cover physical design, verification rules, timing constraints, and implementation diagnostics. Answering them requires identifying relevant files within the project library and combining evidence across multiple files. Table 5 presents representative questions from these four categories.

EDA engineers also wrote a detailed reference answer for each question. Each conclusion in the reference answer is accompanied by a citation to its supporting source in the file library, allowing the conclusion to be checked against the project files.

D Calculation Formulas of Evaluation Metrics

This section presents the formulas used to calculate the evaluation metrics in our experiments.

D.1 Average Answer Score *Sco.*

We uniformly use $Sco.$ to measure answer quality across all the adopted benchmarks. For each benchmark, we follow the calculation strategy in the corresponding paper.

LBv2. It evaluates model performance through exact matching between the predicted multiple-choice option and the ground-truth option. Formally, let $\hat{o}_j$ and o_j^* denote the predicted and reference options for the j-th question, respectively. The score is calculated as

$$Sco. = \frac{1}{N_{\mathrm{Ben}}} \sum_{j=1}^{N_{\mathrm{Ben}}} \mathbb{I}\left(\hat{o}_j = o_j^*\right), \quad \mathrm{Ben} \in \{\mathrm{LBv2}\}, \tag{7}$$

where N_{Ben} denotes the total number of questions in the benchmark Ben and $\mathbb{I}(\cdot)$ is the indicator function.

Loong and RxA. These benchmarks evaluate each generated answer using GPT-5.5, which assigns a score based on answer correctness, hallucination, and completeness. Formally, let $r_j \in \{1, \ldots, 100\}$ denote the rating assigned to the j-th answer. The normalized score is computed as

$$Sco. = \frac{1}{100 N_{\mathrm{Ben}}} \sum_{j=1}^{N_{\mathrm{Ben}}} r_j, \quad \mathrm{Ben} \in \{\mathrm{Loong}, \mathrm{RxA}\}. \tag{8}$$

The evaluation prompt (Wang et al., 2024b) is formatted as follows.

> *[Question]* `question`
> *[Gold Answer]* `reference_answer`
> *[The Start of Assistant's Predicted Answer]*
> `generated_answer`
> *[The End of Assistant's Predicted Answer]*
> *[System]*
> *Evaluate the AI assistant's predicted answer against the gold answer.*
> *Use the following criteria:*
> *1. Accuracy and Hallucinations*
> *- The predicted answer should be semantically consistent with the gold answer.*
> *- Numerical values and their ordering should be correct.*
> *- The answer should not contain hallucinated information.*
> *2. Completeness*
> *- The predicted answer should contain all key points required by the gold answer.*
> *- Additional elaboration beyond those key points is not required.*
> *Treat the gold answer as a correct answer to the question. Assign one overall score from 1 to 100. A higher score represents better performance. If the predicted answer fully satisfies all of the above criteria relative to the gold answer, assign 100.*

> *First, provide an evaluation explanation without bias. Then provide the score. The explanation should be no more than 100 words.*
> *Use this output structure:*
> *[start output]*
> *Evaluation evidence: EVALUATION_EXPLANATION*
> *Rating: [[SCORE]]*
> *[end output]*
> *Now perform the evaluation.*

Here, `question`, `reference_answer`, and `generated_answer` are replaced with the corresponding content of each evaluation sample.

xAFS, FRAMES, and MuDA. These benchmarks use GPT-5.5 to evaluate answer correctness through binary judgments. Formally, let $c_j \in \{0, 1\}$ indicate whether the generated answer for the j-th question is judged correct. The score is calculated as

$$Sco. = \frac{1}{N_{\text{Ben}}} \sum_{j=1}^{N_{\text{Ben}}} c_j, \quad \text{Ben} \in \{\text{xAFS}, \text{FRAMES}, \text{MuDA}\}. \tag{9}$$

Besides, for these three benchmarks, we follow the public evaluation criteria of each benchmark when prompting the evaluation LLM.

D.2 File-level Recall *R* and Precision *P*

For the benchmark $\text{Ben} \in \{\text{xAFS}, \text{FRAMES}, \text{MuDA}, \text{RxA}\}$, the file-level recall R and precision P are calculated as:

$$R = \frac{1}{N_{\text{Ben}}} \sum_{j=1}^{N_{\text{Ben}}} \frac{\left|\hat{\mathcal{F}}_j \cap \mathcal{F}_j^*\right|}{\left|\mathcal{F}_j^*\right|}, \qquad P = \frac{1}{N_{\text{Ben}}} \sum_{j=1}^{N_{\text{Ben}}} \frac{\left|\hat{\mathcal{F}}_j \cap \mathcal{F}_j^*\right|}{\left|\hat{\mathcal{F}}_j\right|}, \tag{10}$$

where $\hat{\mathcal{F}}_j$ denotes the file set referenced in the final answer to the j-th question and $\mathcal{F}_j^*$ denotes the gold file set assigned to the j-th question. When $\hat{\mathcal{F}}_j = \varnothing$, the precision of the j-th question is defined as zero.

To identify $\hat{\mathcal{F}}_j$, we append a file-attribution query after each agent generates its answer in each adopted method. Specifically, we instruct each agent to return the exact paths of all files whose contents contributed to the answer. After duplicate removal, the paths provided alongside the final answer constitute $\hat{\mathcal{F}}_j$. The specific prompt is formatted as follows.

> *After generating the final answer to the problem, you must list all and only the files in the repository whose contents contributed to the answer. Return their exact repository-relative paths as a JSON array of strings, with no additional text. If no file contributed, return an empty array ('[]').*

E Introduction to Variants of Sensemaker

This section introduces eight variants of Sensemaker adopted in our experiments.

Five Module-level Variants. In each variant, one capability is no longer implemented by a dedicated module but by the task-layer agent, which is equipped with the corresponding tools, while the remaining four modules remain unchanged.

Three Architecture-level Variants. These variants remove the entire capability layer of Sensemaker and expose all tools directly to the task-layer agent. In "Sensemaker w/o capability instantiation", the five capability modules retain their original system prompts and tool permissions, but are no longer instantiated as separate sessions. Instead, each MDocAQA task is handled by a single persistent worker session. The task-layer agent alternates between planning and capability execution by switching to the selected role while retaining the complete task trajectory. Consequently, all capabilities are executed sequentially by the same agent within a single execution trajectory. In

"Sensemaker w/o capability layer", the task-layer agent receives the general prompt shown below, which does not explicitly define these capabilities.

> ***Role***
>
> *You are an autonomous, general-purpose agent for Multi-Document Analytical Question Answering (MDocAQA). You operate in a single continuous session and directly use the available tools to answer questions over a file library.*
>
> ***Objective***
>
> *Produce an accurate, complete, and evidence-grounded answer to the question. The required evidence may be distributed across multiple files and may only become identifiable through iterative exploration and cross-file analysis. Questions may involve heterogeneous file types, large repositories, multi-hop reasoning, comparisons, temporal constraints, numerical calculations, aggregation, or other forms of analytical reasoning.*
>
> *Treat the project files as the authoritative source for repository-specific facts. Do not substitute model memory for evidence that should be obtained from the files.*
>
> ***General Instructions***
>
> - *Understand the question before taking action. Identify its scope, target entities, constraints, time period, units, and required answer format.*
> - *Keep brief notes on key facts. Record the exact file and location for each fact, and track the files that directly support the final answer.*
> - *If sources conflict, compare their context, scope, date, and terminology. If the conflict cannot be resolved, clearly report it.*
> - *Never invent file contents, tool results, source paths, or numerical values. If the available evidence is still insufficient after a reasonable search, clearly state this limitation.*
>
> ***Available Tools***
>
> {*Introduction and Tool-Call Syntax of each tool* }
>
> *(*`search_profile`*,* `project_search`*,* `save_profile`*,* `range_read`*,* `regex_extract`*,* `run_python`*, and* `run_command`*)*
>
> ***Final Response***
>
> *You need to provide a concise answer with only the explanation needed to make the result clear and verifiable. Do not reveal hidden reasoning, internal plans, or working notes. Do not mention files that did not contribute to the answer.*
>
> *After generating the final answer to the problem, you must list all and only the files in the repository whose contents contributed to the answer. Return their exact repository-relative paths as a JSON array of strings, with no additional text. If no file contributed, return an empty array ('[]').*

Moreover, "Sensemaker w/ general subagent" extends "Sensemaker w/o capability layer" by moving all tool access from the task-layer agent to a general-purpose subagent. The task-layer agent cannot call tools. It organizes subtasks based on the current task requirements and delegates them to the subagent. Each subagent invocation starts an independent session without inheriting the context of previous invocations.

F Implementation Details of Sensemaker

This section introduces the implementation details of our proposed Sensemaker in the experiments, including reasoning budget, scheduling rules, and system prompts.

Reasoning Budget. For each question, the task-layer agent is allowed to invoke capability modules at most 30 times. Within each module invocation, the corresponding capability-layer agent can make at most 20 tool calls, and each tool execution is subject to a maximum timeout of 60 seconds. In practice, Sensemaker typically terminates well before reaching these upper bounds, as statistics aggregated over all 18 evaluation settings (three backbone models $\times$ six benchmarks) show that it invokes capability modules an average of 7.82 times per task, with an average of 9.46 tool calls per module invocation. We adopt relatively permissive reasoning budgets primarily to accommodate challenging cases involving large file libraries, particularly RxA, whose search space contains more than 13,000 files. The budgets therefore serve as safeguards against premature termination rather than as fixed computational allocations. The efficiency results in Figure 4 further show that the

performance gains of Sensemaker do not arise from consuming a larger reasoning budget. Instead, Sensemaker achieves higher answer quality with substantially lower token usage than the adopted baseline methods, indicating that its gains primarily result from explicit capability specialization and structured coordination.

Scheduling Rules. Both invocations of capability modules by the task-layer agent and tool calls within a capability module can be executed in parallel, with at most five concurrent calls. Each invocation of a capability module is initialized independently and does not inherit the execution context of previous invocations of the same module. Instead, the task-layer agent provides the information required for the current invocation based on the accumulated analysis results. Besides, for each question, the profile repository $\mathcal{D}$ is initialized as empty at the start of every run. Its contents remain available to subsequent module invocations within that run and are not retained across questions or repeated runs.

System Prompts. The system prompts of each agent in Sensemaker are instantiated from the template below.

> ***Role***
>
> *You are a {Role Name} agent for Multi-Document Analytical Question Answering (MDocAQA). You operate in a single continuous session and directly use the available agents/tools to {Responsibility}.*
>
> ***Objective***
>
> *As the responsibility mentioned in **Role**, you need to {Explanation of the Responsibility}.*
>
> *Attention: Use only information explicitly supplied in the current input or returned by a permitted operation during the current invocation. Do not rely on any other hidden memory or undeclared source of information.*
>
> ***General Instructions***
>
> - *Understand the question before taking action. Identify its scope, target entities, constraints, time period, units, and required answer format.*
> - *Keep brief notes on key facts. Record the exact file and location for each fact, and track the files that directly support the final answer.*
> - *If sources conflict, compare their context, scope, date, and terminology. If the conflict cannot be resolved, clearly report it.*
> - *Never invent file contents, tool results, source paths, or numerical values. If the available evidence is still insufficient after a reasonable search, clearly state this limitation.*
>
> *(If applicable)* ***Available Tools***
>
> *{Introduction and Tool-Call Syntax of Each Tool }*
>
> *(If applicable)* ***Available Agents***
>
> *{Introduction and Agent-Call Syntax of Each Agent }*
>
> ***Input Interpretation***
>
> *According to your responsibility, users will provide you with the following information:*
>
> *{Description of the Required Information }*
>
> *If the information provided by the user is incomplete, please do not supplement the missing information by yourself, but return the missing information to the user for supplementation. If the task assigned to you is beyond your scope of responsibility, please do not try to complete the task, but inform the user of the situation and explain the reason.*
>
> ***Output Requirements***
>
> *You need to provide a concise answer with only the explanation needed to make the result clear and verifiable. Do not reveal hidden reasoning, internal plans, or working notes. Do not mention files that did not contribute to the answer.*
>
> *Besides, you need to {Specific Requirements of the Responsibility}.*

G TOOL-CALL SYNTAX OF SENSEMAKER

The modules in the capability layer invoke each tool through a structured function call of the form `tool_call(tool=tool_name, argument1=value1, argument2=value2, ...)`. Tool observations are returned as structured JSON objects.

search_profile. It searches for information in $\mathcal{D}$ using one or more keywords, file names, path fragments, or regular expressions. When multiple patterns are supplied, matching records are merged and ranked according to the number of distinct matched patterns. The JSON representation of this tool call is as follows:

```
{
  "tool": "search_profile",
  "patterns": [
    "clock uncertainty",
    "set_clock_uncertainty"
  ],
  "max_results": 10
}
```

Here, `patterns` specifies one or more search expressions, such as keywords, file names, path fragments, or regular expressions. `max_results` specifies the maximum number of matching profiles returned by the tool. The JSON representation of the tool response is as follows:

```
{
  "tool": "search_profile",
  "status": "ok",
  "profiles": [
    {
      "profile_id": "profile_0001",
      "file_path": "constraints/top.sdc",
      "required":
        "Which clock-related settings are defined?",
      "description":
        "Sets clock uncertainty: set_clock_uncertainty.",
      "matched_patterns": [
        "clock uncertainty",
        "set_clock_uncertainty"
      ],
      "match_count": 2
    }
  ],
  "errors": []
}
```

project_search. It retrieves candidate files within a bounded search scope by filtering file patterns and matching keywords. The JSON representation of this tool call is as follows:

```
{
  "tool": "project_search",
  "patterns": [
    "clock uncertainty",
    "set_clock_uncertainty"
  ],
  "max_results": 20,
  "workdir": "constraints/",
  "timeout_seconds": 60
}
```

Here, `patterns` specifies the keywords or regular-expression patterns to be matched. `max_results` limits the number of search results returned by the tool. `workdir` specifies the working directory in which the search is performed. `timeout_seconds` specifies the maximum execution time, in seconds, allowed for the search. The JSON representation of the tool response is as follows:

```
{
```

```
  "tool": "project_search",
  "status": "ok",
  "workdir": "constraints/",
  "files": [
    {
      "file_path": "constraints/top.sdc",
      "matches": [
        {
          "pattern": "set_clock_uncertainty",
          "target": "content",
          "start_line": 40,
          "end_line": 40,
          "text":
            "set_clock_uncertainty 0.1 [all_clocks]"
        }
      ]
    }
  ],
  "errors": []
}
```


`save_profile`. It stores or updates the profile of a file in $\mathcal{D}$. The JSON representation of this tool call is as follows:


```
{
  "tool": "save_profile",
  "file_path": "constraints/top.sdc",
  "required": "What is the content of the document?",
  "description": "Define clocks and timing exceptions"
}
```


Here, `file_path` specifies the path of the file whose profile is to be stored or updated. `required` describes the required information for the file. `description` provides a concise description of the file. The JSON representation of the tool response is as follows:


```
{
  "tool": "save_profile",
  "status": "ok",
  "action": "created",
  "profile": {
    "profile_id": "profile_0001",
    "file_path": "constraints/top.sdc",
    "required": "What is the content of the document?",
    "description": "Define clocks and timing exceptions"
  },
  "errors": []
}
```


`range_read`. It extracts the content of specified line ranges from one or more files. The content of overlapping or adjacent ranges is merged by default. The JSON representation of this tool call is as follows:


```
{
  "tool": "range_read",
  "merge_overlapping": true,
  "ranges": [
    {
      "file_path": "constraints/top.sdc",
      "start_line": 40,
      "end_line": 65
```

```
      },
      {
        "file_path": "reports/timing.rpt",
        "start_line": 120,
        "end_line": 155
      }
    ]
  }
```

Here, `merge_overlapping` specifies whether overlapping or adjacent ranges should be combined before their content is returned. Each range object contains a `file_path` identifying the target file, a `start_line` specifying the first line to be read, and an `end_line` specifying the last line to be read. The JSON representation of the tool response is as follows:

The JSON representation of the tool response is as follows:

```
{
  "tool": "range_read",
  "status": "ok",
  "ranges": [
    {
      "request_indices": 0,
      "status": "ok",
      "source_span": {
        "file_path": "constraints/top.sdc",
        "start_line": 40,
        "end_line": 65,
        "text": "<full source text from lines 40 to 65>"
      }
    },
    {
      "request_indices": 1,
      "status": "ok",
      "source_span": {
        "file_path": "reports/timing.rpt",
        "start_line": 120,
        "end_line": 155,
        "text": "<full source text from lines 120 to 155>"
      }
    }
  ],
  "errors": []
}
```

`regex_extract`. It extracts matching content and contextual information from specified files using multiple regular-expression patterns. The JSON representation of this tool call is as follows:

```
  {
    "tool": "regex_extract",
    "requests": [
      {
        "file_path": "constraints/top.sdc",
        "patterns": [
          "set_clock",
          "set_false_path"
        ]
      },
      {
        "file_path": "scripts/setup.tcl",
        "patterns": [
```

```
                    "create_clock"
                  ]
                }
              ]
            }
```

Here, each `file_path` identifies a file from which matching content is extracted, while the corresponding `patterns` specify one or more regular expressions to be applied to that file. The JSON representation of the tool response is as follows:

The JSON representation of the tool response is as follows:


```
{
  "tool": "regex_extract",
  "status": "ok",
  "results": [
    {
      "request_index": 0,
      "file_path": "constraints/top.sdc",
      "pattern": "set_clock",
      "status": "ok",
      "matches": [
        {
          "match_text": "set_clock",
          "match_start_line": 40,
          "match_end_line": 40,
          "source_span": {
            "file_path": "constraints/top.sdc",
            "start_line": 40,
            "end_line": 40,
            "text":
              "set_clock_uncertainty 0.1 [all_clocks]"
          }
        }
      ]
    },
    {
      "request_index": 0,
      "file_path": "constraints/top.sdc",
      "pattern": "set_false_path",
      "status": "ok",
      "matches": [
        {
          "match_text": "set_false_path",
          "match_start_line": 58,
          "match_end_line": 58,
          "source_span": {
            "file_path": "constraints/top.sdc",
            "start_line": 58,
            "end_line": 58,
            "text":
              "set_false_path -from [get_ports rst_n]"
          }
        }
      ]
    },
    {
      "request_index": 1,
      "file_path": "scripts/setup.tcl",
      "pattern": "create_clock",
```

```
            "status": "ok",
            "matches": [
              {
                "match_text": "create_clock",
                "match_start_line": 12,
                "match_end_line": 12,
                "source_span": {
                  "file_path": "scripts/setup.tcl",
                  "start_line": 12,
                  "end_line": 12,
                  "text":
                    "create_clock -period 10 [get_ports clk]"
                }
              }
            ]
          }
        ],
        "errors": []
      }
```


run_python. It executes a Python program for structured parsing, numerical computation, or lightweight data transformation. File access is restricted to the current project, while file writes and subprocess creation are prohibited. The JSON representation of this tool call is as follows:


```
      {
        "tool": "run_python",
        "workdir": ".",
        "python_code": "print(sum([0.71, 0.68, 0.73]))",
        "timeout_seconds": 60
      }
```


Here, `workdir` specifies the working directory in which the Python program is executed. `python_code` contains the Python source code to be executed. `timeout_seconds` specifies the maximum execution time, in seconds, allowed for the program. The JSON representation of the tool response is as follows:


```
      {
        "tool": "run_python",
        "status": "ok",
        "workdir": ".",
        "stdout": "2.12\n",
        "python_code": "print(sum([0.71, 0.68, 0.73]))",
        "exit_code": 0,
        "timed_out": false,
        "errors": []
      }
```


run_command. It executes a shell command within the current project directory. Commands that write, modify, move, or delete files, or access paths outside the project boundary are rejected. The JSON representation of this tool call is as follows:


```
    {
      "tool": "run_command",
      "workdir": "constraints/",
      "shell_command": "rg -n 'set_clock|set_path' reports",
      "timeout_seconds": 60
    }
```


Here, `workdir` specifies the working directory in which the command is executed. `shell_command` contains the shell command to be executed. `timeout_seconds` specifies

the maximum execution time, in seconds, allowed for the command. To reinforce capability boundaries, we restrict the commands that each module can execute through `run_command`. Specifically, FL and CE are limited to `rg`. FP is limited to `file`, `stat`, `head`, and `tail`. DP is limited to `sort`, `uniq`, and `wc`. EV is limited to `sed`, `diff`, and `cmp`. The JSON representation of the tool response is as follows:


```
{
  "tool": "run_command",
  "status": "ok",
  "workdir": "constraints/",
  "shell_command": "rg -n 'set_clock|set_path' reports",
  "stdout":
    "40:set_clock_uncertainty 0.1 [all_clocks]\n",
  "timed_out": false,
  "errors": []
}
```


H CASE STUDY

Question: *For the Chinese A-share market, calculate the range of the total operating costs of the companies in the document collection in 2023.*
Answer: *157,478,863,925.30 CNY.*
Description: This question is from MuDABench (Li et al., 2026b). The file library associated with this question contains 14 reports. The maximum is CNY 158,164,000,000.00 for stock 601328, whereas the minimum is CNY 685,136,074.70 for stock 000681. Therefore, the reference answer is $158{,}164{,}000{,}000.00 - 685{,}136{,}074.70 = 157{,}478{,}863{,}925.30$ CNY.

LongAgent (Zhao et al., 2024a). Member agents need to perform all the information processing capabilities within their corresponding file chunks, while the leader agent needs to not only plan the overall process but also perform data processing, statistical analysis, and other operations on all the information returned by member agents.

1. **Leader Agent:** It decomposes the task into (i) exhaustive per-report extraction and (ii) a global maximum–minimum calculation. It broadcasts the same instruction to all the member agents.

   Instruction to Member Agents: "Your assigned chunk belongs to one of the 14 identified 2023 A-share annual reports. Search only this chunk for the current-period item corresponding to *total operating cost*. If it is present, return a tuple $(\text{stock}, \text{total operating cost})$ ... If the requested fact is absent from your chunk, return exactly `No Mention`. Do not answer from model memory."

2. **Member Agents, response in parallel:**
   - **Member Agent 1:** "... As a result, the tuple $(\text{stock}, \text{total operating cost})$ presented in my chunk is $(\text{stock } 000001, \text{CNY}47{,}677{,}000{,}000.00)$"
   - **Member Agent 2:** "... As a result, the tuple $(\text{stock}, \text{total operating cost})$ presented in my chunk is $(\text{stock } 000681, \text{CNY}685{,}136{,}074.70)$"
   - **Member Agent 3:** "... Since there is no relevant information in my chunk, my answer is No Mention."
   - ...
   - **Member Agent 7:** "... As a result, the tuple $(\text{stock}, \text{total operating cost})$ presented in my chunk is $(\text{stock } 601328, \text{CNY}158{,}164{,}000{,}000.00)$"
   - **Member Agent 8:** "... As a result, the tuple $(\text{stock}, \text{total operating cost})$ presented in my chunk is $(\text{stock } 603466, \text{CNY}1{,}959{,}711{,}464.81)$"
   - ...
3. **Leader Agent:** It receives the outputs of all the member agents and integrates the answers.

> **Reasoning Content:** "... The current maximum and minimum are reported by 601328 and 000681. Wait, I find that the total operating cost of stock 601328 is missing. I need to obtain this value ..." **[Redundant Exploration]**
> **Instruction to Member Agents:** "Your assigned chunk belongs to one of the 14 identified 2023 A-share annual reports. Search only this chunk for the *total operating cost* of stock 601328. If it is present, return a tuple $(\text{stock}, \text{total operating cost})$ ... If the requested fact is absent from your chunk, return exactly `No Mention`. Do not answer from model memory."

4. **Member Agents, response in parallel:**
   - **Member Agent 1:** "... Since there is no relevant information in my chunk, my answer is No Mention."
   - **Member Agent 2:** "... Since there is no relevant information in my chunk, my answer is No Mention."
   - **Member Agent 3:** "... Since there is no relevant information in my chunk, my answer is No Mention."
   - ...
   - **Member Agent 7:** "... As a result, the tuple $(\text{stock}, \text{total operating cost})$ presented in my chunk is $(\text{stock } 601328, \text{CNY}158{,}164{,}000{,}000.00)$"
   - **Member Agent 8:** "... Since there is no relevant information in my chunk, my answer is No Mention."
   - ...
5. **Leader:** "... The minimum is reported by 000681 ... The result is 158,164,000,000.00 - 685,136,074.70 = 157,478,876,571.30." **[Incorrect]**

Chain-of-Agents (CoA) (Zhang et al., 2024). Each worker agent needs to perform all the information processing capabilities within its corresponding file chunk.

1. **Worker Agent 1:** It extracts the value of 000001 and initializes the communication unit.
$$\text{max} = 47{,}677{,}000{,}000\ (000001);\ \text{min} = 47{,}677{,}000{,}000\ (000001)$$
2. **Worker Agent 2:** It extracts the value of 000681 and updates the communication unit.
$$\text{max} = 47{,}677{,}000{,}000\ (000001);\ \text{min} = 685{,}136{,}074.70\ (000681)$$
3. **Worker Agents 3–7:** They sequentially extract values and update the communication unit.
$$\text{max} = 158{,}164{,}000{,}000\ (601328);\ \text{min} = 685{,}136{,}074.70\ (000681)$$
4. **Worker Agent 8:** It extracts the value of 601528, CNY 2,203,371,000.00, and rewrites the communication unit **[Data Comparison Error]**
$$\text{max} = 158{,}164{,}000{,}000\ (601328);\ \text{min} = 2{,}203{,}371{,}000.00\ (601528)$$
5. **Worker Agent 9,...:** They sequentially extract values and update the communication unit.
$$\text{max} = 158{,}164{,}000{,}000\ (601328);\ \text{min} = 1{,}959{,}711{,}464.81\ (603466)$$
6. **Manager Agent:** It receives only the last communication unit and calculates the result.
$$158{,}164{,}000{,}000.00 - 1{,}959{,}711{,}464.81 = 156{,}204{,}288{,}535.19.\ \textbf{[Incorrect]}$$

MAIN-RAG (Chang et al., 2025). The predictor agent needs to perform all the information processing capabilities within each file chunk, while the final-predictor agent needs to perform data processing, statistical analysis, and other operations on the retained chunks.

1. **Retriever Module:** It retrieves candidate chunks from the 14 annual reports using the original question.
2. **Predictor Agent, per-chunk prediction:** For each recalled chunk, it generates an answer to the question.

- 601328 chunk: "In this chunk, I need to ... As a result, the 2023 total operating cost presented in this chunk is CNY $158{,}164{,}000{,}000.00$."
- 601528 chunk: "In this chunk, I need to ... As a result, the 2023 total operating cost presented in this chunk is CNY $2{,}203{,}371{,}000.00$." **[Context Forgetting]**
- 000681 chunk: "In this chunk, I need to ... As a result, the 2023 total operating cost presented in this chunk is CNY $685{,}136{,}074.70$; this is a possible minimum."
- ...

3. **Judge Agent, per-triplet assessment:** It scores each triplet according to whether the document supports the question.
   - (601328 chunk, question, prediction generated by the predictor agent): 0.974
   - (601528 chunk, question, prediction generated by the predictor agent): 0.897
   - (000681 chunk, question, prediction generated by the predictor agent): 0.842
   - ...
4. **Final-Predictor Agent:** It receives the contents of the chunks with high scores and generates the final answer to the original question.

   > "From the retained evidence, I need to choose the maximum and minimum values ... As a result, the value of 601328 is the maximum and the value of 601528 is the minimum ... The result is $158{,}164{,}000{,}000 - 220{,}337{,}100 = 157{,}943{,}662{,}900$." **[Incorrect]**

BELLE (Zhang et al., 2025a). In the multi-hop QA executor, the agent of each sub-step typically needs to handle multiple information processing capabilities.

1. **Question-Type Classifier:** "The question asks for extrema across multiple companies followed by subtraction. Type: `Comparison`."
2. **First-level affirmative debater:** "For a Comparison question, use `sub-step` to decompose the task, followed by `single-step` retrieval for the maximum and minimum."
3. **First-level negative debater:** "I disagree. Both extrema are defined over all 14 reports. Two single-step retrievals may return salient candidates without establishing exhaustive coverage. We should use `sub-step + iterative-step` retrieval and verify the resulting extrema."
4. **Fast debater:** "The affirmative plan is consistent with the usual operator choice for Comparison questions, but the negative side identifies a document-coverage risk."
5. **Slow debater:** "Historical summary: both sides agree on decomposition. Their disagreement concerns whether one retrieval step can establish global extrema over the full collection."
6. **Judge:** "Terminate the debate. Execution plan: (1) decompose the task into finding the maximum and minimum; (2) use single-step retrieval for each sub-question; and (3) subtract the two sub-answers."
7. **Multi-hop QA Executor:**
   - **Finding the Maximum:** Extracting the total operating cost from each report and returning the maximum value $47{,}677{,}000{,}000.00$.
   - **Finding the Minimum:** Extracting the total operating cost from each report and returning the minimum value $685{,}136{,}074.70$.
   - **Generating the Final Answer:** The range of the costs is CNY $46{,}991{,}863{,}925.3$. **[Incorrect]**

LongTA (Li et al., 2026a). The adopted agent needs to handle all the information processing capabilities required for the specific task.

1. **Agent, Step 1:** Inspects the file list and observes that all 14 candidates are 2023 annual reports. It then invokes lexical retrieval over a subset of apparently relevant files:

```
retriever(601328.pdf, "total operating cost", ...)
retriever(601998.pdf, "total operating cost", ...)
retriever(603466.pdf, "total operating cost", ...)
...
```

2. **Tool Result:** The retrieved passages provide values of CNY 158,164,000,000.00, CNY 131,001,000,000.00, CNY 1,959,711,464.81, ...
3. **Agent, Step 2:** Calls `read_doc` on multiple files and confirms some values.

   ```
   read_doc(601328.pdf, ...)
   read_doc(603466.pdf, ...)
   ...
   ```

4. **Tool Result:** The contents of "601328.pdf", "603466.pdf", ... are ... , respectively.
5. **Agent, Step 3:** ... Next, I need to find the maximum value and the minimum value ... As a result, the maximum value is 158,164,000,000.00 (601328) and the minimum value is 685,136,074.70 (600919). **[Evidence Mis-attribution]**
6. **Agent, Step 4:** I write Python code *code* to calculate the range and execute it in the Python execution environment.

   `Python execution:` *code*

7. **Execution Result:** The result is 157,478,863,925.30 CNY
8. **Agent, Step 5:** The maximum value is 158,164,000,000.00 from 601328 and the minimum value is 685,136,074.70 from 600919. So the range is 157,478,863,925.30. **[Incorrect: 685,136,074.70 is from 000681]**

Sensemaker. Instantiating different capabilities as specialized and independently executable modules achieves structured coordination, dedicated execution, and explicit intermediate states, which together support reliable and high-quality multi-document analysis.

1. **Task-layer Agent:** "The user asks me to calculate the range of the total operating costs of the companies in the document collection in 2023. First, I need to know which files record the operating costs of the companies in 2023."

   `agent_call("FL",` $\mathcal{X}_{\mathrm{FL}}$`)`

2. **FL:** It locates files related to the total operating costs of the companies in 2023 from the file library.

   Reasoning Content: "I need to select relevant files ..."
   Tool Call:

   ```
   tool_call( "project_search", patterns=["2023", "operating
   costs", ...], ...)
   tool_call( "search_profile", patterns=["2023", "operating
   costs", ...], ...)
   ```

   Tool Result: "The matched files and their matching status are as follows ..."
   Reasoning Content: "... The relevant files are 601328.pdf, . . ."

3. **Task-layer Agent:** "... I need to extract the operating costs from each file."

   `agent_call("CE",` $\mathcal{X}_{\mathrm{CE}}$`)`

4. **CE:** It extracts the operating costs from relevant files.

   Reasoning Content: "I need to extract the value of operating cost ... Therefore, I choose `regex_extract` to get the values."
   Tool Call:

   ```
   tool_call( "regex_extract", requests=[ {
   file_path="601328.pdf", patterns=["operating costs", ...]
   }, ...]  )
   ```

   Tool Result: "The operating costs are 158,164,000,000.00, 685,136,074.70, ..."
   Reasoning Content: "... The extracted values are 158,164,000,000.00, 685,136,074.70, ..."

5. **Task-layer Agent:** "... I already have all the operating costs of the companies in the document collection in 2023. Next, I need to calculate the range."

   `agent_call("DP",` $\mathcal{X}_{\mathrm{DP}}$`)`

6. **DP:** It calculates the range of the total operating costs of the companies based on the extracted values.

Reasoning Content: "... To ensure the accuracy of the results, I will write Python code to find the maximum value and the minimum value and calculate $max_value - min_value$."
Tool Call:
`tool_call( "run_python", python_code=code, ...)`
Tool Result: "The maximum value is 158,164,000,000.00 from 601328, the minimum value is 685,136,074.70 from 000681, and $158{,}164{,}000{,}000.00 - 685{,}136{,}074.70 = 157{,}478{,}863{,}925.30$."
Reasoning Content: "... The maximum value is 158,164,000,000.00 (601328), the minimum value is 685,136,074.70 (000681), and $158{,}164{,}000{,}000.00 - 685{,}136{,}074.70 = 157{,}478{,}863{,}925.30$."

7. **Task-layer Agent:** ... I need to check the evidence.

   `agent_call("EV", `$\mathcal{X}_{\text{EV}}$`)`

8. **EV:** It checks whether the values of operating costs are correct.

   Reasoning Content: "... I need to know the raw source fragment of these values."
   Tool Call:
   `tool_call( "range_read", ranges=[ { file_path="000681.pdf", ...} ], ...)`
   `tool_call( "range_read", ranges=[ { file_path="601328.pdf", ...} ], ...)`
   `...`
   Tool Result: "The raw fragment of the evidence is ..."
   Reasoning Content: "... According to the raw fragment, the operating cost of stock 601328 is 158,164,000,000.00 and the operating cost of stock 000681 is 685,136,074.70. The values are correct."

9. **Task-layer Agent:** ... The maximum value is 158,164,000,000.00 from 601328 and the minimum value is 685,136,074.70 from 000681. So the range is 157,478,863,925.30. **[Correct]**