Gnosis Deepsearch: Structured Search Space Navigation for Complex QA

Umut Halil[¶†‡] **Le Zhang**[§‡] **Jin Huang**[§‡] **Yuwei Zhang**[§‡] **Xinyu Hou**[§‡]
André Melo[§] **Shriram Piramanayagam**[§] **Amela Fejza**[§] **Ruofei Lai**[§] **Yang Ren**[§]
Pavlos Vougiouklis[§⋆] **Jeff Z. Pan**[§◇⋆]
[§]Huawei Technologies R&D (UK) Ltd., Poisson Lab, CSI
[¶]JP Morgan Chase [◇]University of Edinburgh
firstname.lastname@huawei.com
https://github.com/openJiuwen-ai/deepsearch

Abstract

Complex, multi-step reasoning requires agents to be capable of dynamically exploring diverse research directions and adapting plans on the fly. We introduce Gnosis Deepsearch, a schema-guided framework for long-horizon planning in complex Question Answering, where variables, clues, and constraints are captured in a structured state representation maintained across trajectories. This state is cascaded into parallel search paths broadening exploration without relying on sequential execution, thereby mitigating both limited search space coverage and catastrophic forgetting. We couple Gnosis with a retriever that sidesteps challenges associated with retrieval and LLM reading when longer documents are involved. Our retrieval strategy ensures that the most relevant snippets from longer documents are considered, showcasing several benefits within both Gnosis and a more general ReAct framework on the BrowseComp-Plus benchmark. Extensive experiments on several QA datasets of varying difficulty demonstrate the strong performance of Gnosis across different configurations.

1 Introduction

Recent advancements have seen Large Language Models (LLMs) evolve from standalone generators into autonomous orchestrators capable of managing tasks with variable complexity. Integrating external tools, such as web search and browsing, enables LLMs to generalise beyond their internal parameters. This shift facilitates *deep search* and *deep research* capabilities, where agents decompose abstract or multi-hop inquiries into iterative search trajectories (Li et al., 2025b). Instead of a single retrieval step, these systems proactively navigate the web, synthesising information across multiple documents to address questions that require high-level reasoning.

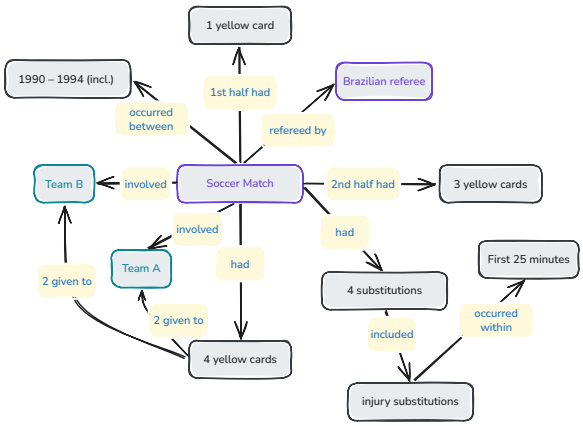


Figure 1: The graph corresponding to a complex question from BrowseComp-Plus: "Between 1990 and 1994 (inclusive), what teams played in a soccer match that had a Brazilian referee, included four yellow cards – two for each team – where three of the total four were not issued during the first half four substitutions, one of which was for an injury during the first 25 minutes of the match?".

However, as the complexity of these agentic workflows increases, so do the computational and architectural challenges. Deep search systems must manage expanding search spaces and the noise inherent in multi-step browsing and retrieval. As trajectories lengthen, maintaining coherence within finite context-limit restrictions becomes critical.

Other approaches have sought to explore the ever-increasing search spaces by computing semi-parallel *rollouts* (DeepSeek-AI et al., 2025; Li et al., 2025a). However, in many cases, the resulting sub-branches require existing rollouts to be completed first, leading to sequential computation bottlenecks (Li et al., 2025a). Furthermore, these systems often offer no guarantees against wasted computation, as different trajectories frequently arrive at similar states and redundantly repeat the same work. To

[†]Work done while at Huawei Edinburgh Research Centre.
[‡]The authors contributed equally to this work.
[⋆]Corresponding author.

address both the explicit context length limitations of LLMs and the implicit issue of forgetting, recent agentic systems introduced step-wise summarisation, referred to also as *folding* (Sun et al., 2026; Ye et al., 2026), of the full history or intermediate results. While this reduces context overload, it can lead to incomplete exploration of alternative reasoning paths (Li et al., 2025b), as potentially promising lines of reasoning may be prematurely discarded when the agent shifts focus to other directions (Zhu et al., 2026).

We formulate deep search as a constraint satisfaction problem, where a set of variables is associated with constraints that may depend on the values of other variables. To this end, we introduce a schema-guided framework: Gnosis Deepsearch for long-horizon planning, where variables containing clues and constraints are captured in a structured state representation maintained across trajectories. Gnosis is fuelled by a semantic parser which is responsible for extracting a structured representation of the input question into an initial state. This initial state is propagated across all subsequent turns and resulting trajectories. Building on these structured states, we incorporate graph-based heuristics and agent-based verification to guide efficient navigation of the search space while minimising redundant computation.

We couple Gnosis with **LEGO**: a **L**ocalised **E**mbedding and **G**lobal **O**rdering document retriever. LEGO sidesteps challenges associated with retrieving and reading long documents, particularly for LLMs with smaller context windows. LEGO feeds *partial* documents into the LLM context by identifying the most important chunks within their original text and subsequently restructuring them while maintaining their original global order. Our main contributions are as follows:

- We propose a deep search engine called Gnosis that solves problems by maintaining a shared, structured state across several trajectories. Gnosis successfully matches the performance of systems relying on significantly more computationally expensive LLMs.[1]
- We propose a methodology for efficient search space exploration by computing and scoring several parallel trajectories without relying on sequential execution.
- We shed light on the impact of the retriever within agentic QA frameworks and propose LEGO, a scalable approach that consistently improves performance across both ReAct and Gnosis while offering considerable token consumption benefits compared to full-document reading.
- We conduct extensive experiments and ablation studies on BrowseComp-Plus, and demonstrate Gnosis's competitive performance across several general-purpose multi-hop QA datasets, highlighting both strengths and weaknesses of our approach.

[1] https://huggingface.co/spaces/Tevatron/BrowseComp-Plus

2 Related Work

ReAct-based agents have been widely applied to complex search and research tasks. In this setting, the system performs multi-step reasoning across several turns, combining intermediate thoughts, tool use, and response generation. However, many earlier approaches relied on interleaving prompts, which led to catastrophic forgetting as the interaction trajectory grew longer (Trivedi et al., 2023). To mitigate context limits and residual forgetting, recent methods employ step-wise *folding* (Ye et al., 2026). However, this risks prematurely discarding promising reasoning branches (Zhu et al., 2026), thereby limiting overall search space exploration (Li et al., 2025a).

Recent approaches perform proactive context folding at each step, enabling fine-grained condensation of individual reasoning steps. In line with this, Li et al. enhance the vanilla ReAct-based Retrieval-augmented Generation pipelines through an integrated Reason-in-Documents module that summarises search results and evaluates their relevance to the running query at each turn. Other works apply deep summarisation to compress multi-step sub-tasks into higher-level representations (Ye et al., 2026). However, such folding operates over a single evolving trajectory, which can limit the exploration of alternative branches and lead to incomplete coverage of the search space.

Other approaches leverage parallel rollouts (DeepSeek-AI et al., 2025), identifying regions of high uncertainty after a single rollout, then branching out to explore additional directions (Li et al., 2025a). In these settings, candidate answers are typically aggregated into a single output by compressing and merging reasoning traces across trajectories, or alternatively through heuristic selection,

for instance by choosing the trajectory with the fewest summarisation steps. However, these methods can still suffer from sequential inefficiencies and wasted computation, as further branching often still depends on prior sequential exploration. In this paper, we introduce a schema-guided framework that scales up search space exploration by maintaining a structured state across *truly* parallel trajectories. Furthermore, we couple our resulting deep search engine with an efficient retriever, which we refer to as LEGO. LEGO sidesteps the bottlenecks of long-document reading, providing a scalable alternative to full-document processing, including in context-limited settings. Unlike retrieved evidence compression frameworks, such as RECOMP and LongLLMLingua that compress retrieved evidence or prompts after retrieval (Xu et al., 2024; Jiang et al., 2024), LEGO operates during retrieval by localising relevant chunks and restoring same-source order while preserving source text.

3 Gnosis Deepsearch

Our system is a deep search engine that is responsible for answering complex natural language questions. We refer to this engine as Gnosis Deepsearch or simply Gnosis. Gnosis aims to answer deep search questions through an intermediary structured state representation which is cascaded across different search trajectories (see Figure 2). An input question $\mathbf{q}$ by definition contains unknown variables. Within Gnosis we treat the question answering problem, across both highly complex and simpler settings, as a constraint satisfaction problem, aiming to find all ground-truth candidate values for the question variables.

As illustrated in Figure 1, complex questions introduce multiple interdependent variables (highlighted in the cyan and purple boxes). These variables cannot be resolved via straightforward path-following exploration (Gutierrez et al., 2024), necessitating advanced path-finding capabilities instead. Crucially, certain clues interconnect distinct variables, illustrating that constraints on a specific variable often depend on the state or values of others. To address this complexity, we propose a methodology to model these diverse cross-variable constraints within a structured state representation, which serves as the foundation for ultimately answering the original input question. Detailed prompts used for any agentic steps are provided in Appendix I.

3.1 Parsing an Initial Gnosis State

We start by constructing a state from the input question. As shown in Figure 2, a **Semantic Parser** takes the input question $\mathbf{q}$ and returns an *initial state*, denoted as: $\mathbf{s}^{\text{init}} := \texttt{parser}(\mathbf{q})$. This initial state possesses the property that all candidate sets for its variables are initialised as empty. Notably, the parser is tasked solely with providing a structured format of the question. It does not attempt to reason about what the variables' true values might be. Any variable constraints defined during the creation of the initial state remain unaltered during subsequent state transitions.

3.2 Structured State Management across Several Trajectories

This resulting state is at the core of our framework. Each variable of a state can have certain properties which include: (i) type (*e.g., whether the variable represents a person, a city, or a date*), (ii) question constraints (*what constraints should the variable satisfy based on the input question*), (iii) candidate (*what do we currently believe is the correct value of this variable*), (iv) candidate strength (*how confident we are about the candidate value*), and (v) clues (*what have we discovered about the variable based on our search findings*). This state schema provides a formal representation of the question and establishes a mechanism for belief updating via iterative candidate assignment.

Candidates and clues are empty in $\mathbf{s}^{\text{init}}$. After the initial state is built, the functionality is carried out in distinct agentic steps, each of which can be implemented as one or more LLM prompting and tool calling steps. Gnosis creates new states given an existing state by altering the variable candidates until we arrive at a solution. Given a state $\mathbf{s}$, the process of transitioning from one structured state representation to the next is based on two distinct agentic steps (displayed in Figure 3).

Action Proposer The agent's job is to analyse the current state and propose the most useful next research steps to improve it. $\mathbf{A^s} := \texttt{ap_agent}(\mathbf{s})$ where $\mathbf{A^s}$ is a set of p action proposals s.t. $\mathbf{A^s} = \{\mathbf{a}_1, \ldots \mathbf{a}_p\}$. Each action proposal, $\mathbf{a}_j \in \mathbf{A^s}$, is expressed as a natural language sentence describing a promising research direction given the current state, along with a confidence score in $[0, 1]$ representing the agent's estimate of its likelihood of leading to the answer. We package the action proposal with the original state, $\mathbf{s}$, to create *actions* α

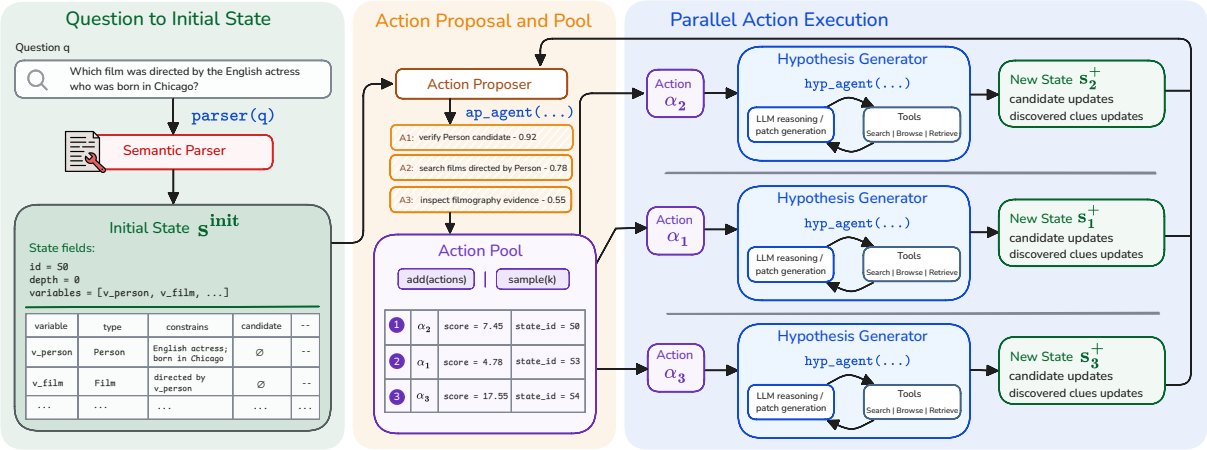


Figure 2: Overview of Gnosis Deepsearch

s.t. $\boldsymbol{\alpha}_j := [\mathbf{s}; \mathbf{a}_j] \; \forall \mathbf{a}_j \in \mathbf{A}^{\mathbf{s}}$. For instance, if our action proposer creates five action proposals given a state, we can consequently create five actions.

Hypothesis Generation Given an action $\boldsymbol{\alpha}_j$, the hypothesis agent executes the proposed research to generate a successor state $\mathbf{s}_j^+$. The agent accomplishes this by utilising its available tools (e.g., a document retriever). If the agent determines that a state transition should occur based on its findings, it outputs the updated state: $\mathbf{s}_j^+ := \texttt{hyp_agent}(\boldsymbol{\alpha}_j)$. For instance, if the action proposal is to obtain birth records for "Jacqueline Cantrelle" (as illustrated in the example of Figure 3), the hypothesis generator executes the retrieval task and integrates the newly discovered information into $\mathbf{s}_j^+$.

From any given state $\mathbf{s}$, the action agent generates a set of p distinct action proposals $\mathbf{A}^{\mathbf{s}}$, each of which is packaged with its parent context into a $\boldsymbol{\alpha}_j$ $\forall \mathbf{a}_j \in \mathbf{A}^{\mathbf{s}}$.

Gist: How can we evaluate trajectories

Because branching by a factor of p at every depth triggers an exponential expansion of the search space, executing every path blindly becomes computationally intractable. This structural bottleneck necessitates a scoring methodology to evaluate and prioritise the most promising research trajectories before they are processed by the hypothesis generator.

3.3 Graph-based Heuristics for Parallel Trajectories Re-ranking

During the creation of new states, we end up with different actions to select from, all of which are handled by the Action Pool Engine. The Action Pool Engine is responsible for deciding which actions across all trajectories to move forward with through score-based sampling (see Figure 2). Once the most suitable actions are selected, they are forwarded to the hypothesis generator, which executes them to create new states. Scores are calculated by aggregating:

- the confidence value associated with the action proposal,
- how many variables in our states have candidates and how confident the hypothesis generator was when it created these candidates (displayed as `Candidate_strength` in Figure 3), and
- the depth of the current state relative to the initial state $\mathbf{s}^{\text{init}}$

We compute the score of an action $\boldsymbol{\alpha}_j$ as follows:

$$\text{score}_{\boldsymbol{\alpha}_j} = \left(g\left(\mathbf{a}_j\right) + \sum_{i=1}^{n} e_i^{\mathbf{s}} \right) \cdot \omega\left(d^{\mathbf{s}}\right) + \mathbb{I}\left(d^{\mathbf{s}} < \rho\right) \cdot \lambda \,, \tag{1}$$

where $g\left(\mathbf{a}_j\right)$ is the confidence value associated with the action proposal, $d^{\mathbf{s}} \in \mathbb{N}$ is the depth of state $\mathbf{s}$, the sum of $e_i^{\mathbf{s}}$ over all $i \in [1, n]$, is the sum of the strengths of all candidate values of the entities in $\mathbf{s}$, $\omega \in (0, 1]$ is a depth penalty function, $d^{\mathbf{s}}$ is the depth of the current $\mathbf{s}$ state, $\mathbb{I}\left(d^{\mathbf{s}} < \rho\right)$ is an indicator function that evaluates to 1 if the depth value is less than a shallow-threshold hyper-parameter, ρ, else 0, and λ is a hyper-parameter intended to boost shallow-depth states. We formulate the depth penalty as a function of d^s:

$$\omega\left(d^{\mathbf{s}}\right) = \begin{cases} \frac{d_{\max} - d^{\mathbf{s}}}{\gamma} & \text{if } d^s > \tau \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

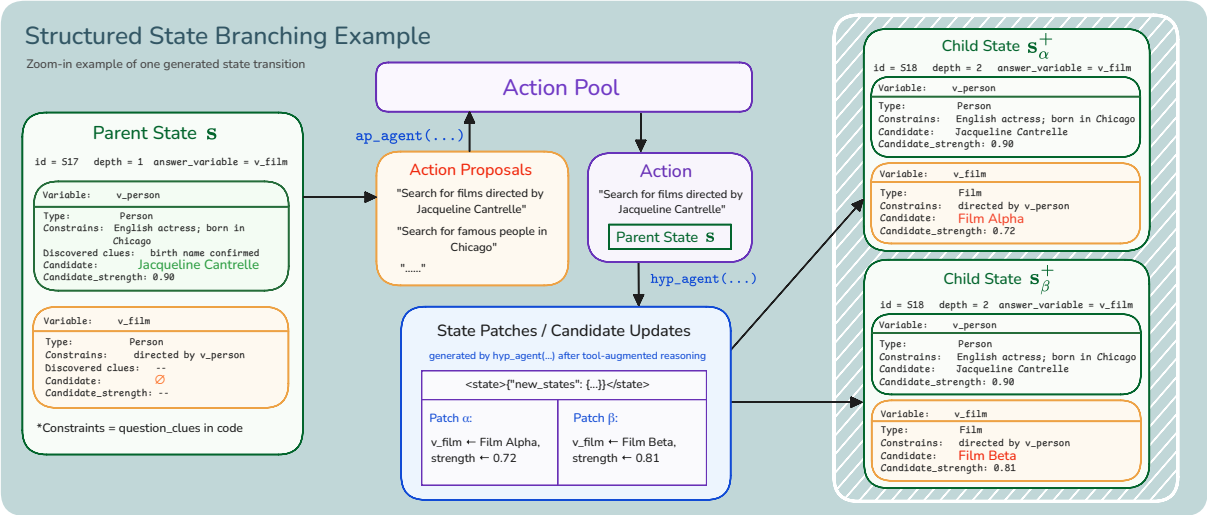


Figure 3: Structured state transition example. A partially resolved parent state grounds an action proposal, whose execution emits candidate updates for the unresolved answer variable. Applying these updates creates parallel child states that inherit shared assignments while branching over alternative candidates.

where $d_{\max}$, τ and γ are hyper-parameters representing the maximum allowed depth, the maximum allowed depth before any penalisation begins and a scaling factor respectively. In our implementation, $\gamma > d_{\max} - 1 - \tau$ to ensure $w < 1$ when $d > \tau$. This formulation encourages Gnosis to explore shallower, unproven trajectories rather than getting stuck in deep rabbit holes, achieving an effective balance between exploration and exploitation. Since each action can be executed independently, multiple workers can process distinct trajectories in parallel. The number of actions run at any given time remains fully user-configurable.

3.4 Answer Selection

The system continues this state expansion until a solution state is generated. Upon reaching this point, the system can either return the answer immediately or continue execution until a predefined number k of solution states are discovered, in which case the state is marked as *complete*. The overall execution can continue until we can aggregate k candidate answers. To select the final output from these candidates, we evaluate four aggregation strategies:

- **Greedy:** Selects the chronologically first valid answer, serving as a baseline equivalent to setting $k = 1$.
- **min(depth):** Selects the candidate with the shortest search trajectory (i.e. the fewest state transitions from the initial state). The strategy is inspired by the Parallel-fewest-step heuristic implemented (DeepSeek-AI et al., 2025).
- **max(depth):** Selects the candidate with the longest search trajectory.
- **max(conf):** Selects the candidate whose answer variable possesses the maximum confidence score. This approach emulates the test-time scaling optimisations demonstrated in BrowseComp (Chen et al., 2025).

4 Piece-by-Piece Retrieval with LEGO

We couple Gnosis with **LEGO**: a **L**ocalised **E**mbedding and **G**lobal **O**rdering document retriever. The system ingests large documents (global) by first segmenting them into chunks (local). Each resulting chunk is then prepended with semi-structured metadata, such as title, author and date, ensuring that any sparse or dense similarity model is able to capture critical high-level document context during the initial vector search phase, even within deeply buried segments.

During the retrieval phase, when multiple chunks from the same source document appear within the top-k results, the retriever dynamically consolidates them. This merging process reconstructs the natural flow of the text by arranging the chunks according to their original sequential order in the document. In order to eliminate redundancy and optimise the context window for downstream language models, the semi-structured metadata is retained only for the leading chunk and seamlessly

omitted from all subsequent merged segments.

5 Experiments

To test the ability of Gnosis on complex QA we primarily focus on the BrowseComp-Plus benchmark (Chen et al., 2025). BrowseComp-Plus incorporates the core question set from BrowseComp (Wei et al., 2025) but avoids reliance on external search APIs, whose fluctuating performance over time can compromise evaluation consistency. In addition, we explore the multi-hop QA capabilities of our framework on three open-domain datasets: MuSiQue (Trivedi et al., 2022), HotpotQA (Yang et al., 2018) and 2Wiki-MultiHopQA (2Wiki) (Ho et al., 2020). For MuSiQue and 2Wiki, we use the data provided in the IRCoT paper (Trivedi et al., 2023) which includes the full corpus, while for HotpotQA, we follow the same setting as HippoRAG (Gutierrez et al., 2024; Gutiérrez et al., 2025; Shen et al., 2025).

We follow standard evaluation practices for all benchmarks. For BrowseComp-Plus, we use the original evaluation script[2] provided by Chen et al., to report QA accuracy, recall and number of search calls. For MuSiQue, 2Wiki and HotpotQA, QA performance is evaluated using Exact Match (EM) and F1 scores (Trivedi et al., 2023). Further dataset details are provided in Appendix A. For direct comparisons, we use the exact McNemar test on paired, per-question correctness outcomes to calculate p-values. The test evaluates whether the two models differ systematically in their success rates while accounting for the shared benchmark questions.

5.1 Experiment Setup

Key hyper-parameters used for Gnosis include the maximum number of actions that can be created per state, which is set to five. We set the shallow-depth bonus value (λ) to 100. This encourages the creation of a diverse population of possible actions before exploiting the most promising ones. Unless otherwise stated, we use $k = 3$ for answer selection with the max(confidence) answer aggregation strategy (see Section 3.4). We provide the complete set of hyper-parameters used for Gnosis in Appendix K. We set up our experiments to evaluate the performance of Gnosis across three independent axes:

Large Language Models We conduct experiments using two models from distinct families, `gpt-oss-20b` (OpenAI, 2025) and `DeepSeek-V4-Flash`[3] (DeepSeek-AI et al., 2026), with context window sizes of 128k and 1 million tokens, respectively. Any mention of `gpt-oss` or `DeepSeek-V4` refers to the above LLMs unless otherwise stated.

Agentic Baselines We compare Gnosis against several agentic baselines. The first is *ReAct* (Yao et al., 2023), implemented with the Discard-all strategy. This is a context-management strategy under which, when the model's context window limit is reached, all prior tool invocation results are purged[4] from the conversation history (DeepSeek-AI et al., 2025). Second, we compare against *Search-o1*, which enhances vanilla ReAct-based Retrieval-augmented Generation pipelines through an integrated Reason-in-Documents module (Li et al., 2025b). To control for the additional test-time computation used by Gnosis, we scale up ReAct by using 12 independently sampled ReAct trajectories followed by majority voting. We refer to this ReAct variant as ReAct (maj@12). This provides a search budget that is more fairly matched to that of Gnosis and even exceeds it beyond the 7th trajectory (see Figure 6). Finally, in Appendix B, we include results from FoldAgent, a multi-agent framework based on context folding (Sun et al., 2026), under matched LLM and retrieval settings.

5.2 Indexing and Retrieval

Following Chen et al., we implement our dense retriever using the `Qwen3-Embedding-8B` embedding model (Zhang et al., 2025), retaining only the first 512 tokens of each document. We implement a hybrid retriever following the design from Biswas and Ramnath. We combine dense similarities from `Qwen3-Embedding-8B` and sparse scores from BM25 using weights of 0.6 and 0.4, respectively.

For LEGO, we employ a sentence-based tokenizer with a maximum chunk size of 2,048 tokens and an overlap of 50 tokens per chunk using the same hybrid retrieval strategy described above. Across all configurations, we consistently retrieve the top-3 documents per query. Since LEGO operates on chunks rather than full documents, we expand the maximum number of retrieved chunks to ensure a fair comparison at the document level, and

[2] https://github.com/texttron/BrowseComp-Plus

[3] The original model released in April of 2026, not to be confused with the updated `DeepSeek-V4-Flash-0731`.

[4] https://www-cdn.anthropic.com/bf10f64990cfda0ba858290be7b8cc6317685f47.pdf

Setup	BrowseComp-Plus		2Wiki		HotpotQA		MuSiQue	
	Acc.	Recall	EM	F1	EM	F1	EM	F1
ReAct								
`gpt-oss`	24.8	31.93	18.8	35.1	26.2	42.7	15.4	25.2
`gpt-oss` + LEGO	31.6	27.33	–	–	–	–	–	–
`DeepSeek-V4`	56.2	62.08	34.2	48.1	38.0	53.9	30.2	45.3
`DeepSeek-V4` + LEGO	66.8	52.47	–	–	–	–	–	–
ReAct (maj@12)								
`gpt-oss`	38.4	40.02	35.4	51.2	–	–	30.6	42.1
`DeepSeek-V4`	69.0	65.81	41.8	52.5	–	–	**46.0**	**68.7**
Search-o1								
`DeepSeek-V4`	21.4	12.60	**70.4**	**80.0**	<u>53.0</u>	<u>67.5</u>	25.6	35.9
`DeepSeek-V4` + LEGO	19.0	20.18	–	–	–	–	–	–
Gnosis								
`gpt-oss`	46.8	45.59	46.2	62.0	46.6	63.1	29.0	41.0
`gpt-oss` + LEGO	61.2	47.01	–	–	–	–	–	–
`DeepSeek-V4`	<u>72.2</u>	**70.57**	<u>62.2</u>	<u>74.6</u>	**54.0**	**69.7**	<u>39.4</u>	<u>54.0</u>
`DeepSeek-V4` + LEGO	**83.4**	<u>70.20</u>	–	–	–	–	–	–

Table 1: Performance comparison across different benchmarks using different LLMs. All setups use hybrid retrieval using `Qwen3-Embedding-8B` for embedding similarities. `DeepSeek-V4` and `gpt-oss` refer to `DeepSeek-V4-Flash` and `gpt-oss-20b` respectively. The best setup is in **bold** and the second best is <u>underlined</u>. For BrowseComp-Plus results are reported using the first 500 queries from the officially released benchmark.

we initially retrieve the top-15 (effectively 5×3) chunks. After merging the associated chunks, we ultimately filter the results to retain only the top-3 unique documents. We conduct experiments with LEGO exclusively on BrowseComp-Plus. Documents in the other multi-hop datasets are inherently short and comfortably fit within a single chunk, making further chunking unnecessary (see Appendix A).

5.3 Results

Table 1 summarises the results. Overall, Gnosis achieves the highest reported score among the evaluated systems on three of the four benchmarks. Even in the case of a shorter-context LLM (i.e., `gpt-oss`) with a 128k context window, Gnosis exhibits very strong performance on the more complex BrowseComp-Plus benchmark, improving upon the performance of base ReAct by 89% (24.8 vs 46.8 accuracy) ($p = 3.4 \times 10^{-23}$). This highlights the effectiveness of Gnosis in helping models navigate complex query spaces, as well as its strong test-time scaling ability.

While Search-o1 exhibits strong performance on the multi-hop QA datasets, it fails to address the highly complex questions within BrowseComp-Plus, even when we purposefully set the maximum number of turns to a very high limit. This may be due to its Reason-in-Documents module struggling to handle the retrieval of chunked documents. Notably, 64% of all search calls issued by the retrieval agent resulted in dead-ends. In our comparison with FoldAgent (Appendix B), we also see that Gnosis outperforms FoldAgent across all tested settings on both BrowseComp-Plus and MuSiQue.

Our LEGO retriever consistently improves both ReAct and Gnosis. Using `gpt-oss` with LEGO led to relative accuracy improvements by 27.4% ($p = 0.005$) and 30.8% ($p = 3.4 \times 10^{-9}$) for ReAct and Gnosis respectively. Similarly for `DeepSeek-V4`, we see relative improvements by 18.9% ($p = 6.2 \times 10^{-5}$) and 15.5% ($p = 4.5 \times 10^{-8}$) for ReAct and Gnosis respectively.

BrowseComp-Plus Leaderboard Gnosis powered by `DeepSeek-V4-Flash` and equipped with our LEGO retrieval strategy achieves an accuracy of 83.4 (83.6 when evaluated across all 830 BrowseComp-Plus queries). Compared to the results on the official BrowseComp-Plus leaderboard[1], Gnosis successfully matches the perfor-

Setup	Acc.	Recall	Avg. Calls
Dense	–	11.1	1.0
Hybrid	–	11.2	1.0
$\text{LEGO}_{\text{dense}}$	–	11.3	1.0
LEGO	–	10.4	1.0
ReAct w/ Dense	29	37.5	9.5
+ $\text{LEGO}_{\text{dense}}$	29	25.4	6.9
ReAct w/ Hybrid	20	29.3	9.0
+ LEGO	30	27.1	6.2
Gnosis w/ Hybrid	44	46.4	183.1 (20.9)
+ $\text{LEGO}_{\text{dense}}$ $k = 1$	47	45.5	93.1 (10.3)
+ $k = 3$ min(depth)	47	<u>46.9</u>	91.3 (9.9)
+ LEGO $k = 1$	56	43.5	94.2 (9.8)
− action pool/scoring	20	18.6	6.3 (6.3)
− action scoring	51	40.7	94.9 (12.2)
+ $k = 3$ min(depth)	<u>57</u>	43.1	92.1 (9.8)
+ $k = 3$ max(depth)	54	**55.1**	124.0 (15.0)
+ $k = 3$ max(conf)	**58**	43.1	124.0 (12.9)

Table 2: Performance ablations for Gnosis on the first 100 BrowseComp-Plus questions using `gpt-oss`. For Gnosis tool calls, we report the average number of calls across all trajectories per question, with the number of search calls for the selected trajectory provided in brackets. The best results are in **bold** and the second best are <u>underlined</u>.

Setup	Input	Output	Acc.	Appr. Cost[6]
Base	1495.2M	21.7M	73/100	$215.4
Full-doc	1742.1M	9.9M	73/100	$246.7
LEGO	1468.3M	11.7M	82/100	$208.8

Table 3: Gnosis token consumption and QA accuracy across different reading setups with `DeepSeek-V4` on the first 100 BrowseComp-Plus questions.

mance of systems relying on significantly more computationally expensive language models, such as GPT-5. This is achieved while maintaining strong performance across all the simpler multi-hop QA datasets.

Gnosis vs ReAct (maj@12) Comparing Gnosis against the more expensive ReAct variant on BrowseComp-Plus: ReAct (maj@12) shows that Gnosis's performance gains are not fully attributed to additional test-time computation or search sampling. Despite comparable compute budget, Gnosis achieves 83.4% accuracy compared with 69.0% for compute-matched ReAct ($p = 5 \times 10^{-11}$) using `DeepSeek-V4`. All Gnosis variants in Table 1 perform better ($p < 0.05$) than ReAct (maj@12) for the same LLM setting. A detailed analysis of how Gnosis's test-time compute is matched across different ReAct trajectory numbers is presented in Figure 6 of Appendix E.

Exploration Cost vs Question Complexity We further analyse how Gnosis's exploration cost scales with question complexity on BrowseComp-Plus. We use the number of question constraints/clues as a proxy for question complexity, dividing the questions by constraint count. Table 4 shows that Gnosis's cost grows with structural complexity, but much more slowly than the implicit combinatorial assignment space. We observe similar trends when approximating question complexity with raw question length, the number of variables parsed into the initial Gnosis state, and the number of gold evidence documents.

Complexity	Exec. Actions	Mater. States	Tokens	Acc.
low	12.3	6.2	1.5M	84.8
medium	13.8	6.6	3.1M	82.8
high	15.6	7.2	3.4M	81.9

Table 4: Exploration vs question complexity cost for Gnosis + LEGO with `DeepSeek-V4-Flash`. The different levels of complexity are determined after binning questions by the number of question constraints. Low-complexity questions contain up to 9 constraints, medium-complexity questions contain between 10 and 15, and high-complexity questions contain 16 or more.

Gnosis across Different Answer Depths We explore the performance of Gnosis across different answer depths (i.e., *how many turns do selected trajectories need to answer the question*, and *what is the performance at different depths/turns*) in Figure 4. We observe that most questions are answered within the first 3–4 turns, predominantly at answer depth 1. This demonstrates that even for the highly complex questions from BrowseComp-Plus, efficient parallel search space exploration (see Section 3.3) can resolve them at very shallow depths, assuming efficient parallel trajectory re-ranking. The inclusion of the LEGO retriever, primarily along with increasing the required number of answer trajectories (k), leads to improvements, ensuring higher accuracy even at deeper answer depths.

6 What makes Gnosis *actually* work?

Table 2 ablates the performance of Gnosis with `gpt-oss` to shed light on the impact of LEGO and answer selection strategies on performance and computational efficiency. We measure the recall performance of different base retrievers (i.e. dense, hybrid, LEGO, as originally implemented with a

hybrid retrieval backbone, and LEGO$_{\text{dense}}$, relying only on dense similarities) by querying them directly using the first 100 BrowseComp-Plus questions. Appendix A.1 includes our analysis about why this sample is representative of the full set.

6.1 Computing Parallel Trajectories at Scale

A primary benefit of Gnosis is that it utilises a structured state to represent a problem, which in turn is cascaded across multiple trajectories in parallel. As illustrated in Table 2, this architecture enables us to scale up tool call counts without suffering from the residual forgetting tendencies exhibited by baseline frameworks at deeper processing depths. We find that removing Gnosis's action pool and scoring mechanism (i.e. $-$ action pool/scoring in Table 2) results in the largest performance drop. This effectively reduces Gnosis to greedily selecting the most promising action at each step along a single trajectory. Random action selection without action scoring and trajectory re-ranking (i.e. $-$ action scoring in Table 2) still achieves 51% showcasing that structured state/action search helps even without score-based ranking.

Interestingly, the average number of search calls along the selected answer trajectory required by Gnosis, especially when augmented with LEGO ($9.8 - 15.0$), remains comparable with ReAct ($6.2 - 9.5$), yet it demonstrates superior performance. This indicates that solving BrowseComp-Plus does not necessarily require more complex reasoning via longer sequences of search calls, but rather a more efficient exploration of the search space, a capability at which Gnosis excels. As shown in Table 2, aggregating answers across successful trajectories ($k > 1$) improves performance, with the max-confidence-based selection slightly edging out the rest of the settings.

From Dense to LEGO Retrieval While there are only minor differences in the recall performance of the base retrievers, we observe that this does not translate into equally minor differences when these retrievers are used within agentic frameworks. The accuracy of both Gnosis and ReAct increases when a hybrid retriever is used with LEGO, however it drops when used without LEGO. This raises an important question about the role of the retriever within agentic frameworks, and whether using recall to measure performance on the original benchmark question is always a reliable metric for estimating its impact on end-to-end accuracy. LEGO also results in a lower average number of search tool calls for both ReAct and Gnosis, thereby reducing the reasoning burden on the backbone LLM. This observation is further corroborated by the output token counts displayed in Table 3.

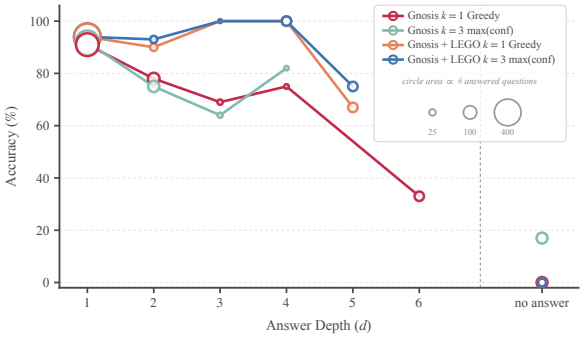


Figure 4: Performance of Gnosis variants with `DeepSeek-V4-Flash` at different answer depths across different retrieval and answer selection strategies. The circle area in each graph is proportional to the number of questions answered at that depth.

LEGO vs Full-Document Reading In Table 3, we compare the performance of Gnosis[5] across different reading strategies: (i) a base setting, where we read only the first 512 tokens from each document; (ii) LEGO; and (iii) full-document reading. Our results show that LEGO improves both performance and efficiency, outperforming full-document reading while reducing LLM cost by more than 15%.[6] Further cost analysis details are provided in Appendix H.

7 Conclusion

We introduced Gnosis, an agentic framework that maintains robust accuracy across standard multi-hop QA datasets while effectively tackling highly complex queries from BrowseComp-Plus with both short- and long-context LLMs. Gnosis takes search space exploration to the extreme by cascading a structured state across several parallel trajectories, avoiding residual forgetting common in traditional frameworks. We couple Gnosis with a retriever designed to alleviate the challenges of retrieving and reading long documents. The resulting system matches the performance of approaches that rely on more computationally expensive LLMs.

[5] We conduct these experiments with `DeepSeek-V4-Flash`, as `gpt-oss-20b`'s limited context window is insufficient for the full-document reading setting proposed by Chen et al.

[6] Cost is approximated based on pricing at `https://api-docs.deepseek.com/quick_start/pricing` in May 2026, assuming no cache hits.

Limitations

While Gnosis uses parallel rollouts to efficiently explore the action space of a question while minimising repeated search directions, this approach can be computationally expensive. Specifically, it averages approximately $7 - 13.5\times$ the cost of our ReAct baseline strategy. Due to the extensive computational cost of running experiments on BrowseComp-Plus, we utilised representative subsets for some of the ablation studies. However, we ensured that the observed trends remain consistent with the performance scores obtained on larger subsets (100 or 500 questions) and the full dataset (830 questions). The estimated full cost[6] of a full run of Gnosis `DeepSeek-V4-Flash` + LEGO is around $\$1445$ ($\$564$ if factoring in cache hits).

Furthermore, Gnosis relies on computing an initial structured state $\mathbf{s}^{\text{init}}$ for an input problem, which is subsequently cascaded across several parallel trajectories. A consequence of this design is that any error in the initial semantic parsing step, $\mathbf{s}^{\text{init}} := \text{parser}(q)$, is shared by every rollout and cannot be recovered downstream, as shown in Appendix F. This yields three recurring failure modes, including dropped sub-questions, mis-assigned answer targets, and missing answer slots. Consequently, trajectory quality is ultimately upper-bounded by the fidelity of the parser on compositionally complex questions. We leave the adaptation of this state formulation to domains beyond deepsearch QA as future work.

Finally, while we include experiments from three additional multi-hop QA datasets, our primary focus remains on BrowseComp-Plus. Consequently, the performance benefits of LEGO are exclusively explored within BrowseComp-Plus, as, to the best of our knowledge, it serves as the most suitable QA dataset for evaluating long-document reading capabilities. We also note that, following the official BrowseComp-Plus protocol, all reported accuracies are computed by an LLM-as-a-judge grader (GPT-5-mini), which is itself imperfect: it can reject correct answers over trivial surface-form differences and occasionally accept wrong numeric values under a lenient tolerance (Appendix G). While we find such errors rare and roughly offsetting, they add minor noise to the absolute scores, so closely-spaced results should be interpreted with corresponding caution.

Ethical Considerations

The development of agents capable of deep search space exploration introduces several important ethical considerations. Primarily, the parallel trajectory generation employed by Gnosis inherently increases the computational overhead and corresponding environmental footprint of inference compared to standard single-trajectory baselines. Although our LEGO retriever partially mitigates this by significantly reducing token consumption, the energy costs of parallel rollouts remain an important consideration for at-scale deployment. Furthermore, because Gnosis relies on both foundational LLMs and external search mechanisms, it inherits potential biases from its pre-training data and the retrieved content. As with any retrieval-augmented agentic system, there is an inherent risk of surfacing or aggregating misinformation. We advocate for the responsible deployment of such frameworks, emphasising the need for continuous bias monitoring and energy-aware resource allocation in real-world applications.

References

Biplob Biswas and Rajiv Ramnath. 2024. Efficient and interpretable information retrieval for product question answering with heterogeneous data. In *Proceedings of the Seventh Workshop on e-Commerce and NLP @ LREC-COLING 2024*, pages 19–28, Torino, Italia. ELRA and ICCL.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. 2025. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent. *Preprint*, arXiv:2508.06600.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. DeepSeek-V3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, and 300 others. 2026. DeepSeek-V4: Towards

highly efficient million-token context intelligence. *Preprint*, arXiv:2606.19348.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From RAG to memory: Non-parametric continual learning for large language models. In *Forty-second International Conference on Machine Learning*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, Bangkok, Thailand. Association for Computational Linguistics.

Baixuan Li, Dingchu Zhang, Jialong Wu, Wenbiao Yin, Zhengwei Tao, Yida Zhao, Liwen Zhang, Haiyang Shen, Runnan Fang, Pengjun Xie, Jingren Zhou, and Yong Jiang. 2025a. ParallelMuse: Agentic parallel thinking for deep information seeking. *Preprint*, arXiv:2510.24698.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025b. Search-o1: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5420–5438, Suzhou, China. Association for Computational Linguistics.

OpenAI. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Zhili Shen, Chenxin Diao, Pavlos Vougiouklis, Pascual Merita, Shriram Piramanayagam, Enting Chen, Damien Graux, Andre Melo, Ruofei Lai, Zeren Jiang, Zhongyang Li, Ye Qi, Yang Ren, Dandan Tu, and Jeff Z. Pan. 2025. GeAR: Graph-enhanced agent for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12049–12072, Vienna, Austria. Association for Computational Linguistics.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2026. Scaling long-horizon llm agent via context-folding.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. BrowseComp: A simple yet challenging benchmark for browsing agents. *Preprint*, arXiv:2504.12516.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Yong Jiang, Pengjun Xie, Fei Huang, Siheng Chen, and Jingren Zhou. 2026. AgentFold: Long-horizon web agents with proactive context management. In *The Fourteenth International Conference on Learning Representations*.

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

Jialiang Zhu, Gongrui Zhang, Xiaolong Ma, Lin Xu, Miaosen Zhang, Ruiqi Yang, Song Wang, Kai Qiu, Zhirong Wu, Qi Dai, Ruichun Ma, Bei Liu, Yifan Yang, Chong Luo, Zhengyuan Yang, Linjie Li, Lijuan Wang, Weizhu Chen, Xin Geng, and Baining Guo. 2026. RE-TRAC: REcursive TRAjectory Compression for deep search agents. *Preprint*, arXiv:2602.02486.

A Dataset Statistics

Table 5 summarizes the structural properties and statistics of the evaluation datasets. For BrowseComp-Plus, we evaluate the first 500 queries out of the 830 queries from the officially released benchmark (Chen et al., 2025) to balance evaluation depth with computational overhead. For all evaluated datasets, we consistently utilise the open-domain setting and restrict our focus to the answerable subset where applicable.

To calculate the EM match for our agent implementations in Table 1 when an answer is found, we parse our answer via a fixed workflow. This mitigates cases where EM fails to recognise a correct answer due to overly verbose model outputs.

A.1 Ablating performance on the smaller subset

We compare the first 100 questions with the remaining 730 questions of the full BrowseComp-Plus dataset using dataset characteristics (question/answer length, evidence document count, and gold document count) and evaluation metrics (accuracy, retrieval recall, answer depth, termination reasons, search calls, and token usage). Continuous variables are evaluated using the Mann–Whitney U test, while categorical outcomes are evaluated using Fisher's exact test. All variables show no statistically significant differences (all $p \geq 0.05$). Accuracy is 58.0% vs. 57.95% ($p = 1.00$), and retrieval recall is 0.42 vs. 0.46 ($p = 0.35$).

B Comparing against FoldAgent

FoldAgent is a long-horizon agent framework based on context folding (Sun et al., 2026). It handles a sub-task in a separate sub-trajectory, then folds the sub-trajectory by returning a short summary of its result to the main context. This keeps the active context small while allowing the agent to carry information across many search steps. We compare with FoldAgent because it addresses the same long-horizon search problem as Gnosis through a different form of context management. FoldAgent keeps past work through folded summaries, while Gnosis maintains structured states across parallel trajectories. This comparison therefore tests two different ways of preserving and using intermediate information during long searches. We run the released FoldAgent inference framework with the same LLM, corpus, retrieval, and evaluation settings as Gnosis. We evaluate 100 BrowseComp-Plus questions using accuracy and evidence recall, and 500 MuSiQue questions using EM and F1. Table 6 presents the results. The BrowseComp-Plus experiment is a separate controlled comparison from the main evaluation in Table 1. Here we see that Gnosis outperforms FoldAgent in all settings tested on both BrowseComp-Plus and MuSiQue. Furthermore, by incorporating the LEGO retriever we notice an increase in FoldAgent's performance on both benchmarks, demonstrating generality beyond Gnosis. Detailed hyper-parameters for FoldAgent are provided in Appendix K.

C Gnosis Deepsearch Running Example

Initial State


```
{
  "depth": 0,
  "id": "0",
  "retrieved_evidence_ids": [],
  "answer_variable": 4,
  "verify_result": null,
  "state": [
    {
      "id": 1,
      "type": "Person",
      "question_clues": [
        "divorced businessman",
        "ex-circus worker",
        "lost their home and life savings",
        "moved to a new continent after 2000 to
             raise their child",
        "gave a keynote speech at a college in
             the first decade of the 21st
             century",
        "founded a small organization in the
             1990s involving roving artists
             hired by various colleges"
      ],
      "discovered_clues": [],
      "candidate": null,
      "candidate_strength": null
    },
    {
      "id": 2,
      "type": "Place",
      "question_clues": [
        "birthplace of the person",
        "changed from a double-barreled name to
             a single-barreled name",
        "experienced an influx of people after
             an earthquake occurred elsewhere
             between 1900 and 1920",
        "had red streetcars that last ran in the
             1940s"
      ],
      "discovered_clues": [],
      "candidate": null,
      "candidate_strength": null
    },
    {
      "id": 3,
      "type": "College",
      "question_clues": [
        "location where the person gave a
             keynote speech in the first decade
             of the 21st century"
      ],
      "discovered_clues": [],
      "candidate": null,
      "candidate_strength": null
    },
    {
```

	BrowseComp-Plus	MuSiQue	2Wiki	HotpotQA
Split Source	BrowseComp-Plus	IRCoT	IRCoT	HippoRAG
# Test Queries	500	500	500	1,000
# Documents	67,707	115,094	430,225	9,221
# Tokens per Doc. (`tiktoken: o200k_base`)	7,354.5	100.3	96.8	123.0
# Chunks	67,707	115,094	430,238	9,222
# Chunks w/ LEGO	290,718	–	–	–

Table 5: Dataset Statistics

Setup	Method	BrowseComp-Plus		MuSiQue	
		Acc.	Recall	EM	F1
`gpt-oss-20b`	FoldAgent	30.0	35.6	25.4	36.2
	Gnosis	**44.0**	**46.4**	**29.0**	**41.0**
`gpt-oss-20b` + LEGO	FoldAgent	36.0	31.8	–	–
	Gnosis	**58.0**	**43.1**	–	–
`DeepSeek-V4-Flash`	FoldAgent	58.0	63.9	35.0	43.2
	Gnosis	**73.0**	**75.7**	**39.4**	**54.0**
`DeepSeek-V4-Flash` + LEGO	FoldAgent	60.0	57.5	–	–
	Gnosis	**82.0**	**70.3**	–	–

Table 6: Comparing Gnosis against FoldAgent under matched LLM and retrieval settings. For BrowseComp-Plus, we report accuracy and evidence recall on the first 100 questions from the officially released benchmark. For MuSiQue, we report EM and F1. LEGO is evaluated only on BrowseComp-Plus, where documents span multiple chunks.


```
      "id": 4,
      "type": "Organization",
      "question_clues": [
        "founded by the person in the 1990s",
        "involved roving artists being hired by
            various colleges",
        "small organization"
      ],
      "discovered_clues": [],
      "candidate": null,
      "candidate_strength": null
    }
  ]
}
```


Action ID	D	Score	Res.	Proposal Direction
`0_0a6f00`	0	100.80	✓	Search for small organizations founded in the 1990s that match the description "roving artists hired by various colleges". Look for terms like "artist roster", "touring artist program", "college artist agency", or "itinerant artist".
`0_c98148`	0	100.75	✓	Search for biographies or interviews of divorced businessmen and ex-circus workers who lost their home and life savings, moved continents after 2000 to raise a child, and gave keynote speeches at colleges in the 2000s.
`0_3012ae`	0	100.85	✓	Search for locations that changed from a double-barreled to a single-barreled name, had red streetcars until the 1940s, and received migrants after a 1900-1920 earthquake elsewhere.
`001_29690c 891d35_d91deb`	1	104.00	✓	Search for any other organizations founded by Derek Sivers in the 1990s that involved artists traveling to college campuses, to verify that "Professional Pests" is the only candidate matching the description.

The selected action below is the depth-1 verification action.

Action Proposals / Action Pool

The run artifact stores completed, pending, and running actions. At the end of this run, `pending` and `running` were empty. The completed action pool included:

Selected Action


```
{
  "id": "001_29690c891d35_d91deb",
  "proposal": {
```

```
    "direction": "Search for any other
        organizations founded by Derek Sivers
        in the 1990s that involved artists
        traveling to college campuses, to
        verify that 'Professional Pests' is the
        only candidate matching the description
        of 'roving artists hired by various
        colleges'.",
    "score": 0.3
  },
  "state": {
    "depth": 1,
    "id": "001_29690c891d35",
    "answer_variable": 4,
    "verify_result": null,
    "state": [
      {
        "id": 1,
        "type": "Person",
        "candidate": "Derek Sivers",
        "candidate_strength": 0.95
      },
      {
        "id": 2,
        "type": "Place",
        "candidate": "Berkeley, California",
        "candidate_strength": 0.85
      },
      {
        "id": 3,
        "type": "College",
        "candidate": "Berklee College of Music",
        "candidate_strength": 0.95
      },
      {
        "id": 4,
        "type": "Organization",
        "candidate": "Professional Pests",
        "candidate_strength": 0.95
      }
    ]
  }
}

// The full local action JSON contains the
    complete question_clues,
// discovered_clues, and retrieved_evidence_ids.
    The block above is a
// compact rendering of the same action object
    for appendix readability.
```

Hypothesis Generation

The action executor generated a retrieval call:

```
{
  "name": "retrieve",
  "arguments": {
    "query": [
      "Derek Sivers founded organizations 1990s
          artists college campuses",
      "Derek Sivers other companies before CD
          Baby",
      "Professional Pests Derek Sivers roving
          artists university"
    ]
  }
}
```

The local retrieval output included this evidence item:

```
{
  "title": "Professional Pests",
  "date": "2014-11-19",
  "text": "The Professional Pests are a group of
      roving artists for university campuses to
      add a random element to an otherwise
      ordinary day. Founded in 1994 by Derek
      Sivers, and hired by over 300 college
      campuses in the Northeast U.S., the
      Professional Pests are currently on
      hiatus."
}
```

The run-action stage then generated this final answer block:

```
<answer>
{
  "answer": "Professional␣Pests",
  "new_state": []
}
</answer>
```

The serialised result recorded the final answer and carried forward one depth-2 state. Note that the raw `<answer>` block above has `"new_state": []`, while the saved `Result.result.new_states` contains the propagated state object:

```
{
  "previous_action_id":
      "001_29690c891d35_d91deb",
  "found_answer": "Professional Pests",
  "summary": null,
  "new_states": [
    {
      "depth": 2,
      "id": "002_29690c9442cf",
      "answer_variable": 4,
      "verify_result": null,
      "state": [
        {
          "id": 1,
          "type": "Person",
          "candidate": "Derek Sivers",
          "candidate_strength": 0.95
        },
        {
          "id": 2,
          "type": "Place",
          "candidate": "Berkeley, California",
          "candidate_strength": 0.85
        },
        {
          "id": 3,
          "type": "College",
          "candidate": "Berklee College of
              Music",
          "candidate_strength": 0.95
        },
        {
          "id": 4,
          "type": "Organization",
          "candidate": "Professional Pests",
          "candidate_strength": 0.95
        }
      ]
    }
  ]
}
```

D Answer Depth Comparison

In the main body, we analysed the performance of Gnosis variants using `DeepSeek-V4-Flash` across different retrieval and answer selection strategies. For comprehensive reference, this section presents the corresponding performance evaluation utilising the `gpt-oss-20b` model, as illustrated in Figure 5.

E Scaling up ReAct: ReAct (maj@12)

A detailed analysis of how Gnosis's test-time compute is matched by aggregating results from different numbers of ReAct trajectories is presented in Figure 6.

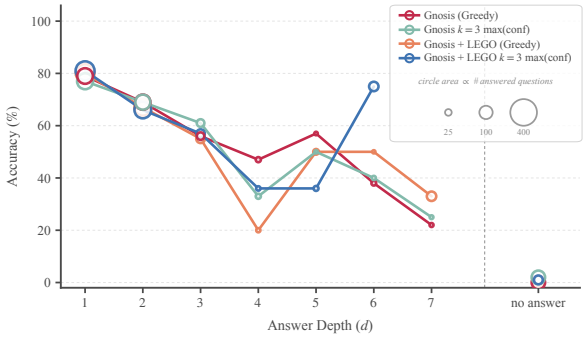


Figure 5: Performance of Gnosis variants with `gpt-oss-20b` at different answer depths across different retrieval and answer selection strategies. The circle area in each graph is proportional to the number of questions answered at that depth.

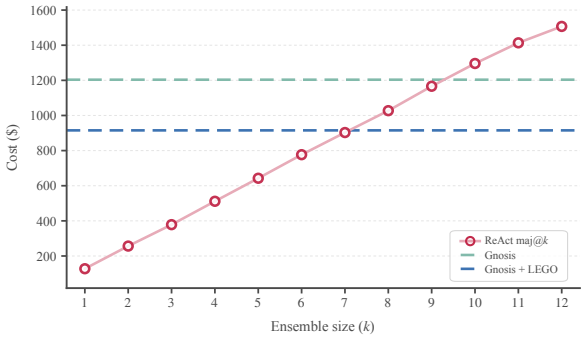


Figure 6: Analysis of how Gnosis's test-time compute budget is matched across different numbers of ReAct trajectories, with $k \leq 12$.

F Failure Mode: Error Analysis of Semantic Parser

The Gnosis pipeline is bootstrapped by a single semantic parsing step, $\mathbf{s}^{\text{init}} := \texttt{parser}(\mathbf{q})$, which decomposes the natural language query $\mathbf{q}$ into a structured initial state: a set of typed *variables* (unknown entities to be resolved), the *clues* attached to each variable and a designated *answer variable*. Every downstream trajectory operates on this shared decomposition. Consequently, errors introduced during semantic parsing propagate to all downstream trajectories. Here we illustrate three recurring failure modes, each with a case from our experimental logs where a first-step parsing error is the direct cause of the final QA failure. These failure modes are summarised in Table 7.

F.1 Dropped Sub-question

Case study (qid 12) The query asks:

> *"... What's the **name** of this potential antagonist, **and how many movements with different effects have they used**?"*

The parser produced five variables with types {**Manga**, **Group of creators**, **Literary work**, **Character**, **Character**} and set `answer_variable` to the antagonist **Character**. Crucially, no variable of type **Number** (or an equivalent count clue on the answer variable) was created: the entire "how many movements" sub-question was dropped. As a result, no downstream action ever searches for or verifies the count. The entity returned by the system was correct, but the requested count could not be reliably recovered because this sub-question was never represented in s_{init}.

F.2 Wrong Answer Target

Case study (qid 337) The query ends as follows:

> *"... the parent company produced a new motivational TV show where the president of the company served as the host and interviewer. **The interviewer's wife herself once hosted a series in the 1970s. Name the series.**"*

The parser created variables {**Television Documentary**, **TV Channel**, **Broadcasting Company**, **Motivational TV Show**, **Person** (president), **Person** (wife)} and set `answer_variable` to the **Motivational TV Show**. The intended answer, the wife's 1970s series, was *not* instantiated as its own variable but instead buried as a clue on the wife **Person** variable. The system therefore answered the wrong question, returning `The Achievers` (the motivational show) instead of the gold `Asian Horizons`.

F.3 Missing Answer Slot

Case study (qid 1190) The query requests:

> *"... identify the **name of the movie and the year it won** that prestigious award in that category."*

Although the parser created twelve variables (one per candidate film and director), it created *no* **Year** or **Award Ceremony** variable and set `answer_variable` to the winning **Movie** alone. The "year it won" component survived only as free text within the movie's clue list, where it is neither searched for nor verified as a first-class quantity. The system returned `Avatar (2009)` using the film's *release year*, whereas the gold answer is `Avatar, 2010` (the year of the award ceremony).

Failure Mode	Description	query id
Dropped Sub-question (Appendix F.1)	For a compound "X and Y" query, only the first sub-question is materialized as a variable; the second is not encoded as a variable or even as a clue, so it is never researched or verified.	qid 12
Wrong Answer Target (Appendix F.2)	When the final interrogative phrase is referentially ambiguous, the parser assigns `answer_variable` to a semantically incorrect (typically the nearest) variable, so the pipeline answers a different question.	qid 337
Missing Answer Slot (Appendix F.3)	The gold answer requires multiple independently verifiable components (e.g., an entity and a year), but the parser models only one component as a variable and compresses the other into a free-text clue that cannot be tracked or verified independently.	qid 1190

Table 7: Three recurring failure modes of the semantic parser $s_{\text{init}} := \textsc{Parser}(q)$. Each can cause an unrecoverable downstream failure because of the missing or mis-assigned structure.

The film is correct, but the temporal slot is wrong precisely because it was never allocated a verifiable variable.

G Reliability of LLM-as-a-Judge Evaluation

Following the official BrowseComp-Plus evaluation protocol, all accuracy numbers in this paper are computed by an LLM-as-a-Judge grader: for each query, a judge model (GPT-5 Mini) is shown the question, the gold answer, and the predicted answer, and decides whether the two answers are equivalent using the official grading prompt provided at: `https://github.com/texttron/BrowseComp-Plus`. While this approach typically yields reliable performance scores, there remains a small subset of instances where it fails. We qualitatively identify that such grading errors are rare and primarily fall into the two patterns below.

False negatives from surface-form differences The judge occasionally rejects a semantically correct answer over a trivial surface difference that the rubric explicitly says to ignore. For instance, for a query asking to name a video game released between 2001 and 2007 in which the player's companion is drawn into a temporal gateway, the judge ruled:

> **Gold answer:** *Billy Blade and the Temple of Time*
> **Predicted answer:** *Billy Blade: Temple of Time*
> **Verdict:** Incorrect

False positives from numeric leniency Conversely, the rubric's underspecified "small margin of error" for numbers occasionally lets a wrong value through. For a query asking for a percentage stated in a company's annual report, the judge ruled:

> **Gold answer:** `65%`
> **Predicted answer:** `66%`
> **Verdict:** Correct

H Cost per Question

On the same 500-question BrowseComp-Plus subset that is used in Table 1, ReAct + LEGO with `DeepSeek-V4-Flash` uses 0.95M tokens and costs \$0.13 per question on average (\$0.050 median), achieving 66.8% accuracy. Gnosis + LEGO uses 12.80M tokens and costs \$1.81 per question on average (\$0.313 median), achieving 83.4% accuracy—a 16.6-point gain at 13.5x the mean cost. Therefore, while Gnosis may be more expensive in absolute terms, the additional test-time compute is budgeted and yields a substantial accuracy gain. Note that the cost calculation here assumes no cache hits. In our runs we observe 33.3% and 46.7% of input tokens for ReAct + LEGO and Gnosis + LEGO respectively being cache hits.

I Prompts

I.1 Prompt for state initialisation

State Initialisation Prompt — `parser(...)`

```
You are initializing the initial **STATE** for a reasoning agent that will attempt to answer a question
     through multi-step web research.

Each **STATE** consists of a list of **VARIABLES** that represent the *unknown entities* mentioned or
     implied in the question.

Each **VARIABLE** contains:

-  `id` : a unique integer identifier
-  `type` : the type or category of the entity (e.g., `Person`, `Book`, `University`, `City`, etc.)
-  question_clues : list of textual hints or logical relationships derived from the question itself that
     help identify or constrain the entity's value
-  discovered_clues : list of clues discovered during research (leave empty at initialization)
-  candidate : the hypothesized value for the variable (leave empty at initialization)
-  candidate_strength : a float in [0,1] indicating how confident we are about the candidate (set to
     null (NOT empty string) at initialization)

In addition, the **STATE** must include:

- `answer_variable`: the id of the **VARIABLE** representing the entity that directly answers the user's
     question.
  This **MUST** correspond to one of the VARIABLE ids in the state list.
  The `answer_variable` is the entity that the user is ultimately asking for-e.g., if the question asks
       for a specific actor, then `answer_variable` should be the id of the VARIABLE whose type is
       `actor` and whose `question_clues` describe that actor.

---

### Your task

1. Analyze the question carefully.
2. Identify all meaningful variables (unknown entities) that **MUST** be resolved to answer it.
   Do not include variables that are not explicitly mentioned in the question.
3. For each variable, describe:
   - What kind of entity it represents (`type`)
   - The clues or relationships that connect it to other entities or facts in the question (these go in
        `question_clues` )
4. Do **not** attempt to guess any candidate values yet.
5. Initialize `discovered_clues` as an empty list for all variables.

---

### Output format

Think step by step inside the `"reasoning"` field: break down the question, identify every entity that
     needs to be resolved, and explain how the entities relate to each other. Then output ONLY valid
     JSON matching the exact schema below.

{
  "reasoning": "<your step-by-step analysis of the question, identifying the entities and their
       relationships>",
  "state": [
    {
      "id": <unique integer starting from 1>,
      "type": "<entity type, e.g. Person, City, Book, University>",
      "question_clues": ["<clue or relationship derived from the question>"],
      "discovered_clues": [],
      "candidate": null,
      "candidate_strength": null
    }
  ],
  "answer_variable": <integer id of the variable that directly answers the question>
}

Rules:
- Do NOT include any explanation or extra text outside the JSON
- Do NOT use markdown formatting
- Do NOT wrap the JSON in code fences
- The output must be directly parseable by a JSON parser
- candidate MUST be null for all variables
- candidate_strength MUST be null for all variables
- discovered_clues MUST be an empty list [] for all variables
- answer_variable MUST match one of the variable `id` values in the `state` list
```

I.2 Prompt for generating action proposal given the current state

Action Proposal Prompt — ap_agent (...)

```
You are an agent tasked with proposing **new research actions** to help solve a question through
     multi-step reasoning and web exploration.

Each question involves discovering and verifying multiple **unknown entities**, called VARIABLES.
Each VARIABLE is defined by:

-  `id` : a unique identifier
-  `type` : what kind of entity it represents (e.g., `Person`, `Book`, `University`, `City`, etc.)
-  `question_clues` : list of textual hints or logical relationships derived from the question itself
-  `discovered_clues` : list of clues discovered during research that help identify or constrain the
     entity
-  `candidate` : a current hypothesis for its value (may be empty if not yet found)
-  `candidate_strength` : confidence in the current candidate (between 0 and 1, may be empty)

You are given the current **STATE**, which consists of a list of these VARIABLES.
This STATE may be:

1. The **initial state** - where no candidates have yet been proposed, or
2. A **derived state** - produced by a research agent, which investigated a certain research direction
     given its state and returned the derived state given to you as input.

If this is not the initial state, you will also receive a **Result** string. This is a concise summary
     of the previous research agent's reasoning process and which variables it updated.

---

### Your task

Analyze the current `State` (and any available context from `Result` ) and propose one or more
     **Action_Proposals**.
Each action proposal describes a **research direction** - a specific focus or next step you would take
     to improve the knowledge of the current state.

**CRITICAL: Adapt your strategy based on candidate density:**

- **When there are FEW candidates** (most variables have empty candidates or very few candidates):
  Propose **broader, less constrained research directions** that can generate many potential candidates.
       Avoid overly specific constraints that would limit the candidate pool. The goal is to cast a wide
       net to discover multiple plausible options before narrowing down.

- **When there are MANY candidates** (variables have multiple candidates or strong candidates):
  Propose **more focused, constrained directions** that can help verify, eliminate, or refine existing
       candidates through contradiction checks or verification.

Each `Action_Proposal` has:

-  `direction` : a **concise, self-contained natural-language description** of one specific research
     action to perform next.
  Each direction must stand on its own - avoid conditional or sequential phrasing such as *"once the
       author is confirmed..."* or *"if initial searches are inconclusive..."*.
  The direction should describe **what to search or analyze next and why**, based only on the current
       state - not on hypothetical future outcomes.
  The direction should be focused on a single variable or a small cluster of variables in the state that
       are still unknown given the current state. It should lead to finding either a contradiction
       between the current candidates or lead to finding new candidates for the single variable or the
       cluster of variables.

  **Examples:**
  - "Investigate all referees that were of French nationality that were active in the years 2018-2022."
  - "Find 2018 interviews discussing Kathleen's writing philosophy."
  - "Suggest candidates that satisfy the following requirements: Find museums that is 2-4 kms away from
       a famous park, that also is the origins of a football club and also is the birthplace of a famous
       director."
    *(Here we are trying to find candidates that satisfy a subset of requirements variables.)*

-  `score` : a float in [0,1] indicating how promising or useful this action is for progressing toward a
     final answer.

You should:

- Do not return more than {{action_proposals_limit}} action proposals.
- Make each action direction unique by either focusing on different variables or by focusing on
     different methods to find candidates.
- Be creative and think outside the box when proposing research directions - consider unconventional
     angles, indirect connections, and alternative approaches that might reveal unexpected insights.

---

### Output format
```

```
Think step by step inside the `"reasoning"` field: analyze the current state, identify which variables
     need attention, assess candidate density, and explain your strategy. Then output ONLY valid JSON
     matching the exact schema below.

{
  "reasoning": "<your step-by-step analysis of the state, candidate density, and strategy for choosing
       research directions>",
  "action_proposals": [
    {
      "direction": "<concise, self-contained description of the research action>",
      "score": <float between 0 and 1>
    }
  ]
}

Rules:
- Do NOT include any explanation or extra text outside the JSON
- Do NOT use markdown formatting
- Do NOT wrap the JSON in code fences
- The output must be directly parseable by a JSON parser
```

I.3 Prompt for action running and tool usage

This is the prompt describing the hypothesis generation process.

Action Running and Tool Usage Prompt — hyp_agent (...)

```
You are a deep research assistant. Your input is a single Action object:
- `question`: the user's question
- `state`: the current State (list of Variables with immutable `id`, `type`, `question_clues`, and
     mutable `discovered_clues`, optional `candidate`/`candidate_strength`)
- `proposal`: an Action_Proposal with one self-contained research direction (no conditional phrasing)

Your job is to execute the proposal via focused research and return EXACTLY ONE of the following Output
     Modes:

================================================================================
OUTPUT MODES (CHOOSE EXACTLY ONE)
================================================================================

IMPORTANT: Before choosing an output mode, you MUST first think through your reasoning explicitly.
     Consider what you know, what you've learned, and why you're choosing this particular output mode.
     Only after this reasoning should you produce the output.

1) TOOL CALL MODE (when you need to gather evidence)
   - Call one of the provided tools to gather information. The orchestrator will execute the tool and
       return the tool output in the next turn.
   - Use tools minimally, with precise goals.

2) FINAL ANSWER MODE (only when you have found the answer to the question that fulfills all of the
     constraints of the question)
   - Check if the information you have gathered is enough to satisfy all of the question clues and the
       requirements of the question. If so end your response with the following tag:
    <answer>
    {
      "answer": "THE FINAL ANSWER TEXT HERE",
      "new_state": [
        {
          "id": <int>,                                # existing variable id
          "candidate": <string or null>,              # new/updated candidate
          "candidate_strength": <float or null>,      # [0,1], evidence-weighted confidence
          "discovered_clues": <complete list of all clue strings> # Optional, only include if you want
               to update the discovered clues
          # Do NOT include 'type' or 'question_clues'
        }
        # Include ONLY variables that changed. Omit all unchanged variables.
      ]
    }
    </answer>
   - If not, instead output a state patch or call a tool to gather more information.


3) STATE PATCH MODE (when you have partial updates but no final answer)
   - Return ONLY a single <state>...</state> block containing a JSON object that represents the patch.
   - The JSON should include ONLY variables that changed relative to the current state; do NOT repeat
       the entire state.
   - `id`, `question_clues` are IMMUTABLE - we cannot add, remove, renumber variables, or modify
       question_clues.
   - Format:
     <state>
     {
       "new_states": [
```

```
        {
          "state": [
            {
              "id": <int>,                                # existing variable id
              "candidate": <string or null>,              # new/updated candidate
              "candidate_strength": <float or null>,      # [0,1], evidence-weighted confidence
              "discovered_clues": <complete list of all clue strings> # Optional, only include if you
                   want to update the discovered clues
              # Do NOT include 'type' or 'question_clues'
            }
            # Include ONLY variables that changed. Omit all unchanged variables.
          ]
        }
        # Optionally multiple patches if multiple plausible hypotheses exist.
      ]
    }
    </state>

**IMPORTANT RULES FOR DISCOVERED_CLUES:**
- `discovered_clues` is REPLACED (not appended) by whatever you provide in the state patch
- If you include `discovered_clues` in the state patch, you MUST retain all information in the
    previously discovered clues unless you discover them to be incorrect or no longer relevant.
- Discovered clues should restrict the candidate pool of a variable by either eliminating specific
    candidates or by eliminating groups of candidates based on a property.
- If a candidate is changed, make sure all discovered clues that are related to that candidate are also
    updated.
- Format: "ELIMINATES: [candidate/group description] - [reasoning/evidence]. [If applicable: Include any
    assumptions about other variables that cause a contradiction or lead to this elimination] |
    Sources: [URL1, URL2, ...]"

Examples of GOOD evidence:
    * "ELIMINATES: John Smith - published work in 1823, but University College London was not
        established until 1826. Assumption: Variable 2 (university) is University College London |
        Sources: url_of_the_source"
    * "ELIMINATES: Chicago - as there is no soccer team that is based in Chicago that competed in the FA
        Cup (requirement stated in question clues) | Sources: url_of_the_source1, ..."
    * "ELIMINATES: Ned Stark - he dies in Season 1 (requirement stated in question clues). Assumption:
        Variable 4 (director) is David Benioff and Variable 6 (tv show) is Game of Thrones | Sources:
        url_of_the_source"

================================================================================
WHEN TO OUTPUT A STATE PATCH
================================================================================
You should output a <state> block when your research direction has been sufficiently explored - this
    includes both:
1. Partial progress - meaning you have executed one or more search or fetch operations and discovered
    potential candidates, new clues, or refinements for one or more variables (but not enough for a
    final answer).
2. No progress - meaning you have explored the research direction but found no new information,
    candidates, or updates to the state.

When you reach a point where the current research direction has been reasonably exhausted -
for example, further searches are returning repetitive, irrelevant, or low-value information, or no
    relevant information is found -
you should return a <state> block instead of continuing to call more tools.

**Always return a state patch to signal completion of the action:**
- If you found updates: return a state patch with the updated variables
- If you found no updates: return an empty state patch: <state>{"new_states":[]}</state>

**You may output a new state patch even if candidate values are not fully verified yet:**
- If you discover candidates for a variable but cannot fully verify it because other related variables
    are missing candidates, you should still record the candidate with an appropriate confidence level
    (typically tentative: 0.4-0.7).
- For example, if you find "John Smith" as a candidate for Variable 1 (the author), but cannot verify it
    because Variable 2 (the university) doesn't have a candidate yet, you should still create a state
    patch with "John Smith" as a candidate for Variable 1 with a tentative confidence score.
- Future actions can use this partial information to make progress on related variables, and
    verification can happen once all related variables have candidates.

The purpose of a <state> block is to signal action completion and incrementally build knowledge across
    multiple reasoning cycles.
It allows the system to:
1. Record new candidate values and confidence scores for variables discovered in this research direction.
2. Signal that the action has been completed (even if no progress was made).
3. Pass the enriched state forward so future agents can continue reasoning or explore alternate
    directions.


Furthermore you should return a <state> block when:
- You've found a contradiction between the current candidates, so you should return a state patch to
    remove/replace the candidate along with the discovered clues.
- You've determined that a candidate or group of candidates is unverifiable with online sources after
    intensive searching (cannot find sufficient information to confirm all question clues are
    satisfied), so you should return a state patch to remove/replace the candidate(s) and add
    discovered clues documenting the unverifiability. Note: This applies when unverifiability is due to
```

```
    lack of online information, NOT when verification is blocked by missing related variables.
- Multiple plausible candidates to a variable exist and should be captured as separate state patches for
    parallel exploration.

================================================================================
CRITICAL RULES
================================================================================
- ALWAYS think explicitly before choosing an output mode. Reason through what you know and why you're
    choosing this output mode.
- Return EXACTLY ONE mode per response; never mix modes.
- Do not return a candidate corresponding to a variable that is associated with a final answer if it has
    a candidate_strength of greater than 0.7, instead return a final answer.
- Every change in a <state> patch must be traceable to concrete evidence discovered.
- Partial verification is acceptable: You may output candidate values even if they cannot be fully
    verified yet because other related variables are missing. Use tentative confidence scores (0.4-0.7)
    for such candidates.
- Action completion: When an action has been explored and yields no updates to the state, you MUST
    return an empty state patch to signal completion:
  <state>
  {"new_states":[]}
  </state>
- Never let an action hang: If you cannot make progress after reasonable exploration, return an empty
    state patch rather than continuing to call tools indefinitely.
- `candidate_strength` semantics: proven >0.9; strong 0.7-0.9; tentative 0.4-0.7; weak <0.4.
- You cannot have a candidate that is eliminated in the discovered_clues and still be a candidate in the
    state.
- Unverifiability handling: If a candidate (or group of candidates across multiple variables) cannot be
    verified with online sources to satisfy all question clues after intensive searching, you MUST
    eliminate it. This applies when you cannot find sufficient information online to confirm that the
    candidate(s) meet all requirements, NOT when verification is blocked by missing related variables.
    Remove the candidate(s) and add a discovered clue documenting the unverifiability.
- State patches should normally change at least one candidate (new or updated). Exception: The special
    empty patch `<state>{"new_states":[]}</state>` is allowed to signal "no progress / no updates". Do
    NOT send a non-empty patch that leaves all candidates effectively unchanged.

================================================================================
OPERATING PRINCIPLES
================================================================================
- Use the proposal of the Action as your research direction.
- Start with targeted searches; fetch and verify before updating.
- Keep steps atomic and grounded in the current state (no conditional chains).

CURRENT DATE: {current_date}
```

I.4 Prompt for answer extraction

The prompt that is used for extracting the final answer from Gnosis and ReAct for the experiments on MuSiQue, 2Wiki and HotpotQA is as follows.

Exact Match Answer Extraction

Final answer extraction prompt for datasets using the exact match (EM) metric.

```
You are given a question, a predicted answer, and retrieved passages.

Question:
<question>
{question}
</question>

Predicted answer:
<prediction>
{prediction}
</prediction>

Retrieved passages:
<passages>
{passages}
</passages>

Instruction:
<instruction>
Select the shortest exact substring from the retrieved passages that best corresponds to the predicted
    answer.
The span must appear verbatim in the passages (case and punctuation must match).
If no suitable substring exists, return an empty string.
</instruction>

Respond with JSON only: {{"span": "<substring>"}}
```

J Retrieval tool schema

Retrieval Tool Schema

```
{
    "type": "function",
    "function": {
        "name": "retrieve",
        "description": (
            "Search a fixed corpus of pre-indexed web content. You "
            "pass natural-language queries; the tool returns the most relevant "
            "text passages from the corpus via similarity search. Use this when "
            "you need evidence or answers from that corpus. Supports 1-3 queries "
            "per call; use multiple queries to target different aspects."
        ),
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "array",
                    "items": {"type": "string"},
                    "minItems": 1,
                    "maxItems": 3,
                    "description": (
                        "One or more natural-language queries. Each query is run "
                        "against the corpus; matching passages are returned for "
                        "all queries in one response."
                    ),
                },
            },
            "required": ["query"],
        },
    },
}
```

K Hyper-parameters

Default hyper-parameters for Gnosis, Search-o1 and FoldAgent are provided below.

K.1 Gnosis and LEGO

Hyper-parameters for Gnosis and LEGO

```
# Maximum Number of Action Proposals Per Call
find_action_agent.action_proposals_limit = 5

# Execution Model (Gnosis / ReAct)
model_name = {"openai/gpt-oss-20b", "deepseek/deepseek-v4-flash"}

# State-Scoring Parameters Based on Depth
states.shallow.threshold = 2  # $\rho$ in Eq. 1
states.shallow.bonus = 100  # $\lambda$ in Eq. 1
states.max_depth = 12  # $d_max$ in Eq. 2
states.depth_penalty.threshold = 5  # $\tau$ in Eq. 2
states.depth_penalty.denominator = 8  # $\gamma$ in Eq. 2


# Termination & Answer Selection
per_question.actions_explore_limit = 125
per_question.answer_mode_top_k = 3
per_question.provide_best_guess = true

# Retriever
retrieval.top_k = 3
retrieval.top_k_multiply_factor = 5
retrieval.mode = {hybrid, dense}
retrieval.include_leading_chunk = true
retrieval.add_instruction = true
retrieval.embedder = "qwen/qwen3-embedding-8b"
retrieval.embed_dim = 4096
retrieval.hybrid_weight.dense = 0.6
retrieval.hybrid_weight.sparse = 0.4

# Evaluation Model
judge_model = "openai/gpt-5-mini"
```

K.2 Search-o1

Hyper-Parameters for Search-o1

```
# Settings Overridden for Fair Comparison
max_tokens = 16384  # 2x default

# Default Settings
temperature = 0.7
top_p = 0.8
top_k_sampling = 20
repetition_penalty = 1.0
max_search_limit = 10
max_doc_length = 3000
max_turn = 15

# Default Retriever
top_k = 10

# BrowseComp-Plus Retriever
retrieval.top_k = 3
retrieval.top_k_multiply_factor = 5
retrieval.mode = hybrid
retrieval.include_leading_chunk = true
retrieval.add_instruction = true
retrieval.embedder = "qwen/qwen3-embedding-8b"
retrieval.embed_dim = 4096
retrieval.hybrid_weight.dense = 0.6
retrieval.hybrid_weight.sparse = 0.4
```

K.3 FoldAgent

Hyper-parameters for FoldAgent

```
# Context-Folding Agent Default Settings
workflow = search_branch
prompt_length = 16384
response_length = 32768
max_turn = 200
max_session = 10
session_timeout = 5400
enable_summary = false
turn_max_new_tokens = -1
temperature = provider default
top_p = provider default
process_reward = none

# Matched Retriever Settings
retrieval.top_k = 3
retrieval.top_k_multiply_factor = 5
retrieval.mode = hybrid
retrieval.include_leading_chunk = true
retrieval.add_instruction = true
retrieval.embedder = "qwen/qwen3-embedding-8b"
retrieval.embed_dim = 4096
retrieval.hybrid_weight.dense = 0.6
retrieval.hybrid_weight.sparse = 0.4
```

L AI Usage Disclosure

AI assistants have been used during code development and to proof-read parts of the paper.