Benefits and Limitations of Communication in Multi-Agent Reasoning

Anonymous Author(s)
Affiliation
Address
email

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 multi-hop reasoning. We derive bounds on (i) the number of agents required, (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 paradigm, 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 chains of thought (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 these challenges, novel approaches such as multi-agent collaboration (e.g. Zhang et al., 2024; Tran et al., 2025; Xiao et al., 2025; Hsu et al., 2025) and adaptive parallel reasoning (Pan 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.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

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 multi-threaded and 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 based on insights from the multi-party communication complexity and parallel processing literature
- For three distinct families of algorithmic tasks—state tracking, recall, 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.

Throughout, we consider the setting where an input of size N is partitioned equally between w agents. Our results reveal three distinct regimes for multi-agent tasks (Table 1). First, there are tasks that can be solved efficiently with minimal chain-of-thought reasoning or communication 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, there are tasks that can be solved through partitioning but require significant communication among agents, such as reasoning over multiple hops.

	Depth	Size	Communication
State tracking	$\Theta(\frac{N}{w} + \log w)$	$\Theta(N)$	$\Theta(w)$
Lookup by query	$\Theta(1)$	$\Theta(w)$	$\Theta(1)$
k-hop reasoning	$\mathcal{O}(k)$	$\Theta(k)$	$\Theta(k)$

Table 1: Summary of results, with w denoting the number of agents. N represents the length of the input. $\mathcal{O}(\cdot)$ indicates the existence of a protocol; $\Theta(\cdot)$ indicates that we prove it optimal.

2 Background

2.1 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.

2.2 Model of Transformers

Transformers. 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

$$\mathrm{Att}(\mathbf{X}) = f^{\mathrm{Att}}(\mathbf{X}\mathbf{W}_Q\mathbf{W}_K^\top\mathbf{X}^\top)\mathbf{X}\mathbf{W}_V^\top \quad (1)$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{m \times d}$ and $f^{\text{Att}}(\cdot) = \text{softmax}(\cdot)$ or $f^{\text{Att}}(\cdot) = \text{UHAT}(\cdot)$. For any matrix $\mathbf{A} \in \mathbb{R}^{N \times M}$, we define the softmax operator row-wise as

$$\text{softmax}(\mathbf{A})_{i,j} = \frac{\exp(\mathbf{A}_{i,j})}{\sum_{k=1}^{M} \exp(\mathbf{A}_{i,k})}.$$

and we define UHAT row-wise as

$$\text{UHAT}(\mathbf{A})_{i,j} = \begin{cases} 1 & \text{if } j = \arg\max \mathbf{A}_{i,:} \\ 0 & \text{else} \end{cases}, \tag{2}$$

where in case of a tie, the rightmost element is selected. For simplicity, we will use $Q(\mathbf{x}_i)$ (and likewise $K(\mathbf{x}_i)$ and $V(\mathbf{x}_i)$) to denote $\mathbf{W}_Q\mathbf{x}_i$. The *width* of the Transformer is $\max(m, d)$, where $m \times d$ is the shape of the projection matrices $\mathbf{W}_Q, \mathbf{W}_K$. Multi-head attention with H heads is defined as $\text{M-Att}_H(\mathbf{X}) = [\text{Att}_1(\mathbf{X}), \dots, \text{Att}_H(\mathbf{X})]\mathbf{W}_O$ 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 layer of Transformer will be $\psi(\text{M-Att}_H(\mathbf{X})) \in \mathbb{R}^{N \times d}$ where $\psi : \mathbb{R}^d \to \mathbb{R}^d$ corresponds to the function computed by the MLP. We use $\mathbf{y}_i^{(l)}$ to denote the ith activation at layer l of a Transformer.

Hard and soft attention Throughout, we will assume a model of Transformers with uniform hard attention, which we will refer to as UHAT for short (e.g. Hahn, 2020; Hao et al., 2022; Yang et al., 2024a; Amiri et al., 2025; Jerad et al., 2025). Although in practice soft attention is easier to train with gradient descent, analysis studies suggest that pretrained models typically concentrate their attention on only a few positions (Voita et al., 2019; Clark et al., 2019) and that the most important heads are those with peaky attention.

Sequence-to-sequence vs. decoder-only The definition above considers an L-layer *sequence-to-sequence* Transformer which sends a sequence of token embeddings to another sequence of token embeddings s.t. $T : \mathbb{R}^{N \times d} \to \mathbb{R}^{N \times d}$. Typically most models commonly used in practice are *decoder-only*: concretely this makes them functions $T : \mathbb{R}^{N \times d} \to \mathcal{V}$ with $\mathcal{V} \subset \mathbb{R}^{|\Sigma|}$. Typically, $\mathcal{V}$ is a set of one hot encoded (OHE) vectors, each associated to a symbol in Σ Concretely, this is implemented by adding an output layer which maps $\mathbf{y}_n^{(L)} \mapsto \mathbf{O}\mathbf{y}_n^{(L)}$ for some linear map $\mathbf{O} \in \mathbb{R}^{|\Sigma| \times d}$.

Constant precision models In this work, we are interested in the expressive power of models with *finite precision*. Our constructions will work with p-bit numbers. Throughout, we will consider *constant precision* (w.r.t. input length): $p = O(1)$.

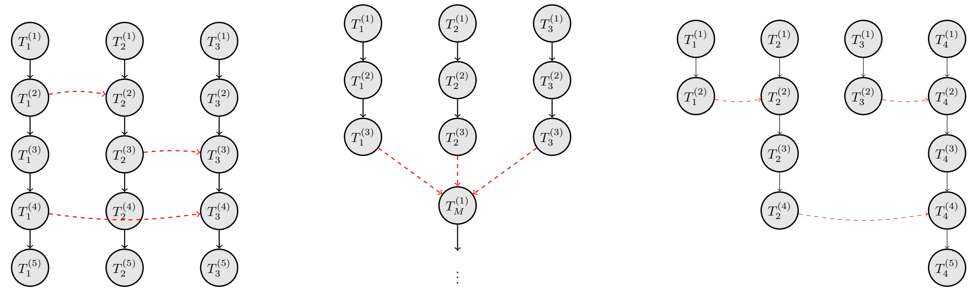
Size preserving functions We say a function $f : \{0,1\}^* \to \{0,1\}^*$ is size *preserving* if and only if there exists c, n such that $\forall x \; |x| \geq n \implies |f(x)| \leq c|x|$. Throughout, we will assume arbitrary size preserving MLPs. This means that the map ψ can compute any function so long as the input and output have a number of bits in the same order of magnitude.

2.3 Formalization of Multi-Thread and Multi-Agent Systems

We define multi-agent systems from a graph perspective:

Definition 2.1 (K-way multi-agent system)**.** Let Σ be a finite alphabet and $\Xi \supset \Sigma$ a CoT alphabet s.t. $|\Xi| \in O(\text{poly}(N))$. A *$K$-way multi-agent system*, denoted $\mathcal{A}_K(\{x^{(i)}\}_{i=1}^K, N, b)$, is a labeled DAG with two edge types. Nodes correspond to the computational model (i.e., Transformers in our case) and edges correspond to a symbol from Ξ outputted by the model. Each node corresponds to a specific model i at a specific decoding step t. We denote $T_i^{(t)}, i \in [K]$ the ith model at timestep t of decoding. We define two types of edge labels: *communication edges* $\{c, \sigma\}, \sigma \in \Xi$ represent communicating a symbol between two different models and *CoT edges* $\{a, \sigma\}, \sigma \in \Xi$ correspond to autoregressive decoding of the model.

Agents can only send or receive one symbol $\sigma \in \Xi$ at a time. If a node receives n communication edges at once, the agent must process each edge one at a time, leading to n CoT steps. A given multi-agent system can communicate in many different ways. We denote $\mathcal{C}(\mathcal{A}_K)$ a specific communication



(a) Example of a K-way multi-agent protocol with $K = 3$, CoT depth 5, and 3 communication edges.

(b) Example of a multi agent setup with 3 worker agents and one manager agent

(c) Example of a prefix sum agent cascade

Figure 1: Example graphs of different multi-agent systems

protocol or strategy implemented under the constraints of $\mathcal{A}_K$. Finally, we define some terminology to characterize the complexity of a multi agent system:

- Let depth represent the longest path on the graph, regardless of the edge type. *Computation depth* is the number of t-edges on this path.
- *Communication depth* or communication rounds is the number of c-edges on the longest path.
- *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.
- *Size* or work corresponds to the number of nodes in the graph.
- *Communication budget* corresponds to the total number of c edges.

Figure 1 illustrates examples of the proposed graph representation for multi-agent systems.

Decision problems We say a K-way multi-agent system $\mathcal{A}_K$ *decides* a function $f : \Sigma^* \to \{0, 1\}$ if for all $x \in \Sigma^*$, and for all partitions $x = x_1 \dots x_K$, there exists a communication protocol $\mathcal{C}$ with a subset $S \subset [K]$ of agents which terminate in $f(x)$. For functions $f : \Sigma^* \to \Sigma$ we say $\mathcal{A}_K$ *evaluates* f. The definition is extended in the straightforward way. More concretely, Transformer models implement the protocol computation in the following way:

Definition 2.2 (Agent computation)**.** Let $T^{(i)}$ denote an agent, represented by a maximal path of CoT edges, possibly augmented with incoming and outgoing communication edges. The computation of $T^{(i)}$ is defined as follows. The first step consists of passing the input chunk $x^{(i)} \in \Sigma^*$ to the agent. The protocol then proceeds according to:

1. Append the agent identifier $\mathsf{ID}(T^{(i)})$.
2. Traverse the nodes along the agent's path in order. Letting $\sigma \in \Xi$ denote a communicated symbol, for each step:
   (a) If there is an outgoing communication edge:
      i. If the message is sent to a single agent $T^{(j)}$, append the token sequence $[\mathsf{send}]\ \sigma\ \mathsf{ID}(T^{(j)})$.
      ii. If the message is broadcast to all agents, append the token sequence $[\mathsf{broadcast}]\ \sigma$.
   (b) If there is an incoming communication edge, append the token sequence $[\mathsf{receive}]\ \sigma$.
   (c) Append the corresponding CoT edge symbol $\{c, \sigma\}$.

A transformer *computes* such a protocol if, when run autoregressively on this string, it predicts all tokens other than those in (a),(b) and (c).

3 Results

3.1 General Results

In this section, we present theoretical results which hold for all task families and all multi-agent systems following Definition 2.1. The first result we present relates to the *size* of the system:

Proposition 3.1 (Conservation of size)**.** *Any protocol can be converted into an equivalent single-agent protocol with the same size up to constant factor.*

Sketch of proof. By constructing a single agent that alternates between simulating each of the agents of the original protocol. □

In essence, this result implies that there is "no free lunch" when it comes to multi-agent systems. Although one can obtain speedups in computation time (or depth), the quantity of work done remains the same. This result is simple, but critical to our analysis of multi-agent systems. The second result in this section situates multi-agent systems within the circuit complexity landscape.

Proposition 3.2. *Consider a decision problem on an input $x \in \{0,1\}^N$ with a multi-agent system $\mathcal{A}_K$ with depth $O(\log^j(N)), i \in \mathbb{N}$. If a UHAT transformer computes $\mathcal{A}_K$, then the decision problem is in AC^i.*

Sketch of proof. The key idea of the proof is to simulate the entire computation graph with a log-depth Transformer and leverage the known circuit complexity results for these models. In order to manage intermediary tokens from the CoT, we allow the model to have $O(\text{poly}(N))$ padding tokens in which in can store such intermediary values. Each depth in the graph is thus simulated by a single Transformer layer which stores the "CoT tokens" in the corresponding padding tokens. Applying the results of Hao et al. (2022). □

3.2 State Tracking

The first family of problems we consider 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. We recall the formal definition of a state tracking problem:

Definition 3.1 (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 $m_0 \cdot m_1 \cdot \dots \cdot m_k \in M$.

This class of problems encompasses deciding membership for all regular languages such as PARITY. Previously, Amiri et al. (2025) showed that for PARITY, UHAT Transformers required a CoT of length $\Omega(N)$. Can a multi-agent system with a large amount of total communication do better? We show that in terms of the *size* of the underlying graph, this cannot be the case:

Proposition 3.3. *Let $K \in \mathbb{N}$, any communication protocol $\mathcal{C}(\mathcal{A}_K)$ deciding* PARITY *using a UHAT Transformer requires size $\Omega(N)$.*

Proof. By proposition 3.1, we know that we can always obtain a serial CoT with equivalent expressivity. By applying Lemma 3.4 of Amiri et al. (2025), we thus directly obtain the result. □

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 3.4. *Let M be a finite monoid. For any word $m_0 \dots m_N \in M^N$, there exists a communication protocol with $\mathcal{A}_N(\{\sigma_i\}_{i=1}^N, \log(N), N)$ which sends $m_0 \dots m_N$ to $m_0 \cdot \dots \cdot m_N$.*

The above protocol has a width of N agents, but we can generalize the above protocol to other widths given by some function $w(N)$ of the input size N:

Proposition 3.5. *Given a monoid M and a constant depth Transformer T with context window of size N, there exists a $O(\log w(N) + \frac{N}{w(N)})$ depth and $w(N)$ (e.g., $\sqrt{N}$) width and $O(N)$ size parallel CoT which solves state tracking on M for sequences of length up to N, with communication budget $w(N)$.*

Effectively, this means that given enough parallel computation budget, we can indeed recover a speedup in terms of effective or wall-clock time. The proof for this result is given in Appendix A.1; Proposition 3.4 is simply a corollary of this proof. The above result is essentially optimal, in that essentially no shorter depth is attainable:

Proposition 3.6 (Optimality)**.** *Assume M is a nontrivial group. Let $w(N)$ be the number of agents, with each receiving a disjoint contiguous part of the string. Then $\mathcal{O}(w(N))$ communication budget, $\mathcal{O}(\log w(N))$ communication depth, and computation depth $\Omega(\frac{N}{w(N)})$ are each optimal.*

Sketch of proof. Optimality of the computation budget holds because each agent's portion matters for the result. An $\Omega(\log w(N))$ lower bound on the communication rounds follows by constructing a tree consisting of only the communication edges, and noting that in each round, an agent receives only one symbol. Now for the time/depth lower bound, we appeal to size conservation:

$$N = Size \leq \text{Computation-Depth} \cdot \text{Agents} \tag{3}$$

hence

$$\frac{N}{w(N)} \leq \text{Computation-Depth} \tag{4}$$

From which the result follows. □

We summarize our results for state tracking below:

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

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

are both realizable and optimal for performing state tracking.

3.3 Simple Retrieval

Another foundational task is to perform simple, associative retrieval. In this case, we obtain a very favorable result:

Proposition 3.7. *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 $O(1)$.*

Sketch of proof. 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 output y. □

Thus:

Tradeoffs for Simple Retrieval

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

is both realizable and optimal for retrieval.

3.4 Multi-Hop Reasoning

A related task is k-hop composition (e.g. Yang et al., 2024b; Wang et al., 2025; 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 facts $f(x) = y$ 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 agents are tasked with jointly evaluating this composition based on the provided facts. Here, the domain and vocabulary may be arbitrarily large filling a context of potentially very large size N, but the computation depth and communication budget depend only on k:

Proposition 3.8. *The k-hop composition task can be solved with computation depth $\mathcal{O}(k)$, communication budget $\mathcal{O}(k)$, and size $\mathcal{O}(k)$. Size and communication budget are optimal. Computation depth and communication depth $\mathcal{O}(k)$ are optimal at least up to a $\log(N + k)$ factor.*

The regime of this task is different from the previous ones in that, in the worst case, there is no reduction of computation depth when increasing the number of agents: Depending on how the facts relevant to the query are distributed among the agents, computation depth and communication budget may be $\Omega(k)$ in the worst case.

We thus have:

Tradeoffs for k-hop Composition for k-hop composition and N facts:

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

   Communication budget $O(k)$
4. Size: $O(k)$

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

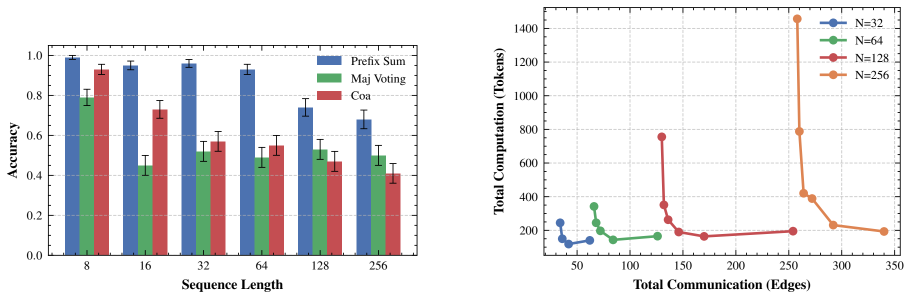
4 Experimental Validation

In this section, we aim to validate experimentally if the proposed communication protocols and constructions of Section 3 also work in practice. To do so, we employ pretrained LLMs that are given a system prompt as well as a query to solve the task. We typically use hard coded communication protocols similar to the protocol implementation of Zhang et al. (2024). For all experiments, we report the mean over 100 runs using the LGAI EXAONE-3.5-32B-Instruct (Research, 2024) model through the TogetherAI API. This model was chosen given it was a free, medium-sized and instruction-tuned. Future work will include analysis on a wider range of models.

4.1 State Tracking

We start by validating experimentally the abilities of different multi-agent systems to perform state tracking tasks. We consider two tasks: (i) PARITY i.e. determining if the number of 1s in a bitstring is even or odd (ii) S_5 permutations, which we frame as a word problem where an 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 bins numbers are only given at the beginning of the task making this a *hard* state tracking problem (Merrill et al., 2024).

We compare our theoretical constructions to two baselines: self-consistency (Wang et al., 2022) with majority voting and Chain-of-Agents (Zhang et al., 2024). We ablate over the branching factor for Prefix Sum, the number of agents for Maj Voting and the chunk size for CoA. For more details about the experiments, please refer to the appendix. We report the mean accuracy over 100 runs for the *best* hyperparameter value found in each sweep.



(a) Accuracy of models on PARITY for different sequence lengths. Prefix Sum represents the theoretically optimal communication protocol, Majority Voting is self-consistency with majority voting decision (Wang et al., 2022) and CoA is Chain-of-agents protocol (Zhang et al., 2024).

(b) Computation depth (calculated by summing the average token usage at each level of the protocol) 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 3.2.

Figure 2: Empirical validation for PARITY.

Parity As we can see in Figure 2(a), the Prefix Sum construction consistently outperforms all other methods. Interestingly, CoA outperforms self-consistency only shorter sequence lengths; as length increases, self-consistency has better accuracy. However, this gain in performance is not noteworthy: both methods have accuracy very close to random chance for large sequence lengths. Only Prefix sum retains a significant advantage over the random chance baseline of 0.5. In terms of communication, Figure 2(b) shows the tradeoff between the computation *depth* and the total amount of communication. This trend is consequent with the theoretical prediction of the tradeoff between communication and computation. Indeed, in Section 3.2 we predict a tradeoff between depth $N/w(N)$ and total communication $w(N)$

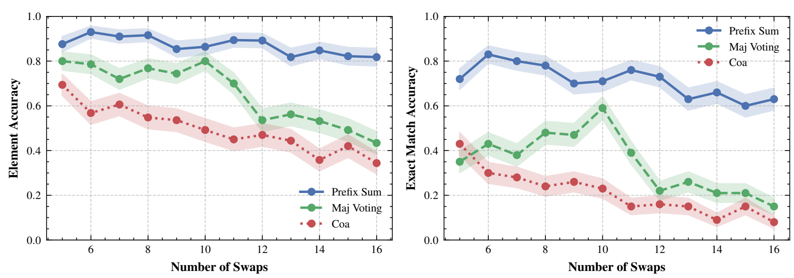


Figure 3: Per-element accuracy (left) and exact match (EM) (right) accuracy for the S_5 permutations task.

S_5 permutations Figure 3 gives the exact match (EM) and the per element accuracy for the permutation task. EM is calculated by either returning 1 if the entire sequence is correct or 0 otherwise. Once again, the prefix sum protocol consistently outperforms both other baselines. Interestingly, majority voting outperforms CoA on this task. This can be explained by the chosen implementation for the permutation problem: worker agents return a dictionary where keys represent bin and values represent balls. The manager agent must then combine together the composition maps given by these dictionaries. When the number of dictionaries to compose is high, this becomes quite difficult, thus limiting the abilities of CoA.

4.2 k-hop Reasoning

Finally, we investigate the abilities of models to perform a k-hop reasoning task. In this task, agents are given a series of *facts* e.g. Paula is the boss of Mary, Mary is a friend of George etc and a *query* e.g. "Who is the boss of the friend of George?". There are two parameters controlling the difficulty of this task; the number of facts and the number of hops in the query. For this task, we consider two baselines. MajorityVoting i.e. self-consistency with majority voting Wang et al.

(2022) and IterativeQuery, a protocol similar to the one optimal for the k-hop task; at each round, multiple agents are given disjoint subsets of the facts and a specific query (e.g. "Who is the friend of George?"). If an agent finds the answer to the query it returns it, agents who do not return a response indicating they did not. The manager then aggregates the answer and updates the query for the next round. This goes on until the final query is answered.

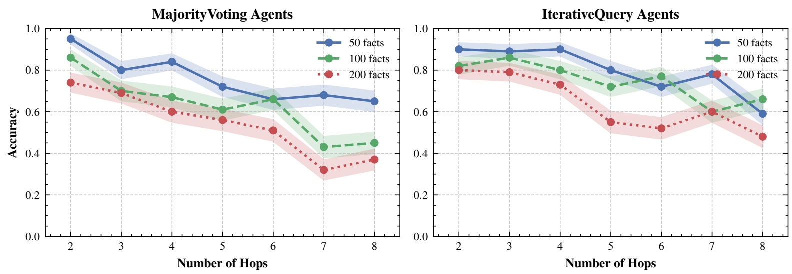


Figure 4: Accuracy vs. number of hops in the query. Left panel shows results for self-consistency with majority voting. Right panel is IterativeQuery, a protocol implementing the optimal k-hop construction. Each line represents a different number of facts in the knowledge base.

As we can see in Figure 4, the IterativeQuery protocol outperforms MajorityVoting. This is especially apparent in the regime where the number of facts is high. This highlights the advantage of separating long contexts for reasoning tasks.

5 Discussion and Conclusion

In summary, our work provides a principled foundation for understanding the algorithmic benefits and limitations of multi-agent reasoning. 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 practical performance.

Practical Considerations Several of our theoretical and empirical observations may be of interest to practitioners or to researchers aiming to design better multi-agent LLM systems. First, we note that setups with multiple worker agents and a single manager (e.g. Zhang et al. (2024)) only shift the context bottleneck to the manager agent; if there is a large amount of workers, the manager must process all of their responses, which can lead to errors. To mitigate this issue, we propose an architecture akin to the prefix sum agent cascade. The key idea is that iterative summarization and processing reduces the bottleneck on the final agent This could be implemented with a constant branching factor and constant depth, left as hyperparameters for the user. We also believe the IterativeQuery protocol we give for k-hop reasoning may be of practical relevance. For tasks with complex queries, it could be interesting to implement a similar architecture, where, at first, a manager model splits the main query into subqueries which are each processed through iterative worker/manager communication rounds, with the manager updating the query after each round.

Limitations and Future Work There are many directions in which this work could be extended. Firstly, it would be exciting to use the practical considerations we provide to design new multi-agent systems, and test them on real-world applications. As for the theoretical side, it would be interesting to extend our analysis to other domains such as graph reachability, where existing literature on parallel processing provides a starting point to analyze optimality of algorithms/number of agents. Finally, the proofs currently assume UHAT and arbitrary MLPs; the analysis could be strengthened by considering softmax attention and RELU feedforward nets.

References

Alireza Amiri, Xinting Huang, Mark Rofin, and Michael Hahn. Lower bounds for chain-of-thought reasoning in hard-attention transformers. *arXiv preprint arXiv:2502.02393*, 2025.

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.

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.

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.

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

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.

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 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.

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.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. *arXiv preprint arXiv:2504.15466*, 2025.

LG AI Research. Exaone 3.5: Series of large language models for real-world use cases. *arXiv preprint arXiv:https://arxiv.org/abs/2412.04862*, 2024.

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.

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.

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*, 2025.

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.

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`.

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.

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, 2024.

A Appendix

A.1 Proofs for State Tracking results

We start by giving a formal proof of Proposition 3.5.

Proposition A.1 (Repeated from Prop 3.5)**.** *Given a monoid M and a constant depth Transformer T with context window of size N, there exists a $O(\log w(N) + \frac{N}{w(N)})$ depth and $w(N)$ (e.g., $\sqrt{N}$) width and $O(N)$ size parallel CoT which solves state tracking on M for sequences of length up to N, 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 $x_{1,j} \dots x_{N/w,j}\#$ where # is the EOS token. The context length of the sequence given to each agent is thus $N/w + 1$.

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 $[\mathsf{send}]$ is a special token to transmit information to other agents. We assume WLOG 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 on 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 # 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 $k = O(\log N)$ 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 through the Johnson-Lindenstrauss Lemma . We define $E(\sigma)$ to be the embedding vector of some symbol $\sigma \in \Xi$. Embeddings have the following structure

$$E(\sigma) = [\text{ohe}(\sigma) \quad \text{ohe}(\sigma) \quad \mathcal{T}(i) \quad \mathbf{0} \quad \mathbf{0} \quad [\mathsf{send}]] \,, \tag{5}$$

where $\text{ohe}(\sigma) \in \{0,1\}^{|\Xi|}$ is the one hot encoding (OHE) of $\sigma \in \Xi$, $\mathcal{T}(i)$ is a quasi orthogonal vector, 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 separator token \$ as

$$E(\#) = [\mathbf{0} \quad \mathbf{0} \quad \mathbf{0} \quad \mathcal{T}(1) \quad \mathcal{T}(2) \quad [\mathsf{send}]] \tag{6}$$

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:

$$\mathbf{W}_Q = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \\ \mathbf{0} \end{bmatrix} \qquad \mathbf{W}_K = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \qquad \mathbf{W}_V = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{7}$$

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

$$E(\#) = [\text{ohe}(\sigma) \quad \mathbf{0} \quad \mathbf{0} \quad \mathcal{T}(i) \quad \mathbf{0} \quad [\mathsf{send}]] \,, \tag{8}$$

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 # symbol which would be

$$E(\#) = [\mathbf{0} \quad \text{ohe}(\sigma) \quad \mathbf{0} \quad \mathbf{0} \quad \mathcal{T}(i) \quad [\mathsf{send}]] \,, \tag{9}$$

The third head trivially computes the identity matrix (but with 0s at the # 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 had. Once this is done, we use the MLP to compute composition.

MLP The MLP uses conditional processing. if $[\mathsf{send}]$ flag is 0, it defines the following map:

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

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

At the last step of composition, using a conditional on based on $\mathcal{T}(i_2 + c) = \mathcal{T}(2N/w)$, the model computes this slightly different map:

$$\begin{aligned}&\left[\text{ohe}(\sigma_{2N/w-1}) \quad \text{ohe}(\sigma_{2N/w}) \quad \mathbf{0} \quad \mathcal{T}(2N/w-1) \quad \mathcal{T}(2N/w) \quad [\mathsf{send}]\right] \mapsto \\ &\left[\text{ohe}([\mathsf{send}]) \quad \text{ohe}(\sigma_{2N/w-1}) \circ \text{ohe}(\sigma_{2N/w}) \quad \mathcal{T}(2N+1) \quad \mathcal{T}(2N/w+1) \quad \mathbf{0} \quad [\mathsf{send}]\right],\end{aligned}$$

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 $[\mathsf{send}]\sigma$

This map is size preserving as it maps elements from Ξ back to Ξ and vectors $\mathcal{T}(i)$ back to vectors from the same set.

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. The agent's context is then wiped and it starts again. 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 A.2. *Let L be a regular language over Σ. For any input $x \in \Sigma^N$, there exists a communication protocol with $\mathcal{A}_N(\{\sigma_i\}_{i=1}^N, \log(N), N)$ which decides L.*

Proof. This statement follows immediately as a consequence of Proposition 3.5. □

A.2 Proofs for Retrieval

Proposition A.3. *The k-hop composition task can be solved with computation depth $\mathcal{O}(k)$, communication budget $\mathcal{O}(k)$, size $\mathcal{O}(k)$. Size and budget are optimal. Computation depth is optimal up to a $\log(N+k)$ factor.*

Proof. A construction is as follows: Each agent checks $f_k(x)$ against the facts in their context using an induction-head-like construction; one agent will find the answer y and reports it back to all agents. Now all agents check whether they have the value of $f_{k-1}(y)$ in their context, and so on. Once the agents have evaluated the final answer, the manager encodes it in its final node. Optimality of size follows because State Tracking is a special case of k-hop composition. Optimality of the communication budget follows because composition of k permutations over $\{1, \ldots, 5\}$ has communication complexity $\Omega(k)$ in the model where one agent has the even positions and the other the odd positions (Tesson & Thérien, 2002). To prove that the depth is worst-case optimal, 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 $\mathcal{O}(\log(N+k))$ bits at a step of time, the communication must be lower-bounded by $\Omega(\frac{k}{\log k})$.

□