

Map-of-Actions: Deliberate Reasoning over Multi-Labeled Graphs

Wuguang Ni and Kai Yang
Tongji University
{wgni, kaiyang}@tongji.edu.cn

Abstract

Multi-step reasoning in large language models (LLMs) is typically expressed as unstructured text, making intermediate states difficult to organize, verify, and revise explicitly. This limitation often leads to redundant reasoning paths, error accumulation, and limited controllability in complex tasks. We propose Map-of-Actions (MoA), a neuro-symbolic reasoning framework that treats reasoning as operations over an explicit structured state space. MoA represents intermediate states as a multi-labeled graph, in which each node corresponds to a semantically labeled reasoning unit. This representation provides LLMs with structured memory, explicit state transitions, and flexible interfaces to external tools. Experiments on multiple complex question answering (QA) benchmarks show that MoA consistently outperforms strong baselines, improving accuracy by up to 17.9 percentage points.

1 Introduction

In recent years, a series of large language models (LLMs), such as GPT-4 (Achiam et al., 2023) and DeepSeek-R1 (Guo et al., 2025), have demonstrated strong performance in question answering (QA) and reasoning tasks. However, most existing approaches rely on unstructured text sequences to carry out reasoning, where intermediate states are implicitly represented in natural language. As a result, the reasoning process is difficult to explicitly organize, verify, and revise, especially in complex scenarios requiring multi-step decomposition and information integration (Mavi et al., 2022; Li et al., 2024; Zheng et al., 2025; Zhang, 2024), and may be further affected by adversarial perturbations when external retrieval is involved (Jiao et al., 2025).

Edward C. Tolman’s Cognitive Map Theory (Tolman, 1948) posits that humans and animals learn and navigate by constructing explicit, structured spatial representations (cognitive maps), which

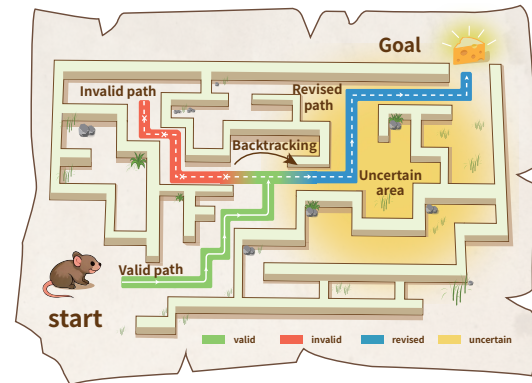


Figure 1: Illustration of cognitive map-based reasoning using a maze navigation example. An agent (mouse) incrementally constructs a structured internal representation of the environment while searching for a target (cheese). Previously explored paths are integrated into a latent cognitive map, where valid paths are marked in green and invalid paths in red. When encountering an incorrect path, the agent performs backtracking to revise its trajectory and explores alternative routes (blue), ultimately reaching the goal.

serve as an internal workspace for organizing knowledge, tracking intermediate states, and guiding behavior. As illustrated in Figure 1, such representations enable agents to flexibly explore and revise paths in a maze-like environment.

Inspired by this perspective, we model the reasoning process of LLMs as a sequence of operations over a structured state space. By introducing explicit state representations, the model is equipped with structured memory, executable operations, and accumulated experience, thereby improving the controllability and interpretability of reasoning. To this end, we propose Map-of-Actions (MoA), a neuro-symbolic reasoning framework based on multi-labeled graphs. Table 1 summarizes the differences between MoA and existing prompt-based and tool-augmented reasoning paradigms.

MoA represents the reasoning process as a multi-labeled graph, where each node corresponds to a

Table 1: Comparison of reasoning paradigms. We categorize existing methods into prompt-based and tool-augmented approaches. We compare these paradigms with MoA along key dimensions, including multi-step reasoning, tool interaction, structured state memory, explicit state transitions, and symbolic validation. Unlike prior approaches that rely on implicit intermediate states, MoA explicitly models structured intermediate states and their transitions, enabling more controllable reasoning.

Paradigm	Representative Methods	Multi-step Reasoning	Tool Interaction	State Memory	State Transitions	Symbolic Validation
Prompt-based	CoT(Wei et al., 2022), ToT(Yao et al., 2023a), GoT(Besta et al., 2024), etc.	✓	✗	✗	✗	✗
Tool-augmented	Self-Ask(Press et al., 2023), ReAct(Yao et al., 2023b), DSP(Khattab et al., 2023), etc.	✓	✓	✗	✗	✗
Neuro-symbolic	MoA (Ours)	✓	✓	✓	✓	✓

semantically labeled reasoning unit, providing explicit structured memory for LLMs. Based on this representation, we define a set of neuro-symbolic operators, including *Add*, *Edit*, *Delete*, *Merge*, and *Split*, which enable reasoning through structured graph editing. Specifically, LLMs generate symbolic operations, while their validity is verified and executed by symbolic programs. This design helps prevent circular dependencies and reduces redundant reasoning, ultimately producing interpretable and visualizable reasoning structures.

Our main contributions are as follows:

- We propose a neuro-symbolic reasoning paradigm that formulates LLM reasoning as operations over an explicit structured state space, rather than as purely linear text generation. This paradigm enables intermediate states to be explicitly organized, verified, and revised during reasoning.
- Based on this paradigm, we introduce Map-of-Actions (MoA), a neuro-symbolic reasoning framework that represents intermediate reasoning states as a multi-labeled graph. MoA equips LLMs with structured memory, explicit state transitions, graph editing operators, and flexible interfaces to external tools for iterative reasoning refinement.
- We evaluate MoA on multiple complex question answering benchmarks. It consistently outperforms strong baselines by up to 17.9 percentage points, and further analyses demonstrate its effectiveness in reasoning-path refinement, error correction, and robustness to corrupted intermediate states.

2 Related Work

Prompt-based Methods. Prompting techniques enhance the reasoning capabilities of LLMs through examples or prompts. Few-shot prompting (Brown et al., 2020) enables in-context learning. Chain-of-Thought prompting (Wei et al., 2022), along with related approaches such as Least to Most prompting (Zhou et al., 2023), Auto-CoT (Zhang et al., 2022), and Program of Thoughts (Chen et al., 2023), incorporates intermediate reasoning steps to enhance performance on complex tasks. Tree of Thoughts (Yao et al., 2023a) explores multiple reasoning paths via branching, while Graph of Thoughts (Besta et al., 2024) further incorporates merging operations to combine these paths.

Tool-augmented Methods. Tool-augmented approaches provide dynamic support through external knowledge bases or tools. Toolformer (Schick et al., 2023) and ReAct (Yao et al., 2023b) use API calls to access external information. Self-Ask (Press et al., 2023) guides the model to ask follow-up questions and use search engines to build a thought chain. Search-in-the-Chain (Xu et al., 2024) constructs a reasoning chain with query-answer pairs and verifies each step through retrieval. Chain-of-Action (Pan et al., 2025) decomposes complex problems into reasoning chains, enhancing answer credibility with multi-source information and credibility scores, and related efforts further explore efficient adaptation of tool-using LLM agents (Liu et al., 2025).

Neuro-symbolic Reasoning. Neuro-symbolic reasoning refers to the reasoning process within the neuro-symbolic computing paradigm, integrating

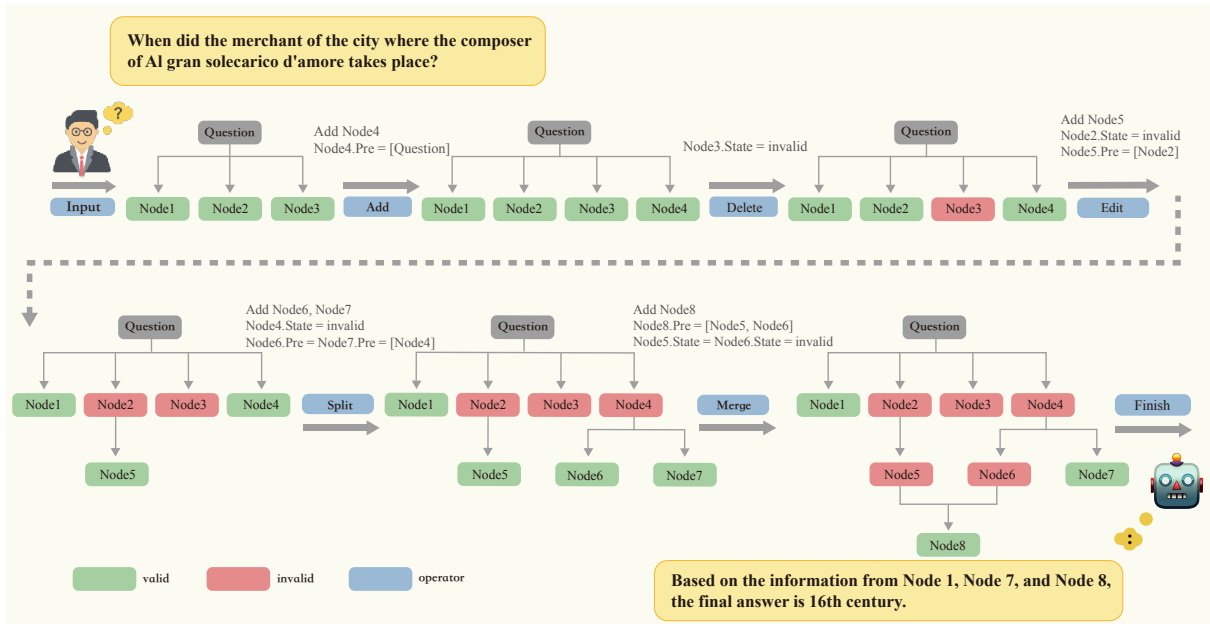


Figure 2: Given an input question, the model incrementally constructs a graph-structured reasoning state, where each node represents a semantically labeled decision unit. Through a sequence of neuro-symbolic operators (e.g., Add, Delete, Edit, Split, and Merge), the model dynamically expands, revises, and reorganizes the reasoning structure. Nodes are explicitly marked as valid (green) or invalid (red), enabling the model to track intermediate states, identify errors, and perform backtracking to explore alternative reasoning paths. This iterative process allows MoA to refine the reasoning trajectory and aggregate information from multiple nodes, ultimately producing the final answer.

neural networks’ perceptual and learning capabilities with symbolic reasoning systems’ strengths in logical reasoning and knowledge representation. Recent advances highlight the integration of LLMs with symbolic processes. For instance, Fang et al. (2024) demonstrated that LLMs with symbolic modules can perform zero-shot symbolic reasoning. Neuro-symbolic computing also encompasses the use of programming languages and symbolic expressions to implement executable constraints (Wang et al., 2024).

3 Map-of-Actions

Overview of MoA. In this section, we provide a detailed introduction to the MoA framework and illustrate its reasoning process in Figure 2. We first define three categories of actions (Section 3.1) and initialize a multi-labeled graph using a graph initialization process (Section 3.2), during which appropriate retrieval actions are executed to supplement missing information. Based on the initialized graph, we employ a graph optimization process (Section 3.4) to iteratively refine the graph structure, guided by the five predefined neuro-symbolic operators and symbolic programs described in Section 3.3. Upon convergence, we perform reasoning over the optimized graph by filtering out invalid

information and leveraging the valid nodes within the multi-labeled graph to produce an answer.

3.1 Action Definition

Complex QA tasks often involve a variety of knowledge sources, including up-to-date open-domain facts, domain-specific content, and abstract cognitive patterns. To effectively integrate these heterogeneous information resources, we introduce three fundamental action types as the core methods for information retrieval: (1) *Web-retrieval*, retrieving open-text information from the internet via search engines; (2) *Local-retrieval*, retrieving relevant texts from local domain-specific knowledge repositories; (3) *Thought-retrieval*, retrieving high-level cognitive patterns extracted from historical successful cases.

Web-retrieval. If the action type of a node is *Web-retrieval* and its *Response* label is empty, we retrieve relevant content from the internet using existing search engines (e.g., Google Search). From the retrieved webpages, we select the top-k ranked entries as candidate results.

Local-retrieval. We collect open literature, public datasets, domain knowledge sources, and common knowledge from Wikipedia. After organiz-

ing them into text format, long texts are split into smaller, more easily searchable paragraphs based on character length, alleviating the pressure on the context window of the language model. We then use the pre-trained embedding model (bge-m3-embedding from BAAI (Chen et al., 2024)) to transform these text blocks into 1024-dimensional vectors, and utilize the open-source Faiss library (Douze et al., 2025) as the underlying vector similarity search engine to build the index. When a retrieval query is input, it is converted into an embedding using the embedding model, and the top- k text blocks are selected as the final retrieval results.

Thought-retrieval. Because reasoning strategies extracted from successful past cases are often reusable and generalizable, we introduce a new action type termed *Thought-retrieval*.

This mechanism is designed to extract and abstract high-level reasoning templates from prior reasoning trajectories to serve as structured cognitive guidance for the current task. In practice, the system concatenates the node’s query with the input question, encodes the resulting text into a vector, and computes its similarity to each template vector $\text{Emb}(T_i)_{i=1}^n$ in the thought template set. It filters out templates whose similarity is below $\sigma = 0.5$, and selects the top- k from the rest:

$$T^* = \text{Top}_k \left(\left\{ T_i \in T \mid \text{Sim}(\text{Emb}(q, Q), \text{Emb}(T_i)) \geq \sigma \right\} \right) \quad (1)$$

where Q is the question, q is the query content, and T is the full set of thought templates. T_i represents the i -th template, $\text{Emb}(T_i)$ is the embedding of template T_i , $\text{Emb}(q, Q)$ denotes the embedding of the concatenated query q and question Q , $\text{Sim}(\cdot, \cdot)$ is a similarity measurement function (e.g., cosine similarity), and T^* is the resulting set of candidate templates. If $T^* = \emptyset$, the system returns: “No strategy available; self-driven solution required.” The prompt for extracting these thought templates is described in Appendix C.

3.2 Graph Initialization

Motivation for Multi-Labeled Graphs. Multi-step reasoning not only requires modeling the dependencies between steps, but also validating intermediate results and revising earlier conclusions when necessary. To this end, we represent the reasoning process as a multi-labeled graph, where

each node is associated with a semantically explicit label. This representation provides the model with an explicitly manipulable reasoning space, enabling it to backtrack, examine, and reorganize previous reasoning steps. These labels allow the model to explicitly capture dependencies between different steps, track the evolution of intermediate states, and determine whether an intermediate result should be accepted or requires further revision.

Multi-labeled Graph Construction. We use a prompt-based approach to guide the LLM to initialize the reasoning steps as a multi-labeled graph, with the specific prompt template for QA tasks provided in Appendix A. The graph has two node types: question nodes and reasoning step nodes. Edges are represented by an adjacency list of predecessor nodes, where each node lists its direct predecessors. Each edge ($u \rightarrow v$) points from a predecessor to the current node, indicating dependency (i.e., u is a premise for v). Each reasoning step node is tagged with *Action*, *Query*, *Response*, *State*, and *Predecessor*. Specifically, *Action* refers to the action type, such as *Web-retrieval*; *Query* specifies the content of the query; *Response* is the anticipated or actual response to the query. If the model can answer confidently, it generates the response; otherwise, the response is obtained by executing the action. *State* indicates the current state of the node, initially set to valid and later updated to invalid with an associated reason during graph optimization.

3.3 Neuro-Symbolic Operators

We define five neuro-symbolic operators for QA tasks: *Add*, *Edit*, *Delete*, *Merge*, and *Split* to facilitate flexible coordination of multi-step reasoning. By allowing the LLM to generate symbolic operations and delegating validation and execution to a symbolic program, the system can dynamically edit the graph structure, continually adjusting and refining the reasoning path. The symbolic program determines whether to execute a symbolic operation by verifying that its format is correct and that it adheres to the properties of a logical directed acyclic graph. If the conditions are not met, execution is rejected. In addition, we can also design other neuro-symbolic operators of various types and purposes. For example, a *Verify* operator can be used to check reasoning steps or subgraphs, thereby improving accuracy and reducing hallucinations. Meanwhile, a *Rollback* operator allows

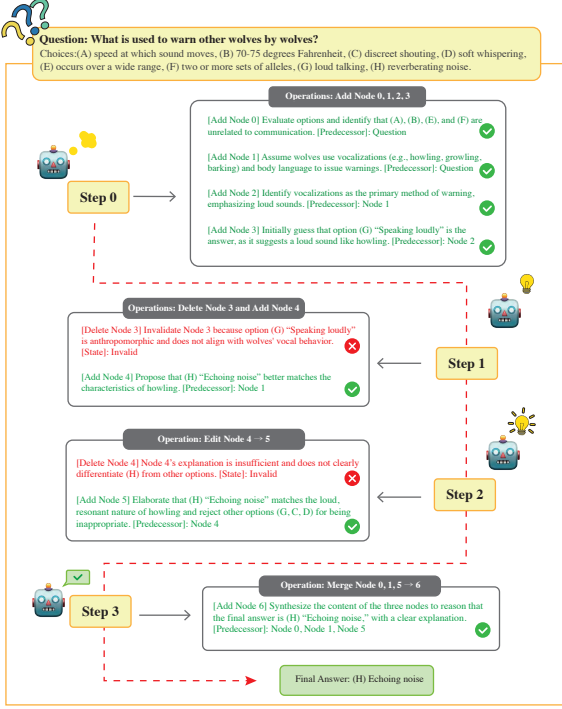


Figure 3: Illustration of the reasoning process for a QA task within the MoA framework. Starting from an initial interpretation of the question and options, the model incrementally refines its intermediate conclusions by revisiting, rejecting, and reorganizing earlier reasoning steps, and finally arrives at the correct answer.

for the reversal of previous operations, especially when a current path is found to be invalid or suboptimal. As shown in Figure 3, we demonstrate how the LLM performs symbolic operations at each step of a specific task within the MoA framework. By recording the symbolic operations at each step, MoA provides an interpretable reasoning structure for complex tasks. We discuss how LLMs collaborate with symbolic programs to preserve acyclicity during reasoning in Appendix F.

Add Operator. The Add operator inserts a new reasoning node into a multi-labeled graph. It can be applied to any predecessor node, including the user question, or a valid or invalid node. After the operation, a new valid node n' is added, while the labels of all existing nodes remain unchanged. We can formally represent the Add operator as:

$$\text{Add}(n') \Rightarrow \text{Create new node } n', \text{State}(n') = \text{valid}. \quad (2)$$

Edit Operator. The Edit operator modifies an existing valid node to correct mistakes or improve its content. After the operation, the original node n is marked as invalid and a new valid node n' is

added containing the revised information. We can formally represent the Edit Operator as:

$$\begin{aligned} \text{Edit}(n) &\Rightarrow \text{State}(n) = \text{invalid}, \\ &\text{Create new node } n', \text{State}(n') = \text{valid}. \end{aligned} \quad (3)$$

Delete Operator. The Delete operator deactivates a valid node by marking it as invalid and providing a reason. After the operation, the state of node n changes to invalid, making it ineligible for further editing operations, although it may still serve as a predecessor for subsequent Add operations. We can formally represent the Delete Operator as:

$$\text{Delete}(n) \Rightarrow \text{State}(n) = \text{invalid}. \quad (4)$$

Merge Operator. The Merge operator consolidates two or more valid nodes into a single node, reducing redundancy and summarizing the relevant information. After the operation, the original valid nodes $\{n_1, \dots, n_k\}$ are marked invalid, and a new valid node n' is introduced to represent the merged content. We can formally represent the Merge Operator as:

$$\begin{aligned} \text{Merge}(\{n_1, \dots, n_k\}) &\Rightarrow \\ &\left\{ \begin{array}{l} \forall i \in \{1, \dots, k\}, \text{State}(n_i) = \text{invalid}, \\ \text{Create new node } n', \text{State}(n') = \text{valid}. \end{array} \right. \end{aligned} \quad (5)$$

Split Operator. The Split operator breaks down a complex valid node into multiple simpler ones, helping clarify the reasoning structure. After the operation, the original node n is marked invalid, and several new valid nodes $\{n'_1, \dots, n'_k\}$ are added to represent each component. We can formally represent the Split Operator as:

$$\text{Split}(n) \Rightarrow \left\{ \begin{array}{l} \text{State}(n) = \text{invalid}, \\ \text{Create nodes } \{n'_1, \dots, n'_k\}, \\ \forall i \in \{1, \dots, k\}, \text{State}(n'_i) = \text{valid}. \end{array} \right. \quad (6)$$

3.4 Graph Optimization

Based on the five neuro-symbolic operators and three action types defined above, we design a prompt template to iteratively optimize multi-labeled graphs, as detailed in Appendix B. This prompt template guides the model to refine the graph step by step by selecting appropriate operations, following the algorithmic process described in Appendix G. The optimization process terminates either when the model determines that reasoning is complete or when the maximum number

of iterations has been reached. After optimization, valid nodes are retained while invalid information is discarded. Finally, we use a prompt template provided in Appendix D to generate the answer, where the model aggregates the retained information to produce the final result. Appendix H presents case studies with several specific examples, showcasing a variety of reasoning structures and successful symbolic editing sequences.

4 Experiments

Datasets and Tasks. To validate the effectiveness of the MoA framework on QA tasks, we employ a diverse collection of datasets, chosen to challenge various aspects of multi-step reasoning and knowledge integration. We evaluate both single-hop and multi-hop QA tasks: (1) **Single-Hop QA:** Natural Questions (NQ)(Kwiatkowski et al., 2019), AI2 Reasoning Challenge (ARC)(Clark et al., 2018), TruthfulQA (Truth)(Lin et al., 2022), StrategyQA (Strategy)(Geva et al., 2021); (2) **Multi-Hop QA:** Question Answering via Sentence Composition (QASC)(Khot et al., 2020), HotpotQA(Yang et al., 2018), 2WikiMultiHopQA (2WikiMQA)(Ho et al., 2020), MuSiQue(Trivedi et al., 2022).

Implementation and Baselines. To avoid potential test data contamination, we apply a filtering procedure in our experiments (Deng et al., 2024). By testing the model under zero-shot conditions once and excluding instances that the backbone model already answers correctly under zero-shot prompting, we minimize contamination risks. For fair comparisons with previous methods, MoA and all baselines in this study use Qwen2.5-Turbo(Yang et al., 2024) as the backbone. We compare our MoA with two categories of baselines: (1) **Prompt-based Methods**, which include Few-shot(Brown et al., 2020), Chain-of-Thought (CoT)(Wei et al., 2022), Tree of Thoughts (ToT)(Yao et al., 2023a), Graph of Thoughts (GoT)(Besta et al., 2024), Least-to-Most(Zhou et al., 2023), and Self-Refine(Madaan et al., 2023). These methods prompt LLMs to solve complex questions. (2) **Tool-augmented Methods**, which include Toolformer(Schick et al., 2023), Self-Ask(Press et al., 2023), ReAct(Yao et al., 2023b), DSP(Khattab et al., 2023), Chain-of-Action (CoA)(Pan et al., 2025), and Search-in-the-Chain (SearChain)(Xu et al., 2024). These methods enhance reasoning by integrating information retrieval techniques.

4.1 Experimental Analysis

Scenarios without Information Retrieval. As shown in Table 2, in scenarios without information retrieval, MoA generally outperforms other methods, indicating that its reasoning advantage does not fully depend on access to external knowledge. The gains are particularly pronounced on the Strategy and QASC datasets, where MoA surpasses the strongest prompt-based baselines by 9.5 and 6.1 percentage points, respectively. The main exception is the ARC dataset, where Self-Refine slightly outperforms MoA. This suggests that when the reasoning chain is relatively short and the search space is limited, the benefit of explicit graph-based operations becomes less pronounced.

Scenarios with Information Retrieval. As shown in Table 2, when information retrieval is introduced, the advantage of MoA becomes substantially more pronounced. MoA achieves the best performance among the compared baselines across all datasets, with particularly strong gains on multi-hop reasoning tasks, including an improvement of 17.9 percentage points over the best baseline on the MuSiQue dataset. Since all methods share the same retrieval backend, these improvements are more likely attributable to more effective organization and dynamic refinement of retrieved evidence. Specifically, semantic labels associated with nodes enable the model to better analyze reasoning chains, identify and eliminate misleading paths, and reorganize alternative reasoning branches. This advantage becomes increasingly important as tasks require deeper integration of evidence across multiple steps and sources.

Effect of the Maximum Number of Iterations. Iterative analysis shows that the optimal reasoning budget depends on task complexity. As shown in Figure 4, for NQ, high accuracy can be achieved with only a small number of iterations; further increasing the budget may even degrade performance, likely because additional operations introduce more opportunities for error accumulation. In contrast, MuSiQue continues to benefit from larger iteration budgets, indicating that more complex multi-hop tasks require longer refinement processes. Notably, the growth rate of the average number of iterations used is much lower than that of the maximum budget. As shown in Figure 5, on MuSiQue, the ratio of the average number of iterations to the maximum budget decreases from 60% when the maximum is

Table 2: By comparing MoA with previous methods across various tasks, considering both retrieval and non-retrieval scenarios, we conducted extensive evaluations on eight datasets, including single-hop and multi-hop QA tasks. The results average over three runs, presented with variance values omitted for clarity (all $\leq 3\%$). The "w/o IR" refers to the MoA framework without information retrieval. The maximum number of iterations for MoA is 5. In the retrieval scenarios, all methods use the same retrieval backend to access information from different sources.

Method	Single-Hop QA				Multi-Hop QA			
	NQ	ARC	Truth	Strategy	QASC	HotpotQA	2WikiMQA	MuSiQue
Without Information Retrieval								
Zero-shot	12.7	29.3	15.7	27.1	15.1	5.4	18.6	6.5
Few-shot	5.2	38.6	21.3	30.6	31.5	11.6	20.5	4.2
CoT	16.0	40.2	20.2	38.9	32.2	15.5	17.5	8.1
ToT	19.3	43.5	20.5	39.8	30.1	15.2	22.0	7.1
GoT	13.8	39.1	18.2	33.0	30.8	13.0	20.1	6.2
Least-to-Most	12.5	41.3	19.1	34.8	32.9	12.6	21.8	7.5
Self-Refine	12.1	44.5	16.4	35.6	21.9	6.8	20.7	6.8
MoA w/o IR	20.2	42.2	23.6	49.3	39.0	16.9	23.1	9.5
Interaction with Information Retrieval								
Toolformer	25.8	35.1	16.9	26.1	30.8	19.5	12.2	11.5
Self-Ask	15.6	42.5	24.5	45.3	25.3	22.0	23.5	15.5
ReAct	19.5	44.3	27.6	36.2	36.3	11.5	31.6	14.1
DSP	28.7	46.3	28.5	45.8	52.0	30.6	27.1	13.8
CoA	41.2	47.6	31.7	49.5	56.2	27.1	28.2	19.3
SearChain	37.0	45.7	30.1	47.2	50.7	40.0	30.8	21.7
MoA (Ours)	45.3	48.8	34.4	57.1	61.6	52.7	41.7	39.6

10 to 47% when it is 20. This trend suggests that MoA can stop early once the graph becomes sufficiently informative, thereby partially mitigating the overhead introduced by a larger budget.

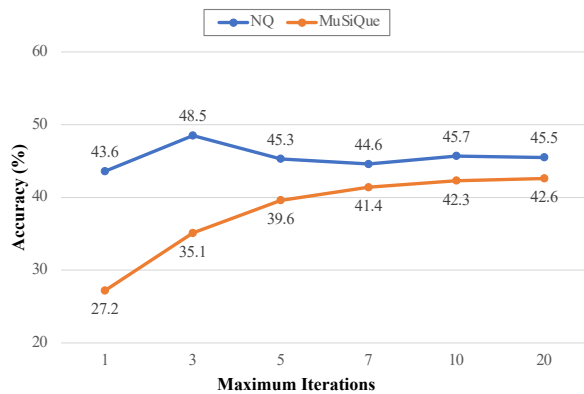


Figure 4: Accuracy under different maximum iteration budgets.

Robustness to Corrupted Node States. To analyze the robustness of the MoA framework in handling autonomous inference errors in LLMs, we design a controlled perturbation experiment on the QASC dataset. After the LLM completes graph initialization, we randomly flip valid nodes to invalid states at rates ranging from 0% to 100%, thereby systematically evaluating the impact of different levels of intermediate state errors on the reasoning process. The results are shown in Table 3. MoA

exhibits strong robustness even under node state corruption: even when all initial nodes are mislabeled, the model accuracy only slightly decreases from 61.6% to 59.4%. Meanwhile, as the perturbation rate increases, both the number of graph operations performed by the model and the inference time increase significantly, indicating that the model actively performs additional corrections to the graph structure to recover corrupted intermediate states. Notably, the average graph depth remains relatively stable, suggesting that MoA primarily recovers from erroneous states through local corrections and lateral structural adjustments (rather than significantly deepening the reasoning chain), thereby effectively mitigating the negative impact of early errors.

4.2 Ablation Study

We conduct ablation studies to assess the contribution of both the key structural components and the graph editing operators in MoA. Table 4 reports results on five representative datasets, where each variant removes one component or operator from the full model. We further analyze the impact of different LLM backbones in Appendix E.

Impact of Key Components. Removing any core component from MoA leads to varying degrees of performance degradation. First, removing the thought-retrieval mechanism results in signifi-

Table 3: Performance under different node state corruption rates on the QASC dataset.

Corruption Rate	Accuracy	Avg. Time	Avg. Operations	Avg. Nodes	Avg. Edges	Avg. Depth
0%	61.6	21.5s	6.9	6.0	7.6	4.3
30%	60.3	28.8s	9.3	6.7	8.7	4.4
50%	59.6	34.5s	11.5	7.1	9.0	4.5
70%	59.1	35.2s	12.8	7.5	9.5	4.4
100%	59.4	35.7s	13.0	7.9	9.1	4.6

Table 4: Ablation results for key components and graph editing operators. Each variant removes one component or operator from the full model. Results are averaged over three runs; variances are omitted for clarity (all $\leq 3\%$).

Dataset	MoA	w/o Thought	w/o State	w/o Predecessor	w/o Add	w/o Edit	w/o Delete	w/o Merge	w/o Split
NQ	45.3	44.5	43.2	40.8	43.0	41.9	42.7	43.6	40.8
QASC	61.6	58.9	58.2	54.1	57.5	58.8	60.2	58.2	55.4
HotpotQA	52.7	46.9	47.8	46.2	48.0	51.3	50.2	50.9	49.8
2WikiMQA	41.7	34.6	39.5	36.1	39.5	38.2	34.4	36.7	38.3
MuSiQue	39.6	35.1	37.0	32.4	36.5	35.5	35.8	36.2	36.2

cant performance drops on multi-hop tasks such as HotpotQA and MuSiQue, indicating that reusable reasoning templates provide effective high-level guidance in scenarios that require integrating evidence across multiple steps. Second, removing state memory also reduces accuracy, suggesting that explicitly retaining nodes identified as incorrect helps the model avoid revisiting previously rejected reasoning paths. Furthermore, predecessor information has a substantial impact, with a maximum drop of 7.5 percentage points on the QASC dataset, indicating that explicit dependency structures are crucial for maintaining coherent multi-step reasoning.

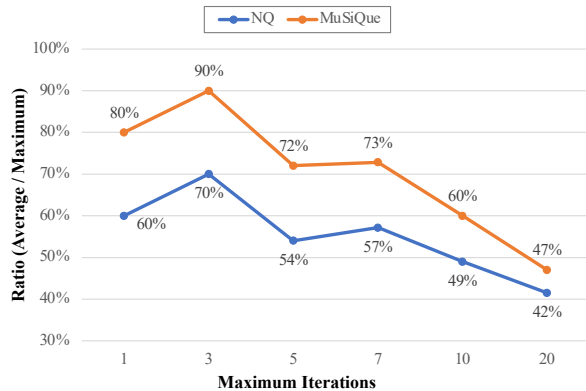


Figure 5: Ratio of average utilized iterations to the maximum iteration budget.

Impact of Graph Editing Operators. Ablation experiments on graph editing operators further demonstrate that the ability to dynamically revise the graph structure is crucial for MoA. Removing the *Add* operator degrades performance, indicat-

ing the importance of introducing new nodes when the current graph lacks sufficient information. Removing the *Edit* operator also leads to a significant performance drop, suggesting that the ability to revise earlier intermediate conclusions is important for correcting reasoning trajectories. The *Delete* operator is particularly important on 2WikiMQA and MuSiQue, as it prevents error propagation by pruning misleading nodes. Finally, *Merge* and *Split* restructure the graph at different levels of granularity: *Merge* consolidates redundant information, while *Split* decomposes complex nodes into simpler substeps.

5 Conclusion

We propose Map-of-Actions (MoA), a neuro-symbolic reasoning framework that models the reasoning process of LLMs as the iterative construction and optimization of explicit multi-labeled graphs. Experimental results show that MoA outperforms both prompt-based methods and tool-augmented methods, with particularly notable advantages in challenging multi-hop reasoning scenarios. Our findings suggest that reasoning in an explicit structured state space is a promising direction for building more reliable and structure-aware LLM systems. In future work, we plan to further strengthen structured reasoning by integrating advanced optimization techniques, such as bilevel preference optimization (Jian et al., 2025).

6 Limitations

This paper studies complex question answering as a representative testbed for structure-aware reasoning in LLMs. As a result, the generality of MoA beyond QA remains to be established through evaluation on a broader range of tasks and application settings. Although the core idea of representing reasoning as graph construction and graph editing is task-agnostic in principle, different tasks may require task-specific graph schemas, node semantics, and operator designs. The current formulation of MoA does not yet provide a unified recipe for adapting these components across domains.

In addition, MoA introduces iterative graph construction and graph optimization, which increase inference complexity relative to standard prompting methods. As problems become more complex and the reasoning graph grows larger, the framework may incur higher latency. Under such conditions, designing effective mechanisms to maintain both short-term and long-term memory remains challenging, in order to enable the model to focus on the current reasoning state without repeatedly revisiting the full reasoning history.

Finally, the current framework still relies on the underlying LLM to propose graph structures and editing operations. Although symbolic execution helps verify operation validity and improves controllability, the overall reasoning quality remains sensitive to the quality of the model’s intermediate decisions. In domains that require stronger factual grounding or stricter formal guarantees, MoA may need tighter integration with external tools, retrieval systems, or symbolic modules than we consider in this work.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. *Graph of thoughts: Solving elaborate problems with large language models*. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Aspell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. *Language models are few-shot learners*. *Preprint, arXiv:2005.14165*.
- Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. *Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation*. *Preprint, arXiv:2402.03216*.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. *Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks*. *Transactions on Machine Learning Research*.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.
- Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.
- Chunyu Deng, Yilun Zhao, Xiangru Tang, Mark Gestein, and Arman Cohan. 2024. Investigating data contamination in modern benchmarks for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8698–8711.
- Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The faiss library. *IEEE Transactions on Big Data*.
- M Fang, S Deng, Y Zhang, Z Shi, L Chen, M Pechenizkiy, and J Wang. 2024. Large language models are neurosymbolic reasoners. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence (AAAI).
- Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. *Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies*. *Transactions of the Association for Computational Linguistics*, 9:346–361.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in

- llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- 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.
- Chengtao Jian, Kai Yang, Tianhao Gao, Wuguang Ni, Keying Yang, Bowen Xiao, Jiajun Liu, and Ye Ouyang. 2025. Stable preference optimization: A bilevel approach to catastrophic preference shift. *arXiv preprint arXiv:2507.07723*.
- Yang Jiao, Xiaodong Wang, and Kai Yang. 2025. Pr-attack: Coordinated prompt-rag attacks on retrieval-augmented generation in large language models via bilevel optimization. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 656–667.
- Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2023. [Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp](#). *Preprint*, arXiv:2212.14024.
- Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. 2020. Qasc: A dataset for question answering via sentence composition. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. [Natural questions: A benchmark for question answering research](#). *Transactions of the Association for Computational Linguistics*, 7:452–466.
- Yinghao Li, Haorui Wang, and Chao Zhang. 2024. [Assessing logical puzzle solving in large language models: Insights from a minesweeper case study](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, page 59–81. Association for Computational Linguistics.
- Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. [TruthfulQA: Measuring how models mimic human falsehoods](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.
- Ya Liu, Kai Yang, Yu Zhu, Keying Yang, Chengtao Jian, Wuguang Ni, Xiaozhou Ye, and Ye Ouyang. 2025. A nested zeroth-order fine-tuning approach for cloud-edge llm agents. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 3–15. Springer.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. [Self-refine: Iterative refinement with self-feedback](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Vaibhav Mavi, Anubhav Jangra, and Adam Jatowt. 2022. Multi-hop question answering. *arXiv preprint arXiv:2204.09140*.
- Zhenyu Pan, Haozheng Luo, Manling Li, and Han Liu. 2025. [Chain-of-action: Faithful and multimodal question answering through large language models](#). In *The Thirteenth International Conference on Learning Representations*.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). *Preprint*, arXiv:2302.04761.
- Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.
- Edward C. Tolman. 1948. [Cognitive maps in rats and men](#). *Psychological Review*, 55(4):189–208.
- 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.
- Wenguan Wang, Yi Yang, and Fei Wu. 2024. Towards data-and knowledge-driven ai: a survey on neuro-symbolic computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Shicheng Xu, Liang Pang, Huawei Shen, Xueqi Cheng, and Tat-Seng Chua. 2024. [Search-in-the-chain: Interactively enhancing large language models with search for knowledge-intensive tasks](#). In *The Web Conference 2024*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. [React: Synergizing reasoning and acting in language models](#). *Preprint*, arXiv:2210.03629.

Haoteng Zhang. 2024. [Large language models still face challenges in multi-hop reasoning with external knowledge](#). *Preprint*, arXiv:2412.08317.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.

Da Zheng, Lun Du, Junwei Su, Yuchen Tian, Yuqi Zhu, Jintian Zhang, Lanning Wei, Ningyu Zhang, and Hua-jun Chen. 2025. [Knowledge augmented complex problem solving with large language models: A survey](#). *Preprint*, arXiv:2505.03418.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. [Least-to-most prompting enables complex reasoning in large language models](#). In *The Eleventh International Conference on Learning Representations*.

A Prompt for Graph Initialization

Prompt for Graph Initialization:

User: [Problem Description]

To solve the complex reasoning tasks need obtain multiple pieces of information. Therefore, you should construct an reasoning graph for the user’s problem in JSON format to facilitate information management. Each node in the graph represents

a action query, and the adjacency between nodes indicates the logical sequence. Each node in the graph should contain the following attributes: **Node ID** – a unique identifier for the node and ID is a unique number. **Action** – the type of action to be performed at this step; **Query** – the specific content or information to be retrieved; **Response** – the anticipated or action response to the query; **State** – the current status of the node (initially set to “valid”); **Predecessor** – the preceding node(s), represented as a list comprising both the key of the user’s original problem or the keys of other preceding queries.

Action List:

— **Web-retrieval** (Search for real-time and open-domain information from the internet)

— **Local-retrieval** (Search for domain-specific content from local knowledge repositories.)

— **Thought-retrieval** (Search for high-level cognitive strategies and reasoning patterns to guide problem-solving.)

Note:

- If you can confidently answer the query, provide the response; otherwise, leave it empty.

- The predecessor field must reference the key of the user’s problem or prior nodes.

After completing all the steps, output the final reasoning graph in the following JSON format:

```
Graph = {
  "Question": "User’s problem",
  "Node 0": {
    "Action": "Action type",
    "Query": "Query content",
    "Response": "Answer if available",
    "State": "valid",
    "Predecessor": List[str]
  },
  ... }
```

B Prompt for Graph Optimization

Prompt for Graph Optimization:

User: [Multi-Labeled Graph]

You are a structured thinking expert, optimizing a multi-labeled graph that represents the process of solving a complex problem. Each node in the graph represents an action query, and the adjacency between nodes indicates logical sequencing. Each node should include the following attributes: **Node ID** - A unique identifier for the node; the ID is a distinct number. **Action** - The type of operation to be executed in this step. **Query** - The specific content or information to be retrieved. **Response** - The expected answer to the query (fill in if confidently known; otherwise, leave blank). **State** - The current state of the node, either valid or invalid with a reason. **Predecessor** - This denotes the preceding steps of this node, represented as a list. It must refer to either the user's question key or the key of preceding nodes.

List of Action Types:

Web-retrieval - Retrieve real-time and open-domain information from the internet.

Local-retrieval - Retrieve domain-specific content from a local knowledge base.

Thought-retrieval - Retrieve higher-order cognitive strategies and reasoning patterns to guide problem-solving.

Your current task is to inspect the graph for missing or incorrect information and decide if any of the following operations are required: **Add** - Insert new reasoning nodes to fill logical gaps or complete the reasoning path. **Edit** - Modify a specific valid node by marking the original as invalid and adding a new node to correct local errors without affecting the overall reasoning structure. **Delete** - Remove redundant or incorrect valid nodes by marking them as invalid. Nodes already marked invalid due to deletion do not require repeat operations. **Merge** - Combine duplicate or logically related valid nodes to reduce redundancy and derive more advanced conclusions. **Split** - Break down a complex valid node into smaller, simpler sub-steps. **Finish** - Choose this if you believe all necessary informa-

tion has been collected and the reasoning process is complete.

Instructions:

In the **<think>** section, explain your reasoning process, provide specific analysis, and clearly state the chosen operation type (Add, Edit, Delete, Merge, Split, or Finish). In the **<revise>** section, based on your chosen operation and analysis, provide Python code to represent the symbolic operation you intend to carry out. The Predecessor attribute is generated at the time of node creation and must not be modified for existing nodes. For each query, if information is not found or does not meet expectations, try modifying the query or using another action type. If you can confidently answer a query, provide the response; otherwise, leave it blank. Be cautious when selecting finish; only choose it after thoroughly reviewing all responses and ensuring there are no errors. If you choose Finish, the **<revise>** section must be completely empty (do not retain or repeat the original graph content), for example: **<revise>** **</revise>**.

Final Output Format:

```
<think>
(Your detailed analysis and chosen operations)
</think>
<revise>
Example 1: Add (add a missing step)
Graph['Node 1'] = {
  "Action": "Action_Type",
  "Query": "Query_Content",
  "Response": "Answer_If_Available",
  "State": "valid",
  "Predecessor": ["Predecessor_Key"]
}
Example 2: Edit (edit a wrong node and replace with a new one)
Graph['Node 0']['State'] = "invalid (error query)"
Graph['Node 1'] = {
  "Action": "Action_Type",
  "Query": "Updated_Query_Content",
  "Response": "Answer_If_Available",
  "State": "valid",
  "Predecessor": ["Node 0"]
}
```

Example 3: Delete (remove a redundant or incorrect node)

```
Graph['Node 0']['State'] = "invalid (the reason is ...)"
```

Example 4: Merge (merge two redundant nodes into one)

```
Graph['Node 0']['State'] = "invalid (merged with Node 1)"
```

```
Graph['Node 1']['State'] = "invalid (merged with Node 0)"
```

```
Graph['Node 2'] = {  
  "Action": "Action_Type",  
  "Query": "Merged_Query_Content",  
  "Response": "Answer_If_Available",  
  "State": "valid",  
  "Predecessor": ["Node 0", "Node 1"]  
}
```

Example 5: Split (break a complex node into sub-steps)

```
Graph['Node 0']['State'] = "invalid (query too broad)"
```

```
Graph['Node 1'] = {  
  "Action": "Action_Type",  
  "Query": "Sub_Query_Content",  
  "Response": "Answer_If_Available",  
  "State": "valid",  
  "Predecessor": ["Node 0"]  
}
```

```
Graph['Node 2'] = {  
  "Action": "Action_Type",  
  "Query": "Sub_Query_Content",  
  "Response": "Answer_If_Available",  
  "State": "valid",  
  "Predecessor": ["Node 0"]  
}
```

</revise>

C Prompt for Thought Extraction

Prompt for Thought Extraction:

User: [Problem Description] + [Solution Contents]

To understand the problem-solving logic from a general perspective, even though the answer is already known. Please do not provide the specific answer, but instead analyze the general solution approach applicable to this type of problem.

Please complete the following content:

1. In the <analyse> </analyse> tags, thor-

oughly analyze the typical solving process, key computational logic, commonly used formulas, theorems, algorithms, or properties for this type of problem.

2. In the <answer> </answer> tags, provide the problem keywords and summarize three concise, transferable Strategies (each one within 50 words and not related to the specific question) that represent abstract thinking commonly used by humans when tackling such problems.

Output format:

<analyse>

(Fill in the general analysis for this type of problem here)

</analyse>

<answer>

Question Keywords: xxx

Strategies:

1. ...

2. ...

</answer>

D Prompt for Final Answer Generation

Prompt for Final Answer Generation:

User: [Question] + [Query] + [Response]

Begin your response with the [Final answer], and support it with evidence by citing query-response numbers in the format:

Reasoning: xxx [1], xxx [2]. Therefore, the final answer is: xxx.

E Impact of Different LLM Backbones

As shown in Table 5, we evaluate the effect of different LLM backbones on MoA using the NQ and QASC datasets. The evaluated models include open-source models from the Gemma2 (Team et al., 2024) and Qwen2.5 (Yang et al., 2024) series, as well as closed-source models such as GPT-3.5-turbo (Brown et al., 2020) and Gemini-2.5-flash (Comanici et al., 2025). Results show that MoA consistently improves the performance of all base models, both with and without information retrieval.

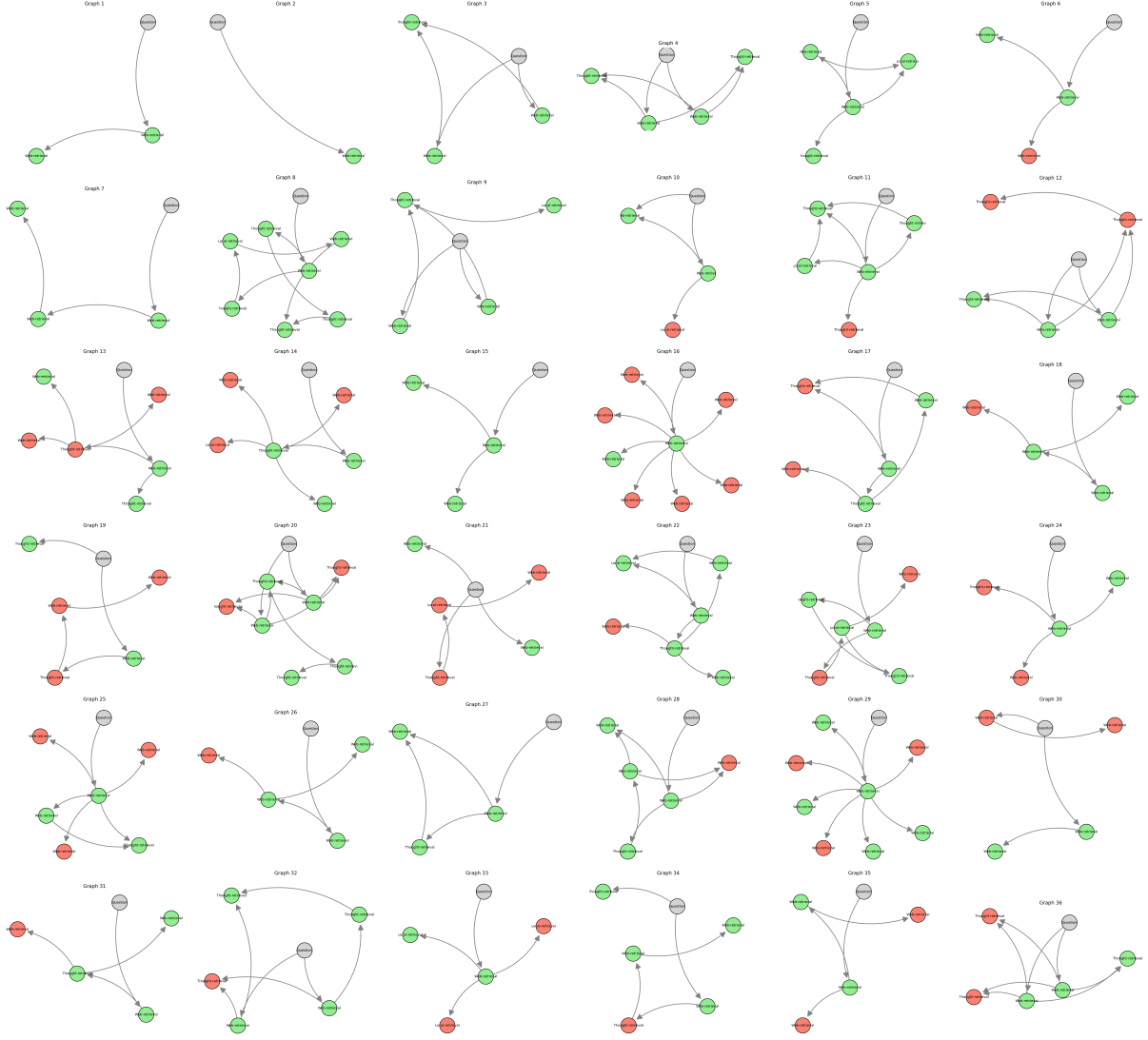


Figure 6: Various Reasoning Processes of Large Language Models under the MoA Framework.

F Acyclicity Preservation in Neuro-Symbolic Reasoning

F.1 Acyclicity of the graph structure

We define a multi-labeled directed graph $G = (V, E)$, where each node $n \in V$ represents a reasoning step, and each edge $(u \rightarrow v) \in E$ denotes a dependency (i.e., u serves as a premise for v). Each node is assigned a label $\text{label}(n) \in \{\text{valid}, \text{invalid}\}$. Our objective is to demonstrate that the graph G_t remains a directed acyclic graph (DAG) at every time step $t \in \mathbb{N}$.

Assume that the initial graph $G_0 = (V_0, E_0)$ is acyclic. The graph evolves through a sequence of operations:

$$G_{t+1} = \mathcal{O}_t(G_t), \quad (7)$$

$$\mathcal{O}_t \in \{\text{Add}, \text{Edit}, \text{Delete}, \text{Merge}, \text{Split}\}.$$

Each operation \mathcal{O}_t transforms the current graph $G_t = (V_t, E_t)$ into the next graph $G_{t+1} = (V_{t+1}, E_{t+1})$, subject to the following constraints: (i) Any newly added node $n' \in V_{t+1} \setminus V_t$ must have zero out-degree, and its incoming edges must originate exclusively from nodes in the valid set $V_t^{\text{valid}} = \{n \in V_t \mid \text{label}(n) = \text{valid}\}$; (ii) The set of predecessor nodes for any node is fixed at the time of its creation and cannot be modified thereafter.

By analyzing each operation individually for its effect on the acyclic nature of the graph, the following observations can be made: in the Add operation, the newly introduced node has zero out-degree and receives edges only from existing nodes, thus no cycles can form. In the Edit operation, the original node is marked as invalid, and a new node is added that receives incoming edges solely from the

Table 5: Performance of MoA with different LLM backbones on NQ and QASC.

Model	NQ	QASC
Gemma2-9b	18.5	28.8
Gemma2-9b MoA w/o IR	21.7	30.1
Gemma2-9b MoA	25.6	35.6
Gemma2-27b	23.3	30.8
Gemma2-27b MoA w/o IR	29.7	39.7
Gemma2-27b MoA	42.1	51.4
Qwen2.5-14b	15.2	30.1
Qwen2.5-14b MoA w/o IR	22.7	33.6
Qwen2.5-14b MoA	30.2	37.0
Qwen2.5-72b	16.5	31.5
Qwen2.5-72b MoA w/o IR	26.2	42.5
Qwen2.5-72b MoA	38.5	53.4
GPT-3.5-turbo	19.5	28.2
GPT-3.5-turbo MoA w/o IR	29.1	36.7
GPT-3.5-turbo MoA	34.8	45.8
Gemini-2.5-flash	27.9	49.1
Gemini-2.5-flash MoA w/o IR	33.5	56.5
Gemini-2.5-flash MoA	48.7	63.2

original node and emits none, thereby preserving acyclicity. The Delete operation marks the target node as invalid without modifying the edge structure, so no cycles are introduced. In the Merge operation, the merged nodes are marked as invalid, and a new node is created that receives incoming edges from the merged nodes; since this node has no outgoing edges, the acyclic structure is maintained. In the Split operation, the original node is marked as invalid and replaced with multiple new nodes, each receiving edges only from the original node. As these edges do not form a path back to the source, no cycles are introduced. During the reasoning process, Python program is used to verify whether the symbolic operation format generated by the large language models (LLMs) is correct to determine whether to execute the symbolic operation; if the requirements are not met, execution is rejected.

Therefore, under the structural constraints imposed by neuro-symbolic operators, the multi-labeled graph G_t is guaranteed to remain acyclic at all time steps t .

F.2 Graph Structures for Reasoning

Definition 1. We define an infinite directed graph $G = (V, E)$, where the node set $V = \{s_1, s_2, s_3, \dots\}$ represents all possible reasoning steps, forming a countably infinite set. The edge set $E \subseteq V \times V$ represents the dependencies between these steps. A directed edge $(s_i, s_j) \in E$ signifies that reasoning step s_j is dependent on step s_i .

The Problem of Cyclic Dependencies. In many real world reasoning problems, the solution process consists of a finite sequence of steps with a clear beginning and end. For the overall reasoning process to be logically sound and rigorous, the dependencies between steps must be unidirectional and, crucially, must not form cycles. The presence of a cyclic dependency, for example, when step A depends on B, B depends on C, and C ultimately depends on A, creates a logical loop. This prevents the correctness of any individual step from being verified independently, thereby invalidating the entire chain of reasoning. In more severe cases, such cycles may introduce logical contradictions, making any final conclusion impossible to establish. This leads to our core assumption:

Assumption 1. *The solution process for a general reasoning problem consists of a finite series of steps, and the dependencies between these steps are strictly unidirectional.*

Modeling Reasoning as a Graph. Based on Assumption 1, we can model the process of solving a general reasoning problem as the search for a finite, feasible directed acyclic subgraph $G' = (V', E')$ within the infinite graph $G = (V, E)$. In this subgraph, $V' \subseteq V$ is the finite set of reasoning steps, and $E' \subseteq E \cap (V' \times V')$ represents the dependencies between them, satisfying the directed acyclic property. Since the complete structure of the infinite graph is unknown, solving a reasoning problem essentially involves an exploratory process of incrementally building this finite and feasible directed acyclic subgraph G' .

To aid large language models in this construction, the MoA framework introduces a multi-labeled graph representation. In this graph structure, each node is annotated with semantic labels to explicitly define the semantic role and dependency relationship of each reasoning step.

G Algorithm for Multi-labeled Graph Optimization

H Case Study

H.1 Diversified reasoning structures

To facilitate the observation of the detailed reasoning process of large language models, we present in Figure 6 a selection of experimental results on the HotpotQA dataset. Each subfigure represents a single reasoning process: gray nodes indicate the

Algorithm 1 Multi-labeled Graph Optimization Workflow

Require: Multi-labeled Graph G , Question Q , Optimization Prompt p_t , Operator Set O , Max Steps k

Ensure: Optimized Multi-labeled Graph G'

```
1:  $G' \leftarrow G, iter \leftarrow 0$ 
2: while  $iter < k$  do
3:    $operators \leftarrow \text{LLM}(p_t, G')$  {LLM selects from  $O$  or returns Finish}
4:   if  $operators = \text{Finish}$  then
5:     break
6:   end if
7:   for all  $operator \in operators$  do
8:      $\text{APPLYOPERATOR}(G', operator)$ 
9:   end for
10:  for all  $node \in G'$  do
11:    if  $node \neq Q$  and  $node.response$  is empty then
12:      switch  $node.action$ 
13:        case Web-retrieval:
14:           $node.response \leftarrow \text{WEBRETRIEVAL}(node.query)$ 
15:        case Local-retrieval:
16:           $node.response \leftarrow \text{LOCALRETRIEVAL}(node.query)$ 
17:        case Thought-retrieval:
18:           $node.response \leftarrow \text{THOUGHTRETRIEVAL}(node.query, Q)$ 
19:      end if
20:    end for
21:   $iter \leftarrow iter + 1$ 
22: end while
23: return  $G'$ 
```

question, green nodes denote valid steps, and red nodes represent invalid steps. The label of each node corresponds to the action type chosen by the large language model.

H.2 Graph Structures After Completing Reasoning

Question 1. *What class of instrument does Apatim Majumdar play?*

Node 0 Action: Web-retrieval; *Query:* Apatim Majumdar musical instrument; *Response:* Apatim Majumdar (born 1978), is an Indian classical musician from Kolkata, India. He plays the Sarod. His “Dhrupadee Veenkar” style of Ustad Alauddin Khan’s Seni Veenkar Gharana: analytically imaginative prosodic progression with

metrical waves of melodic phrases interwoven with highest order of technical brilliance and maturity reminds the great legends of the Gharana. He is an internationally recognized instructor of classical Indian music.; *State:* valid; *Predecessor:* Question 1.

Node 1 Action: Thought-retrieval; *Query:* Identify the class of musical instruments; *Response:* Classification, Instrument. 1. Break down complex domains into smaller, manageable categories for easier classification. 2. Cross-reference multiple sources to ensure consistency and accuracy in categorization. 3. Use context clues and associated terminology to infer relationships between entities.; *State:* valid; *Predecessor:* Node 0.

Node 2 Action: Local-retrieval; *Query:* Musical instrument classes; *Response:* String, Percussion, Woodwind, Brass; *State:* valid; *Predecessor:* Node 1.

Node 3 Action: Thought-retrieval; *Query:* Extract class of musical instrument from Node 0’s response; *Response:* String; *State:* valid; *Predecessor:* Node 0.

Question 2. *Who is the writer of this song that was inspired by words on a tombstone and was the first track on the box set Back to Mono?*

Node 0 Action: Local-retrieval; *Query:* The first track on the box set Back to Mono.; *Response:* Back to Mono surveys recordings leading up to and featuring the Wall of Sound, Spector’s famed production trademark. Starting with “He’s A Rebel” on disc one, through “I’ll Never Need More Than This” and including disc four, all of the musical arrangements are by Jack Nitzsche.; *State:* valid; *Predecessor:* Question 2.

Node 1 Action: Thought-retrieval; *Query:* identify song based on inspiration and album context; *Response:* Brand, Album, Song. 1. Identify key entities and their relationships in the context. 2. Use chronological or categorical filters to narrow down options. 3. Cross-reference multiple data points to validate connections logically.; *State:* invalid (vague thought process); *Predecessor:* Node 0.

Node 2 Action: Web-retrieval; *Query:* The Beatles Back to Mono tracklist; *Response:* Mono Mas-

ters is a double CD compilation from The Beatles in Mono box set, featuring only tracks with dedicated mono mixes. It serves as the mono counterpart to Past Masters, omitting songs like “Old Brown Shoe,” “The Ballad of John and Yoko,” and “Let It Be” which had no true mono versions. Instead, it includes tracks prepared for a scrapped 1969 mono Yellow Submarine EP (tracks 9–12, 15), and ends with “Get Back” / “Don’t Let Me Down,” the band’s final mono single.; *State*: valid; *Predecessor*: Node 0.

Node 3 *Action*: Web-retrieval; *Query*: writer of the first track on The Beatles Back to Mono; *Response*: Back to Mono (1958–1969) is a box set (4 compact discs or 5 vinyl LPs) compilation of the recorded work of record producer Phil Spector, through the 1960s, released in 1991 by ABKCO as #7118-2. The first track, “To Know Him Is to Love Him,” released in 1958, features Spector performing as part of the group the Teddy Bears. Initially a vinyl album-sized package, the box contained a booklet with photographs, complete song lyrics, discographical information, and a reproduction of the essay on Spector by Tom Wolfe, “The First Tycoon of Teen.” The package also contained a small, round, red “Back to Mono” pin.; *State*: valid; *Predecessor*: Node 2.

Node 4 *Action*: Thought-retrieval; *Query*: Identify the songwriter of the first track on The Beatles’ “Back to Mono” using contextual clues; *Response*: Identify the song title and its connection to the album Back to Mono by cross-referencing known sources such as liner notes and historical documents, and incorporating biographical details of Phil Spector.; *State*: valid; *Predecessor*: Node 3.

Node 5 *Action*: Thought-retrieval; *Query*: Confirm the writer of the first track on The Beatles’ “Back to Mono”; *Response*: Phil Spector; *State*: valid; *Predecessor*: Node 4, Node 2.

H.3 Successful Symbolic Operation Processes

Question 1. *A hot rock is dropped into a pail of cool water. Heat energy is transferred from the rock to the water by choices: (A) boiling, (B) evaporation, (C) conduction, (D) radiation*

Step 0: Initial Graph Construction

- **Operations:** Add Node 0, 1, 2, 3, 4.

- **[Add Node 0]** Hypothesize that heat transfer occurs primarily through direct contact between the hot rock and cool water, suggesting conduction. [Predecessor]: Question.
- **[Add Node 1]** Define conduction as the transfer of heat energy through direct physical contact between substances, without the movement of the substances themselves. [Predecessor]: Node 0.
- **[Add Node 2]** Confirm the physics definition of conduction: heat transfer via molecular collisions in solids or fluids in contact, applicable to a hot rock submerged in water. [Predecessor]: Node 1.
- **[Add Node 3]** Validate conduction as the primary method for heat transfer in this scenario, as the rock is in direct contact with the water. [Predecessor]: Node 2.
- **[Add Node 4]** Identify option (C) “conduction” as the most likely heat transfer method among the given choices. [Predecessor]: Node 3.

Step 1: Refine Node 4 for Precision

- **Operations:** Delete Node 4, Add Node 5.
- **[Delete Node 4]** Invalidate Node 4 because its answer, while correct, lacks explicit alignment with the multiple-choice option “C. conduction.” [State]: Invalid.
- **[Add Node 5]** Explicitly state that option (C) “conduction” is the correct answer, as it matches the mechanism of heat transfer through direct contact. [Predecessor]: Node 3.

Step 2: Enhance Explanation

- **Operations:** Delete Node 5, Add Node 6.
- **[Delete Node 5]** Invalidate Node 5 because its explanation is too brief and lacks detailed justification for why conduction is correct and other options are not. [State]: Invalid.
- **[Add Node 6]** Elaborate that (C) “conduction” is correct because heat transfers from the hot rock to the cool water through direct molecular contact. Reject (A) “boiling” as it requires water to reach 100°C, which is unlikely here; (B) “evaporation” as it involves phase change at the surface, not bulk heat transfer; and (D) “radiation” as it is minimal compared to conduction in this scenario. [Predecessor]: Node 5.

Step 3: Resolve Redundancy and Finalize

- **Operations:** Delete Node 6, Merge Nodes 0, 1, 6 → 7, Edit Node 0.
- **[Delete Node 6]** Invalidate Node 6 because it partially overlaps with earlier nodes and can be synthesized for clarity. [State]: Invalid.
- **[Add Node 7]** Synthesize Nodes 0, 1, and 6 to confirm that (C) “conduction” is the answer. Explain that conduction occurs due to direct contact between the rock and water, transferring heat via molecular collisions. Other options—(A) boiling, (B) evaporation, and (D) radiation—are unsuitable due to their specific requirements or minimal relevance. [Predecessor]: Node 0, Node 1, Node 6.
- **[Edit Node 0]** Update Node 0 for clarity to emphasize that conduction involves heat transfer through direct physical contact between the hot rock and water. [Predecessor]: Question.

Step 4: Generate Final Answer

- **Operations:** Finish.
- **[Finish]** Synthesize reasoning to confirm (C) “conduction” as the final answer. Resolution: Reasoning is complete, with no errors.

Final Answer: (C) conduction

Question 2. *What kind of weather can cause forest fires? Choices: (A) blizzard, (B) tornado, (C) rain, (D) a cold environment, (E) thunderstorm, (F) tropical cyclone, (G) heated by the sun, (H) very complex*

Step 0: Initial Graph Construction

- **Operations:** Add Node 0, Add Node 1.
- **[Add Node 0]** Hypothesize that weather conditions causing forest fires are typically hot, dry, and windy. This points toward options that introduce heat or dry conditions. [Predecessor]: Question.
- **[Add Node 1]** Define the key components of weather that can ignite fires: a source of ignition (like lightning), dry conditions (low humidity), and fuel (dry vegetation). [Predecessor]: Node 0.

Step 1: Evaluate Options Against Hypothesis

- **Operations:** Add Node 2, Add Node 3.
- **[Add Node 2]** Analyze the given options: blizzard, tornado, rain, and a cold environment are unlikely. Thunderstorm, tropical cyclone, and being heated by the sun are plausible candidates. [Predecessor]: Node 1.
- **[Add Node 3]** Specifically, a thunderstorm can generate lightning, a well-known natural cause of wildfires, especially when accompanied by little rain (dry thunderstorms). A tropical cyclone can bring strong winds that fan existing fires, but also heavy rain that can extinguish them, making its effect variable. Being heated by the sun is a contributing factor (high temperatures) but not a direct “cause” like lightning. [Predecessor]: Node 2.

Step 2: Select the Most Direct Cause

- **Operations:** Add Node 4.
- **[Add Node 4]** Conclude that a thunderstorm is the most direct and common natural cause of forest fires listed in the options, due to lightning strikes acting as an ignition source. While other options like tropical cyclones and high temperatures can contribute, a thunderstorm provides the most direct ignition mechanism. [Predecessor]: Node 3.

Step 3: Finalize the Answer

- **Operations:** Add Node 5.
- **[Add Node 5]** Based on the analysis, option (E) Thunderstorm is the correct answer. The primary reason is the lightning strike, which acts as a direct ignition source for dry vegetation. [Predecessor]: Node 4.

Step 4: Generate Final Answer

- **Operations:** Finish.
- **[Finish]** Synthesize reasoning to confirm (E) Thunderstorm as the final answer. The key factor is the ignition source provided by lightning strikes, a characteristic of thunderstorms.

Final Answer: (E) Thunderstorm

I Experimental Details

Experiments were conducted on a workstation equipped with NVIDIA RTX 4090 GPU (24 GB), an Intel Core i9-13900K CPU (32 logical cores), 125 GB of RAM, and running Ubuntu 20.04.6 LTS with CUDA 12.2, NVIDIA driver 535.183.01, and Python 3.8.10.