GSM-AGENT: UNDERSTANDING AGENTIC REASONING USING CONTROLLABLE ENVIRONMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

As LLMs are increasingly deployed as agents, agentic reasoning—the ability to combine tool use, especially search, and reasoning—becomes a critical skill. However, it is hard to disentangle agentic reasoning when evaluated in complex environments and tasks. Current agent benchmarks often mix agentic reasoning with challenging math reasoning, expert-level knowledge, and other advanced capabilities. To fill this gap, we build a novel benchmark, GSM-AGENT, where an LLM agent is required to solve grade-school-level reasoning problems, but is only presented with the question in the prompt without the premises that contain the necessary information to solve the task, and needs to proactively collect that information using tools. Although the original tasks are grade-school math problems, we observe that even frontier models like GPT-5 only achieve 67% accuracy. To understand and analyze the agentic reasoning patterns, we propose the concept of *agentic reasoning graph*: cluster the environment's document embeddings into nodes, and map each tool call to its nearest node to build a reasoning path. Surprisingly, we identify that revisit, returning to a previously visited node after leaving–widely taken as a crucial pattern in static reasoning, is a missing ability for agentic reasoning among many models. Based on the insight, we propose a tool-augmented test-time scaling method to improve LLM's agentic reasoning performance by adding tools to encourage models to revisit. We expect our benchmark and the agentic reasoning framework to aid future studies of understanding and pushing the boundaries of agentic reasoning.

1 INTRODUCTIONS

Large language models (LLMs) have demonstrated remarkable performance on challenging reasoning tasks (Wei et al., 2022; Srivastava et al., 2023), from arithmetic word problems (Cobbe et al., 2021) to multi-hop question answering (Yang et al., 2018) and program synthesis (Chen et al., 2021). Most previous work focuses on reasoning tasks (Cobbe et al., 2021; Hendrycks et al., 2021; Saxton et al., 2019) that evaluate LLMs' *static reasoning* capability, where the model receives all necessary information from the prompt and conducts reasoning without external help. Yet, as LLMs are increasingly deployed as *agents* – systems that plan, use external tools, and iteratively refine their hypotheses – the form of reasoning that matters in practice gradually shifts from *static reasoning* to *agentic reasoning* that couples logical inference with decisions about what to read, what to ask next, when to verify, and how to recover from unproductive directions.

In this paper, we aim to understand to what extent strong static reasoning abilities of an LLM can be adapted to the agentic setting, and identify the key skills that may enable this. To achieve this, we aim to (1) compare a model's reasoning ability on the same or similar tasks under static and agentic settings; (2) identify the important skills that contribute to the performance gap between the two settings; (3) improve the model's skill in the agentic setting to enhance its agentic reasoning ability. The above steps bring two major challenges, and in this paper, we propose solutions to each of them.

Challenge 1: Existing benchmarks fail to provide an apples-to-apples comparison of reasoning abilities under the two settings.

Solution: To this end, we introduce GSM-AGENT, a novel benchmark that transforms GSM8K problems into agentic tasks. Specifically, during dataset construction, each original problem is decomposed into a question and several premises; each premise is then converted into a context-rich

document and inserted into a database (the environment). During evaluation, the agent sees only the question and needs to use the provided tools (a `Search` tool and a `NextPage` tool) to discover the relevant documents before solving the math problem. Importantly, we can control the difficulty of the agentic task through careful construction of the database, e.g., by adding distracting documents. Across a broad suite of models, we observe substantial performance drops compared to the static setting where the question and all necessary documents are provided in the prompt. For example, a frontier model like GPT-5 loses roughly 33% absolute accuracy, whereas some models (e.g., DeepSeek-V3) lose up to 80%. The results demonstrate a clear and consistent gap between static and agentic reasoning in a clean and controllable setting.

Challenge 2: We lack a framework to identify and quantify the core skills that contribute to agentic reasoning capability.

Solution: To understand and analyze the core reasons of the performance gap between the two settings and what drives such significant differences in performance across models under agentic settings, inspired by Minegishi et al. (2025), we propose the concept of *agentic reasoning graph*: cluster the environment's document embeddings into nodes, and map each tool call (`Search` or `NextPage`) to its nearest node, yielding a discrete reasoning path. This framework allows us to label each reasoning step as *exploration* (first visit to a node), *exploitation* (staying within a node), or *revisit* (returning to a previously visited node after leaving). Our analysis reveals that the *revisit ratio* strongly correlates with the accuracy on GSM-AGENT, which indicates that revisit might be a core skill for strong agentic reasoning. Based on the insight, we propose a tool-augmented method, where we add a new tool that encourages the model to revisit, to improve LLMs' performance. Experimental results demonstrate that our tool-augmented method exhibits better performance than interaction-round scaling, which enforces agents to interact with the environment for more rounds without considering the quality of each interaction step.

We summarize our contributions as follows:

- We propose GSM-AGENT, a novel benchmark with a controllable environment for evaluating and analyzing the agentic reasoning capability of LLMs and providing a clear comparison between static and agentic reasoning.
- We introduce the concept of *agentic reasoning graph*, which induces a topology over the environment via clustering of document embeddings and maps tool-use traces to discrete paths. This yields interpretable, quantitative measures of *exploration*, *exploitation*, and *revisit* during the reasoning procedure at step resolution to facilitate analysis of agentic reasoning.
- Our analysis of reasoning patterns on agentic reasoning graphs reveals that revisit is an important reasoning skill that strongly correlates with agentic reasoning capability. Based on the insight, we propose a tool-augmented method to improve LLMs' agentic reasoning capability by encouraging revisit.

2 RELATED WORK

Reasoning with incomplete information. Multiple works have studied the ability of LLMs to look for missing information. Most relevant to our work, Li et al. (2025) evaluate the ability to ask the right question, including on a variant of GSM8K with missing information. However, their focus is on evaluating whether the model asks specific questions, rather than overall reasoning abilities. Zhou et al. (2025b) compare "passive" and "active" reasoning, similar to our "static" vs "agentic" reasoning, although they use different tasks for the two setups, while our dataset can be used in both scenarios, leading to a better apples-to-apples comparison.

Agentic reasoning benchmarks. Several benchmarks have recently been established for evaluating agentic reasoning capabilities of LLMs (Jimenez et al., 2023; Yao et al., 2024; Lu et al., 2024; Trivedi et al., 2024; Patil et al., 2025). In contrast to these works, our benchmark aims to provide a controllable environment that enables direct comparison of agentic reasoning with static reasoning.

Understanding of reasoning. Many recent works have focused on understanding the reasoning abilities of LLMs, including the ability to self-correct (Huang et al., 2024), the reliability of reasoning on GSM8K beyond the original benchmark via synthetic extensions (Zhou et al., 2025a;

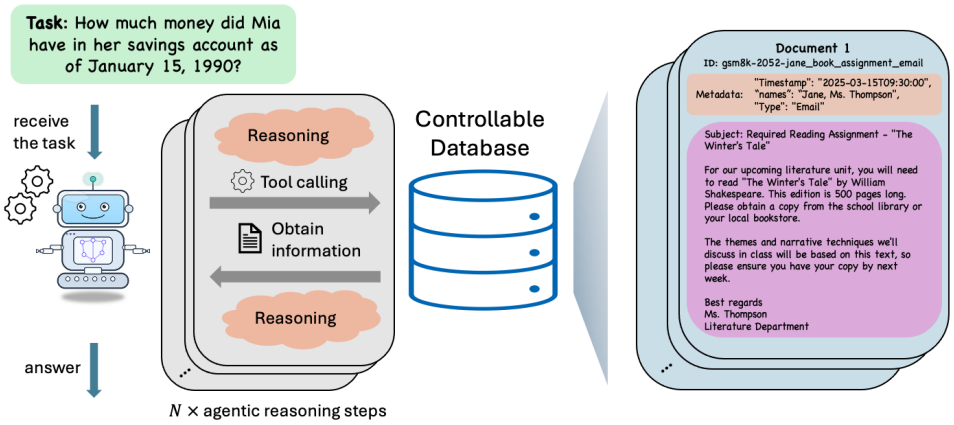


Figure 1: An overview of evaluation tasks in our GSM-AGENT benchmark. The LLM agent receives a task that only contains a question. At each *agentic reasoning* step, the agent needs to decide what information is needed, call the tool to search for information in the database, and reason about the retrieved documents. The agent also needs to decide whether all the necessary information has been collected and when to give the final answer to the task.

Mirzadeh et al., 2025), or the reasoning behaviors or long chain-of-thought models (Yeo et al., 2025; Sun et al., 2025; Minegishi et al., 2025). Our graph-based analysis of explore, exploit, and revisit patterns was partly inspired by these works, though we extend this to the agentic setting with search tools by definiting the graph through document embeddings.

3 GSM-AGENT BENCHMARK

In this section, we introduce GSM-AGENT, a novel benchmark with controllable environments for comprehensively evaluating the agentic reasoning capabilities of LLMs. In particular, our dataset aims to test LLM agents' abilities to combine reasoning and tool-use (mainly search) ability to solve mathematical reasoning problems by proactively interacting with the environment using tools. Below, we provide an overview of our benchmark tasks in Section 3.1, and introduce our dataset construction process in Section 3.2.

3.1 OVERVIEW

Our dataset $\mathcal{D} = (\mathcal{T}, \mathcal{E}, \mathcal{F})$ consists of a set of tasks $\mathcal{T}$, an environment $\mathcal{E}$, and a set of tools $\mathcal{F}$ that LLM agents can use to interact with the environment.

Tasks. Each task $T = (q, (p_1, \ldots, p_k), a) \in \mathcal{T}$ consists of a question q, k premises $p_1, \ldots, p_k$ (k can vary for different task instances) and the ground-truth answer a. Figure 2 provides an example of a task instance that consists of a question and three premises. For a grade-school-level math problem, it is easy for an advanced LLM to solve the task if all premises $p_1, \ldots, p_k$ are provided in the prompt along with the question q. In our benchmark, the LLM agent will only see the question q without premises $p_1, p_2, \ldots p_k$ in the prompt, and it needs to use tools in $\mathcal{F}$ to find all necessary information in the environment $\mathcal{E}$ to solve the task (see Figure 1 for a pictorial illustration).

Environments. The environment $\mathcal{E} = \{D_1, D_2, \ldots, D_m\}$ consists of a set of documents, where each document corresponds to a premise of a task in $\mathcal{T}$. Let $g_D(\cdot)$ be a document generator, where $g_D(T) = g_D(q, (p_1, \ldots, p_k)) = (D_1, D_2, \ldots, D_k)$ and the generated document D_i contains all necessary information of the premise p_i for all $1 \leq i \leq k$. See the document generation part of Figure 2 for a pictorial illustration. We will introduce the details of our implementation of the document generator $g_D(\cdot)$ in Section 3.2. Since our environment $\mathcal{E}$ is a set of documents, we also call $\mathcal{E}$ a database in the rest of the paper.

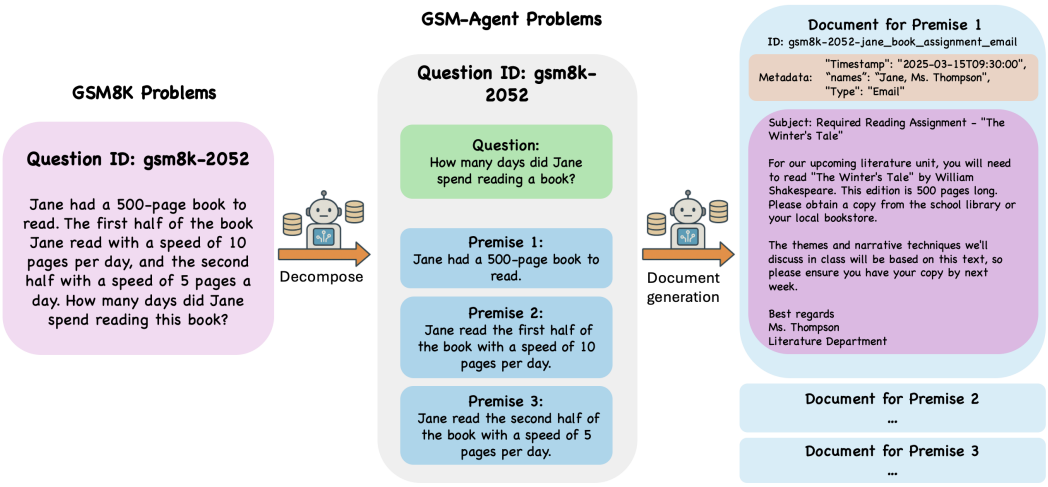


Figure 2: Data processing overview. We first decompose a GSM8k problem into a question and several premises, and then generate a document for each premise to cover its essential information.

Tools. In our benchmark, we provide two tools $\mathcal{F} = \{\texttt{Search}(\cdot), \texttt{NextPage}(\cdot)\}$. For the search tool, an LLM agent can specify a query prompt x (which can be of any format, such as a sentence or only several keywords) and call $\texttt{Search}(x)$. The search engine will return the top 5 most relevant documents in $\mathcal{E}$ to the query x. The agent can also use the $\texttt{NextPage}(\cdot)$ tool, which will return the next five most relevant documents. The LLM agent is allowed to call $\texttt{Search}(x)$ for multiple different query prompts x that are decided by the agent itself. For each call of $\texttt{Search}(x)$, the agent is also allowed to call $\texttt{NextPage}(\cdot)$ at most 19 times for that search, resulting in retrieving up to the top 100 most relevant documents in the database. See Figure 1 for a pictorial illustration and Section 3.2 for more details about the search engine.

During evaluation, for each task T, an LLM agent will receive the corresponding question q (without premises) in the prompt, then reason about what information is missing, call tools $\texttt{Search}(x)$ with appropriate search query x and $\texttt{NextPage}(\cdot)$ to collect information from the environment $\mathcal{E}$, and solve the problem with information extracted from the retrieved document. In our dataset, the ground-truth answer a is a numerical value for each task, so we directly compare the final answer given by an LLM agent $\hat{a}$ to a, where the task is solved iff $\hat{a} = a$.

3.2 Dataset construction

In this section, we introduce our dataset construction procedure in detail. Our GSM-Agent dataset is built upon the well-known GSM8k problem set, where each task T in our dataset is constructed based on a problem instance in GSM8k. A simplified dataset construction pipeline is illustrated in Figure 2, while the whole pipeline consists of five stages: (1) data preprocessing; (2) problem decomposition and sharding; (3) document generation; (4) data filtering; (5) database construction. Note that some steps in our pipeline involve using LLMs to process data. Unless otherwise specified, we use `Claude-3.5-Sonnet` as our default LLM to facilitate data processing.

3.2.1 Data preprocessing

As shown in Figure 2, for each GSM8k problem, we decompose it into a question and several premises, and convert each premise into a document which will be added to our database. However, naively processing each problem will cause issues.

First, different problems might share the same name of the protagonist(s). Although this is not an issue in the original GSM8k problem since each problem is independent, it could cause conflict or ambiguity in our database, as documents for different tasks will be added to the same database. For example, Alice might spend 5 dollars on ice cream in document D_1 for one task T_1, and also pay 20

dollars for a book in document D_2 for another task T_2, which renders the question "How much did Alice spend in total?" ambiguous.

Second, some problems only contain a generic entity without a specific name. For example, consider a task such as "a bookshelf has 20 books at the top and 40 books at the bottom, and how many books are there in the shelf in total". While the original problem is self-contained, separating the question from premises renders the question "how many books are there in the shelf in total" again confusing and ambiguous since it is unclear which specific bookshelf the question refers to.

To address the above two issues, we carefully design the following three data preprocessing steps to disambiguate the tasks. **Step 1: Entity detection.** In this step, we use LLM to detect the main character of each problem. For problems with a generic main character without a specific name, we flag it as generic. **Step 2: Name assignment for generic entities.** In this step, we assign different names to generic entities. **Step 3: Timestamps assignment to differentiate problems sharing the same entity.** At this step, we assign different timestamps to problems sharing the same entity to ensure no conflict between documents from different problems. The details can be found in Appendix B.

3.2.2 Problem decomposition and sharding

After systematic data preprocessing to ensure no ambiguity in our tasks and no conflicting documents in our environment, our next step is to decompose each preprocessed problem into a question q and several self-contained premises $p_1, p_2, \ldots, p_k$. See the "decompose" part of Figure 2 for an example. The decomposition is executed by an LLM agent, where the agent receives a preprocessed problem along with its timestamp as the input, then breaks down the problem narrative into a list of individual self-contained and consistent premises, rephrases the core question, and carries over the timestamp. In particular, if the problem shares an entity name with another problem, its timestamp will be explicitly stated in the question q outputted by the agent to make sure the question itself is unambiguous.

3.2.3 Document generation

The next stage is document generation. The main purpose of this stage is to convert each premise of each problem into a context-rich document, which will be added to our database (i.e., the environment $\mathcal{E}$) in the final stage. We conduct the following three steps to ensure a high-quality database and a reasonable level of difficulty for our tasks. **Step 1: Hierarchical document generation.** At this step, we generate a high-level coherent story for a problem. **Step 2: Independence verification.** We prompt Claude-3.5-Sonnet, giving it each document-premise pair and the original question, and asks it to judge whether the document contains extra information that is not covered by the premise. By doing so, we make sure that documents are independent, with no overlapping information. **Step 3: Document anonymization.** To ensure the difficulty of our benchmark, we randomly anonymize a subset of the document to avoid LLM agents from "cheating" by blindly querying the name of the main character. The details of each step can be found in Appendix B.

3.2.4 Data filtering

Since the previous stages involve using LLM agents to process data, and original premises are converted into much longer documents, some of the generated tasks may turn out to be problematic or unsolvable. To ensure the quality of our dataset after the above data processing stages, a rule of thumb is to ensure that all generated tasks are solvable when provided with complete information (i.e., all documents corresponding to their premises). Therefore, for each problem, we test whether `claude-3.5-sonnet` can solve it given the question and its corresponding documents. We only keep the problems that `claude-3.5-sonnet` can correctly solve to ensure a high-quality dataset. After our data filtering stage, there are 7323 problems left in our dataset, with 32315 unique problems stored in the chroma database.

3.2.5 Database construction

The final stage is to build the environment $\mathcal{E}$ (i.e., the database) using generated documents. We use Chroma to build our database, where the content (excluding document ID and metadata)

of each document is embedded into a vector that will be used for document retrieval. We use `text-embedding-3-large` as our default embedding model.

Moreover, we built three datasets that reflect different levels of difficulty of our benchmark: **GSM-AGENT-Full**: contains all problems after data filtering; **GSM-AGENT-Medium**: contains 25% of problems after data filtering; **GSM-AGENT-Small**: contains 6.25% of problems after data filtering.

For each dataset, its environment is a database that contains all the documents of problems in the dataset. For the results reported in Section 4, we evaluate models on GSM-AGENT-Full unless otherwise specified.

4 RESULTS & ANALYSIS

In this section, we first present the main evaluation results on a variety of mainstream LLMs on our GSM-AGENT dataset (Section 4.1), then analyze the agentic reasoning pattern and identify the core skill, which is *revisit*, that correlates to strong agentic reasoning capabilities (Section 4.2), and finally propose a tool-augmented method to improve models' agentic reasoning capability by encouraging models to revisit (Section 4.3). We use LangChain ReAct agent for all evaluation settings, with temperature set to 0.4 and max tokens 4096.

4.1 OVERALL PERFORMANCE

Table 1: Evaluation results under zero-shot prompting with ReAct agent across models.[1] Acc and FF are shown as percentages; other metrics use the units indicated. Acronyms: Acc=Accuracy; SR=Search Rounds, which is the number of tool calls to solve a task; Dur(s)=Duration (seconds), which is the time the agent spent to solve the task; SC=Search-Complete rate, which is the proportion of tasks that an gent find all relevant documents; ER=Extra Rounds, which is the number of tool calls after all relevant documents are found; FF=Follow-Format rate, which is the proportion of tasks that an agent follows the required format to solve; PremT=Premature-Total rate, which is the proportion of tasks that an agent attempted to provide a premature answer before making the final decision; TotTok=Total Generated Tokens; Tok/R=Mean Tokens per Round. All results are averaged over three random seeds. For each metric, ↑ indicates higher is better and ↓ means lower is better.

Setting	Acc ↑	SR	Dur(s)	SC ↑	ER ↓	FF ↑	PremT ↓	TotTok ↓	Tok/R ↓
Solvable by any model	88.00%	Nan	Nan	Nan	Nan	Nan	Nan	Nan	Nan
o3	68.46%	13.33	117.85	53%	4.89	95%	0%	5775.75	386.03
GPT-5	66.78%	9.98	116.00	52%	2.18	100%	1%	7184.10	615.99
Grok-4	53.00%	7.19	126.01	42%	2.86	100%	0%	3817.42	599.72
Claude-4-sonnet (fewshot)[2]	51.50%	9.27	47.90	40%	4.41	100%	10%	1028.52	118.65
Gemini-2.5-Pro	38.33%	2.93	51.59	25%	0.20	82%	3%	Nan	Nan[3]
Kimi-K2-Instruct	37.42%	5.41	31.00	24%	0.53	92%	0%	245.34	56.18
Gemini-2.5-Flash	25.33%	1.88	17.13	14%	0.12	99%	4%	Nan	Nan
GPT-4o	22.67%	1.92	21.27	22%	2.72	94%	1%	135.20	92.22
Llama-4-Maverick	20.00%	2.10	21.94	17%	0.26	97%	3%	504.93	211.30
DeepSeek-V3	19.42%	0.94	14.30	8%	0.00	82%	0%	38.95	41.33
Qwen3-235B	19.30%	1.13	25.76	19%	4.40	96%	0%	184.82	173.19
Claude-4-Sonnet	18.67%	2.46	21.27	14%	3.14	33%	1%	243.93	98.23
Llama-4-Scout	12.54%	2.07	14.93	9%	1.76	86%	4%	215.48	118.96

Table 1 highlights the large performance gaps between different models. While some strong agents achieve relatively high accuracy, many open models remain surprisingly weak. This result is puz-

[1] We observe that zero-shot prompting renders the most stable result across most models, better than fewshot prompt or multi-agent system. However, it cannot render meaningful results for DeepSeek-R1 and Claude-Opus. DeepSeek expects a different tool-use format than other models. Claude frequently asks the user for additional information and thus requires careful prompting to fix. To ensure a fair comparison across models, we excluded the two models in the evaluation.

[2] We observed that Claude-4-sonnet requires special few-shot prompt strategies to achieve decent performance. However, such a prompt can hurt the performance of many other models compared to the default zero-shot prompting. To ensure fair comparison, we report both zero-shot and few-shot prompting results for Claude-4-sonnet.

[3] Number missing due to LangChain output format mismatch for Gemini model series.

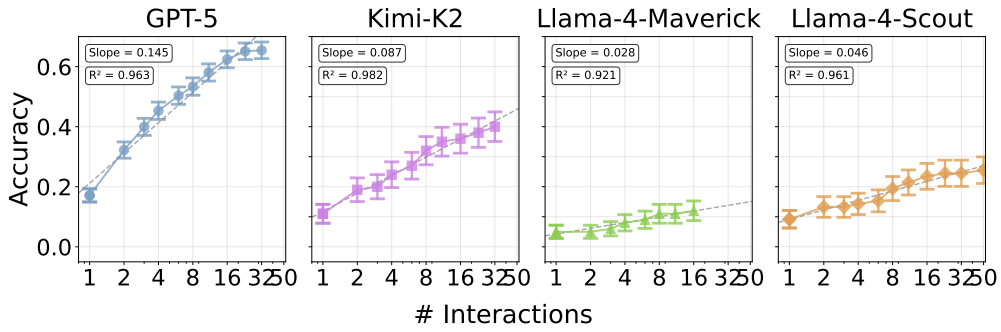
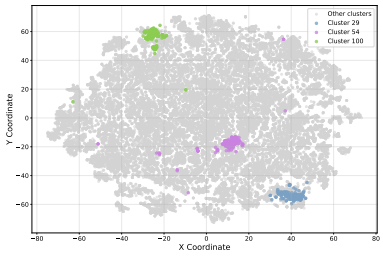


Figure 3: Interaction round scaling. The number of iteration rounds is defined as the number of tool calls. For each model, we first collect the reasoning trajectory on each task under zero-shot prompting. For a specified number of interaction rounds n, a trajectory is considered correct either if it answers the task correctly within n rounds, or it successfully collects all necessary documents within n rounds and gives a correct answer eventually. GPT-5 exhibits a much stronger interaction-round scaling than the other three models. Note that x-axis is in logarithmic scale.

(a) Visualization of Nodes



(b) Representative Nodes from The Database Graph

Node	Sample Document IDs
Grocery Receipts (Node 29)	laura_flour_purchase_receipt chocolate_purchase_receipt
Fruit Counting (Node 54)	xena_total_fruit_count chads_apple_inventory_log
Youth Sports Stats (Node 100)	wario_kick_direction_analysis james_touchdown_stats

Table 2: *Left (a)*: A t-SNE visualization of the search database embeddings. It highlights three clusters whose centroids define the embeddings for nodes 29, 54, and 100. *Right (b)*: A summary of documents for these nodes. The documents have semantic coherence within each cluster.

zling, since our GSM-AGENT environment—adapted from GSM8K—requires only grade-school math and basic common sense reasoning. Nevertheless, even frontier models fall short of perfect performance, with the best accuracy reaching just 68.46% (o3), while Llama-4-Scout only achieves 12.54%. These discrepancies raise an important question:

What drives such big differences in performance across models, given such a simple environment?

Simple interaction-time scaling does not improve agentic reasoning. Table 1 shows that models tend to achieve higher accuracy when they perform more search rounds. A natural hypothesis for the observed performance gap is therefore differences in interaction-time scaling. To test this, we selected three open models (Kimi-K2-Instuct, Llama-4-Maverick, and Llama-4-Scout) and prompted them to continue searching whenever they attempted to stop. For comparison, we also measured the test-time scaling behavior of GPT-5, a representative strong proprietary model. As shown in Figure 3, the accuracy of the open models improves only marginally with additional search rounds, exhibiting far weaker interaction-time scaling than the proprietary models (GPT-5). This finding suggests that we must look beyond just interaction time and instead examine the quality of interaction choices-this motivates the design of our *agentic reasoning graph*.

4.2 IDENTIFYING AND MEASURING CORE SKILLS CONTRIBUTING TO AGENTIC REASONING

In this section, we propose a new framework to understand and analyze models' agentic reasoning patterns and, thus, identify the core skills that contribute to a model's reasoning ability in agentic settings. Inspired by Minegishi et al. (2025), we define the *agentic reasoning graph* below.

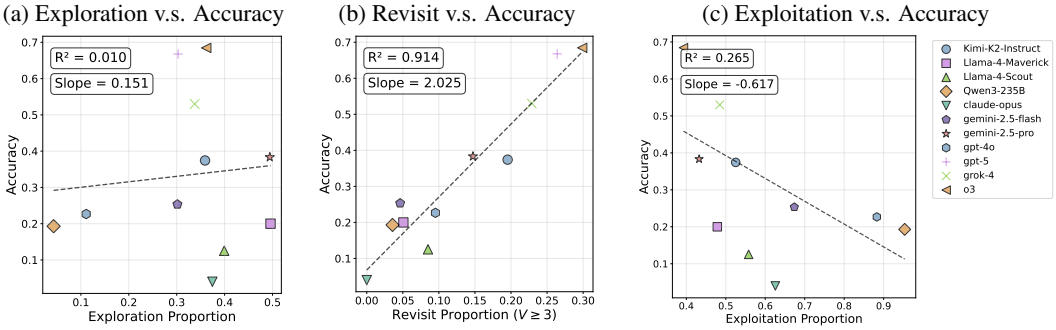


Figure 4: Correlation between accuracy and exploration, exploitation, and revisit ratio. The three ratios are defined as the proportion of exploration steps (visit a node that has never been reached), exploitation steps (visit the same node as the last step), and revisit steps (revisit a previously reached node after leaving) to the total reasoning steps. We plot their correlation to the models' accuracy on our GSM-AGENT benchmark. The plots show that the model accuracy has a weak correlation to the exploration ratio, a strong correlation to the revisit ratio and a negative correalation to the exploitation ratio.

Nodes of the agentic reasoning graph. Assume the environment $\mathcal{E} = \{D_i\}_{i=1}^N$ contains N documents. Denote the embedding model to be $e_\theta(\cdot)$ and thus $e_i = e_\theta(D_i) \in \mathbb{R}^d$ is the embedding vector of document D_i. We run K-means ($K = 250$) on $\{e_i\}_{i=1}^N$ to get clusters $\{C_k\}_{k=1}^K$ with centroid $\{c_k\}_{k=1}^K$, and each centroids $c_k \in \mathbb{R}^d$ correspond to a node v_k in the graph. Therefore, the vertex set of the agentic reasoning graph is $V = \{v_1, \ldots, v_K\}$. See Table 2 for a visualization of the vertex set of our database and examples for the semantic meaning of representative nodes.

Agentic reasoning path. Assume the agent makes T tool calls in total in the whole reasoning trace. For the t-th tool call, if it calls `Search`(x), then the agentic reasoning node p_t for the t-th tool call is defined as $p_t = \arg\min_{v_k \in V} \|q(x) - c_k\|_2$, where $q(x) \in \mathbb{R}^d$ is the embedding of the query prompt x. If the agent calls `NextPage`$(\cdot)$ for the t-th tool call, then the agentic reasoning node is defined as $p_t = p_{t-1}$. The agentic reasoning path is then defined as $\pi = (p_1, \ldots, p_T)$.

Exploration, exploitation and revisit. For each step p_t in the reasoning path, we classify it into one of the three categories. For the first step p_1, it is always classified as an *exploration step*. For $t > 1$, if $p_t \notin \{p_1, \ldots, p_{t-1}\}$, i.e., p_t has never been visited in previous steps, then p_t is also an exploration step. If $p_t \in \{p_1, \ldots, p_{t-1}\}$, it is considered as an *exploitation step* if $p_t = p_{t-1}$, and otherwise a *revisit step*. The exploration ratio is defined as the proportion of exploration steps among the total number of steps T. Similarly, we can define the exploitation and revisit ratio. These three ratios can thus be used to quantify the reasoning pattern and facilitate deeper analysis.

Figure 4 shows that models' accuracy has a strong correlation to the revisit ratio during the reasoning trace, which implies that revisit is an important skill in agentic reasoning.

4.3 IMPROVING AGENTIC REASONING CAPABILITY VIA TOOL-AUGMENTED SCALING

Given the insight from the analysis in Section 4.2, instead of naively scaling up the interaction rounds that is widely adopted for static reasoning, we propose to use tool-augmented scaling, which might be a more efficient scaling paradigm for agentic reasoning.

Thinking tool, exploration tool, and revisit tool. We introduce three new tools for our experiments. (1) `Thinking`$(\cdot)$ is a thinking tool, which will copy a model's preceding tokens to enforce thinking whenever called. (2) `Explore`(x) is an exploration tool which has the same effect as `Search`(x) while the system prompt will instruct the model to use `Explore`(x) to explore different search queries. (3) `Revisit`(x) is a revisit tool which has the same effect as `Search`(x) while the system prompt will instruct the model to use `Revisit`(x) to revisit previously called queries.

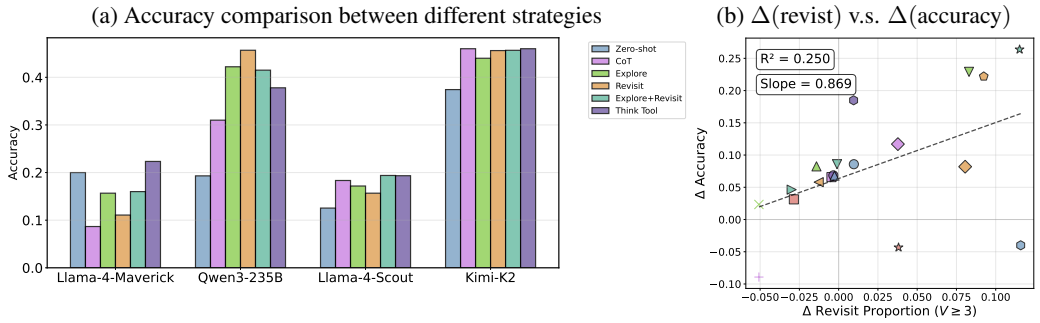


Figure 5: Visualization of performance gain via encouraging the revisit reasoning pattern. In Figure 5a, we compare five different strategies to zero-shot prompting on four different models. CoT is a prompt-only strategy where the prompt will instruct the model to think more. The remaining four strategies are all tool-augmented methods by adding different combinations of the three tools, $\texttt{Thinking}(\cdot), \texttt{Explore}(\cdot), \texttt{Revisit}(\cdot)$, to the tool set. For Llama-4-Maverick and Qwen3-235B, tool-augmented methods consistently outperform the prompt-based CoT strategy. For Llama-4-Scout and Kimi-K2, tool-augmented methods achieve comparable performance to the CoT method. For most cases, both the tool-augmented method and prompt-based CoT improve over zero-shot prompting. Figure 5b plots the correlation between the increase in revisit ratio and the increase in the accuracy for any of the strategies. It shows a strong correlation between the enhancement of revisit ability and performance improvement.

We tested four combinations of the above three tools: (1) adding $\texttt{Thinking}(\cdot)$ only to the tool set $\mathcal{F}$; (2) adding $\texttt{Explore}(\cdot)$ only; (3) $\texttt{Revisit}(\cdot)$ only; (4) adding both $\texttt{Explore}(\cdot)$ and $\texttt{Revisit}(\cdot)$.

Figure 5a shows that adding tools outperforms or achieves similar performance to the CoT prompt strategy in most cases. Moreover, Figure 5b shows a strong correlation between the increase of accuracy and revisit ratio, which further indicates that revisit is an important skill for agentic reasoning. The above results indicate that the tool-augmented method may serve as a more efficient test-time scaling paradigm than interaction-time scaling for agentic reasoning.

5 Conclusions

In this paper, we study LLMs' agentic reasoning capability, where an LLM agent needs to combine tool-use and reasoning ability to solve tasks. We first propose a novel benchmark, GSM-Agent, where an LLM agent is required to solve grade-school math reasoning problems but must proactively search for necessary information from the environment. Our comprehensive evaluation of various models shows a significant gap in performance across different models in the seemingly simple environment. We further analyze the reasoning patterns of different models using the agentic reasoning graph and identify revisit as an important skill for agentic reasoning. Finally, we propose a tool-augmented scaling method that adds new tools to encourage the model to revisit, which improves agents' performance on our benchmark for different models. We hope that our benchmark can serve as a controllable and clean environment for future study of agentic reasoning, and our framework of agentic reasoning graph can bring new insights into better understanding and improvement of reasoning ability for LLM agents.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique P. de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. In *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems*, 2021.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Belinda Z Li, Been Kim, and Zi Wang. Questbench: Can llms ask the right question to acquire information in reasoning tasks? *arXiv preprint arXiv:2503.22674*, 2025.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024.

Gouki Minegishi, Hiroki Furuta, Takeshi Kojima, Yusuke Iwasawa, and Yutaka Matsuo. Topology of reasoning: Understanding large reasoning models through reasoning graph properties. *arXiv preprint arXiv:2506.05744*, 2025.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations (ICLR)*, 2025.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning (ICML)*, 2025.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*, 2019.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Ahmed Shoeb, Abubakar Abid, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

Yiyou Sun, Shawn Hu, Georgia Zhou, Ken Zheng, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Omega: Can llms reason outside the box in math? evaluating exploratory, compositional, and transformative generalization. *arXiv preprint arXiv:2506.18880*, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*, 2025.

Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. Gsm-infinite: How do your llms behave over infinitely increasing context length and reasoning complexity? In *International Conference on Machine Learning (ICML)*, 2025a.

Zhanke Zhou, Xiao Feng, Zhaocheng Zhu, Jiangchao Yao, Sanmi Koyejo, and Bo Han. From passive to active reasoning: Can large language models ask the right questions under incomplete information? In *International Conference on Machine Learning (ICML)*, 2025b.