BENEFITS AND LIMITATIONS OF COMMUNICATION IN MULTI-AGENT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-thought prompting has popularized step-by-step reasoning in large language models, yet model performance still degrades as problem complexity and context length grow. By decomposing difficult tasks with long contexts into shorter, manageable ones, recent multi-agent paradigms offer a promising near-term solution to this problem. However, the fundamental capacities of such systems are poorly understood. In this work, we propose a theoretical framework to analyze the expressivity of multi-agent systems. We apply our framework to three algorithmic families: state tracking, recall, and k-hop reasoning. We derive bounds on (i) the number of agents required to solve the task exactly, (ii) the quantity and structure of inter-agent communication, and (iii) the achievable speedups as problem size and context scale. Our results identify regimes where communication is provably beneficial, delineate tradeoffs between agent count and bandwidth, and expose intrinsic limitations when either resource is constrained. We complement our theoretical analysis with a set of experiments on pretrained LLMs using controlled synthetic benchmarks. Empirical outcomes confirm the tradeoffs between key quantities predicted by our theory. Collectively, our analysis offers principled guidance for designing scalable multi-agent reasoning systems.

1 INTRODUCTION

Chain-of-thought (CoT) prompting has become the de facto standard for tackling complex reasoning problems. By encouraging models to "think step-by-step", CoT significantly improves performance on tasks requiring mathematical and logical reasoning (Wei et al., 2022). Building on this, recent approaches view reasoning as a structured traversal over thoughts, exploring methods such as self-consistency (Wang et al., 2022), tree-of-thoughts (Yao et al., 2023), and stream-of-search (Gandhi et al., 2024). In parallel, post-training for large reasoning models (LRMs) increasingly relies on reinforcement learning over generated CoTs (OpenAI, 2025; Guo et al., 2025).

Despite these advances, several limitations have emerged. The reasoning abilities of LRMs degrade as the complexity of problem instances increases or as the context length grows (Shojaee et al., 2025; Sun et al., 2025). To address this, new approaches based on multi-agent collaboration (e.g., Zhang et al., 2024b; Tran et al., 2025; Xiao et al., 2025; Hsu et al., 2025) decompose complex tasks into simpler subproblems, coordinating multiple agents to achieve stronger performance. These frameworks offer promising near-term solutions, yet the theoretical underpinnings of their expressive capacity remain poorly understood. While the expressive power of Transformers with CoT prompting has been studied in depth (Merrill & Sabharwal, 2023; Amiri et al., 2025), little is known about the fundamental limits and tradeoffs of communication and resource allocation in multi-agent reasoning schemes. This gap motivates the central question of our work:

From an algorithmic perspective, are there tasks that provably benefit from communication and dynamic resource allocation in multi-agent reasoning systems?

We address this question by proposing a theoretical framework for analyzing the expressivity of collaborative multi-agent reasoning strategies. Our analysis applies to settings where both problem complexity and context size scale, and focuses on three representative algorithmic families: state tracking, recall, and k-hop reasoning. For each task family, we establish bounds on the number of agents and the quantity of communication required, and we characterize the tradeoffs between these

quantities. Finally, we complement our theoretical results with empirical validation using pretrained large language models. Our contributions are as follows:

- We propose a formalization of multi-agent reasoning systems grounded in the rich literature on Transformer expressivity.
- For three distinct families of algorithmic tasks—recall, state tracking and k-hop reasoning—we derive bounds on the number of agents and the communication required, highlighting the tradeoffs between these resources. These tasks capture key aspects of practical reasoning problems, making the results broadly applicable.
- We provide empirical validation of our theoretical insights by implementing the optimal communication protocols given by theory. Our analysis shows the performance in terms of accuracy, communication and token usage closely aligns with theoretical predictions.

Our work focuses on Transformer-based multi-agent systems which partition an input of size N equally between w agents, an abstraction of many settings where multiple agents cooperate by taking responsibility for different parts of the input, such as different document chunks in long-context summarization or question answering, corpus shards or knowledge-graph subgraphs in multi-agent RAG, web pages or site sections in browser-style agents, and map–reduce pipelines where workers process disjoint partitions before a coordinator aggregates the partial results (e.g., Zhou et al., 2025; Chang et al., 2025; Zhang et al., 2024a; Guo et al., 2024; Salve et al., 2024; Yang et al., 2025b; Xiao et al., 2025; Liu et al., 2025; Xu et al., 2025).

Our results reveal three distinct regimes for multi-agent tasks, each instantiated by natural tasks of broad relevance (Table 1). First, some tasks require almost no communication overhead even when the input is partitioned between agents, such as key-query retrieval. Second, some tasks not only allow partitioning but also benefit from it, achieving reduced wall-clock time compared to a single-agent setup; state tracking is a prime example. Finally, some tasks can be solved through partitioning but require significant communication among agents, such as reasoning over multiple hops.

	Size	Depth	Communication
Associative recall	$\Theta(w)$	$\Theta(1)$	$\Theta(1)$
State tracking	$\Theta(N)$	$\mathcal{O}(\frac{N}{w} + \log w)$	$\Theta(w)$
k-hop reasoning	$\mathcal{O}(wk)$	$\mathcal{O}(k)$	$\Theta(k)$ (when $w > 1$)

Table 1: Summary of results. w denotes the number of agents. N represents the length of the input. *Size* corresponds to total computation. *Depth* loosely corresponds to wall-clock time. *Communication* refers to the overall amount of communication between agents. We will define these formally in Section 3. $\mathcal{O}(\cdot)$ indicates existence of a protocol; $\Theta(\cdot)$ indicates that we prove it optimal.

2 Model of Transformers

We assume causally masked (decoder-only) unique hard attention Transformers (UHAT) (e.g., Hahn, 2020; Hao et al., 2022; Yang et al., 2024a; Amiri et al., 2025; Jerad et al., 2025; Bergsträßer et al., 2024), a popular abstraction where attention heads concentrate attention on the position maximizing the attention score. Analysis studies suggest that pretrained models concentrate their attention only few positions (Voita et al., 2019; Clark et al., 2019). Importantly, UHAT subsumes expressivity of ordinary softmax Transformers with fixed precision (Jerad et al., 2025), making it a plausible model of Transformers operating in the regime of long contexts and large reasoning problems.

Each layer of a Transformer has an attention block followed by an MLP block. The attention block takes as input $\mathbf{X} \in \mathbb{R}^{N \times d}$ and applies the operation $\text{Att}(\mathbf{X}) = f^{\text{Att}}(\mathbf{X}\mathbf{W}_Q\mathbf{W}_K^\top\mathbf{X}^\top)\mathbf{X}\mathbf{W}_V^\top$ where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times m}$ and $f^{\text{Att}}(\cdot) = \text{UHAT}(\cdot)$, where for any matrix $\mathbf{A} \in \mathbb{R}^{N \times M}$:

$$\text{UHAT}(\mathbf{A})_{i,j} = \begin{cases} 1 & \text{if } j = \arg\max \mathbf{A}_{i,:} \\ 0 & \text{else} \end{cases}, \tag{1}$$

where in case of a tie, the rightmost element is selected. Multi-head attention with H heads is defined as $\text{M-Att}_H(\mathbf{X}) = [\text{Att}_1(\mathbf{X}), \ldots, \text{Att}_H(\mathbf{X})]\mathbf{W}_O$ where each $\text{Att}_i(\mathbf{X})$ has its own set of parameters.

The matrix $\mathbf{W}_O \in \mathbb{R}^{mH \times d}$ projects the concatenated vector to a vector of dimension d. For an input $\mathbf{X} \in \mathbb{R}^{N \times d}$, the output of a Transformer layer is $\psi(\text{M-Att}_H(\mathbf{X})) \in \mathbb{R}^{N \times d}$ where $\psi : \mathbb{R}^d \to \mathbb{R}^d$ corresponds to the function computed by the MLP. The model has access to arbitrary positional embedding vectors $p_i \in \mathbb{R}^d$, for each $i \in [1, N_{max}]$, where N_{max} is the model's context window.

3 Formalization of Multi-Agent Systems

We formalize multi-agent systems as graphs, with a node representing an agent at a given timestep, and edges describing both the emission of CoT tokens, and communication between different agents. We discuss connections to applied systems in Section 3.1, and illustrate the definition in Figure 1.

Definition 3.1 (Multi-agent system)**.** Let Σ be a (finite or infinite) input alphabet and $\Xi \supset \Sigma$ an (infinite) CoT alphabet. For broad generality, depending on the task, we're allowing both input and CoT alphabets to grow with the input length, such that $\Sigma_1 \subset \Sigma_2 \subset \cdots \subset \Sigma$ and $\Xi_1 \subset \Xi_2 \subset \cdots \subset \Xi$, where $|\Sigma_N|, |\Xi_N| \in \mathcal{O}(\text{poly}(N))$. We write the set of input strings as $\mathcal{S} := \bigcup_{N=1}^{\infty} (\Sigma_N)^N$. We reserve agent identifiers $\text{ID}_1, \ldots, \text{ID}_N \in \Xi_N$.

A *multi-agent system* $\mathcal{A}$ maps strings $x \in \mathcal{S}$ to labeled DAGs $\mathcal{A}(x)$ with $w(x) \leq |x|$ agents where:

1. Each node is uniquely labeled as $T_i^{(t)}$, where $i \in [1, w(x)]$ and $t \in \mathbb{N}$. Informally, it represents agent i's state at time t. For each agent $i \in [1, w(x)]$, there is $D_i \in \mathbb{N}$ such there are nodes $T_i^{(t)}$ exactly for $t \in [1, D_i]$ and for no other t.
2. We define two types of edges:
   (a) *communication edges* $\{c, \sigma\}$ ($\sigma \in \Xi_{|x|}$) from $T_i^{(t)}$ to $T_j^{(t+1)}$, which represent communicating a symbol between two different agents ($i \neq j$)
   (b) *CoT edges* $\{a, \sigma\}$ ($\sigma \in \Xi_{|x|}$), which correspond to autoregressive decoding of the model from $T_i^{(t)}$ to $T_i^{(t+1)}$
3. Every node $T_i^{(t)}$ ($t > 1$) has exactly one incoming edge.
4. Every node $T_i^{(t)}$ can have (i) no outgoing edge, (ii) one outgoing edge, (ii) outgoing edges edge with the same label $\{c, \sigma\}$ to each other agent, $j \neq i$.
5. One agent $i \in [1, w(x)]$ is designated as a *manager agent*.

By definition, agents can only send or receive a single token $w \in \Xi$ at a time.* Every agent can only receive one incoming edge at a time. Intuitively, the symbol provided by the incoming edge at time t (whether it is a CoT or communication edge) is appended to the agent's context at this time step.

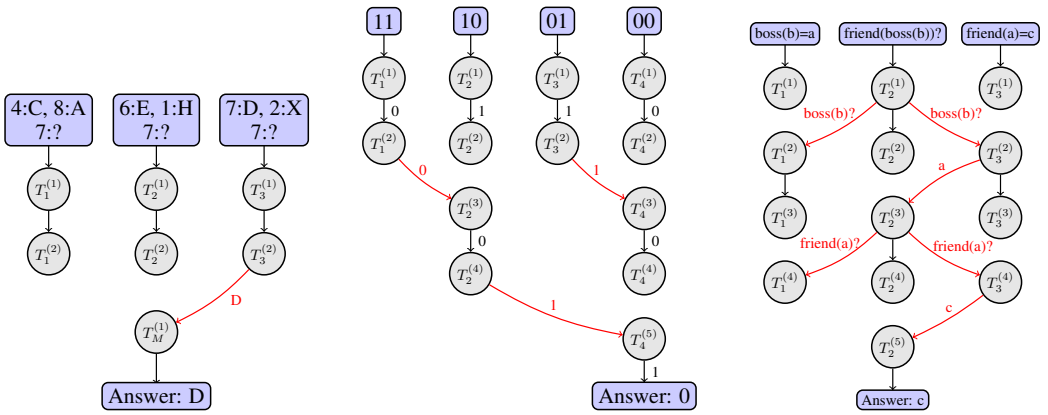
Definition 3.2 (Complexity of Multi-Agent System)**.** We use the following notions to characterize the complexity of a multi-agent system:

- *Computation depth* is the length of the longest path in the graph, irrespective of edge type. We write $Depth(N)$ for the maximum depth on any input x of size $|x| \leq N$. Computation depth is a proxy for the wall-clock time.
- *Width* of the graph corresponds to the number of agents in the system. Typically we use $w(N)$ when the number of agents is a function of input length. As the input is partitioned into chunks of size N/w, we consider $w(N) \in [1, N]$.
- *Size* corresponds to the number of nodes in the graph. We write $Size(N)$ for the maximum size on any input of size N.
- *Communication budget* is the number of nodes with an outgoing communication edge.

We say a multi-agent system $\mathcal{A}$ *computes* a function $f : \mathcal{S} \to \Sigma$ if for all $x \in \mathcal{S}$, the last CoT edge of the manager agent in $\mathcal{A}(x)$ has the label $f(x) \in \Sigma$.

Definition 3.3 (Agent computation)**.** Given a multi-agent system $\mathcal{A}(x)$ on input $x \in \mathcal{S}$, for each agent $i \in [1, w(x)]$, we construct a string $\xi(i) \in (\Xi_{|x|})^*$. Intuitively, this is the sequence of tokens that is processed by this agent over the course of reasoning. The string $\xi(i)$ is constructed as follows:

*An extension to words of bounded length $w \in \Xi^{\leq C}$ would be straightforward, but we find it easiest to formalize the setup with single-token messages. Our theoretical results hold irrespective of this choice



(a) Graphical representation of recall protocol (Sec. 4.2). T_1, T_2 and T_3 are worker agents given chunks of 2 key-value pairs. Only T_3 has the query in its context and thus communicates the answer to the manager T_M after reasoning.

(b) Example of a prefix sum protocol for state tracking (Sec. 4.3) on input 11100100. Here T_2 and T_4 act as intermediate managers composing together the answers of T_1 and T_3 with their own. T_4 also acts as the final manager, providing the final output.

(c) Example of the Iterative Query protocol (Sec. 4.4). Each agent holds one fact. The full query, friend(boss(b)), is managed by T_2, which receives answers at $t = 3, 5$ and broadcasts followup queries at $t = 2, 4$.

Figure 1: Graphical representations of the protocols analyzed in Section 4.

1. First, take the input chunk $x_{ceil(|x|\cdot i/w),\cdots,ceil(|x|\cdot(i+1)/w-1)} \in \mathcal{S}$
2. Then append $\mathrm{ID}_i \in \Xi_{|x|}$
3. Then traverse the nodes associated to the agent, $T_1^{(i)}, T_2^{(i)}, \ldots$:
   (a) If there is an incoming communication edge $\{c, \sigma\}$, append the token $\mathsf{RECEIVE\_}\sigma$.
   (b) If there is an outgoing CoT edge $\{a, \sigma\}$, append the symbol σ.
   (c) If there is an outgoing communication edge,
      i. Either the message is sent to a single agent j – in this case, append the token $\mathsf{SEND\_}\sigma\mathsf{\_ID}_j$ – or,
      ii. the message is broadcast to all agents, append the token $\mathsf{BROADCAST\_}\sigma$.
4. We append the EOS symbol.

We assume all tokens to be part of $\Xi_{|x|}$.[†] A Transformer T *implements* $\mathcal{A}(x)$ on input $x \in \mathcal{S}$ if and only if each of these strings $\xi(i)$ fits into the Transformer's context size, and the transformer predicts all tokens arising from outgoing edges and EOS (cases 3b,3c,4) when run autoregressively on $\xi(i)$. Intuitively, in each reasoning step, the transformer generates the next token, unless it is overridden by incoming communication.

A protocol $\mathcal{A}$ is *expressible* in UHAT if, for each input length n, there is a UHAT Transformer T_n implementing $\mathcal{A}(x)$ on each input x with length $|x| \leq n$, and the numbers of heads and layers are uniformly bounded across all Transformers T_n. We do not require the width d to stay bounded; e.g., growing width can allow positional encodings to keep unboundedly many positions distinct.

The above generalizes the setup of Amiri et al. (2025) to the multi-agent setup. Requiring the system to be implemented by models with bounded layers and heads across input lengths is a very weak assumption, much weaker than the uniformity of the Transformer across input lengths often required in theoretical work on CoT (e.g., Pérez et al., 2019; Merrill & Sabharwal, 2023) – nonetheless, it allows us to prove essentially matching upper and lower bounds on the cost of multi-agent systems.

3.1 Connection to Applied Works

Our formalization covers a broad range of LLM-based protocols which (i) split long contexts into non-overlapping chunks, (ii) process these chunks in parallel with worker agents, (iii) re-

[†]We assume single tokens for simplicity; they could be bounded-length words without change to our results.

lay info to a manager to generates the final answer. The primary distinctions lie in coordination: CoA (Zhang et al., 2024b), LLM×MapReduce (Zhou et al., 2025), NexusSum (Kim & Kim, 2025), AgentSimp (Fang et al., 2025), and Multi^2 (Cao et al., 2025) run workers in parallel with minimal communication, whereas LongAgent (Zhao et al., 2024), XpandA (Xiao et al., 2025), AgentGroupChat-V2 (Gu et al., 2025), and certain task-specific pipelines (e.g., DocAgent (Yang et al., 2025a), Multi-Agent QG (Wang et al., 2025a)) support targeted message-based communication to resolve conflicts. Each of these approaches can be described by a communication graph as in Figure 1. Notably, all of these implement communication using messages written to the recipient agent's context, consistent with our framework.

4 Results

4.1 General Results: Three Regimes for Depth and Communication

In this section, we present theoretical results which hold for all tasks and all multi-agent systems which follow from Definition 3.1. Throughout, by "multi-agent systems", we always refer to systems computed by Transformers. The first result we present relates to the *size* of the system:

Proposition 4.1 (Conservation of size)**.** *Any protocol can be converted into an equivalent single-agent protocol with the same size up to constant factor.*

The proof idea is to construct a single agent that alternates between simulating each of the agents of the original protocol, see Appendix C.2. Thus, the size of the protocol cannot be reduced using multiple agents beyond a constant factor. However, we will see that it may increase in some tasks with the number of agents. In a multi-agent protocol, two other key determinants of cost are (i) depth, and (ii) communication budget. There are two fundamental *a priori* considerations about these quantities. First, any protocol as defined in Definition 3.1 satisfies the inequality:

$$\frac{Size(N)}{w(N)} \leq Depth(N) \tag{2}$$

Proof. The size is upper-bounded by the number of agents times the maximum number of nodes assigned to any individual agent, which is upper-bounded by the maximum length of any path. □

This inequality might lead one to hope that multi-agent protocols reduce depth even if size cannot be reduced. We show that such a gain in depth is realizable in some tasks (Section 4.3), but there are other tasks where *no asymptotic gain in depth is possible* (Section 4.4). A second fundamental observation is that reduction in depth due to multi-agent reasoning is only possible at an increase in communication. In fact, any task solvable at bounded communication cost can already be solved at bounded depth by a single agent, ruling out gains in depth from a multi-agent setup:

Proposition 4.2. *Consider a task with a multi-agent system whose communication budget is $\mathcal{O}(1)$ in N across all $w \in [1, N]$. Then this task has a single-agent CoT with depth (and hence size) $\mathcal{O}(1)$.*

See proof in Appendix C.2. Taken together, this leaves us with *three distinct feasible regimes* of multi-agent reasoning (Table 2). The first one (Section 4.2) is where both depth and communication are $\mathcal{O}(1)$ as the number of agents increases. In this setting, multi-agent setups simply enable processing larger contexts, *without associated cost*. The second regime (Section 4.3) is where depth can be reduced by using more agents (almost up to (2)), though at the cost of increased communication. That is, there is a *depth-communication tradeoff*. The third regime (Section 4.4) is where, at least in the worst case, multi-agent setups require a large amount of communication, without reduction to the required depth. In this regime, multi-agent setups allow processing larger contexts, with a *high cost of communication and no reduction in depth*. A seeming fourth regime, where communication stays $\mathcal{O}(1)$ but depth decreases as (2), is impossible by Proposition 4.2. Importantly, we will next demonstrate that all three regimes are instantiated by naturalistic tasks.

	Communication	
Depth	$\mathcal{O}(1)$	*increases*
$\sim$Size$/w$	impossible	Section 4.3
No Gain	Section 4.2	Section 4.4

Figure 2: Three possible and one impossible regimes for depth-communication tradeoffs.

4.2 Associative Recall

The first task we consider is simple, associative recall. In this setup, given multiple key value pairs, and a queried key, agents must return the associated value. In this case, multi-agent setups permit processing a larger input without associated cost in communication or depth:

Proposition 4.3. *Given an input consisting of N pairs $(x_i, y_i) \in \Sigma_N \times \Sigma_N$, and a query x, consider the task of retrieving the (unique) y such that (x, y) appears in the input. Assume that the input is partitioned disjointly into parts provided to k agents, which also have access to the query. Then they can solve the task with depth $\mathcal{O}(1)$ and communication $\mathcal{O}(1)$.*

Sketch of proof (Full proof in Appendix C.3). Each agent uses attention to check if the query x appears in the input, and uses an induction head to retrieve the associated y if it appears. By design, only one agent will find such a y; it then reports it to a designated manager agent that outputs y. □

Tradeoffs for Simple Retrieval

1. Computation depth $\mathcal{O}(1)$
2. Number of agents $w(N)$ and chunk size: $\frac{N}{w(N)}$
3. Communication budget $\mathcal{O}(1)$
4. Size: $\mathcal{O}(w(N))$

is both realizable and optimal for retrieval.

4.3 State Tracking

Another foundational task is state tracking. State tracking is at the heart of many reasoning problems, such as tracking chess moves in source-target notation, evaluating Python code, or entity tracking (Kim & Schuster, 2023; Merrill et al., 2024). State tracking can be conveniently formalized in terms of evaluation over finite monoids (e.g., Merrill et al., 2024; Grazzi et al., 2025):

Definition 4.1 (State tracking problem)**.** Let M be a finite set, and $(M, \cdot)$ a finite monoid (M with an identity element and associativity). A state tracking problem on M is defined as sending a sequence $m_0 m_1 \dots m_k \in M^*$ to $m_0 \cdot m_1 \cdot \dots \cdot m_k \in M$. Here, the input alphabet is $\Sigma := M$.

Elements of the monoid represent operations (e.g., list manipulation instructions in Python or chess moves). Composing them leads to new monoid elements (e.g., compositions of instructions, or a sequence of chess moves). This problem class subsumes deciding membership for all regular languages, such as Parity, which corresponds to the monoid $(\{0, 1\}, \oplus)$. Amiri et al. (2025) showed that Parity requires a CoT of length $\Omega(N)$. Can a multi-agent system with a large amount of total communication do better? In terms of the *size* of the graph, this cannot be the case:

Proposition 4.4. *Any multi-agent system computing* Parity *requires size $\Omega(N)$.*

The proof is in Appendix C.4. However, if we consider a parallel computation budget, we can obtain a speedup in the *depth* of the computation graph. We assume the setup where each agent receives a disjoint contiguous substring of the input. Then:

Proposition 4.5. *Let M be a finite monoid. There exists a communication protocol with $w(N) = N$, depth $\mathcal{O}(\log N)$ computing the state tracking for M.*

The key idea for this protocol is to compute the prefix sum (or recursive parallel scan) algorithm with the LLM agents, as shown in Figure 1(c). The above protocol has a width of N agents, but it can be generalized to other widths given by some function $w(N)$ of the input size N:

Proposition 4.6. *Given a finite monoid M and any number of agents ($w : \mathbb{N} \to \mathbb{N}$ with $w(N) \in [1, N]$), there exists a $\mathcal{O}(\log w(N) + \frac{N}{w(N)})$ depth and $\mathcal{O}(N)$ size multi-agent system computing state tracking on M with communication budget $w(N)$.*

This means that given enough parallel computation budget, we indeed recover a speedup in terms of effective or wall-clock time. The proof for this result is given in Appendix C.4; Proposition 4.5 is a corollary. The above result is essentially optimal, in that essentially no shorter depth is attainable:

Proposition 4.7 (Optimality, see App. C.4 for proof)**.** *Assume the finite monoid M is a nontrivial group, and $\mathcal{A}$ a multi-agent system computing state tracking over M. Then $\mathcal{A}$ has $\Omega(w(N))$ communication budget, and computation depth $\Omega(\frac{N}{w(N)})$.*

We summarize our results for state tracking below:

Tradeoffs for State Tracking Assume $w(N) \in [1, N]$ agents, each provided a disjoint contiguous portion of the input. Then

1. Computation depth $\mathcal{O}\left(\log w(N) + \frac{N}{w(N)}\right)$
2. Number of agents: $w(N)$ and chunk size: $\frac{N}{w(N)}$
3. Communication budget $\mathcal{O}(w(N))$
4. Size: N

are realizable for performing state tracking. Communication budget and size are optimal. Computation depth is optimal at least up to $\mathcal{O}(\log w(N))$.

4.4 Multi-Hop Reasoning

We instantiate the third regime with k-hop reasoning (e.g., Yang et al., 2024b; Wang et al., 2025b; Yao et al., 2025). In this task, we have a domain $\mathcal{D}$ of objects and a vocabulary $\mathcal{F}$, intended to denote functions. We have a set of N facts $f(x) = y$ ($f, x, y \in \Sigma_N$) contextually given, where for each x and f at most one such fact is included. Each agent receives a disjoint equal sized partition of the set of facts, and a common query of the form $f_1(\dots(f_k(x))\dots)$ where $f_i \in \mathcal{F}$, $x \in \mathcal{D}$. The overall size of the input is $N + k$; each agent has $\frac{N}{w}$ facts and the k-hop query.

Proposition 4.8. *Let the number of agents be $w : \mathbb{N} \to \mathbb{N}$ ($w(N) \in [1, N]$). The k-hop composition task with N facts can be solved with computation depth $\mathcal{O}(k)$, communication budget $\mathcal{O}(k)$, and size $\mathcal{O}(w(N) \cdot k)$. The communication budget is optimal. The computation depth $\mathcal{O}(k)$ is optimal at least up to a $\log(N + k)$ factor.*

The proof is in Section C.5. The idea is that worker agents perform an iterative lookup, where each agent tries to find the next answer in the own context, with one step for each of the k hops $f_k(x)$, $f_{k-1}(f_k(x))$, …, $f_1(\dots(f_k(x))\dots)$. The regime of this task is different from the previous ones; in the worst case, there is no reduction of computation depth when increasing the number of agents: Depending on how the facts are distributed among the agents, computation depth and communication budget may be $\Omega(k)$ in the worst case, as long as more than one agent are involved. The intuition here is that the relevant facts can be distributed between different agents, making iterative lookup the optimal strategy. We thus have:

Tradeoffs for k-hop Composition for k-hop composition and N facts, when $w(k) > 1$:

1. Computation depth $\mathcal{O}(k)$
2. Number of agents: $w(k)$ and chunk size: $\frac{N}{w(k)}$
3. Communication budget $\mathcal{O}(k)$
4. Size: $\mathcal{O}(wk)$

are realizable for k-hop composition. Communication budget is optimal. Computation depth is optimal at least up to a $\log(N + k)$ factor.

5 Experimental Validation

In this section, we experimentally validate whether the protocols of Section 4 work in practice and if computation depth and communication exhibit the three predicted regimes. We evaluate Llama-3.3-70B-Instruct-Turbo and Llama-3.1-8B-Instruct-Turbo (results in Appendix E) on associative recall, state tracking and k-hop reasoning tasks in order to empirically validate each of the three regimes analyzed in theory. We employ pretrained LLMs which are prompted with their roles in the protocol and the instructions to solve the task. We use hard coded communication protocols similar to the protocol implementation of Zhang et al. (2024b). For more details please refer to Section D.

5.1 RECALL

We start by validating experimentally the abilities of different multi-agent systems to perform associative recall. Given a string of key-value pairs and a queried key, models must return the associated value. We use self-consistency with majority voting (Wang et al., 2022) as our baseline and use an implementation of Chain-of-Agent (CoA) for the optimal protocol, given its similarity. For shorter sequences (64–512), both models perform similarly, with Majority Voting sometimes outperforming CoA, but the multi-agent approach gains an edge as length increases. This trend is consistent with theoretical understanding. Recall is a task easily solved by Transformers, even with limited CoT (Arora et al., 2023; Bhattamishra et al., 2024). Thus, at shorter sequence lengths, the communication overhead may be detrimental by e.g., leading to hallucinations in models that do not have the key-value pair in their context.

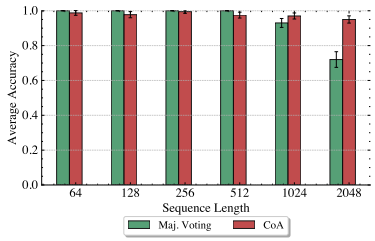
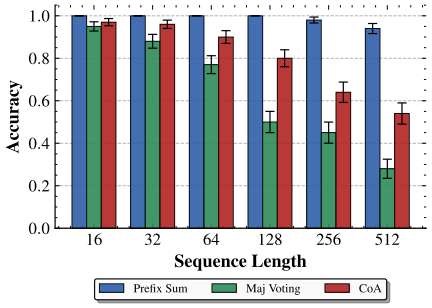


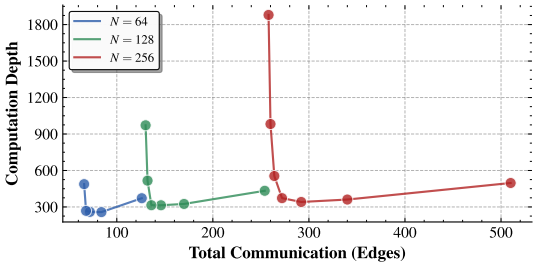
Figure 3: Llama-70B accuracy on RECALL across sequence lengths. CoA is the theoretically optimal protocol.

5.2 STATE TRACKING

We next experimentally evaluate multi-agent systems on state tracking tasks. We evaluate models on PARITY i.e., determining if the number of 1s in a bitstring is even or odd.



(a) Llama-70B accuracy on PARITY for different sequence lengths. Prefix Sum represents the theoretically optimal communication protocol.



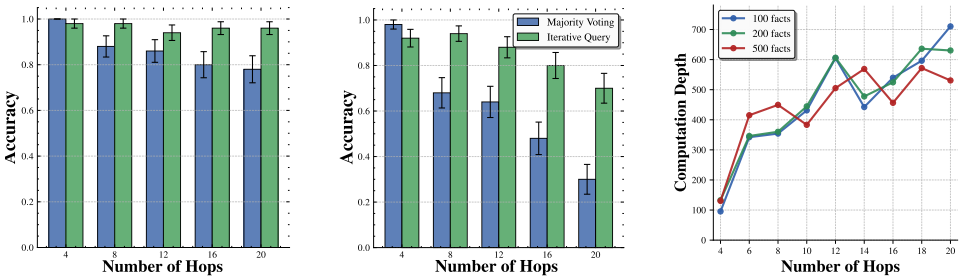
(b) Computation depth against the total amount of communication used. This trend is consistent with the $N/w(N)$ computation depth vs $w(N)$ total communication tradeoff predicted in Section 4.3.

Figure 4: Empirical validation for PARITY.

Figure 4(a) shows that Prefix Sum consistently outperforms all other methods, especially as sequence length grows. Compared to Majority Voting, CoA degrades less with longer sequences, supporting our intuition that chunking complex reasoning into shorter parts helps. In terms of communication, Figure 4(b) shows the tradeoff between the computation *depth* and the total amount of communication for Prefix Sum, calculated by summing the average token usage at every level. This trend is consequent with the theoretically predicted tradeoffs between communication and computation. Indeed, in Section 4.3 we predict a tradeoff between depth $N/w(N)$ and total communication $w(N)$. We note, however, a slight increase in computational depth for high levels of communication. This is due to poor instruction following; models add a constant token overhead by repeating the query and explaining the procedure, especially noticeable in high-communication regimes.

5.3 k-HOP REASONING

Finally, we evaluate models on a k-hop reasoning task, where agents are given *facts* (e.g., "Paula is the boss of Mary") and a *query* (e.g., "Who is the boss of the friend of George?"). Task difficulty depends on the number of facts and query hops. We compare two protocols: Majority Voting and Iterative Query, an implementation of the protocol proved optimal in Section 4.4. As we can see



(a) Accuracy vs. number of hops in the query for 100 facts. Iterative Query outperforms Majority Voting baseline as number of hops increases.

(b) Accuracy vs number of hops in the query for 500 facts. The difference in performance is more pronounced in this regime.

(c) Computation depth vs. number of hops in the query. Computation depth shows an increasing trend as hop count increases.

Figure 5: Empirical validation for k-hop reasoning.

in Figure 5, Iterative Query generally outperforms Majority Voting. This trend is accentuated as the number of hops increases. We note that for the smallest number of hops (four), there are cases where Majority Voting outperforms Iterative Query; we posit that the probability of failing at a given round seems to outweigh the difficulty of retrieving facts in a larger corpus in this regime. Finally, we analyze how the computation depth varies as a function of number of hops in the query. As can be seen in Figure 5(c), the trend we observe is consistent with theory; as depth of queries increases, so does the computation depth.

6 Discussion and Conclusion

In summary, our work provides principled foundations for understanding the algorithmic benefits and limitations of collaborative multi-agent systems. By formalizing communication and resource tradeoffs, we bridge theoretical analysis with empirical observations, shedding light on when collaboration enhances reasoning efficiency and when it imposes inherent costs. These results open new avenues for designing reasoning systems that balance scalability, expressivity, and performance..

Relationship to Self-Consistency Our results show that multi-agent systems with sophisticated communication protocols outperform majority-vote strategies; Mirtaheri et al. (2025) find that self-consistency with polynomially many agents yields limited gains on tasks hard for Transformers, whereas advanced protocols achieve substantial improvements.

Implications for Protocol Design Our results characterize depth (wall-clock time) and communication cost, identifying three regimes of multi-agent systems (Figure 2) with implications for multi-agent LLM design. Systems with many workers and a single manager (e.g., CoA Zhang et al. (2024b), LLM×MapReduce (Zhou et al., 2025), NexusSum (Kim & Kim, 2025), AgentSimp (Fang et al., 2025), Multi^2 (Cao et al., 2025)) only shift the context bottleneck to the manager, risking errors when aggregating many responses. To address this, we propose a prefix-sum–style cascade: iterative summarization reduces the final-agent bottleneck, with branching factor and depth as tunable hyperparameters. We also believe the Iterative Query protocol for k-hop reasoning could have practical relevance. For complex queries, a similar architecture may be promising: a manager first decomposes the main query into subqueries, each processed through iterative worker–manager communication rounds, with the manager updating the query after every round

Limitations and Future Work There are many directions in which this work could be extended. Empirically, ideas discussed in the above paragraph could be incorporated into the design of novel multi-agent systems. Theoretically, our work could be extended to different algorithmic tasks e.g., graph reachability or to different multi-agent paradigms such as adversarial games or cooperative reinforcement learning tasks, where agents collaborate to reach a common goal.

Reproducibility Statement

We provide a complete reproducibility package to facilitate replication of our results. All code will be released alongside the camera-ready version of the paper. Appendix D details the experimental setup, including the hyperparameters and parameter ranges considered, as well as all system prompts used. Complete proofs for all propositions presented in the paper are included in Appendix C.2.

References

Alireza Amiri, Xinting Huang, Mark Rofin, and Michael Hahn. Lower bounds for chain-of-thought reasoning in hard-attention transformers. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Oh9sG5ae2b.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

Pascal Bergsträßer, Chris Köcher, Anthony Widjaja Lin, and Georg Zetzsche. The power of hard attention transformers on data sequences: A formal language theoretic perspective. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=NBq1vmfP4X.

Satwik Bhattamishra, Michael Hahn, Phil Blunsom, and Varun Kanade. Separations in the representational capabilities of transformers and recurrent architectures. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Juntai Cao, Xiang Zhang, Raymond Li, Chuyuan Li, Shafiq Joty, and Giuseppe Carenini. Multi^2: Multi-agent test-time scalable framework for multi-document processing. *arXiv preprint arXiv:2502.20592*, 2025. doi: 10.48550/arXiv.2502.20592. URL https://arxiv.org/abs/2502.20592.

Chia-Yuan Chang, Zhimeng Jiang, Vineeth Rakesh, Menghai Pan, Chin-Chia Michael Yeh, Guanchu Wang, Mingzhi Hu, Zhichao Xu, Yan Zheng, Mahashweta Das, and Na Zou. MAIN-RAG: Multi-agent filtering retrieval-augmented generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2607–2622, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.131. URL https://aclanthology.org/2025.acl-long.131/.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3563–3578, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.212. URL https://aclanthology.org/2024.findings-acl.212/.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

Dengzhao Fang, Jipeng Qiang, Xiaoye Ouyang, Yi Zhu, Yunhao Yuan, and Yun Li. Collaborative document simplification using multi-agent systems. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 897–912, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.60/.

Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pp. 114–118, 1978.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg KH Franke, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear rnns through negative eigenvalues. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zhouhong Gu, Xiaoxuan Zhu, Yin Cai, Hao Shen, Xingzhou Chen, Qingyi Wang, Jialin Li, Xiaoran Shi, Haoran Guo, Wenxuan Huang, Hongwei Feng, Yanghua Xiao, Zheyu Ye, Yao Hu, and Shaosheng Cao. Agentgroupchat-v2: Divide-and-conquer is what llm-based multi-agent system need. *arXiv preprint arXiv:2506.15451*, 2025. doi: 10.48550/arXiv.2506.15451. URL https://arxiv.org/abs/2506.15451.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Tiezheng Guo, Chen Wang, Yanyi Liu, Jiawei Tang, Pan Li, Sai Xu, Qingwen Yang, Xianlin Gao, Zhi Li, and Yingyou Wen. Leveraging inter-chunk interactions for enhanced retrieval in large language model-based question answering. 2024. doi: 10.48550/arXiv.2408.02907. URL https://arxiv.org/abs/2408.02907v1. Version v1.

Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

Michael Hahn and Mark Rofin. Why are sensitive functions hard for transformers? *arXiv preprint arXiv:2402.09963*, 2024.

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810, 2022.

Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Feng-Ting Liao, Yi-Chang Chen, Sattar Vakili, and Da-shan Shiu. Group think: Multiple concurrent reasoning agents collaborating at token level granularity. *arXiv preprint arXiv:2505.11107*, 2025.

Sungjin Im, Ravi Kumar, Silvio Lattanzi, Benjamin Moseley, Sergei Vassilvitskii, et al. Massively parallel computation: Algorithms and applications. *Foundations and Trends® in Optimization*, 5 (4):340–417, 2023.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 977–996, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-252-7. doi: 10.18653/v1/2025.acl-short.76. URL https://aclanthology.org/2025.acl-short.76/.

William B Johnson, Joram Lindenstrauss, et al. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Hyuntak Kim and Byung-Hak Kim. NexusSum: Hierarchical LLM agents for long-form narrative summarization. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10120–10157, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.500. URL https://aclanthology.org/2025.acl-long.500/.

Najoung Kim and Sebastian Schuster. Entity tracking in language models. *arXiv preprint arXiv:2305.02363*, 2023.

Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

Pei Liu, Xin Liu, Ruoyu Yao, Junming Liu, Siyuan Meng, Ding Wang, and Jun Ma. Hm-rag: Hierarchical multi-agent multimodal retrieval augmented generation, 2025. URL https://arxiv.org/abs/2504.12330.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024.

Parsa Mirtaheri, Ezra Edelman, Samy Jelassi, Eran Malach, and Enric Boix-Adsera. Let me think! a long chain-of-thought can be worth exponentially many short ones. *arXiv preprint arXiv:2505.21825*, 2025.

Phuong Nguyen and Stephen Cook. Theories for tc0 and other small complexity classes. *Logical Methods in Computer Science*, 2, 2006.

Thang Nguyen, Peter Chin, and Yu-Wing Tai. Ma-rag: Multi-agent retrieval-augmented generation via collaborative chain-of-thought reasoning. *arXiv preprint arXiv:2505.20096*, 2025. URL https://arxiv.org/abs/2505.20096.

OpenAI. OpenAI o3 and o4-mini System Card. Technical report, OpenAI, San Francisco, CA, April 2025. URL https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf. PDF available online.

David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*, 2019.

Michael Rizvi-Martel, Maude Lizaire, Clara Lacroce, and Guillaume Rabusseau. Simulating weighted automata over sequences and trees with transformers. In *International Conference on Artificial Intelligence and Statistics*, pp. 2368–2376. PMLR, 2024.

Aniruddha Salve, Saba Attar, Mahesh Deshmukh, Sayali Shivpuje, and Arnab Mitra Utsab. A collaborative multi-agent approach to retrieval-augmented generation across diverse data. 2024. doi: 10.48550/arXiv.2412.05838. URL https://arxiv.org/abs/2412.05838. Version v1.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

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

Pascal Tesson and Denis Thérien. Diamonds are forever: The variety da. In *Semigroups, algorithms, automata and languages*, pp. 475–499. World Scientific, 2002.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8): 103–111, 1990.

Gal Vardi, Daniel Reichman, Toniann Pitassi, and Ohad Shamir. Size and depth separation in approximating benign functions with neural networks. In *Conference on Learning Theory*, pp. 4195–4223. PMLR, 2021.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

Haotian Wang, Xiyuan Du, Weijiang Yu, Qianglong Chen, Kun Zhu, Zheng Chu, Lian Yan, and Yi Guan. Learning to break: Knowledge-enhanced reasoning in multi-agent debate system, 2024a. URL https://arxiv.org/abs/2312.04854.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities, 2024b. URL https://arxiv.org/abs/2406.04692.

Kesen Wang, Daulet Toibazar, Abdulrahman Alfulayt, Abdulaziz S. Albadawi, Ranya A. Alkahtani, Asma A. Ibrahim, Haneen A. Alhomoud, Sherif Mohamed, and Pedro J. Moreno. Multi-agent interactive question generation framework for long document understanding. *arXiv preprint arXiv:2507.20145*, 2025a. doi: 10.48550/arXiv.2507.20145. URL https://arxiv.org/abs/2507.20145.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Zixuan Wang, Eshaan Nichani, Alberto Bietti, Alex Damian, Daniel Hsu, Jason D Lee, and Denny Wu. Learning compositional functions with transformers from easy-to-hard data. *arXiv preprint arXiv:2505.23683*, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Sibo Xiao, Zixin Lin, Wenyang Gao, and Yue Zhang. Long context scaling: Divide and conquer via multi-agent question-driven collaboration. *arXiv preprint arXiv:2505.20625*, 2025.

Zhen Xu, Shang Zhu, Jue Wang, Junlin Wang, Ben Athiwaratkun, Chi Wang, James Zou, and Ce Zhang. When does divide and conquer work for long context llm? a noise decomposition framework. 2025. doi: 10.48550/arXiv.2506.16411. URL https://arxiv.org/abs/2506.16411v1. Version v1.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=FBMsBdH0yz.

Dayu Yang, Antoine Simoulin, Xin Qian, Xiaoyi Liu, Yuwei Cao, Zhaopu Teng, and Grey Yang. Docagent: A multi-agent system for automated code documentation generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 460–471, Vienna, Austria, 2025a. Association for Computational Linguistics. URL https://aclanthology.org/2025.acl-demo.44/.

Ruiyi Yang, Hao Xue, Imran Razzak, Hakim Hacid, and Flora D. Salim. Divide by question, conquer by agent: Split-rag with question-driven graph partitioning. *arXiv preprint arXiv:2505.13994*, 2025b. doi: 10.48550/arXiv.2505.13994. URL https://arxiv.org/abs/2505.13994.

Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. Do large language models perform latent multi-hop reasoning without exploiting shortcuts? *arXiv preprint arXiv:2411.16679*, 2024b.

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

Yuekun Yao, Yupei Du, Dawei Zhu, Michael Hahn, and Alexander Koller. Language models can learn implicit multi-hop reasoning, but only if they have lots of training data. *arXiv preprint arXiv:2505.17923*, 2025.

Yizhou Zhang, Lun Du, Defu Cao, Qiang Fu, and Yan Liu. Prompting large language models with divide-and-conquer program for discerning problem solving. 2024a. doi: 10.48550/arXiv.2402.05359. URL https://arxiv.org/abs/2402.05359v3. Version v3.

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

Jun Zhao, Can Zu, Xu Hao, Yi Lu, Wei He, Yiwen Ding, Tao Gui, Qi Zhang, and Xuanjing Huang. LONGAGENT: Achieving question answering for 128k-token-long documents through multi-agent collaboration. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16310–16324, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.912. URL https://aclanthology.org/2024.emnlp-main.912/.

Zihan Zhou, Chong Li, Xinyi Chen, Shuo Wang, Yu Chao, Zhili Li, Haoyu Wang, Qi Shi, Zhixing Tan, Xu Han, Xiaodong Shi, Zhiyuan Liu, and Maosong Sun. LLM $times$MapReduce: Simplified long-sequence processing using large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27664–27678, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1341. URL https://aclanthology.org/2025.acl-long.1341/.

APPENDIX

A USE OF LARGE LANGUAGE MODELS

We used a large language model (GPT-5) in two ways during the research and writing of this paper. First, we used it to refine phrasing and to provide revision suggestions on draft manuscripts. Second, we used it to conduct preliminary literature reviews of the field. Research ideas, proof, and empirical analysis were conducted by the authors.

B CLARIFICATIONS

(i) *Similar work has been already been done in the parallel computation and communication complexity literatures. What is the key difference here?*

Indeed, there are various frameworks for modeling parallel and cooperative computation (such as communication complexity, Parallel RAM (Fortune & Wyllie, 1978), Massively Parallel Computation (Im et al., 2023), BSP Valiant (1990), LOCAL (Peleg, 2000)). There are some conceptual similarities, for instance our analysis of Computation Depth and Size mirrors the concepts of Time (number of parallel steps) and Work (total operations across all processors) in the Parallel RAM model.

Our work differs in making essential use of arguments about the expressivity of the Transformer architecture, and no prior framework would have enabled us to show the set of results in this paper. For instance, the very different behavior between Simple Retrieval (Section 2) and State Tracking (Section 4.3) builds on properties of the Transformer architecture: A Transformer performs retrieval in one step with attention, whereas state tracking is challenging unless a full reasoning chain is used. An unconstrained agent performs both in constant time, a Turing machine or a RAM needs a linear number of steps in both cases.

Models based on unconstrained individual agents (with memory and communication as the primary bottleneck, as in PRAM) predict low computation depth in both tasks; models where individual agents are based on RAM (as in PRAM) predict similarly high depth in both tasks. It is only by considering the specific abilities of Transformers that we achieve a more detailed analysis appropriate to LLM-based multiagent systems, predicting tradeoffs realized by actual LLMs.

(ii) *Why is there no evaluation of the protocols defined in the paper on known QA/Reasoning benchmarks?*

The aim of the experimental section is to validate the theoretical claims we provide. Thus, the experiments are conducted on synthetic setups aimed to mimic the algorithmic tasks found in said sections. Moreover, using synthetic experiments allows us to have full control over the parameters of the problem (e.g. length of sequences, complexity of the problem), which allows us to understand the effects of different problem parameters in a more fine-grained way. Designing practical protocols based on those proved optimal and evaluating them on known benchmarks is, however, an exciting line of research we hope to pursue in future work.

(iii) *What is the practical impact of the results? Do the results relate to any specific NLP tasks?*

RECALL, PARITY and k-hop reasoning are fundamental problems that serve as simple models of reasoning tasks and are of broad interest. RECALL has been shown by Arora et al. (2023) to play in important part in the impressive linguistic abilities of modern transformer-based LLMs. PARITY and other state tracking problems are of great interest as they are directly connected to reasoning problems such as code evaluation, entity tracking in linguistics and generally require some form of world modeling abilities (Merrill et al., 2024; Kim & Schuster, 2023; Rizvi-Martel et al., 2024). Moreover, PARITY is a prime example of a sensitive function, which have been shown to be difficult for Transformer models (Hahn, 2020; Hahn & Rofin, 2024). Finally, k-hop reasoning is foundational to many aspects of reasoning involving the composition of multiple reasoning steps.

(iv) *Why are there no experiments on large reasoning models such as OpenAI-o3 or DeepSeek-R1?*

In the experiments, we typically prompt models to use a specific reasoning strategy for consistency. LRMs typically reason in the way they see best, not necessarily respecting the reasoning strategy given in the prompt. Moreover, using the TogetherAI API, such models typically reason using a large CoT which is hidden to the user, thus rendering it impossible to monitor the exact CoT used by the model.

Moreover, in many experiments, we evaluate token usage as a proxy for certain metrics e.g. computation depth. The long reasoning chains produced by these models make it difficult to see any trends in the token usage as problem complexity increases.

(v) *Section 3 assumes that each agent can only receive bounded communication at each step. What about protocols where each agent can see each other agent's tokens, as in GroupThink (Hsu et al., 2025)?*

One of our key aims here is to understand the cost of communication introduced in multi-agent setups and how it trades off with computation depth, which is easiest to do if we consider a framework that explicitly counts each communication link. In contrast, this is not straightforward in scenarios where all agents have access to the same information.

It may be possible to extend the current framework to protocols with unbounded communication, but this would require careful consideration of what it would mean for transformers to access the contexts of all agents, and how to quantify the amount and cost of communication in scenarios where, at least in principle, all agents have access to the same information.

(vi) *The paper assumes that the input is partitioned between agents. What about frameworks such as multi-agent debates and voting, where multiple agents process the same input?*

Indeed, various papers have proposed strategies where different agents process the same input and solve the problem through mechanisms such as debating or voting (Wang et al., 2022; Dhuliawala et al., 2024; Du et al., 2023; Wang et al., 2024b). An important special case is Self-Consistency (Wang et al., 2022), where agents solve a problem independently and vote at the end.

These techniques fall outside of the scope of our research, which focuses on the *expressivity* of collaborative and multi-agent reasoning strategies. In contrast, these strategies fundamentally leverage the inherent *stochasticity* of trained LLMs to reduce noise and overcome failures. However, these techniques do not increase the intrinsic expressivity of the multi-agent system (e.g., using multiple agents to solve the same state tracking problem and performing voting may reduce error rate, but does not increase the worst-case expressivity of the model.).

(**vii**) *Some of the protocols described in the theory seem quite intricate. Can the protocols described in the theory be linked to protocols from the applied literature?*

Indeed, our results provide a rigorous foundation for certain approaches that have links to ideas from the applied literature. Related to the first regime in our theory (Section 4.2), in various studies, a single query is broadcast and worker agents each inspect only their local shard for an answer, returning a candidate and confidence to a lightweight aggregator that selects or fuses the outputs (e.g., (Zhou et al., 2025; Chang et al., 2025; Zhang et al., 2024a; Salve et al., 2024; Yang et al., 2025b). In our second regime (Section 4.3), recursive aggregation is optimal; this again is related to some existing approaches in the applied literature: agents iteratively compose partial states via tree-structured reduce/merge operators (e.g., parallel scan/prefix-sum and divide-and-conquer), so a global answer is assembled from local summaries with shallow, e.g. logarithmic, communication depth (e.g., (Kim & Kim, 2025; Zhou et al., 2025; Xiao et al., 2025; Xu et al., 2025)). In our third regime (Section 4.4), we prove sequential, multi-hop handoffs optimal; this again links to approaches where the system must iteratively query different agents' shards in a back-and-forth chain—passing intermediate facts as state—so communication depth scales with the hop count (e.g., Yang et al. (2025b); Liu et al. (2025); Nguyen et al. (2025); Wang et al. (2024a)).

Importantly, our results rigorously clarify in which regimes such communication protocols are optimal. Simultaneously, as discussed in Section 6, our results show how, in a rigorous sense, such more sophisticated protocols may improve over simpler but popular strategies such as chain-of-agents or majority voting.

C Proofs

C.1 Notation

We denote with $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the set of natural, integers and real numbers, respectively. We use bold letters for vectors (*e.g.* $\mathbf{v} \in \mathbb{R}^{d_1}$), bold uppercase letters for matrices (*e.g.* $\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$). All vectors considered are column vectors unless otherwise specified. The i-th row and the j-th column of a matrix $\mathbf{M}$ are denoted by $\mathbf{M}_{i,:}$ and $\mathbf{M}_{:,j}$. Let Σ be a fixed finite alphabet of symbols, Σ^* the set of all finite strings (words) with symbols in Σ and Σ^n the set of all finite strings of length n. We use ε to denote the empty string. Given $p, s \in \Sigma^*$, we denote with ps their concatenation.

C.2 Proofs for General Results (Section 4.1)

Proposition C.1 (Conservation of size, repeated from Proposition 4.1)**.** *Any protocol can be converted into an equivalent single-agent protocol with the same size up to constant factor.*

Proof. By constructing a single agent that alternates between simulating each of the agents of the original protocol, computed by a Transformer $T_{\text{SingleAgent}}$.

Recall that all agents have the same Transformer parameters, $T_{\text{MultiAgent}}$.

We extend Ξ with tokens $\langle$`AgentID`$\rangle$, one for each agent ID. We assume some fixed arbitrary ordering over all agent IDs.

The tokens of the reasoning chain record for each step which agent it is associated with. It alternatingly consists of $\langle$`AgentID`$\rangle$ and $\langle$`CoT/communication token`$\rangle$. Say, the even positions are the CoT/communication tokens (the nodes of the original chains); the odd positions are agent indices.

Now, the transformer $T_{\text{SingleAgent}}$ alternates between two modes, depending on the position.

At an $\langle \texttt{AgentID} \rangle$ position, the transformer knows from the input which agent to simulate. It then restrict attention to those tokens that are relevant to this agent, and otherwise performs the computations done by $T_{\text{MultiAgent}}$. The top layer then outputs whatever $T_{\text{MultiAgent}}$ would have output.

In the other mode, $T_{\text{SingleAgent}}$ needs to figure out which agent's turn it is next: It is the first agent in the ordering which is higher than the last agent but hasn't terminated yet. To achieve this, first retrieve the last agent's last previous turn, using a single attention head. Then find the first non-TERMINATE action succeeding that turn; this gives us the agent whose turn it is. The transformer then outpts this agent's ID as the next token. □

Proposition C.2 (Repeated from Proposition 4.2)**.** *Consider a task with a multi-agent system whose communication budget is $\mathcal{O}(1)$ in N across all $w \in [1, N]$. Then this task has a single-agent CoT with depth (and hence size) $\mathcal{O}(1)$.*

Proof. We take w towards N, so that the chunk size becomes 1. Assume that the communication budget is $\mathcal{O}(1)$ in N across all $w \in [1, N]$. Then we can find a protocol with communication budget $\mathcal{O}(1)$ where the depth at $w = N$ is $\mathcal{O}(1)$, as each agent only has a bounded number of tokens to process, which can be hard-coded into a UHAT transformer.

Now we use this to construct a protocol computing the function at depth $\mathcal{O}(1)$ at $w = 1$. To realize this, we increase the Transformer's number of layers to immediately compute the relevant agent's $\mathcal{O}(1)$ computation steps when processing the input in the context window; we code the agent identifiers into the positional embeddings. The only exception here is in the communication edges; for this, the transformer flags places where it would emit a communication edge. After reading the full context, the transformer performs $\mathcal{O}(1)$ CoT steps to simulate the $\mathcal{O}(1)$ communication steps and re-simulating the affected agents. □

We also present here a technical lemma from the work of Vardi et al. (2021) we will use for the MLPs in some of our constructions

Lemma C.1 ((Adapted from Lemma 22 of Vardi et al. (2021))**.** *Let T be a threshold circuit with d inputs, q outputs, depth m and size s. There is a neural network N with q outputs, depth $m + 1$ and size $2s + q$, such that for every input $\mathbf{x} \in \{0, 1\}^d$ we have $N(\mathbf{x}) = T(\mathbf{x})$. Moreover, for every input $\mathbf{x} \in \mathbb{R}^d$ the outputs of N are in $[0, 1]$.*

C.3 Proof for Associative Recall (Section 4.2)

Lemma C.2. *Given an input consisting of N key-value pairs $(\mathbf{x}_i, \mathbf{y}_i)$, and a queried key $\mathbf{x}_{query}$, there exists a two layer Transformer with $\mathcal{O}(\log N)$ width which returns the associated value.*

Proof. We consider that the Transformer agent receives the following input

$$\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{y}_1 \quad \dots \quad \mathbf{x}_n \quad \mathbf{y}_n \quad \mathbf{x}_{\text{query}}]^\top \in \mathbb{R}^{2n+1 \times d}, \tag{3}$$

where $\mathbf{x}_i$ is a vector representing the key and $\mathbf{y}_i$ is a vector representing the value in a sequence of key-value pairs. We assume that every key value $\mathbf{x}_i$ has a unique associated value $\mathbf{y}_i$. We use the following embeddings for tokens:

$$\mathbf{x}_i = [\mathcal{T}(i) \quad \mathcal{T}(\mathbf{y}_\emptyset) \quad P_1(t) \quad P_2(t)] \tag{4}$$

$$\mathbf{y}_i = [\mathbf{0} \quad \mathcal{T}(\mathbf{y}_i) \quad P_1(t) \quad P_2(t)], \tag{5}$$

where both vectors are of size $\log n + \log(n + 1) + 2$, . The first $\log n$ dimensions are Johnson-Lindenstrauss (J-L) vectors Johnson et al. (1984) with the property that $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \leq 1/4$ for $i \neq j$ and $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \geq 3/4$ for $i = j$. These are used to embed the position i and perform retrieval. The set of $\log(n + 1)$ dimensions are used to embed the semantic information of each value vector. These also satisfy the same property as above. Each unique value is embedded in a different vector. Moreover we also define $\mathcal{T}(\mathbf{y}_\emptyset)$ to be the embedding vector corresponding to "no value found" Finally the positional encodings are defined as $P_1(t) = \cos \frac{\pi t}{N}$ and $P_2(t) = \sin \frac{\pi t}{N}$. Here, t corresponds to the index of each token s.t. $t \in [2n + 1]$. Not that this should not be confused with i, which indexes key-value *pairs*. The proof uses a 2-layer transformer model. The first layer

copies the $\mathcal{T}(\mathbf{y}_i)$ vector from $\mathbf{y}_i$ over to the corresponding $\mathbf{x}_i$. This is done with two heads which we index (L) and (R) (for left and right respectively. We set the following:

$$\mathbf{W}_Q^{(L)} = \mathbf{W}_Q^{(R)} = \mathbf{W}_K^{(R)} = [\mathbf{0} \quad \mathbf{I}_2]^\top \tag{6}$$

$$\mathbf{W}_K^{(L)} = [\mathbf{0} \quad \boldsymbol{\rho}_\theta]^\top \tag{7}$$

where $\boldsymbol{\rho}_\theta$ is the rotation matrix given by

$$\boldsymbol{\rho}_\theta = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}, \tag{8}$$

and θ is defined as

$$\theta = -\frac{\pi}{N}, \tag{9}$$

where N is the length of the full sequence. The three first matrices directly select the positional embeddings from the input, and the last matrix selects the positional embeddings and shifts them by 1. Finally, we set

$$\mathbf{W}_V^{(L)} = \begin{bmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{10}$$

$$\mathbf{W}_V^{(R)} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{|\mathcal{D}|} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{11}$$

The MLP for the first layer trivially computes an identity map. For the second layer, we use a construction similar to that of the retrieval heads used in the proof for Prop. 4.7. In essence we select the first $\log n$ dimensions of the vectors with both key and query matrices. This gives us an attention matrix which selects key-value vector which is equivalent to the query and puts it in the last component. Then, the MLP for the second layer extracts the $\mathcal{T}(\mathbf{y}_i)$ vector. The output matrix mapping to tokens is defined s.t. each row is a J-L vector $\mathcal{T}(\mathbf{y}_i), \forall i \in [n+1]$. Thus when computing the dot product with the extracted J-L vector, the majority of the probability mass is put on the correct output token. If the key vector $\mathbf{x}_i$ was not in the agent's chunk, the last token should still encode $\mathcal{T}(\mathbf{y}_\emptyset)$, which corresponds in the output matrix to a special "query not found" token. Using arg max decoding, this thus provides the correct behavior.

Note on MLPs Both computing the identity map and selecting a constant number of values in a vector are tasks that are trivially computable by a TC0 circuit. By appealing to Lemma C.1, we can thus obtain RELU-FFNs that can also perform such operations. □

Lemma C.3. *Given a sequence of tokens $\{x_1, \ldots, x_n\}$, where all but one $x_i = x_\emptyset$, there exists a one-layer Transformer which can retrieve $x_i \neq x_\emptyset$*

Proof. Consider the Transformer agent receives the following input

$$\mathbf{X} = [\mathbf{x}_1 \quad \ldots \quad \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}, \tag{12}$$

where for some $i^* \in [n]$, $\mathbf{x}_{i^*}$ is a one-hot encoding of an entity in $\mathcal{D}$ and $\mathbf{x}_i = \mathbf{x}_\emptyset$ otherwise. We use the following embeddings for tokens:

$$\mathbf{x}_i = [\mathcal{T}(i) \quad \mathcal{T}(\mathbf{x}_i)], \tag{13}$$

where both vectors are of size $\log n + \log(n+1)$. The first $\log n$ dimensions are J-L vectors similar to those used in the proof for Proposition 4.7, with the property that $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \leq 1/4$ for $i \neq j$ and $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \geq 3/4$ for $i = j$. The second $\log(n+1)$ dimensions are J-L vectors corresponding an encoding of each possible entity plus an encoding for the "not found" token $x_\emptyset$. The construction is essentially the same as the second layer of Lemma C.2. The key difference being that there are just "value" vectors and no keys, thus the first layer which copies the one-hot vector for the value vectors to the key vectors is not necessary. □

Proposition C.3. *Given an input consisting of N pairs (x_i, y_i), and a query x, consider the task of retrieving the (unique) y such that (x, y) appears in the input. Assume that the input is partitioned disjointly into parts provided to k agents, which also have access to the query. Then they can solve the task with depth $\mathcal{O}(1)$ and communication $\mathcal{O}(1)$.*

Proof. The proof follows from Lemmas C.2 and C.3. The protocol is as follows:

Consider a setup with w agents, and an input consisting of N key-value pairs. Thus each agent receives a contiguous non-overlapping chunk of size N/w as well as the same queried x. Each agent searches their subset of key-value pairs. The agent that finds the correct key communicates the associated value to a manager agent. All other agents return a special "null" token indicating they did not find the key. The manager then searches through the worker outputs and returns the value.

The worker agents can be simulated using Lemma C.2, and the manager agent extracting the correct answer from all returned agent tokens can be simulated using Lemma C.3. □

C.4 Proofs for State Tracking results (Section 4.3)

Proposition C.4 ((Repeated from Proposition 4.4)**.** *Any multi-agent system computing* PARITY *requires size $\Omega(N)$.*

Proof. By proposition 4.1, we know that any protocol computing PARITY can be converted into an equivalent single agent protocol with some CoT length L. By Theorem 4.2 of Amiri et al. (2025), we have that any UHAT CoT for PARITY has length $\Omega(N)$.

Thus we must have $L \in \Omega(N)$. □

Proposition C.5 (Repeated from Prop 4.6)**.** *Given a finite monoid M and any number of agents ($w : \mathbb{N} \to \mathbb{N}$ with $w(N) \in [1, N]$), there exists a $\mathcal{O}(\log w(N) + \frac{N}{w(N)})$ depth and $\mathcal{O}(N)$ size multi-agent system computing state tracking on M with communication budget $w(N)$.*

Proof. Let an input x of length N be given, where each symbol is an element of M. We assume for simplicity (otherwise padding) that N is a multiple of the number w of agents. We build a DAG as follows.

The context given to agent j is $x_{1,j}, \ldots, x_{N/w,j}, \mathsf{EOS}$ where EOS is the end of sequence (EOS) token. The context length of the sequence given to each agent is thus $N/w+1$ (chunk size plus EOS token).

For each agent j, we create nodes $n_{1,j}, n_{2,j}, \ldots, n_{N/w,j}$, with CoT edges $n_{i,j} \to n_{i+1,j}$ with $\{t, x_{1,j} \ldots x_{i+1,j}\}$.

An agent can use a call $\mathsf{SEND}\sigma$, where SEND is a special token to transmit information to other agents. Without loss of generality, we assume that this command transmits the symbol σ to the next agent with ID $j + 1$. The final agent, which we call the receiver, only receives information and does not transmit. The protocol computes a prefix sum algorithm with branching factor 2: at the beginning of runtime, all agents compute the composition of their N/w elements. Then the agents with odd indices j send their result to those with even indices, who compute the composition of their result with that of their odd index neighbor and so forth in a prefix sum fashion.

We show this is implementable in UHAT with 3 heads and a single layer, with width $\mathcal{O}(\log N)$. Essentially we use 2 heads to extract the value of the monoid elements and then store them in the EOS token and use the MLP to perform the rest of the processing.

Embeddings We will use quasi-orthogonal vectors to keep track of the positions of different elements in the sequence. Formally, let $\mathcal{T}(1), \ldots, \mathcal{T}(2N/w + 1)$ be $2N/w + 1$ vectors of dimension $k = \mathcal{O}(\log N)$ such that $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \leq 1/4$ for $i \neq j$ and $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \geq 3/4$ for $i = j$. Such vectors can be obtained using the Johnson-Lindenstrauss Lemma. We define $E(\sigma)$ to be the embedding vector of some symbol $\sigma \in \Xi$. Embeddings have the following structure

$$E(\sigma) = [\text{ohe}(\sigma) \quad \text{ohe}(\sigma) \quad \mathcal{T}(i) \quad \mathbf{0} \quad \mathbf{0} \quad \mathsf{SEND}], \tag{14}$$

where $\text{ohe}(\sigma) \in \{0,1\}^{|\Xi|}$ is the one hot encoding (OHE) of $\sigma \in \Xi$, $\mathcal{T}(i)$s are quasi orthogonal vectors, the two last dimensions are also of dimension k and where $[\mathsf{send}] \in \{0,1\}$ are flags which are set to 0 by default. Equally, we define the embedding of the end of sequence token as

$$E(\mathsf{EOS}) = [\mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad \mathcal{T}(1) \quad \mathcal{T}(2) \quad \mathsf{SEND}] \tag{15}$$

Construction for composition of monoid elements The construction for composition requires one layer and three heads. The key idea of the construction is to use two heads to extract the two elements to be composed at a given timestep, then concatenate them in the embedding of the \$ token. The MLP can then perform the composition, which it returns in the embedding of the last token. The third head is only there to copy back the remaining embedding values. For the first head, we would have the following key, query and value matrices:

$$\mathbf{W}_Q = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \\ \mathbf{0} \end{bmatrix} \qquad \mathbf{W}_K = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \qquad \mathbf{W}_V = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{16}$$

The output of the attention layer is thus all zeros except for the embedding at the EOS symbol which would be

$$E(\mathsf{EOS}) = [\text{ohe}(\sigma) \quad \mathbf{0} \quad \mathbf{0} \quad \mathcal{T}(i) \quad \mathbf{0} \quad \mathsf{SEND}], \tag{17}$$

The construction for the second head is very similar, with the main differences being the query matrix has the all 0s and identity at the *last* block and the value matrix is like that of the previous head with the two last columns swapped. This would give us a similar sequence of all 0 vectors, except for the embedding at the EOS symbol which would be

$$E(\mathsf{EOS}) = [\mathbf{0} \quad \text{ohe}(\sigma) \quad \mathbf{0} \quad \mathbf{0} \quad \mathcal{T}(i) \quad \mathsf{SEND}], \tag{18}$$

The third head trivially computes the identity matrix (but with 0s at the EOS position) by using both key and query matrices to extract the J-L vectors found at the "third" embedding block. We then use the $\mathbf{W}_O$ matrix to select the relevant parts of out of each head. Once this is done, we use the MLP to compute composition.

MLP The MLP computes a map as defined below. If $\mathsf{SEND} = 0$:

$$[\text{ohe}(\sigma_1) \quad \text{ohe}(\sigma_2) \quad \mathbf{0} \quad \mathcal{T}(i_1) \quad \mathcal{T}(i_2) \quad \mathsf{SEND}] \mapsto$$
$$[\text{ohe}(\sigma_1) \circ \text{ohe}(\sigma_2) \quad \text{ohe}(\sigma_1) \circ \text{ohe}(\sigma_2) \quad \mathbf{0} \quad \mathcal{T}(i_1 + c) \quad \mathcal{T}(i_2 + c) \quad \mathsf{SEND}],$$

where c is the token count between the first token and the EOS token, with $\text{ohe}(\sigma) \circ \text{ohe}(\sigma) \mapsto \text{ohe}(\sigma)$ and $\mathbf{0} \mapsto \mathbf{0}$ in the last two J-L positions.

If $\mathcal{T}(i_2 + c) = \mathcal{T}(2N/w)$, the model computes this slightly different map:

$$\big[\text{ohe}(\sigma_{2N/w-1}) \quad \text{ohe}(\sigma_{2N/w}) \quad \mathbf{0} \quad \mathcal{T}(2N/w - 1) \quad \mathcal{T}(2N/w) \quad \mathsf{SEND}\big] \mapsto$$
$$\big[\text{ohe}(\mathsf{SEND}) \quad \text{ohe}(\sigma_{2N/w-1}) \circ \text{ohe}(\sigma_{2N/w}) \quad \mathcal{T}(2N+1) \quad \mathcal{T}(2N/w + 1) \quad \mathbf{0} \quad \mathsf{SEND}\big],$$

Thus at the next step of decoding the final vector would stay the same.

If the SEND flag is equal to 1, the MLP simply swaps the values in the first $|\Xi|$ dimensions with those in the second $|\Xi|$ dimensions. Thus, once it is time to communicate the model outputs $\mathsf{SEND}\sigma$.

Such a map can be defined by a threshold circuit. Composition of elements from a finite monoid can trivially be evaluated constant time by framing the problem as constant lookup. The shifting of indices is also in TC0 as this is reducible to counting/addition which is known to be in TC0 Nguyen & Cook (2006). Finally, TC0 is closed under boolean combination. Thus performing the conditional routing of which circuit to used based on the flag value is also in TC0. Finally, by Lemma C.1, we know that if such a circuit exists, it can be converted into an MLP which is linear in the circuit parameters.

Output matrix Every row of the output matrix is a OHE of one of the symbols in Ξ. The output matrix is a combined transformation which first selects the top $|\Xi|$ dimensions and uses the OHE vector found there to put a 1 at the underlying position in the output vocabulary vector. Only the last token is used for prediction.

Receiving and sending communication We assume all agents decode synchronously. When an agent receives a symbol, the protocol takes the agent's last symbol, and appends the received symbol as well as a EOS token. For simplicity, we assume that each agent is given a fresh new context at the beginning of a new round of communication. The construction could easily be extended by adding a layer which zeros out the embedding values of vectors from the previous query. To make sure all the agents only send symbols at the appropriate time, one can easily change the number of J-L vectors which the agent receives as these decide at what point the agent sends information. □

Proposition C.6. *Let L be a regular language over Σ. For each $w : \mathbb{N} \to \mathbb{N}$ ($w(N) \in [1, N]$), there is a multi-agent system with $w(N)$ agents that computes membership in L.*

Proof. This statement follows immediately as a consequence of Proposition 4.6. □

Proposition C.7 (Optimality (Repeated from Prop. 4.7))**.** *Assume the finite monoid M is a nontrivial group, and $\mathcal{A}$ a multi-agent system computing state tracking over M. Then $\mathcal{O}(w(N))$ communication budget, and computation depth $\Omega(\frac{N}{w(N)})$ are each optimal.*

Proof. Optimality of the communication budget holds because each agent's portion matters for the result: each agent must send at least one message, hence the communication budget scales linearly with the number of agents. The computation depth lower bound follows from the Proposition 4.1 on size conservation:

$$N = Size \leq \text{Computation-Depth} \cdot \text{Agents} \tag{19}$$

hence

$$\frac{N}{w(N)} \leq \text{Computation-Depth}. \tag{20}$$

From which the result follows. □

C.5 Proofs for k-hop Reasoning (Section 4.4)

Proposition C.8. *Let the number of agents be $w : \mathbb{N} \to \mathbb{N}$ ($w(N) \in [1, N]$). The k-hop composition task with N facts can be solved with computation depth $\mathcal{O}(k)$, communication budget $\mathcal{O}(k)$, and size $\mathcal{O}(w(N) \cdot k)$. The communication budget is optimal. The computation depth $\mathcal{O}(k)$ is optimal at least up to a $\log(N + k)$ factor.*

Proof. We start by exposing the communication protocol which we prove optimal. Then, we give the constructions of the worker and manager agents which implement this protocol. The protocol is as follows:

Let N be the number of total key-value pairs in the context and let w be the number of worker agents. Each worker agent receives a chunk N/w as well as first query $f_1(\mathbf{x}_{\text{query}})$. In the first round, the each agent searches for $f_1(\mathbf{x}_{\text{query}})$ in its chunk. The agent that finds the queried key communicates its associated value to a manager agent. The manager agent then updates the query s.t. $f_2(f_1(\mathbf{x}_{\text{query}})) = f_1(\mathbf{x}'_{\text{query}})$ and broadcasts this new queried key to all worker agents. This process repeats k times in total until the entire k-hop query is resolved.

Worker agent construction The worker agent essentially performs recall on a series of key-value vectors, letting

$$\mathbf{X} = [f(\mathbf{x}_1) \quad \mathbf{y}_1 \quad \dots \quad f(\mathbf{x}_n) \quad \mathbf{y}_n \quad f(\mathbf{x}_{\text{query}})]^\top \in \mathbb{R}^{2n+1 \times d}, \tag{21}$$

we can straightforwardly apply Lemma C.2. We note that the same agent can be used across hops; if we simply append the new queried key to the end current sequence, the worker construction will return the value for the rightmost queried key. This is true because the construction uses rightmost tie-breaking in attention.

Manager construction The manager agent receives a sequence of functions

$$[\mathbf{f}_1 \quad \dots \quad \mathbf{f}_n \quad \mathbf{y}_{\text{query}} \quad E(\#)], \tag{22}$$

where $\mathbf{f}_i$ is an embedding of the function that sends an entity to another through their relationship, $\mathbf{y}_{\text{query}}$ is the current known entity and $E(\#)$ is the embedding of the EOS token. The key idea of the manager construction is to compose together the last function with the query entity in order to return a new entity value. To do so, we define a transition map as

$$\mathbf{q}_i = \sum_{x \in \mathcal{D}} f(x)\mathbf{e}_{i_x} \in \mathbb{R}^{|\mathcal{D}|}, \tag{23}$$

with the full embedding vector being

$$\mathbf{f}_i = [\mathbf{q}_i \quad \mathbf{0} \quad \mathcal{T}(i) \quad \mathbf{0}] \in \mathbb{R}^{2|\mathcal{D}|+2\log N} \tag{24}$$

where $\mathbf{e}_{i_x}$ is the canonical basis vector corresponding to the entity x for some indexing $\mathcal{I}$. Consequently, we have that $\mathbf{y}_{\text{query}} = \begin{bmatrix}\mathbf{0} & \mathbf{e}_{i_y} & \mathcal{T}(n+1) & \mathbf{0}\end{bmatrix} \in \mathbb{R}^{2|\mathcal{D}|+2\log N}$ i.e. a canonical basis vector with a one in the position of the corresponding entity in the second half of the vector. Similarly to the worker agent construction, we define the embedding of EOS token as

$$E(\#) = [\mathbf{0} \quad \mathbf{0} \quad \mathcal{T}(n) \quad \mathcal{T}(n+1)] \tag{25}$$

where $\mathcal{T}(1), \dots, \mathcal{T}(n+1)$ are J-L vectors s.t. $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \leq 1/4$ for $i \neq j$ and $\langle \mathcal{T}(i), \mathcal{T}(j) \rangle \geq 3/4$ for $i = j$. The attention layer is defined in a similar manner to that of the one in the proof for Proposition 4.7 and uses two heads to retrieve both the nth and $n+1$th elements in the sequence using the positional encoding given through J-L vectors.

The MLP then computes the composition of $\mathbf{q}_n$ with $\mathbf{e}_{i_y}$ and outputs the OHE vector of the resulting token. This can easily be done leveraging Lemma 5 of Liu et al. (2022).

Optimality The protocol described above has size $\Theta(wk)$, communication budget $\Theta(k)$, and computation depth $\Theta(k)$.

Optimality of the communication budget follows because composition of k permutations over $\{1, \dots, 5\}$ has communication complexity $\Omega(k)$ in the model where one agent has the even positions and the other the odd positions (Tesson & Thérien, 2002). Thus, communication budget is $\Omega(k)$ even when $w(N) \equiv 2$. Extension to larger $w(N)$ follows by considering the case where all relevant facts happen to be distributed between two agents.

To prove that the depth is worst-case optimal up to a logarithmic factor, we consider the case where all relevant facts happen to be distributed between two agents. Hence, these two agents must jointly emit $\Omega(k)$ communication bits. Because an agent emits only $\mathcal{O}(\log |\Xi_{N+k}|) = \mathcal{O}(\log(N+k))$ bits at a step of time, the number of communication steps between these two agents (and hence the computation depth) must be lower-bounded by $\Omega(\frac{k}{\log(N+k)})$. □

D Experimental Details

General details:

- All experiments were run on TogetherAI API
- Models used: Llama-3.1-8B-Instruct-Turbo, Llama-3.3-70B-Instruct-Turbo, EXAONE-3.5-32B-Instruct
- All experiments are run 100 times with a seed set to 42 for consistency.
- Three multi-agent architectures tested: Majority Voting, Chain-of-Agents, and Prefix Sum
- For all experiments, examples are generated on the fly given the length and difficulty parameters specified in the script.

Throughout, we typically use 8 agents in the majority voting setup.

D.1 Associative Recall Task

Key-value strings are generated at random. A recall query is sampled uniformly from the keys. Models are prompted their roles (manager or worker) explaining what they must do and the communication is handled deterministically.

In experiments, we test sequence lengths (i.e., the number of key-value pairs) as powers of two ranging from 2^4 to 2^{11}. We use 8 agents for Majority Voting. For Chain-of-Agents, we select the optimal chunk size—chosen from powers of two between 8 and 64—for each sequence length.

Associative Recall Majority Vote Agent Prompt

You are a reasoning agent responsible for analyzing a portion of a document. Your task is to detect a specific value in a sequence of key-value pairs, given a corresponding key. Follow these steps:

1. Identify if the key is present in the sequence of key-value pairs.
2. If the key is present, return the value corresponding to the key.
3. If the key is not present, return "NOT_FOUND".
4. Present the final answer in the format "The answer is: [your answer]"

You MUST use the following template. ONLY OUTPUT THE ANSWER. Here is an example for "23 42 12 34 56 78 90 12 | Query: 56":

```
The answer is: 78
```

Associative Recall Chain-of-Agents Worker Prompt

You are a reasoning agent responsible for analyzing a portion of a document. Your task is to detect a specific value in a sequence of key-value pairs, given a corresponding key. Follow these steps:

1. Identify if the key is present in the sequence of key-value pairs.
2. If the key is present, return the value corresponding to the key.
3. If the key is not present, return "NOT_FOUND".
4. Present the final answer in the format "The answer is: [your answer]"

You MUST use the following template. ONLY OUTPUT THE ANSWER. Here is an example for "23 42 12 34 56 78 90 12 | Query: 56":

```
The answer is: 78
```

Associative Recall Chain-of-Agents Manager Prompt

You are a manager agent responsible for synthesizing information from multiple workers. Your task is to combine their provided values and determine the value corresponding to the query. To compute the final value, follow these steps:

1. Collect the value results from all worker agents.
2. Each worker will return either a value or "NOT_FOUND".
3. Exactly one worker will return the value corresponding to the query, the rest will return "NOT_FOUND".
4. Report the value corresponding to the query as your output.
5. Present the final answer in the format "The answer is: [your answer]"

You MUST use the following template. ONLY OUTPUT THE ANSWER. Here is an example for "NOT_FOUND NOT_FOUND 78 NOT_FOUND":

```
The answer is: 78
```

D.2 Parity Calculation Task

String of bits of fixed length are sampled uniformly at random. Ground truth is computed with a function which evaluates parity. Models are prompted their roles (manager or worker) explaining what they must do and the communication is handled deterministically.

The experiments test sequence lengths as powers of 2, ranging from 2^4 to 2^9 (i.e., length 8) with support for index hints to aid model reasoning. We use Majority Voting over a total of 8 agents. For Chain-of-Agents, binary strings are split into chunks of size 8, while Prefix Sum uses a branching factor of 4 for hierarchical processing. The task evaluates models' ability to accurately count 1-bits and determine even/odd parity across different architectural approaches.

The data for the Pareto frontier plots (computation depth vs. total communication) was generated by ablating over the branching factor for the prefix sum protocol. The total communication (number of edges) is can be computed straightforwardly as the sum of all edges of the log-depth tree with the corresponding branching factor. Sequence lengths are taken to be powers of two. For each sequence length 2^n, we ablate over powers of two from 2 to 2^{n-1}.

Parity Majority Vote Agent Prompt

You are a reasoning agent responsible for analyzing a portion of a document. Your task is to provide an analysis of the binary string provided in your chunk and determine if it is even or odd parity. To compute the parity, follow these steps:

1. Count the number of 1's in the binary string.
2. If the count is even, return 0.
3. If the count is odd, return 1.
4. Present the final answer in the format "The answer is: [your answer]"

You MUST use the following template. Here is an example for "1011":

```
1: 1 (count: 1)
2: 0 (count: 1)
3: 1 (count: 2)
4: 1 (count: 3)
Final count: 3
The answer is: 1
```

Parity Prefix Sum Prompt

You are a manager agent responsible for synthesizing the results of previous workers. Your task is to return the parity of the binary string provided by the worker agents. You may think step by step, but your final answer should be concise and clear. To compute the parity, follow these steps:

1. Collect the results from the worker agents. This should be a list of binary digits (0 or 1).
2. If the parity of the list is even, return 0.
3. If the parity of the list is odd, return 1.
4. Present the final answer on a new line in the format "The answer is: [your answer]"

IMPORTANT: Show your work step by step to demonstrate thorough analysis:

1. Go through each bit position and note its value
2. Keep a running count of 1s encountered
3. State the final count
4. Determine if the count is even or odd

You MUST use the following template. Here is an example for "1011":

```
1: 1
0: 1
1: 2
1: 3
Final count: 3
The answer is: 1
```

Parity Chain-of-Agents Worker Prompt

You are a worker agent responsible for analyzing a portion of a document. Your task is to provide an analysis of the binary string provided in your chunk and determine if it is even or odd parity. To compute the parity, follow these steps:

1. Count the number of 1's in the binary string.
2. If the count is even, return 0.
3. If the count is odd, return 1.
4. Provide your result in a clear and concise manner.
5. Present the final answer in the format "The answer is: [your answer]"

You MUST use the following template. Here is an example for "1011":

```
1: 1
0: 1
1: 2
1: 3
Final count: 3
The answer is: 1
```

Parity Chain-of-Agents Manager Prompt

You are a manager agent responsible for synthesizing information from multiple workers. Your task is to combine their provided parities and determine the overall parity of the binary string. To compute the aggregate parity, follow these steps:

1. Collect the parity results from all worker agents.
2. Each worker will return either 0 or 1.
3. Count the number of 1 responses.
4. If the count of 1 responses is even, the overall parity is 0.
5. If the count of 1 responses is odd, the overall parity is 1.
6. Present the final answer in the format "The answer is: [your answer]"

You MUST use the following template. Here is an example for "1011":

```
1: 1
0: 1
1: 2
1: 3
Final count: 3
The answer is: 1
```

D.3 S_5 Permutation Tracking Task

We frame the S_5 permutations task as a word problem where each agent is given a prompt explaining there are 5 balls in 5 distinct bins and a sequence of swap commands such as "swap ball 1 and 3, swap ball 2 and 4". In this task the agents must return the correct value of the ball in each bin. The bin numbers are only given at the beginning of the task making this a *hard* state tracking problem (Merrill et al., 2024).

The experiments test permutations with varying numbers of swaps ranging from 4 to 12 swaps with a step size of 2. By default, the task is constrained to A_5 (even permutations only) by forcing an even number of swaps. For Chain-of-Agents, swap sequences are processed in chunks of 2 swaps per worker, while Prefix Sum uses a branching factor of 2 with each worker handling exactly one swap operation. The task evaluates models' ability to maintain accurate state tracking through sequential ball position updates.

Permutation Majority Vote Agent Prompt

You are a reasoning agent responsible for tracking ball positions through a sequence of swaps.
Your task is to determine the final position of each ball after performing all the given swap operations.
Initial state: Each ball starts in its corresponding bin (ball 1 in bin 1, ball 2 in bin 2, etc.).
To solve this problem:

1. First, identify which balls are mentioned in the swap operations - ONLY track these balls
2. Start with balls in their initial positions (e.g., if balls 1, 2, 3 are mentioned: {1:1, 2:2, 3:3})
3. For each swap operation "Swap ball X and ball Y":
   - Find the current bins of ball X and ball Y
   - Exchange their positions
4. Continue until all swaps are processed
5. Present your final answer as a dictionary mapping ONLY the balls mentioned in swaps to their final positions

IMPORTANT: Only include balls that appear in the swap operations. Do not add extra balls.
Present the final answer in the format "The answer is: {ball1:bin1, ball2:bin2, ...}" for only the balls involved in swaps.

Permutation Chain-of-Agents Worker Prompt

You are a worker agent responsible for processing a portion of swap operations in a larger sequence.
Your task is to carefully track ball positions through your assigned swap operations and report the precise current state.
You will receive:

- Current ball positions as a dictionary (e.g., {1:2, 2:1, 3:3})
- A sequence of swap operations to process

Instructions:

1. Start with the EXACT positions given to you - this is the state after previous swaps
2. Process each swap operation "Swap ball X and ball Y" in order:
   - Find the current bins of ball X and ball Y
   - Exchange ONLY their positions
   - Keep all other balls in their current positions
3. Track each swap carefully - one mistake will affect the final result
4. Report the state after processing ALL your assigned swaps

CRITICAL: Only include the balls that are present in the input positions. The exact same balls, no more, no less.
Present the final answer in the format "The answer is: {ball1:bin1, ball2:bin2, ...}" with the exact same ball numbers as your input.

Permutation Chain-of-Agents Manager Prompt

You are a manager agent responsible for determining the final ball positions from worker results.
Your task is to identify the final state of all balls after all swap operations have been processed by your workers.
You will receive position dictionaries from multiple workers who processed different portions of the swap sequence in order. The workers processed swaps sequentially, so:
Instructions:

1. The workers processed swaps in chronological order (worker 1 $\rightarrow$ worker 2 $\rightarrow$ worker 3, etc.)
2. Each worker started with the positions left by the previous worker
3. The LAST worker's result contains the final positions after all swaps
4. Simply report the last worker's position dictionary as the final answer

CRITICAL: Take the position dictionary from the last (final) worker only. This represents the complete final state.
Present the final answer in the format "The answer is: {ball1:bin1, ball2:bin2, ...}" exactly as reported by the final worker.

Permutation Prefix Sum Worker Prompt

You are a worker agent in a hierarchical system processing ONE swap operation.
IMPORTANT: Initially, balls start in their corresponding bins (ball 1 in bin 1, ball 2 in bin 2, etc.). Through swaps, balls can move to different bins.
Your task is to apply exactly one swap operation and report the resulting ball positions with perfect accuracy.
You will receive:

- Current ball positions as a dictionary (e.g., {1:3, 2:1, 3:2, 4:5, 5:4})
- ONE swap operation: "Swap ball X and ball Y"

Process the swap step-by-step using this exact reasoning template:

1. Current state: [copy the input dictionary]
2. Operation: [copy the swap operation]
3. Ball X is currently in bin: [identify bin number]
4. Ball Y is currently in bin: [identify bin number]
5. After swap: Ball X moves to bin [Y's old bin], Ball Y moves to bin [X's old bin]
6. Verification: Check that only these two balls changed positions, all others remain the same
7. Final state: [complete updated dictionary]

CRITICAL CONCEPT: You are tracking which BALL is in which BIN.

- BALLS are the moving objects (numbered 1, 2, 3, 4, 5)
- BINS are the fixed locations (numbered 1, 2, 3, 4, 5)
- When you swap "ball X and ball Y", you move those balls to different bins
- The bins stay in place - only the balls move between them

CRITICAL: Include ALL balls from input with exact same ball numbers. One swap affects exactly two positions.
Present the final answer in the format "The answer is: {ball1:bin1, ball2:bin2, ...}" with all balls from your input.

Permutation Prefix Sum Manager Prompt

You are a manager agent in a hierarchical ball-tracking system combining results from workers.
IMPORTANT: Initially, balls start in their corresponding bins (ball 1 in bin 1, ball 2 in bin 2, etc.). Through swaps, balls move to different bins.
Your task is to determine the final ball positions after your workers processed their assigned swaps in sequence.
You will receive position dictionaries from workers who processed swaps in chronological order. Each worker:

- Started with the ball positions left by the previous worker
- Applied exactly one swap operation
- Reported the updated positions

Your job requires careful validation and explicit reasoning:

1. Validate that each worker's result is a logical continuation of the previous worker's output
2. Show the complete sequence of states from initial to final
3. Verify that each step represents exactly one swap operation
4. Report the last worker's result as your final output

Use this reasoning template:

1. Initial state: [first worker's input state]
2. After worker 1: [worker 1's result] - validate this is one swap from initial
3. After worker 2: [worker 2's result] - validate this is one swap from worker 1's result
4. Final result: [last worker's result]

CRITICAL: Output exactly the position dictionary from the final (last) worker. This contains the cumulative effect of all swaps.
Present the final answer in the format "The answer is: {ball1:bin1, ball2:bin2, ...}" exactly as reported by the last worker.

D.4 k-HOP REASONING TASK

We create a list of entities and a list of relations. We create the fact base by sampling at random two entities and a relation and then deterministically generating a string. This procedure is done as many times as the number of facts needed. Then relationships are sampled in order to form a valid query that has its answer in the fact base.

The experiments test the models' ability to follow multi-step reasoning chains with varying numbers of hops (relationships to traverse). We use 50 balanced single-token entity names (25 male, 25 female) and 20 diverse single-token relations (boss, instructor, teacher, etc.). The task generates problems with k hops where k ranges from 4 to 20 hops with a step size of 2, using 100 total facts per problem. For the IterativeQueryAgents approach, facts are divided into chunks of 20 facts per worker.

K-hop Majority Vote Agent Prompt

You are an expert at logical reasoning and following relationship chains.
Your task is to answer questions about relationships between people by following chains of connections through the given facts.
You will be given:

1. A set of facts describing relationships between people (e.g., "Alice's boss is Bob")
2. A query asking about a multi-step relationship chain

Instructions:

- Read all the facts carefully
- Follow the relationship chain step by step
- Track each connection to find the final answer
- Output your answer in the exact format: Answer: [PersonName]

Example: Facts: "John's boss is Mary. Mary's supervisor is Tom." Query: "Who is the supervisor of the boss of John?" Reasoning: John's boss is Mary → Mary's supervisor is Tom Answer: Tom
Be systematic and double-check your reasoning chain.

K-hop IterativeQuery Worker Agent Prompt

You are a helpful assistant that answers questions based ONLY on the given facts.
IMPORTANT: You have been given only a small subset of all available facts. It is very likely that the fact needed to answer the query is NOT in your subset.
You will be given:

1. A limited set of facts about relationships between people
2. A specific query about one relationship

Instructions:

- ONLY look through the facts provided to you
- If you find the EXACT fact needed to answer the query, extract the answer
- If the exact fact is NOT in your subset (which is very common), respond with "Not Found"
- DO NOT guess or infer answers from similar facts

- DO NOT make assumptions about relationships not explicitly stated
- DOUBLE-CHECK: Before giving your final answer, carefully re-read the facts to ensure you have the correct match
- Always format your response as: Answer: [YourAnswer]

Example (found): Facts: ”John’s boss is Mary. Alice’s teacher is Bob.” Query: ”Who is John’s boss?” Response: Answer: Mary
Example (not found - very common): Facts: ”John’s boss is Mary. Alice’s teacher is Bob.” Query: ”Who is Sarah’s mentor?” Response: Answer: Not Found
Example (not found - don’t guess): Facts: ”John’s boss is Mary. Alice’s teacher is Bob.” Query: ”Who is Mary’s supervisor?” Response: Answer: Not Found
CRITICAL: Before responding, double-check your work:

1. Re-read the query to understand exactly what is being asked
2. Scan through ALL the facts again to verify your answer or confirm it’s not found
3. Make sure the relationship type matches exactly (e.g., ”boss” vs ”supervisor”)
4. Only provide an answer if you are completely certain it appears in the facts

Remember: Most queries will not have their answer in your subset of facts. Only answer if the exact fact is present and you have double-checked it.

K-hop IterativeQuery Manager Agent Prompt

You are a manager agent that coordinates multi-hop reasoning queries.
Your task is to take an answer from a previous query and generate the next query in the reasoning chain.
You will be given:

1. The original multi-hop question
2. The current intermediate answer
3. The current step number

Instructions:

- Use the intermediate answer to construct the next query
- Format your response as: Next Query: [YourQuery]

Example: Original question: ”Who is the supervisor of the boss of John?” Current answer: ”Mary” (John’s boss) Response: Next Query: Who is Mary’s supervisor?

E Additional Experiments

In this section, we provide experiments from tasks and models that are not featured in the main paper.

E.1 Parity

In this section, we provide Parity results similar to those in the main text, but with llama-8B as the base model for agents. The plots here show trends similar to those in the main text. We note that llama-8B has poorer accuracy on the task; this is due to the deviating from the prompted guidelines and making mistakes (i.e. flipping bits) more frequently

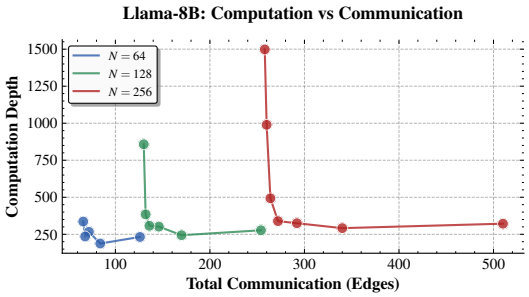


Figure 6: Communication vs Computation tradeoff for Llama-8B showing the relationship between communication budget and computation depth across different multi-agent protocols for the parity task.

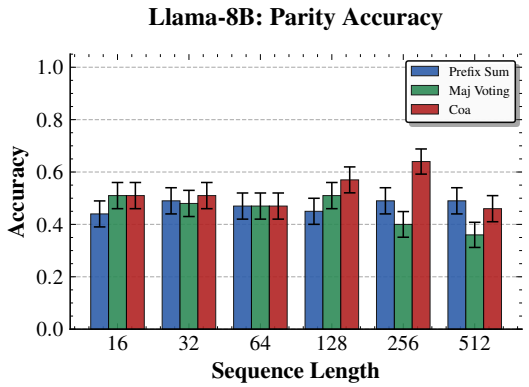
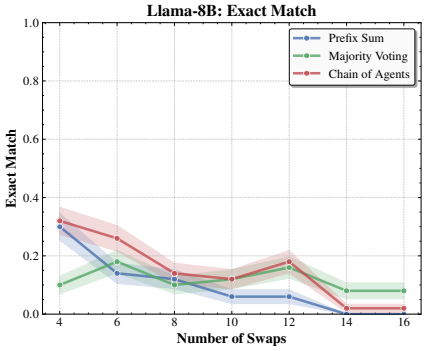


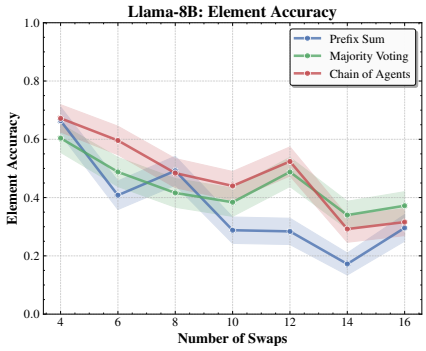
Figure 7: Parity calculation accuracy for Llama-8B across different sequence lengths, comparing single-agent vs multi-agent performance.

E.2 S_5 PERMUTATIONS

Figures 8 and 9 provide detailed comparisons between Llama-8B and Llama-70B models across all three multi-agent approaches.

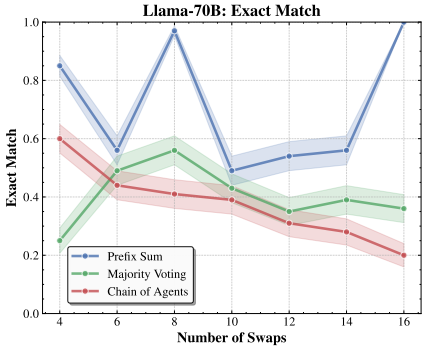


(a) Exact match accuracy for Llama-8B on the S_5 permutations task. Prefix Sum represents the theoretically optimal communication protocol, Majority Voting is self-consistency with majority voting decision, and CoA is Chain-of-agents protocol. Performance degrades as the number of swaps increases, with Prefix Sum maintaining superior performance.

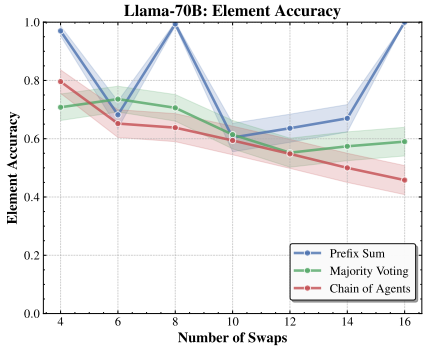


(b) Per-element accuracy for Llama-8B on the S_5 permutations task. Element accuracy measures the fraction of correctly placed elements in the permutation. Shaded regions represent standard error bounds across multiple runs.

Figure 8: Performance comparison of multi-agent approaches on the S_5 permutations task using Llama-8B.



(a) Exact match accuracy for Llama-70B on the S_5 permutations task. The larger model demonstrates consistently improved performance across all agent types compared to Llama-8B, with the performance gap being most pronounced for the Chain of Agents approach.



(b) Per-element accuracy for Llama-70B on the S_5 permutations task. The improved reasoning capabilities of the larger model help mitigate the composition complexity challenges inherent to multi-agent coordination.

Figure 9: Performance comparison of multi-agent approaches on the S_5 permutations task using Llama-70B.

E.3 k-HOP REASONING

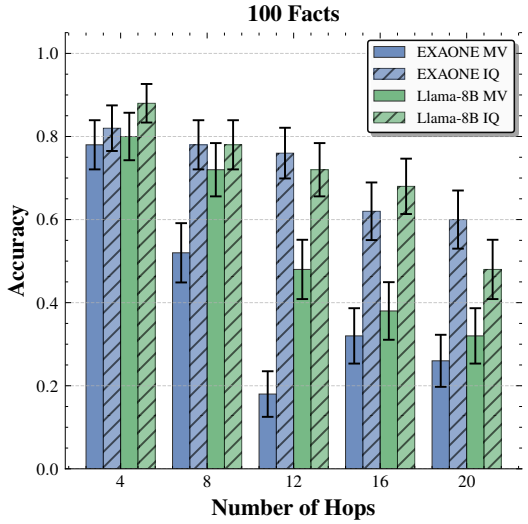


Figure 10: Comparison of multi-agent approaches for k-hop reasoning with 100 facts. Shows performance across different hop lengths for Llama-8B and EXAONE models.

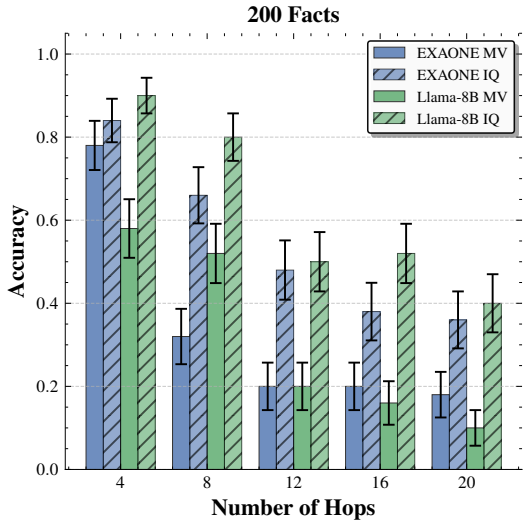


Figure 11: Comparison of multi-agent approaches for k-hop reasoning with 200 facts. Shows performance across different hop lengths for Llama-8B and EXAONE models.

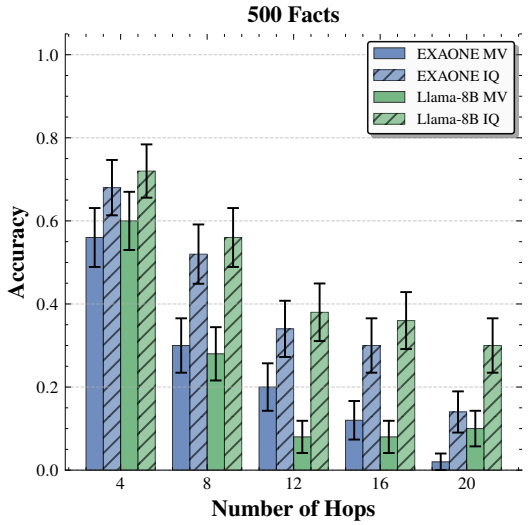


Figure 12: Comparison of multi-agent approaches for k-hop reasoning with 500 facts. Shows performance across different hop lengths for Llama-8B and EXAONE models.

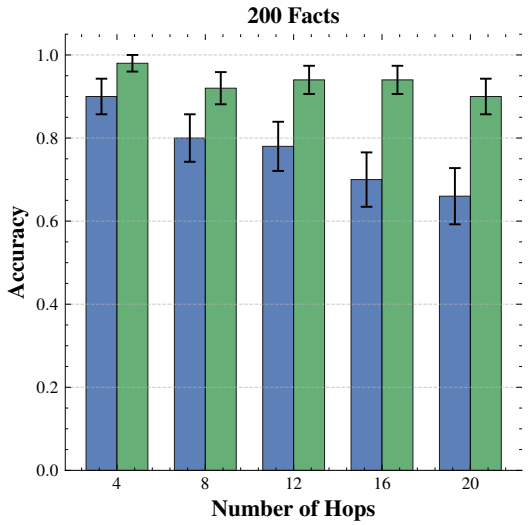


Figure 13: Llama-70B k-hop reasoning accuracy with 200 facts.

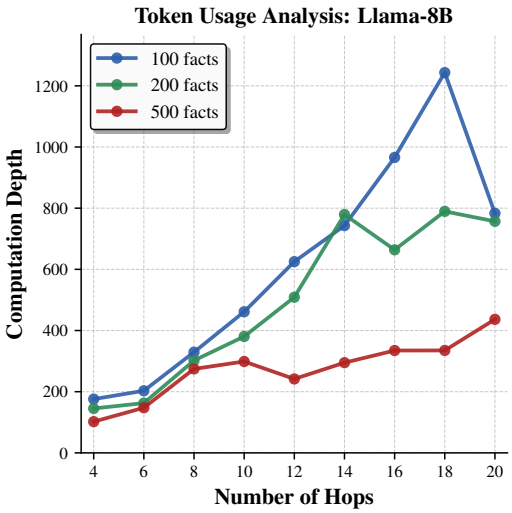


Figure 14: Computation depth vs. number of hops in the query for Lama-8B.