ToM: Leveraging Tree-oriented MapReduce for Long-Context Reasoning in Large Language Models

Jiani Guo[1]*, Zuchao Li[2]†, Jie Wu[3]*, Qianren Wang[4],
Yun Li[5], Lefei Zhang[1], Hai Zhao[6], Yujiu Yang[3]
[1]School of Computer Science, Wuhan University, Wuhan, China
[2]School of Artificial Intelligence, Wuhan University, Wuhan, China
[3]Tsinghua University; [4]Shanghai Huawei Technologies, China; [5]Cognitive AI Lab
[6]School of Computer Science, Shanghai Jiao Tong University, Shanghai, China
{guojiani, zcli-charlie, zhanglefei}@whu.edu.cn

Abstract

Large Language Models (LLMs), constrained by limited context windows, often face significant performance degradation when reasoning over long contexts. To address this, Retrieval-Augmented Generation (RAG) retrieves and reasons over chunks but frequently sacrifices logical coherence due to its reliance on similarity-based rankings. Similarly, divide-and-conquer frameworks (DCF) split documents into small chunks for independent reasoning and aggregation. While effective for local reasoning, DCF struggles to capture long-range dependencies and risks inducing conflicts by processing chunks in isolation. To overcome these limitations, we propose ToM, a novel Tree-oriented MapReduce framework for long-context reasoning. ToM leverages the inherent hierarchical structure of long documents (e.g., main headings and subheadings) by constructing a DocTree through hierarchical semantic parsing and performing bottom-up aggregation. Using a Tree MapReduce approach, ToM enables recursive reasoning: in the **Map** step, rationales are generated at child nodes; in the **Reduce** step, these rationales are aggregated across sibling nodes to resolve conflicts or reach consensus at parent nodes. Experimental results on 70B+ LLMs show that ToM significantly outperforms existing divide-and-conquer frameworks and retrieval-augmented generation methods, achieving better logical coherence and long-context reasoning. Our code is available at https://github.com/gjn12-31/ToM.

1 Introduction

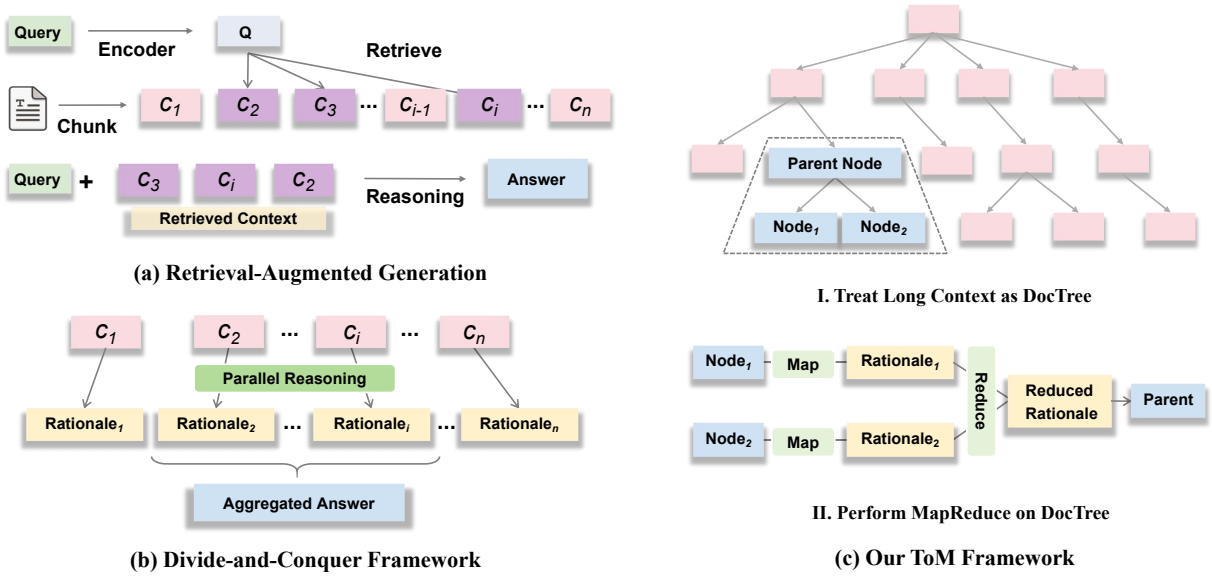
Large Language Models (LLMs) with limited context windows (e.g., 8k, 32k) struggle with reasoning over long contexts. As context length increases, the performance declines due to difficulties in processing information far from the text's beginning or end (Liu et al., 2024a; He et al., 2024). To address these limitations, two mainstream training-free approaches have been introduced: Retrieval-Augmented Generation (RAG) (Li et al., 2024c; Guo et al., 2024) and Divide-and-Conquer Frameworks (DCF) (Zhao et al., 2024; Zhou et al., 2024; Zhang et al., 2024c).

RAG addresses the challenge of long-context reasoning by using a retriever to identify the most relevant chunks from a long document, focusing only on the most pertinent information for reasoning. However, RAG relies on similarity rankings and often neglects the logical coherence between retrieved chunks. In contrast, DCF processes long contexts by splitting them into smaller chunks, reasoning over each independently, and synthesizing local insights into a global understanding. While effective to some extent, DCF treats chunks in isolation and overlooks the relationships between non-adjacent or long-range segments, leading to conflicts and incomplete understanding due to its limited local perspective. Inspired by the idea that human thinking is inherently complex and multi-dimensional, extending beyond what can be captured by flat reasoning or independent reasoning processes (Barsalou, 1999). Reflecting this complexity, long contexts naturally exhibit a hierarchy, such as main headings and subheadings, well suited for reasoning with a tree-based representation.

We introduce ToM, a novel framework for tree-based long-context reasoning. Leveraging a tree-structured MapReduce approach, ToM performs recursive reasoning over documents to enhance long-context understanding. It consists of two key components: 1). **DocTree Construction**: ToM first applies Hierarchical Semantic Parsing to convert each chunk into a structured subtree, then combines these subtrees into a hierarchical DocTree through Bottom-up Aggregation. 2). **Recursive Reasoning via MapReduce**: ToM performs recursive reasoning on the DocTree in a MapReduce fashion, en-

*Equal contribution.
†Corresponding Author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 17793–17812
November 4-9, 2025 ©2025 Association for Computational Linguistics



(a) Retrieval-Augmented Generation

(b) Divide-and-Conquer Framework

(c) Our ToM Framework

Figure 1: Comparison between ToM and existing approaches: LLMs enhanced with RAG (a) rely on sequential reasoning over retrieved chunks, while current Divide-and-Conquer frameworks (b) process chunks in isolation. In contrast, ToM (c) leverages the hierarchical structure of DocTree for tree-based reasoning, mitigating conflicts and preserving contextual coherence through recursive MapReduce reasoning.

abling systematic aggregation of rationales across the hierarchy. In the *Map* step, ToM generates rationales from child nodes; in the *Reduce* step, these rationales are aggregated across sibling nodes to resolve conflicts or reach consensus at the parent node. As shown in Figure 1, ToM enables focal reasoning across the hierarchy, enhancing information utilization beyond the flat reasoning employed by RAG. Compared to DCF, ToM considers sibling and parent-child relationships between chunks and allows long-range chunks to connect to the same parent node through bottom-up reasoning, thereby improving fact aggregation and reducing conflicts.

Experimental results on 70B+ LLMs show that ToM achieves significant performance gains over current divide-and-conquer methods and LLMs enhanced with retrieval-augmented generation. Further analysis shows that ToM balances efficiency and effectiveness by combining DocTree compression with embedding techniques.

Our contributions are shown as follows:

1. **DocTree Representation.** We propose DocTree, a hierarchical representation for organizing long documents. This structure leverages Hierarchical Semantic Parsing and Bottom-up Aggregation to transform chunked text into well-structured trees for bottom-up reasoning.

2. **ToM Framework.** We introduce ToM, a tree-oriented MapReduce framework for long-context reasoning. By leveraging parent-child and sibling relationships, ToM enables structured reasoning, resolves conflicts more effectively, and improves information utilization.

3. **Experimental Validation.** Experiments on 70B+ LLMs demonstrate that ToM outperforms existing divide-and-conquer methods and RAG-enhanced LLMs, achieving significant performance gains in long-context reasoning. Comprehensive ablation studies, complexity analysis, and case studies collectively highlight ToM's effectiveness.

2 Related Work

Long-context Reasoning (Li et al., 2024a,b; Hu et al., 2025) has become essential, as LLMs with limited context windows often suffer performance degradation in long-input scenarios(Liu et al., 2024a). Training-based methods, such as LongLoRA (Chen et al., 2024b), extend context windows via fine-tuning using position encoding refinement, sparse attention, or learnable embeddings. While effective, they demand large datasets support and high computational cost. Training-free methods, like InfLLM (Xiao et al., 2024), retrieve token-relevant content from memory to optimize attention. Divide-and-conquer strategies such as LongAgent (Zhao et al., 2024), LLM×MapReduce (Zhou et al., 2024), and Chain-of-Agents (Zhang et al.,

2024c) split long contexts into chunks, process them locally, and aggregate the results.

MapReduce (Dean and Ghemawat, 2008; Chase, 2023) simplifies large-scale data processing by dividing tasks into two phases: the Map step processes data into key-value pairs, and Reduce aggregates the results. LongChain first introduces MapReduce for multi-document scenarios, applying the Map step to each document and combining outputs into a single result via Reduce. Methods like LLM×MapReduce(Zhou et al., 2024) and XL^{3}M (Wang et al., 2024a) extend this to long documents by splitting them into sub-contexts, selecting relevant segments, and combining them chronologically. The major challenge for MapReduce is that chunks are processed independently, which may break essential long-range dependencies and interconnections between them.

Retrieval Augmented Generation (RAG) (Lewis et al., 2020; Jeong et al., 2024; Wang et al., 2024c,b; Sun et al., 2023; Edge et al., 2024; Guo et al., 2024; Zeng et al., 2025; Asai et al., 2024; Xia et al., 2024; Liu et al., 2024b, 2025; Zhang et al., 2025, 2024a) enhances response quality by combining retrieval and in-context learning. Inspired by the idea of recursive summarization in Raptor (Sarthi et al., 2024), we go further by capturing hierarchical relationships within each chunk and applying bottom-up aggregation, aiming to represent the long document as a structured DocTree. While Raptor uses summary nodes for retrieval-augmented generation, we employ detailed, recursive bottom-up reasoning with a MapReduce approach, fully utilizing the constructed tree.

3 ToM Framework

3.1 Overview.

For effective tree-based reasoning, ToM employs the following two steps: 1) **Representing long contexts as DocTree (Section 3.2)**. ToM begins with *Hierarchical Semantic Parsing* to structure each chunk into a subtree, where lower levels correspond to subheadings and higher levels summarize main headings. It then applies *Bottom-up Aggregation* to merge subtrees into a higher-level hierarchy. 2) **Recursive MapReduce Reasoning (Section 3.3)**. ToM performs recursive reasoning on the DocTree using a MapReduce-style process. This involves iteratively applying the Map and Reduce steps across the hierarchy: the *Map* step transforms information from child nodes into rationales that capture key supporting facts, while the *Reduce* step aggregates sibling rationales to resolve conflicts and reach consensus. By applying recursive MapReduce to the DocTree, ToM enables focused fact aggregation and conflict resolution, thereby facilitating effective long-context reasoning.

3.2 Representing Long Contexts as DocTree

Representing long contexts as a hierarchical DocTree forms the foundation of the ToM framework. This transformation converts a flat, sequential document into a structured, tree-based representation, better reflecting the complexity of human cognition and enabling more organized, hierarchical reasoning. The construction of the DocTree involves two key components: a *Hierarchical Semantic Parser* that parses each chunk into an initial subtree, and a *Bottom-up Tree Aggregation* process that merges these subtrees into a coherent hierarchy.

Hierarchical Semantic Parser. Let D denote a long document, which is segmented into fixed-length token chunks $\{c_1, c_2, c_3, \ldots, c_n\}$, where n is the total number of chunks, and each c_i $(1 \leq i \leq n)$ contains a fixed number of tokens (e.g., 4k). Existing DCF approaches process $\{c_i\}$ independently, treating each c_i as a discrete unit without considering its internal semantic structure.

In contrast, ToM introduces a semantic hierarchy within each chunk c_i by leveraging a Hierarchical Semantic Parser (HSP), a 3B-scale LLM distilled from GPT-4o. The HSP processes each chunk c_i and organizes it into a set of structured semantic subtrees $\{t_{i,1}, t_{i,2}, \ldots, t_{i,m}\}$, where m is the number of subtrees within c_i. Each subtree $t_{i,j}$ $(1 \leq j \leq m)$ encodes hierarchical relationships, such as parent-child links between main headings and subheadings. For simpler chunks, $m = 1$, resulting in a single hierarchical tree, while for more complex chunks, $m > 1$, forming a semantic forest. As shown in Figure 2, each chunk is processed by the HSP to extract its internal hierarchy and transform the hierarchical relationships into a subtree containing structured information such as Title, Keywords, Summary , and Context from the original document. This hierarchical organization allows D to be represented as a collection of semantic subtrees $\{t_{i,j}\}$, facilitating deeper structural understanding. Detailed implementation and quality assessment of HSP are discussed in Appendix B.4, with prompts and illustrative cases provided in Appendix C.2.

Bottom-up Tree Aggregation. After individual

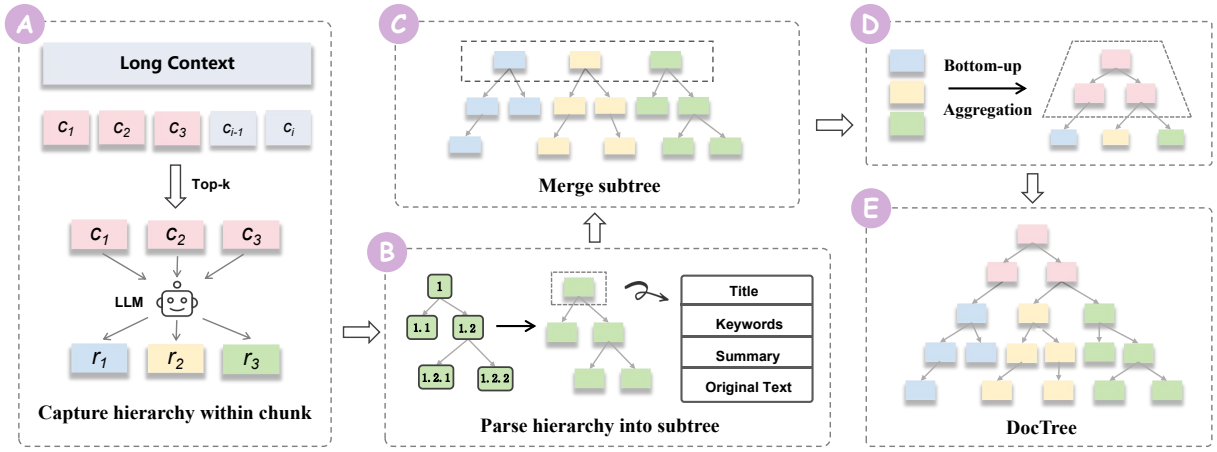


Figure 2: Illustration for DocTree Construction: The process begins with preparation for compression, where a retriever selects Top-k chunks as the foundation for the tree. Each chunk is then processed (A) using the Hierarchical Semantic Parser to capture its internal hierarchy. Next, (B) the hierarchical structure of each chunk is parsed into a subtree, with nodes capturing structured information. These subtrees are merged (C) by placing root nodes at the same level. Following this, (D) a Bottom-up Aggregation integrates information across levels. Finally, the complete DocTree is constructed (E) by combining low-level subtrees with higher-level summaries, ready for reasoning.

subtrees $\{t_{i,j}\}$ are generated from each chunk c_i, the next step is bottom-up aggregation to construct the hierarchical DocTree. This step involves embedding, clustering, and summarizing the root nodes of each subtree, recursively building higher-level summaries to form the complete DocTree.

The initial layer for DocTree, denoted as $\mathcal{L}_0$, is composed of the root nodes of all parsed subtrees:

$$\mathcal{L}_0 = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} \{\text{SubtreeRoot}(t_{i,j})\},$$

where n is the number of chunks, m_i is the number of subtrees within chunk c_i, and $\text{SubtreeRoot}(t_{i,j})$ represents the root node of the j-th subtree $t_{i,j}$. At this stage, $\mathcal{L}_0$ contains only the root nodes of all subtrees, while their child nodes remain encapsulated within their respective subtrees.

To construct the higher layers $\mathcal{L}_k$, we perform the following steps iteratively:

1. **Embedding**: Each root node $r \in \mathcal{L}_{k-1}$ is embedded as a vector e_r using a pre-trained embedding model.

2. **Clustering**: A clustering algorithm groups the embeddings into clusters $\mathcal{I}_{k-1} = \{I^1_{k-1}, I^2_{k-1}, \ldots, I^t_{k-1}\}$, where t is the number of clusters at layer $k-1$.

3. **Summarization**: For each cluster $I^j_{k-1} \in \mathcal{I}_{k-1}$, the evaluated LLM generates a summarized node s^j_k, where:

$$\mathcal{L}_k = \{s^1_k, s^2_k, \ldots, s^t_k\}.$$

Edges are created between the root nodes in I^j_{k-1} at layer $\mathcal{L}_{k-1}$ and their corresponding summary node s^j_k at layer $\mathcal{L}_k$.

This bottom-up aggregation is recursive and continues until the number of clusters stabilizes, resulting in the top layer $\mathcal{L}_K$, which contains a single root node or a small number of high-level summary nodes representing the entire context. The final hierarchical DocTree can be represented as:

$$\mathcal{T} = \{\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_K\}.$$

The key steps for constructing DocTree are outlined in Algorithm 1. The complete DocTree is constructed by organizing lower-level subtrees beneath higher-level summaries. By hierarchically organizing information during aggregation, the lower levels of the DocTree retain detailed information, while the higher levels provide condensed summaries. The DocTree effectively captures both local and global relationships across subtrees, facilitating reasoning from local to global.

DocTree Compression. Short-cutting techniques reduce computational overhead in tree-based reasoning, particularly for large-scale DocTrees with millions of tokens. Drawing inspiration from Retrieval-Augmented Generation methods, our approach integrates DocTree with retrieval techniques like BM25 (Robertson et al., 2009) and embedding models, enabling the construction of a sparser, more concise DocTree. This process filters out irrelevant information, focusing on content closely aligned with the query, thereby improving both effi-

Algorithm 1: DocTree Construction Process

Input: Long context C, Query q
Output: Constructed DocTree $\mathcal{T}$

Step 1: Subtree Generation
Split C into fixed-length token chunks $\{c_1, c_2, \ldots, c_n\}$;
foreach *chunk* c_i **do**
  Use the Hierarchical Semantic Parser to identify semantic structures
  Generate subtrees $\{t_{i,1}, t_{i,2}, \ldots, t_{i,m_i}\}$ for each chunk c_i ;

Step 2: Bottom-Up Tree Aggregation
Initialize the bottom layer:

$$\mathcal{L}_0 = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} \{\text{SubtreeRoot}(t_{i,j})\}$$

for $k = 1, 2, \ldots$ *until the number of clusters stabilizes* **do**
  foreach *node* $r \in \mathcal{L}_{k-1}$ **do**
    Compute embeddings e_r ;
  Cluster embeddings into groups $\mathcal{I}_{k-1} = \{I^1_{k-1}, I^2_{k-1}, \ldots, I^t_{k-1}\}$; Summarize each cluster I^j_{k-1} into a parent node s^j_k ; Update $\mathcal{L}_k = \{s^1_k, s^2_k, \ldots, s^t_k\}$;

Step 3: Final DocTree Construction
Merge all aggregated nodes $\mathcal{L}_K$ into the unified DocTree $\mathcal{T}$, maintaining hierarchical relationships ;

return DocTree $\mathcal{T}$

ciency and relevance. Algorithm 2 in Appendix B.5 outlines the query-aware DocTree compression process, where the most relevant chunks are selected to build the DocTree. Constructing a DocTree from these aligned chunks incurs a small embedding cost while enhancing reasoning performance. Additionally, compressed DocTrees reduce token usage compared to those constructed from full documents, improving computational efficiency.

An example of DocTree construction is illustrated in Appendix C.3. A detailed analysis of the computational complexity involved in constructing the DocTree is also provided in Appendix B.1.

3.3 Recursive MapReduce Reasoning

ToM employs reasoning in a MapReduce style with two key steps: **Map** and **Reduce**. These steps are applied recursively from the bottom up in the DocTree, propagating information and ensuring semantic coherence across the hierarchy.

In the Map step, rationales are generated in parallel from child nodes, while in the Reduce step, these rationales are aggregated across sibling nodes to resolve conflicts or reach consensus at the parent node. Each reasoning step produces a structured output, including key information, rationale, intermediate answer, and confidence estimates for conflict resolution, thereby enabling a more clear and interpretable reasoning process. Map and Reduce are defined as follows:

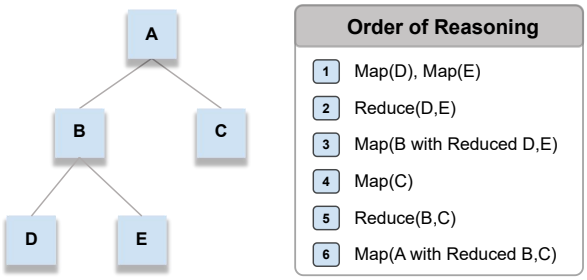


Figure 3: Illustration of the order of recursive reasoning. Nodes at the same hierarchy level, like D and E, can be processed in parallel for accelration.

1. **Map:**
   Input: For a leaf node, the input is info(node). For a non-leaf node, the input is info(node) $\cup$ Reduce(children(node)).
   Operate: Perform reasoning based on the current node's information.
   Output:
   {key_info, rationale, answer, confidence}

2. **Reduce:**
   Input: Results of the Map step from sibling nodes at the same hierarchy level.
   Operate: Resolve conflicts and reach consensus among sibling nodes.
   Output: Aggregated {key_info, rationale, answer, confidence}
   Special Case: Skip Reduce if $|\text{siblings}| = 0$.

As illustrated in Figure 3, recursive reasoning begins at the leaf nodes and propagates upward through Map and Reduce operations. Nodes at the same hierarchical level, such as D and E, are processed in parallel, enhancing computational efficiency. This combined application of Map and Reduce transforms the hierarchical DocTree into a structured reasoning framework, integrating local details into a cohesive global perspective.

For further details about Map and Reduce, see Appendix C.2 for the prompt, Appendix C.3 for example cases, and Appendix B.2 for a comparison of ToM's average reasoning time.

4 Experiment

4.1 Experiment setup

Models. Open-source LLMs with 70B+ parameters, including Qwen2.5-72B-Instruct (Qwen

	HQA (9.1k)		2Wiki (4.9k)		MuSi (11.2k)		Inf.QA (192k)		Inf.MC (184k)
	F1	RL	F1	RL	F1	RL	F1	RL	Acc
LLMs with Full Document									
Llama3-70B-Instruct	21.66	21.29	21.48	21.41	11.33	11.14	8.32	8.09	32.00
Qwen2.5-72B-Instruct	23.99	23.87	21.75	21.67	10.52	10.34	12.47	12.12	36.00
Deepseek-V3	41.41	41.02	37.20	37.05	32.88	32.77	16.89	16.09	47.00
GPT-4o	55.21	54.95	44.28	44.29	40.26	39.95	13.45	12.99	45.00
Deepseek-R1	55.70	55.99	59.59	59.24	40.12	39.46	12.58	13.09	57.00
LLMs Equipped with RAG									
Llama3-70B-Instruct	32.69	32.38	25.39	25.47	12.00	12.77	13.32	14.99	40.00
Qwen2.5-72B-Instruct	35.14	36.07	22.58	23.78	23.78	23.60	19.65	19.94	41.00
Deepseek-V3	42.33	41.17	40.95	40.86	28.26	29.18	17.48	18.69	53.00
GPT-4o	53.73	54.19	54.99	53.79	35.64	35.43	26.03	25.47	65.00
Deepseek-R1	51.84	51.03	56.12	56.97	36.28	37.15	23.14	24.49	62.00
Divide-and-Conquer Frameworks									
+ LongAgent	45.31	45.68	43.72	42.06	26.81	25.60	23.33	23.69	62.00
+ LongAgent	47.19	48.38	48.94	47.98	28.16	27.60	25.68	26.12	65.00
+ LongAgent	57.04	52.71	56.87	55.23	42.99	41.20	29.63	27.12	69.00
+ LongAgent	55.25	52.30	54.53	51.16	45.44	44.03	38.00	33.34	72.00
R1 + LongAgent	59.72	58.66	61.90	62.00	45.41	44.86	29.16	30.20	76.00
+ ToM	50.40	52.71	49.36	50.15	30.89	31.20	27.80	27.12	68.00
+ ToM	52.19	52.08	50.22	51.36	28.29	29.50	30.17	31.58	71.00
+ ToM	60.87	60.43	58.20	58.20	**50.15**	**50.04**	38.60	38.14	77.00
+ ToM	61.07	60.73	59.31	59.55	47.27	47.27	**41.17**	**46.04**	**85.00**
R1 + ToM	**61.91**	**61.55**	**63.33**	**63.25**	44.74	44.86	34.54	34.61	80.00

Table 1: Long document reasoning performance (%) on Longbench and InfiniteBench, with average token lengths from 4.9k to 192k. 100 samples are randomly selected for Inf.QA and Inf.MC, while the full sets are used for the other tasks. Bold denotes the best performance, and underlined results indicate runner-ups.

et al., 2025), Llama3-70B-Instruct, DeepSeek-V3 (DeepSeek-AI et al., 2024), and GPT-4o-2024-05-01. Additionally, reasoning model Deepseek-R1 is also evaluated for comparison.

Benchmarks. We evaluate the performance on HotpotQA, 2WikiMQA, and Musique from LongBench (Bai et al., 2024). For more challenging tasks, we test on the Question-Answering (Inf.QA) and Multi-Choice (Inf.MC) tasks from InfiniteBench (Zhang et al., 2024b), where the average input length is 190k tokens. This poses a significant challenge for LLMs to process, understand, and reason over ultra-long contexts. Additional comparisons on Long In-Context Learning and Long Dialogue History Understanding tasks are provided in Appendix A.1.

Baselines. We compare ToM with the following baselines: **(i) Reasoning on Full Document**: Query and full documents are concatenated for reasoning. Contexts exceeding the window size are truncated. **(ii) LLM enhanced with RAG**: For the RAG baseline, we perform chunking using a fixed number of tokens: 1k for Longbench and 4k for InfiniteBench. We adopt BGE-M3 with an 8k token window as the retriever and select the Top 5 chunks for augmented generation. We report the results of 70B+ LLMs, including Qwen2.5-72B-Instruct, Llama3-70B-Instruct, Deepseek-V3, GPT-4o, and Deepseek-R1. Additionally, the Appendix C.1 provides an in-depth comparative analysis between RAG and ToM. **(iii) Divide-and-Conquer Frameworks**: Recent divide-and-conquer framework LongAgent (Zhao et al., 2024) is evaluated. LongAgent utilizes multiple member agents, each responsible for processing assigned context chunks. Following the chunking process, a leader judge agent synthesizes the final answer through a multi-round discussion. For Inf.QA and Inf.MC, the chunk length is set to 4k tokens, while for other datasets in Longbench, it is configured to 1k tokens. Details about ToM, including the embedding technique, chunking size, and clustering algorithm, are provided in Appendix B.

Metrics. We use F1-score and RougeL score to evaluate question-answering performance and Accuracy for multiple-choice tasks.

4.2 Main Results

The overall results in Table 1 highlight ToM's performance compared to existing methods. We analyze the results and present the following findings:

Challenge Exists in Long Document Reasoning. LLMs still face significant challenges in long-context reasoning, ranging from 10k tokens to ultra-

long scenarios. Qwen2.5-72B-Instruct performs poorly on 10k-token documents like HOTPOTQA and MUSIQUE, with F1-scores between 10.52% and 23.99%. DeepSeek-V3 outperforms Qwen by 17.4% on HOTPOTQA and 15.5% on 2WIKIMQA with full-document context.

This challenge becomes more apparent when handling ultra-long contexts exceeding 100k tokens. Tasks like INF.QA and INF.MC, averaging 190k tokens, present significant challenges for all LLMs. Llama3-70B-Instruct and Qwen2.5-72B-Instruct score 8.32% and 12.47%, respectively, while GPT-4o achieves 13.45%, and DeepSeek-V3 leads with 16.89% on INF.QA. Even DeepSeek-R1, with state-of-the-art performance still struggles with ultra-long contexts, achieving only 12.58% on INF.QA. These results highlight the limitations of current LLMs and the need for improved methods to handle such extensive contexts.

RAG Benefits in Long Document Reasoning. On short document reasoning tasks, with the exception of GPT-4o, most LLMs show significant gains from RAG techniques. For example, on HOTPOTQA, Llama3-70B-Instruct's F1 score increases from 21.66% to 32.69%, and Qwen2.5-72B-Instruct's from 23.99% to 35.14%. Interestingly, DeepSeek-R1 sees a slight drop, from 55.70% to 51.84%, suggesting that its strong in-context performance may be disrupted by the fragmentation introduced by chunking.

For ultra-long documents, RAG provides a straightforward solution by helping LLMs focus on the most relevant chunks. With RAG, Llama3-70B-Instruct and Qwen2.5-72B-Instruct achieve notable improvements on INF.QA, reaching F1 scores of 13.32% and 19.65%, respectively, while GPT-4o sees a boost to 26.03%. Similarly, on INF.MC, all LLMs benefit from RAG, with performance gains ranging from 5.0% to 20.0%.

However, while RAG improves performance in long-context reasoning, it still faces two key limitations: 1) Although RAG improves the recall of relevant chunks, it does not ensure that the retrieved chunks are truly useful for reasoning, often introducing irrelevant information that adds noise to the process. 2) RAG-enhanced LLMs reason over flat-organized chunks but often sacrifice logical coherence due to their reliance on similarity-based rankings, leaving interconnections between chunks underutilized. These limitations constrain the upper bound of RAG-based methods, highlighting the need for a more fine-grained reasoning approach that fully leverages the chunks and improves information aggregation.

DCF Outperform One-Step Reasoning in Long Contexts Scenarios. Divide-and-conquer methods consistently outperform one-step reasoning approaches. LongAgent significantly improves performance across all evaluated LLMs. On HOTPOTQA, LongAgent boosts DeepSeek-R1 to 59.72% F1, surpassing its RAG baseline by 7.88%. Llama3-70B-Instruct and Qwen2.5-72B-Instruct see even larger gains of 12.62% and 12.05%, respectively. In ultra-long document reasoning, LongAgent also proves effective. With DeepSeek-R1, it achieves 29.16% F1 on INF.QA and 76.00% accuracy on INF.MC. GPT-4o performs best on INF.QA at 38.00%, while maintaining 72.00% on INF.MC. These results underscore the advantage of divide-and-conquer strategies in complex, long-context reasoning.

LongAgent enhances long-context reasoning performance by breaking long documents into smaller components for parallel reasoning and employing a Leader LLM to resolve conflicts and refine results across multiple turns. Though effective for local reasoning, LongAgent processes chunks in isolation, limiting its ability to capture long-range dependencies and increasing the risk of conflicts, thus capping its potential.

ToM Achieves Significant Performance Gains. With GPT-4o, ToM achieves state-of-the-art performance, reaching 41.17% F1 on INF.QA and 85.0% accuracy on INF.MC. When paired with Qwen2.5-72B-Instruct and DeepSeek-V3, ToM also demonstrates strong performance, achieving 30.17% and 38.60% F1 on INF.QA, and 71.00% and 77.00% accuracy on INF.MC, respectively, consistently outperforming their LongAgent counterparts. Compared to RAG-based methods, ToM with GPT-4o improves performance by 15.14% on INF.QA and 20.0% on INF.MC. Relative to LongAgent with GPT-4o, it achieves gains of 11.97% in F1 and 13.0% in accuracy, highlighting ToM's advantage in ultra-long document reasoning.

ToM's effectiveness stems from its recursive tree-based MapReduce framework. In the Map phase, relevant information is extracted from chunks; in the Reduce phase, results are merged, conflicts resolved, and refined using confidence scores. This structured approach reduces noise and enhances integration, making it well-suited for long context reasoning tasks.

Compared to RAG, ToM performs fine-grained

reasoning on informative content identified by HSP within each chunk, fully leveraging relevant content while filtering out noise based on confidence estimates. Compared to LongAgent, its tree-based MapReduce framework enables more effective conflict resolution and enhances fact aggregation from local to global through the DocTree structure, thereby achieving promising performance gains.

Equip ToM with Reasoning Models. We evaluate both LongAgent and ToM using DeepSeek-R1. On shorter tasks like HOTPOTQA and 2WIKIMQA, DeepSeek-R1 combined with either framework outperforms other models, with ToM achieving 61.91% and 63.33% F1, respectively. As document length increases, ToM with GPT-4o surpasses DeepSeek-R1 on ultra-long tasks such as INF.QA and INF.MC. While the long chains of thought produced by R1 benefit understanding in shorter contexts, they also introduce overthinking and hallucinations during reasoning, allowing incorrect ideas to propagate upward and ultimately impairing overall reasoning.

5 Ablation Study

In this section, we discuss the contributions of key components, analyze efficiency, and examine the impact of compression, with additional ablations on chunk size provided in Appendix B.3.

Contribution of Key Components. We conduct ablations on bottom-up aggregation and the confidence measure, with results shown in Table 2. Removing the in-context confidence measure leads to substantial performance degradation across all datasets (–6.9% on INF.QA, –7.0% on INF.MC). In the tree-based MapReduce reasoning pattern, the reduce phase is essential for merging viewpoints and resolving conflicts, which is key to strong performance. Divide-and-conquer reasoning often encounters local conflicts, and confidence scores help guide resolution for more consistent results.

Removing bottom-up aggregation results in consistent performance drops (–2.0% on INF.QA, –6.0% on INF.MC). It contributes by providing global context through hierarchical summaries, which are difficult to infer solely from detailed subtree information. By integrating low-level details with high-level summaries, ToM enables more effective reasoning over the DocTree.

Aggre.	Conf.	HQA	2Wiki	MuS	Inf.QA	Inf.MC
✔	✔	**61.1**	**59.3**	**47.3**	**38.6**	**85.0**
✗	✔	55.8	53.0	42.5	36.6	79.0
✔	✗	56.5	51.4	39.3	31.7	78.0

Table 2: Effect of the in-context confidence measure and bottom-up aggregation with DeepSeek-V3.

Efficiency Analysis. ToM introduces the Hierarchical Semantic Parser (HSP) to capture the internal hierarchy within each chunk. While adding some computational cost, the impact is minimal. First, we use a lightweight 3B-scale LLM for parsing, which is more affordable than relying heavily on expensive GPT API calls. Second, we leverage technologies like vLLM to enable parallel processing, further optimizing efficiency. The average DocTree construction time across different stages and input lengths is provided in Appendix B.1. ToM requires fewer LLM calls than LongAgent, making 4.2k calls on 100 INF.QA samples compared to LongAgent's 6.3k.

Compression Impact. We compress Doctree by selecting relevant chunks for construction, with the reasoning performance versus the number of selected chunks on INF.MC shown in Figure 4. There is a trade-off between reasoning performance and computational costs: while increasing the number of chunks raises overhead, it also improves performance. Notably, as the number of chunks increases from Top-3 to Top-7, reasoning performance grows steadily for GPT-4o, highlighting the benefits of adding more context despite the additional costs.

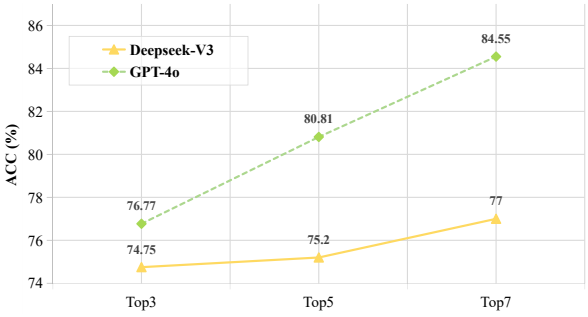


Figure 4: Effect of the number of selected chunks.

6 Conclusion

In this paper, we introduced ToM, a tree-oriented MapReduce framework for effective long document reasoning with large language models. By leveraging tree-based representations and recursive MapReduce reasoning, ToM enhances fact utilization and conflict resolution across extended documents. Extensive experiments show that ToM achieves significant performance gains over existing divide-and-conquer and RAG-enhanced frame-

works, highlighting the potential of tree-based frameworks to overcome the limitations of current LLMs in long-context reasoning and paving the way for more effective, scalable solutions.

Limitations

ToM significantly enhances long-context reasoning through its novel tree-based MapReduce framework. However, the construction of the DocTree, which involves hierarchical semantic parsing and bottom-up aggregation, introduces computational overhead, particularly when processing ultra-long documents. Although the query-aware DocTree compression improves efficiency, it may inadvertently omit intermediate information that is essential for complex multi-hop reasoning. Additionally, the granularity of text chunking and the potential for error propagation from earlier analytical stages can affect the overall performance of the framework. Future research includes extending ToM's capabilities to handle multimodal documents for more comprehensive long-context understanding.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 62306216), and the Technology Innovation Program of Hubei Province (No. 2024BAB043). Hai Zhao was funded by The Major Program of Chinese National Foundation of Social Sciences under Grant 'The Challenge and Governance of Smart Media on News Authenticity' [No. 23&ZD213]. Yujiu Yang was supported by the National Key Research and Development Program of China (No. 2024YFB2808903) and the research grant No. CT20240905126002 of the Doubao Large Model Fund.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Lawrence W. Barsalou. 1999. Perceptual symbol systems. *Behavioral and Brain Sciences*, page 577–660.

Harrison et al. Chase. 2023. Langchain: Building applications with llms through composability. GitHub repository.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. BGE m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *CoRR*, abs/2402.03216.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024b. LongloRA: Efficient fine-tuning of long-context large language models. In *The Twelfth International Conference on Learning Representations*.

Jeffrey Dean and Sanjay Ghemawat. 2008. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, and et al. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. *CoRR*, abs/2410.05779.

Junqing He, Kunhao Pan, Xiaoqun Dong, Zhuoyang Song, LiuYiBo LiuYiBo, Qianguosun Qianguosun, Yuxin Liang, Hao Wang, Enming Zhang, and Jiaxing Zhang. 2024. Never lost in the middle: Mastering long-context question answering with position-agnostic decompositional training. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 13628–13642. Association for Computational Linguistics.

Zhiyuan Hu, Yuliang Liu, Jinman Zhao, Suyuchen Wang, WangYan WangYan, Wei Shen, Qing Gu, Anh Tuan Luu, See-Kiong Ng, Zhiwei Jiang, and Bryan Hooi. 2025. LongRecipe: Recipe for efficient long context generalization in large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11857–11870, Vienna, Austria. Association for Computational Linguistics.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of*

the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). Association for Computational Linguistics.

Omri Koshorek, Adir Cohen, Noam Mor, Michael Rotman, and Jonathan Berant. 2018. Text segmentation as a supervised learning task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 469–473, New Orleans, Louisiana. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Patrick S. H. Lewis, Ethan Perez, and Aleksandra Piktus et al. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. 2024a. Loogle: Can long-context language models understand long contexts? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 16304–16333. Association for Computational Linguistics.

Yanyang Li, Shuo Liang, Michael R. Lyu, and Liwei Wang. 2024b. Making long-context language models better multi-hop reasoners. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2462–2475. Association for Computational Linguistics.

Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. 2024c. Retrieval augmented generation or long-context llms? A comprehensive study and hybrid approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: EMNLP 2024 - Industry Track, Miami, Florida, USA, November 12-16, 2024*, pages 881–893. Association for Computational Linguistics.

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

Wanlong Liu, Junying Chen, Ke Ji, Li Zhou, Wenyu Chen, and Benyou Wang. 2024b. Rag-instruct: Boosting llms with diverse retrieval-augmented instructions. *arXiv preprint arXiv:2501.00353*.

Wanlong Liu, Junxiao Xu, Fei Yu, Yukang Lin, Ke Ji, Wenyu Chen, Yan Xu, Yasheng Wang, Lifeng Shang, and Benyou Wang. 2025. Qfft, question-free fine-tuning for adaptive reasoning. *arXiv preprint arXiv:2506.12860*.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, and et al. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. 2024. RAPTOR: recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. 2023. Recitation-augmented language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Vincent A. Traag, Ludo Waltman, and Nees Jan van Eck. 2018. From louvain to leiden: guaranteeing well-connected communities. *CoRR*, abs/1810.08473.

Shengnan Wang, Youhui Bai, Lin Zhang, Pingyi Zhou, Shixiong Zhao, Gong Zhang, Sen Wang, Renhai Chen, Hua Xu, and Hongwei Sun. 2024a. XL3M: A training-free framework for LLM length extension based on segment-wise inference. *CoRR*, abs/2405.17755.

Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, Ruicheng Yin, Changze Lv, Xiaoqing Zheng, and Xuanjing Huang. 2024b. Searching for best practices in retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17716–17736, Miami, Florida, USA. Association for Computational Linguistics.

Zheng Wang, Shu Xian Teo, Jieer Ouyang, Yongjun Xu, and Wei Shi. 2024c. M-RAG: reinforcing large language model performance through retrieval-augmented generation with multiple partitions. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 1966–1978. Association for Computational Linguistics.

Peng Xia, Kangyu Zhu, Haoran Li, Hongtu Zhu, Yun Li, Gang Li, Linjun Zhang, and Huaxiu Yao. 2024. RULE: Reliable multimodal RAG for factuality in

medical vision language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1081–1093, Miami, Florida, USA. Association for Computational Linguistics.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shuang Zeng, Xinyuan Chang, Mengwei Xie, Xinran Liu, Yifan Bai, Zheng Pan, Mu Xu, and Xing Wei. 2025. Futuresightdrive: Thinking visually with spatio-temporal cot for autonomous driving. *arXiv preprint arXiv:2505.17685*.

Le Zhang, Bo Wang, Xipeng Qiu, Siva Reddy, and Aishwarya Agrawal. 2025. Rearank: Reasoning re-ranking agent via reinforcement learning. *arXiv preprint arXiv:2505.20046*.

Le Zhang, Yihong Wu, Qian Yang, and Jian-Yun Nie. 2024a. Exploring the best practices of query expansion with large language models. *arXiv preprint arXiv:2401.06311*.

Xinrong Zhang, Yingfa Chen, Shengding Hu, and et al. 2024b. ∞Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277, Bangkok, Thailand. Association for Computational Linguistics.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö. Arik. 2024c. Chain of agents: Large language models collaborating on long-context tasks. *CoRR*, abs/2406.02818.

Jun Zhao, Can Zu, Xu Hao, Yi Lu, Wei He, Yiwen Ding, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. LONGAGENT: Achieving question answering for 128k-token-long documents through multi-agent collaboration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16310–16324, Miami, Florida, USA. Association for Computational Linguistics.

Zihan Zhou, Chong Li, Xinyi Chen, Shuo Wang, Yu Chao, Zhili Li, Haoyu Wang, Rongqiao An, Qi Shi, Zhixing Tan, Xu Han, Xiaodong Shi, Zhiyuan Liu, and Maosong Sun. 2024. Llm$\times$mapreduce: Simplified long-sequence processing using large language models. *Preprint*, arXiv:2410.09342.

Appendix

This appendix provides supplementary material to further elaborate on the ToM framework. Section A presents additional performance evaluations of ToM on long-context QA tasks. Section B offers implementation details and an in-depth analysis of ToM's key components, including its computational complexity and chunking strategy, the Hierarchical Semantic Parser's implementation and quality, and the DocTree construction process covering aggregation and compression techniques. Finally, Section C provides a deeper dive into ToM's methodology through a detailed comparison with RAG, an exposition of its core operational prompts, and illustrative case studies.

A Additional Performance Evaluations

This section extends the performance evaluations presented in the main paper, showcasing ToM's effectiveness on further challenging long-context question answering tasks.

A.1 Performance on User Guide QA and Dialogue History QA

In addition to HQA, 2Wiki, MuSi, Inf.QA, and Inf.MC, we also evaluated ToM on additional QA tasks: the Long In-context Learning task—User Guide QA and the Long-dialogue History Understanding task—Dialogue History QA. The experimental results in Table 3 demonstrate that on User Guide QA, our ToM approach achieved an accuracy of 65.0%, outperforming the baseline GPT-4o model by 12.5 percentage points (from 52.5%) and GPT-4o+LongAgent by 5.0 percentage points (from 60.0%). Similarly, on Dialogue History QA, ToM reached 68.4% accuracy, surpassing the baseline GPT-4o by 15.8 percentage points (from 52.6%) and GPT-4o+LongAgent by 5.2 percentage points (from 63.2%).

These statistically significant and consistent performance improvements across diverse long-context tasks provide empirical evidence for ToM's robust generalizability. The observed enhancements can be attributed to the fundamental architectural advantages of our framework: specifically, the tree-oriented MapReduce methodology enables hierarchical document structuring through semantic parsing and bottom-up aggregation, preserving critical semantic relationships that would otherwise be lost in traditional approaches. Unlike conventional RAG methods that rely on sequential reasoning or divide-and-conquer frameworks that process chunks in isolation, ToM's recursive reasoning mechanism explicitly maintains parent-child relationships between document segments and facilitates long-range information flow across the entire DocTree. This structural coherence is particularly beneficial for dialogue-based tasks, as evidenced by the substantial 15.8 percentage point improvement on Dialogue History QA, where maintaining contextual continuity across multiple conversational turns is essential for accurate comprehension and reasoning. The experimental findings thus substantiate our hypothesis that tree-based reasoning offers a more effective paradigm for complex long-context understanding tasks that require sophisticated multi-hop inference capabilities.

	User Guide QA	Dialogue History QA
GPT-4o	52.5	52.6
+ LongAgent	60.0	63.2
+ ToM	**65.0**	**68.4**

Table 3: Accuracy on User Guide and Dialogue History.

B Implementation Details and Component Analysis of ToM

This section delves into specific implementation details and characteristics of the ToM framework's key components. We analyze its computational complexity alongside the chunking strategy, detail the training, implementation, and quality of the Hierarchical Semantic Parser, and discuss the DocTree construction process, focusing on aggregation and compression techniques along with their implications.

B.1 Computational Complexity

When comparing our approach, ToM, to traditional methods like RAG or divide-and-conquer frameworks, one noticeable difference is the additional tree construction step that ToM incorporates. The construction of the DocTree in ToM consists of three steps: 1) **embedding chunks before constructing**; 2) **Inferencing during hierarchical semantic parsing**; and 3) **bottom-up summarizing**.

Table 4 details the average time for DocTree construction over varying document lengths.

From the experimental results shown in Table 4, it is clear that the construction time increases with document length. The total time required for constructing the DocTree increases steadily, with pro-

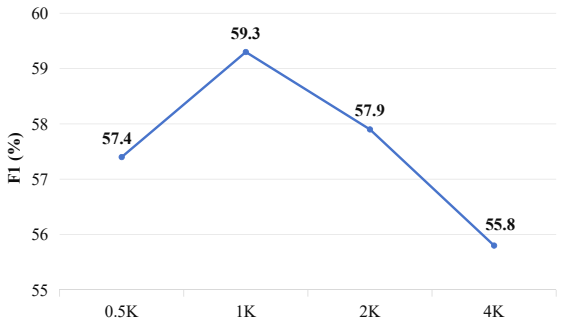


Figure 5: Effect of the chunk size.

cessing times of 18.5 seconds for 10k tokens and 75.4 seconds for 250k tokens.

Chunking Size The chunk size for input documents was adapted based on the benchmark: input documents were segmented into chunks of 1,000 tokens for Longbench tasks and 8,000 tokens for InfiniteBench tasks.

B.2 Reasoning Time on Ultra Long Document

Table 5 presents the average reasoning time compared to other methods.

Turning to the reasoning performance presented in Table 5, ToM achieves reasoning times of 145.0 seconds for 100k tokens. While RAG is faster (5.4s), LongAgent is slower (236.8s). Although the construction and reasoning time for ToM can be higher, our experimental evaluations (presented in the main paper and Section A) show that the added time is justified by the significant improvements in performance, offering a better balance between efficiency and effectiveness.

B.3 Impact of Chunk Size.

We evaluate different chunk sizes (0.5k-4k tokens) on the 2WikiMQA dataset (average length 4.9k tokens) using ToM with GPT-4o. Results in Figure 5 show that 1k tokens achieves optimal performance (59.3% F1), while both smaller (0.5k, 57.4% F1) and larger chunks (4k, 55.8% F1) lead to degraded performance. Smaller chunks likely fragment semantic coherence, making it difficult to capture complete logical units. Conversely, larger chunks may reduce the effectiveness of the hierarchical structure by creating fewer but more complex nodes in the DocTree, potentially obscuring the parent-child relationships that facilitate effective reasoning. The optimal 1k chunk size creates a more balanced hierarchical structure with clearer semantic boundaries between nodes, allowing for more effective information organization for this particular dataset.

Time (Component)	10k	80k	120k	250k
Embedding	0.6	1.9	2.7	5.3
Inference (HSP)	9.9	29.4	34.3	37.2
Summarization	8.0	27.0	28.1	32.9
Total Construction	18.5	58.3	65.1	75.4

Table 4: Average DocTree constructing time (s) within different stages on varying lengths.

	RAG	LongAgent	Our ToM
Time (s)	5.4	236.8	145.0

Table 5: Time costs on Inf.QA over 100k tokens.

B.4 Hierarchical Semantic Parser (HSP): Implementation and Quality Assessment

HSP Training and Implementation The Hierarchical Semantic Parser (HSP) was trained using the Wiki727 dataset (Koshorek et al., 2018) for initial data generation, with semantic chunking performed by GPT-4o (OpenAI, 2024). To enhance operational efficiency, we distilled a 3B-scale model, Qwen2.5-3B-Instruct. This distillation involved fine-tuning the model on 18,000 query-response pairs generated by GPT-4o, employing a full-parameter supervised approach with an 8k context window. For parallel processing tasks related to the HSP and other components, vLLM (Kwon et al., 2023) was incorporated. Following the HSP parsing, subtrees were constructed from the parsed chunks using regular expressions to identify parent-child relationships; this method proved robust, yielding no matching failures across thousands of parsing instances.

HSP Quality Current LLMs can perform stable parsing without tuning and can achieve better performance after distillation. Figure 6 shows the loss curves for our HSP, indicating effective learning and stabilization of parsing quality.

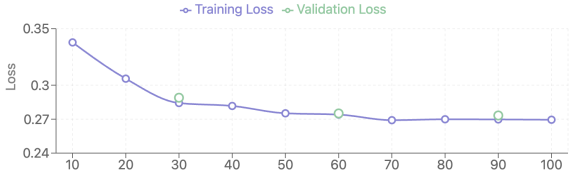


Figure 6: Loss curves of the HSP using the Qwen2.5-3B-Instruct.

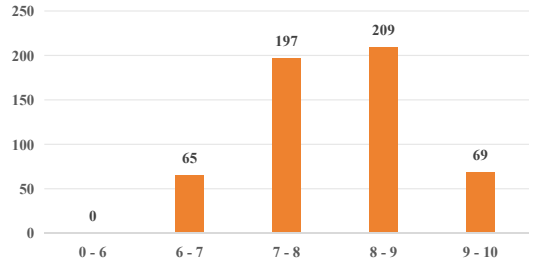


Figure 7: Score distribution of 500 HSP samples evaluated by Claude-Sonnet-3.7, showing a mean score of 8.02.

To further evaluate parsing reliability, we conducted a quality assessment adopting the LLM-as-judge approach using Claude-Sonnet-3.7 as an independent evaluator which is shown in Figure 7. Our analysis of 500 parsed samples, scored on a scale of 0-10, reveals strong performance with no samples below 6.0, and the majority (406 samples, 81.2%) scoring between 7.0-9.0. The highest concentration appears in the 8.0-9.0 range (209 samples), with an overall mean score of 8.02. This distribution confirms that our HSP consistently produces high-quality hierarchical structures, effectively capturing semantic relationships essential for the subsequent MapReduce reasoning process.

B.5 DocTree Construction Details: Aggregation and Compression

Recursive Summarization for Aggregation In the recursive summarization process, which forms part of the bottom-up tree aggregation in DocTree construction, node clustering was performed using the Leiden (Traag et al., 2018) algorithm. The main evaluated LLM (as specified in the main paper's experimental setup) then performed the subsequent summarization steps for the clustered nodes.

DocTree Compression Algorithm The DocTree compression strategy is crucial for managing computational resources. Algorithm 2 outlines our query-aware compression technique. For this mechanism, the parameter k (number of selected chunks) was set to 7. Similarity scores, crucial for query-aware chunk selection during compression, were computed between query and node embeddings using the BGE-M3 model (Chen et al., 2024a).

Analysis of Information Loss during Compression Our DocTree compression strategy, which employs query-aware top-k selection using embeddings, does introduce a deliberate tradeoff between computational efficiency and information completeness. By selectively focusing on the most relevant chunks for DocTree construction, we reduce processing time while maintaining high performance on most reasoning tasks. However, this process may potentially exclude intermediate information that could be valuable for capturing certain long-range dependencies, particularly in cases requiring complex multi-hop reasoning across seemingly unrelated document sections.

To address this concern, we emphasize the flexible nature of our framework, which allows practitioners to adjust the compression parameters based on their specific requirements. For applications prioritizing maximum reasoning accuracy where computational resources are less constrained, compression can be made optional, permitting the construction of complete DocTrees that preserve all document information.

Our experiments with varying compression levels indicate that increasing the number of selected chunks generally improves performance, with diminishing returns beyond a certain threshold. This suggests that while some information loss occurs during compression, our embedding-based selection effectively identifies most critical chunks, and the hierarchical structure of DocTree helps mitigate the impact of missing intermediate information by establishing connections between distant but semantically related chunks during the bottom-up aggregation process.

Algorithm 2: Query-Aware DocTree Compression

Input: Long context C, Query q, Retriever $\mathcal{R}$, Selection scale k

Output: Compressed set of chunks $\{c'_1, c'_2, \ldots, c'_k\}$ to form DocTree $\mathcal{T}$

Step 1: Embedding Generation

Generate query embedding e_q for q using $\mathcal{R}$

Split C into fixed-length token chunks $\{c_1, c_2, \ldots, c_n\}$

foreach *chunk* c_i **do**

  Generate embedding e_{c_i} for c_i using $\mathcal{R}$

  Compute cosine similarity $\text{Sim}(e_{c_i}, e_q)$ for e_{c_i} and e_q

Step 2: Chunk Selection

Select the top-k chunks based on similarity scores:

$$\{c'_1, c'_2, \ldots, c'_k\} = \text{TopKSim}(q, \{c_1, c_2, \ldots, c_n\})$$

Step 3: Build Compressed DocTree $\mathcal{T}$ using $\{c'_1, c'_2, \ldots, c'_k\}$

Return: Compressed DocTree $\mathcal{T}$ (built from selected chunks)

C ToM Methodology: In-depth and Comparative View

This section offers a more detailed exposition of ToM's methodology. We provide an in-depth comparison with traditional Retrieval-Augmented Generation (RAG) to highlight ToM's unique advantages, present the core prompts that guide its reasoning steps, and illustrate its operation through case studies.

C.1 Comparison with Retrieval-Augmented Generation (RAG)

ToM demonstrates superior performance across all benchmarks as evidenced by our experimental results. An intriguing aspect of this comparison is that ToM initially employs retrieval mechanisms similar to those in RAG for chunk selection, yet achieves significantly better outcomes in downstream reasoning tasks.

Retrieval-Augmented Generation (RAG) approaches have made significant contributions to question answering tasks by effectively enhancing recall—successfully retrieving chunks containing potentially relevant information. However, QA tasks inherently demand high precision in utilizing retrieved information to generate accurate answers. While RAG excels at gathering diverse relevant content, its flat concatenation of retrieved chunks and single-pass reasoning process presents challenges for complex queries requiring nuanced information integration. It lacks the ability to capture the logical structure between different information segments, struggles to establish connections between distantly related facts, and cannot effectively resolve contradictions that may appear across different chunks. Consequently, RAG's performance degrades significantly when handling queries that require integrating information from multiple document segments or performing multi-hop reasoning.

Our Tree-oriented MapReduce (ToM) framework builds upon RAG's strong retrieval foundation while addressing this precision challenge through hierarchical processing. By organizing the same retrieved chunks into a structured DocTree, ToM enables a two-phase reasoning process that systematically improves precision. In the Map phase, ToM conducts fine-grained reasoning on individual sibling nodes in parallel, generating detailed rationales that capture essential supporting facts from each document segment. The subsequent Reduce phase then systematically aggregates these rationales across sibling nodes, resolving conflicts and reaching consensus at parent nodes. To more intuitively illustrate the MapReduce reasoning process, a case is provided in Figure 9.

This recursive process enables ToM to progressively refine understanding from leaf nodes to root nodes, preserving context coherence throughout the document hierarchy. By facilitating information flow between parent-child relationships and across sibling nodes, ToM transforms high-recall retrieved content into high-precision answers, particularly excelling at complex queries requiring integration of distributed information and resolution of apparent contradictions.

C.2 Core Operational Prompts

The Map, Reduce, and Hierarchical Semantic Parsing (HSP) steps in ToM are guided by specific prompts provided to the LLM. These prompts are crucial for structuring the LLM's processing and output at each stage. Figure 10 shows the prompt used for the Map step, Figure 11 for the Reduce step, and Figure 12 for the Hierarchical Semantic Parsing of chunks.

C.3 Illustrative Case Studies

To provide a concrete understanding of ToM's internal workings, we present two case studies. Figure 8 illustrates an example of DocTree construction. Figure 9 demonstrates the MapReduce reasoning process on a constructed DocTree. These examples visually walk through the key stages of our framework.

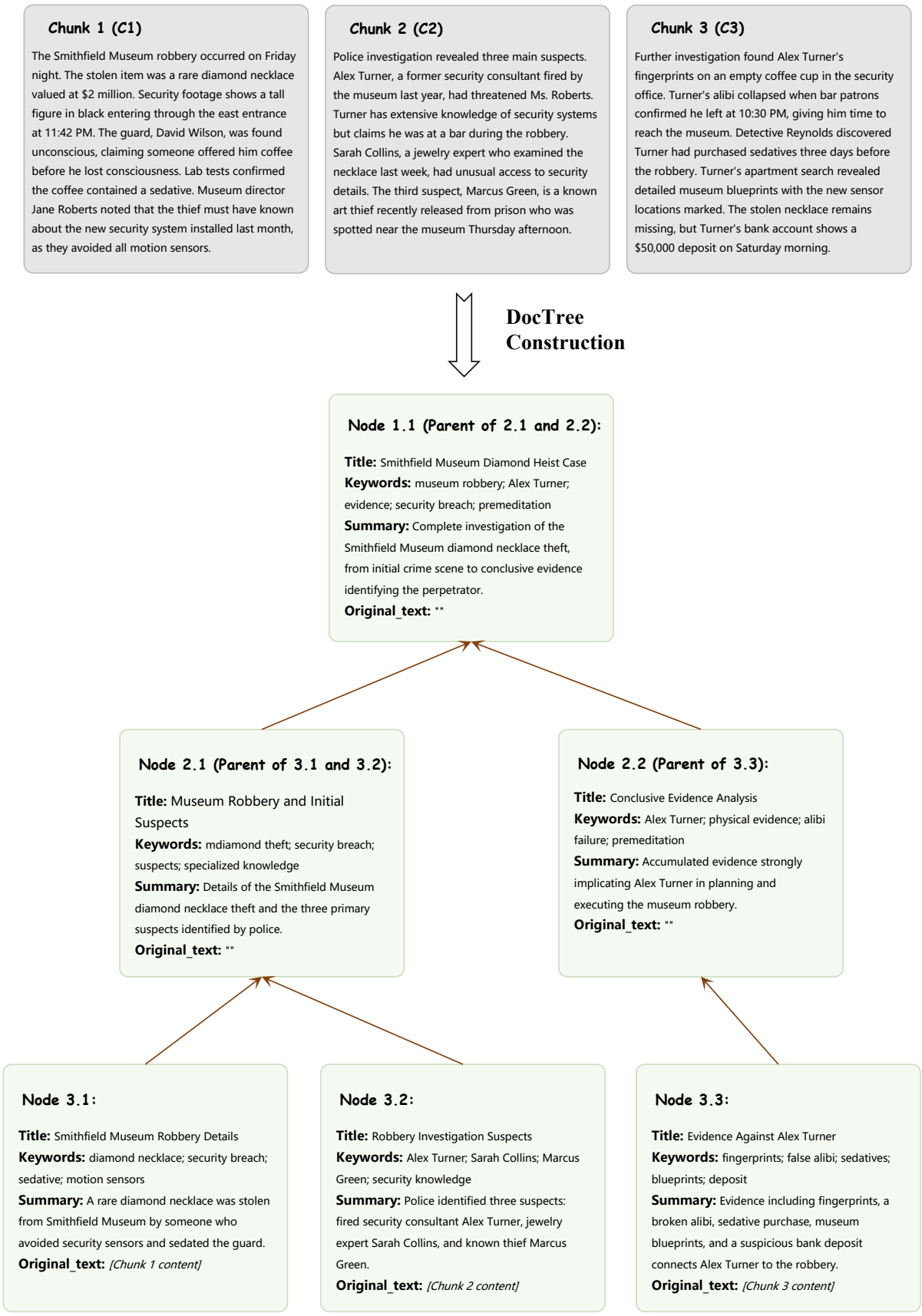


Figure 8: Case for DocTree Construction.

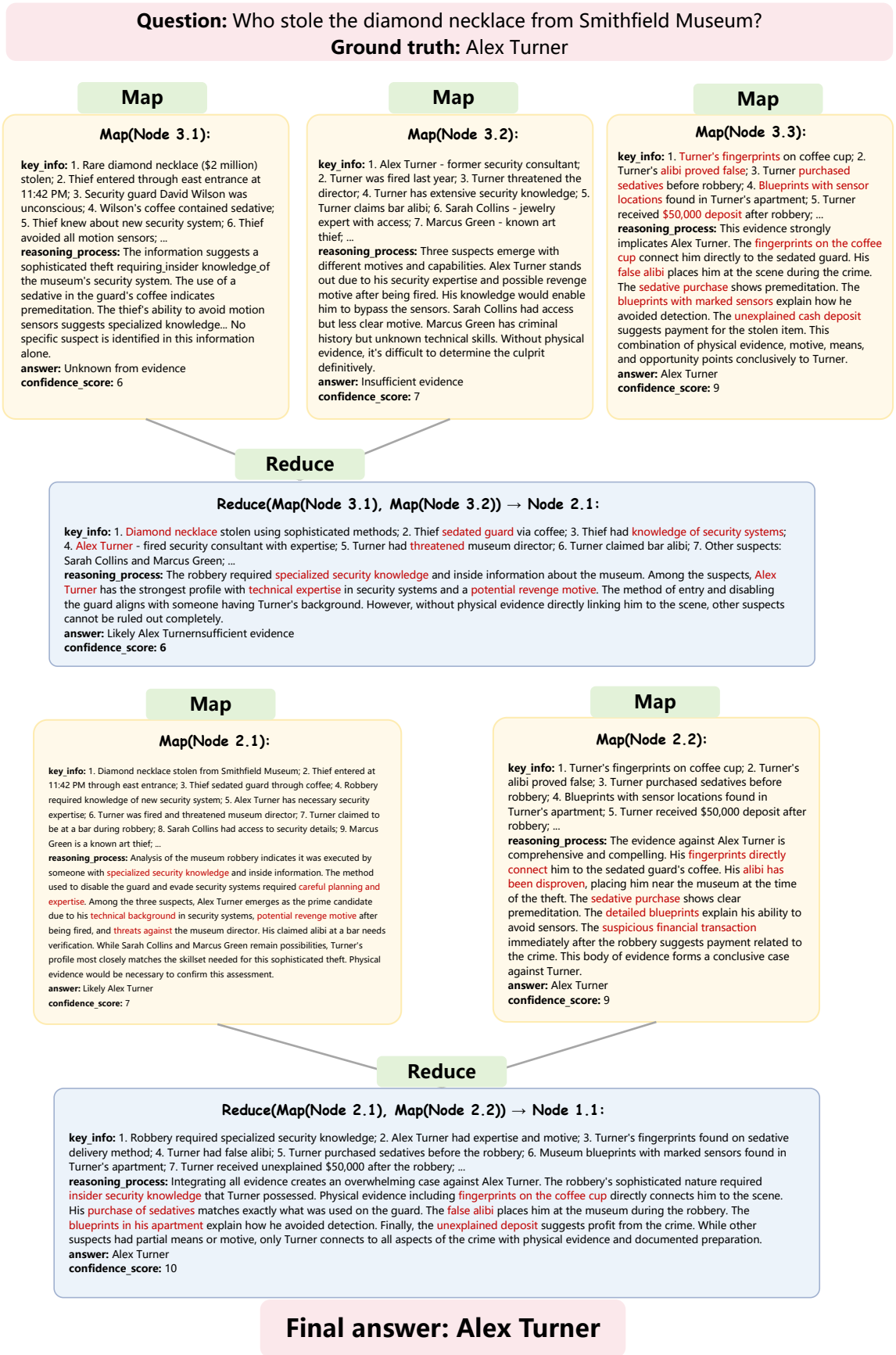


Figure 9: Case Study.

Map Prompt

You are given a portion of an article, previous reasoning results, and a question. Use the original text as the primary source for reasoning, supplemented by other sections (title, keywords, summary). Follow these instructions:

1. Extract Relevant Information (key_info):

Identify at least 10 key details relevant to the question from the original text and reasoning results. The more information, the better.

2. Provide a Rationale (reasoning_process):

Analyze the extracted information and explain how it answers the question.

If information is incomplete, include assumptions or inferences.

3. Answer the Question:

Provide the best possible answer in less than 5 words, concise and concrete, without explanations.

4. Assign a Confidence Score:

Rate your answer out of 10 based on information reliability and reasoning strength (higher score = higher confidence).

5. Example Scoring:

➢ Text: [Jerry is 18 years old this year. He can swim and wants to be an athlete.]

- Jerry can swim: 10 points
- Jerry will be an athlete: 7 points
- Jerry likes chess: 0 points

6. Output format:

Strictly follow the structure below:

```
***start output***
 **key_info**: ;
 **reasoning_process**: ;
 **answer**: ;
 **confidence_score**: ;
***end output***
```

Figure 10: Prompt for Map step.

Reduce Prompt

You are given a question and extracted information from multiple chunks. Your task is to integrate and reason through this data to provide a final answer. Follow these steps:

1. Extract Relevant Information (key_info):

Gather at least 20 key details from the chunks that are relevant to the question.

Combine partial data to improve future reasoning.

2. Provide a Rationale (reasoning_process):

Analyze the gathered information, addressing inconsistencies and weighing confidence scores to form a detailed explanation.

3. Answer the Question:

Provide a concise and concrete answer in less than 6 words.

4. Assign a Confidence Score:

Rate your answer out of 10 based on the reliability and completeness of the information and reasoning.

5. Example Scoring:

- Text: [Jerry is 18 years old this year. He can swim and wants to be an athlete.]
- Jerry can swim: 10 points
- Jerry will be an athlete: 7 points
- Jerry likes chess: 0 points

6. Output format:

Strictly follow the structure below:

```
***start output***
 **key_info**: ;
 **reasoning_process**: ;
 **answer**: ;
 **confidence_score**: ;
***end output***
```

Figure 11: Prompt for Reduce step.

Chunk Prompt

You are a text structuring expert. Organize the following long text into a clear, hierarchical format while preserving all original information (characters, events, timelines, etc.). Below are the requirements:

1. Overall Analysis:

- Primary Title: Reflect the main theme (≤15 words).
- Keywords: Extract 3-5 core keywords (sorted by importance, separated by semicolons).
- Summary: Summarize in one sentence (≤100 words).

2. Hierarchy Rules:

Secondary Titles:

- Distinguish thematic modules (≤15 words).
- Include 3-5 keywords and a short summary (≤100 words).
- If no lower levels, retain the full original text.

Tertiary Titles: Same as secondary titles but refine content further.

3. Structural Guidelines:

- All original content must be preserved without omission.
- Use numbered headings (e.g., 1.; 1.1.; 1.1.1.) for hierarchy.
- Limit to 5 main sections, supporting up to 4 hierarchical levels.
- Keep each section's word count between 500-1,000 words.

4. Output Format:

Strictly follow the structure below:

```
***[General Title]***
**Keywords**: [Keyword1; Keyword2; Keyword3]
**Summary**: [Summary Content]

***1. [First Title]***
**Keywords**: [Keyword1; Keyword2; Keyword3]
**Summary**: [Summary Content]
**Original Text**: [If no lower-level titles, include the full text]

***1.1 [Secondary Title]***
**Keywords**: [Keyword1; Keyword2; Keyword3]
**Summary**: [Summary Content]
**Original Text**: [Include full text if no lower levels]
...
```

Figure 12: Prompt for Hierarchical Semantic Parsing.