

SOLVING A MILLION-STEP LLM TASK WITH ZERO ERRORS

Elliot Meyerson*
Cognizant AI Lab

Giuseppe Paolo
Cognizant AI Lab

Roberto Dailey
Cognizant AI Lab

Hormoz Shahrzad
UT Austin & Cognizant AI Lab

Olivier Francon
Cognizant AI Lab

Conor F. Hayes
Cognizant AI Lab

Xin Qiu
Cognizant AI Lab

Babak Hodjat
Cognizant AI Lab

Risto Miikkulainen
UT Austin & Cognizant AI Lab

ABSTRACT

LLMs have achieved remarkable breakthroughs in reasoning, insights, and tool use, but chaining these abilities into extended processes at the scale of those routinely executed by humans, organizations, and societies has remained out of reach. The models have a persistent error rate that prevents scale-up: for instance, recent experiments in the Towers of Hanoi benchmark domain showed that the process inevitably becomes derailed after at most a few hundred steps. Thus, although LLM research is often still benchmarked on tasks with relatively few dependent logical steps, there is increasing attention on the ability (or inability) of LLMs to perform long range tasks. This paper describes MAKER, the first system that successfully solves a task with over one million LLM steps with zero errors, and, in principle, scales far beyond this level. The approach relies on an extreme decomposition of a task into subtasks, each of which can be tackled by focused microagents. The high level of modularity resulting from the decomposition allows error correction to be applied at each step through an efficient multi-agent voting scheme. This combination of extreme decomposition and error correction makes scaling possible. Thus, the results suggest that instead of relying on continual improvement of current LLMs, massively decomposed agentic processes (MDAPs) may provide a way to efficiently solve problems at the level of organizations and societies.

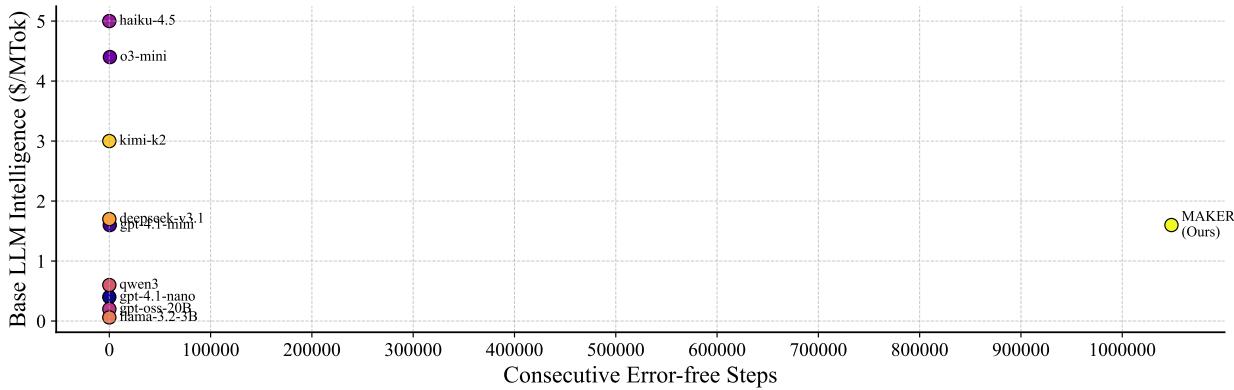


Figure 1: *Orthogonal directions to scaling AI.* The predominant approach to scaling AI is to make more and more ‘intelligent’ base LLMs. This paper introduces a framework and implementation of an orthogonal approach: MAKER, which solves the full task (described in Section 4) with zero errors. In this figure, API cost per output token (as of 10/2025, from openai, anthropic, and together) is used as a proxy for intelligence, and consecutive error-free steps for base LLMs are computed from their per-step error rate (Figure 6b). Appendix A gives a log scale version of the plot.

*Correspondence to: elliot.meyerson@cognizant.com

1 Introduction

Technological achievements of advanced societies are built on the capacity to reliably execute tasks with vast numbers of steps. Whether constructing a skyscraper, airplane, particle accelerator, or iPhone (which relies on tangible contributions from $\approx 1B$ people in an enormous supply chain spanning hundreds of suppliers across multiple countries [1, 2]), running a hospital or medical research organization, processing tax returns and delivering social benefits at a national scale, or even something as seemingly simple as producing a loaf of bread [3], the precise execution of detailed plans and policies is critical to producing high-value outcomes and maintaining societal trust, as the impact of an error in such tasks can range from inconvenience to economic harm to physical harm to death.

Large language models (LLMs) are increasingly inserted into large and complex real-world processes like these. To maximize the benefit of using LLMs in these roles, it is critical to understand the limits of where and how LLMs can be reliably deployed. This paper focuses on the question of how/whether LLMs can execute large tasks with extreme precision, e.g., when a 1% per-step error rate is not acceptable.

This question is investigated by applying LLM-based reasoning to a task whose solution requires more than one million LLM steps with zero errors. The Towers of Hanoi problem, recently proposed as a benchmark for LLM reasoning [4], provides such a task. Most benchmarks for evaluating the quality of LLMs use independent examples, each requiring not many more than a few dependent logical execution steps [5], with a resulting score like accuracy averaged over all examples. Such a benchmark might be considered solved if the accuracy is 99%. However, a system with a 1% per-step error rate is expected to fail after only 100 steps of a million-step task. Therefore, solving a million-step task with zero errors requires a fundamentally different approach.

Such an approach is proposed in this paper: **Massively decomposed agentic processes** (MDAPs). The main contributions of this paper are as follows:

- A design of the **MDAP framework**, which consists of three core components: (1) decomposition into minimal subtasks; (2) error correction based on subtask-level voting; and (3) red-flagging to reduce correlated errors.
- A formalization of this framework that yields **scaling laws**, e.g., how probability of success and expected cost change w.r.t. the number of total steps and level of task decomposition. Under this formalization we find effective scaling under extreme decomposition and infeasibility without it.
- The empirical applications of the framework to **successfully solve a task with over one million steps with zero errors**. One main takeaway is that ‘state-of-the-art’ reasoning models are not required; relatively small non-reasoning models suffice.

This paper provides a first implementation of the MDAP framework: **MAKER** (for Maximal Agentic decomposition, First-to-ahead-by-K Error correction, and Red-flagging), and evaluates it in the Towers of Hanoi domain. MAKER is a system of *agents* in which each agent is assigned a single subtask to solve. In other words, the *role* of each agent is defined by the subtask it is assigned. As advocated for in prior work [6], by avoiding anthropomorphizing agents (i.e., by assigning them human-level roles) and instead assigning them tiny ‘micro-roles’, it is possible to exploit the inherent machine-like nature of LLMs. It may then be possible to take advantage of the kinds of error-correction methods that have been essential to scaling in many domains of computing, i.e., by assigning multiple agents to independently solve the same subtask.

The results demonstrate an instance of **multi-agent advantage** (akin to quantum advantage [7]), that is, a solution to a problem that is not solvable by a monolithic single-agent system. This demonstration sets the stage for a new paradigm of scaling AI: instead of relying on continual improvement of simple underlying LLMs, more powerful AI is achieved through massively decomposed agentic processes (MDAPs).

2 Background

While current LLM agents suffer from catastrophic errors on large tasks, there may be an opportunity for a multi-agent approach that decomposes the tasks into small steps. Error correction is critical for this process, as it is in many complex digital and biological systems.

2.1 Large Agentic LLM Tasks

As large language models have improved, increasing consideration has been given towards real world economic tasks that require multi-step, long horizon reasoning [8]. Research in this direction has repeatedly confirmed an inherent property of LLMs: their performance deteriorates significantly (and often exponentially) with the length of the task

horizon, regardless of task complexity [9, 10]. This observation has led to the recent focus on the ability (and failure) of LLMs to *execute*, i.e., failing to complete many-step tasks, even when a correct plan to follow is explicitly provided to them [11]. While this work identified a fundamental liability of LLMs in long-horizon execution, it also presented an opportunity: Even small improvements in individual subtask performance could lead to exponential improvements in achievable task lengths [11].

At the same time, recent theoretical work has claimed that decomposing large tasks into the smallest possible subtasks can have substantial efficiency benefits [6]. The rise of decomposing tasks into subtasks solvable by focused “small language model” (SLM) agents in industry, motivated by both reliability and cost [12], as well as the burgeoning study of multi-agent LLM systems in academia [13, 14], provides evidence for the practicality of this idea. This paper continues this line of work. It is based on the premise that tasks should be broken up into the smallest possible elements, so that an LLM agent can focus on them one step at a time, improving per step error rate, and thus enabling scaling, reliability, and efficiency in the limit. Critical to the feasibility of such an approach is the granularity of the decomposition, i.e., what defines a single *step*. Since this paper focuses on execution, it is assumed the definition of step is given *a priori*; an orthogonal open question is how to automatically discover optimal decompositions [15, 16]. Informally, the main condition required for the methods in Section 3 to work is that steps are small enough that, for each step, a correct solution is likely to be sampled, and no incorrect solution is more likely.

Now, when an agentic system is applied to a long and expensive multi-step task, there is a natural desire to extract relevant information from responses even when formatted incorrectly. As a result, substantial work has gone in to generating correctly structured output from an LLM. Grammar-constrained (JSON/CFG) decoding reliably enforces structure and often improves downstream pipelines [17, 18], LLMs are fine-tuned to get the format right [19, 20], sampling is performed in a way that only tokens respecting the required format are selected [21], and Python packages are developed dedicated to fixing an LLM’s output post-hoc so that it can be parsed in a meaningful way [22, 23, 24]. However, as described in Section 3.3, when an agent makes an error in the output format, this error may indicate that its other reasoning is wrong as well. When a task has been broken into tiny pieces, this property may be exploited to mitigate errors.

2.2 Error Correction

Error correction is a critical capability across many important areas of computing, including communication [25, 26], memory storage [27], and quantum computing [28]. Error correction makes it possible to pretend that digital communication and classical computation are deterministic, when in fact, single bits are getting lost and flipped all the time [29, 30]. Similarly, improved error correction is the single most important ingredient to achieving scalable quantum computing [31]. In biological systems, error correction is critical to large processes growing and persisting over time. Error correction is necessary both at the population level, e.g., through the error-correcting effects of recombination [32], and at the individual level, e.g., the cancer-fighting ability of mammals. At the individual level, it correlates highly with lifespan and body size (i.e., scale), with elephants showing the most impressive resistance [33]. LLMs now serve as the basis of another substrate of computing, *linguistic computing*, whose constituent processes are *language-based algorithms* (LbAs) [6, 34]. It should then come as no surprise that error correction is critical to achieving LbAs that scale, mitigating for the inherent nondeterminism that results from producing language by pulling from a probability distribution.

Many possible LbA error correction methods can be derived from instances in other fields [35]. One way to reduce errors is for an LLM to reflect on its output and explicitly correct any error it sees [36]. Another approach is to quantify and exploit LLM uncertainty explicitly [37, 38]. For example, work on *semantic density* shows that the semantic content most consistently sampled from an LLM for a given prompt is more likely to be correct than a greedy decoding [38]. This promise of semantic consistency in sampling makes a third, simpler, approach possible: voting, or *ensembling*, which has been a core machine learning technique for decades [39, 40, 41], and is now commonly used to boost the accuracy of LLM-based systems [42]. To date, ensembling has mostly been implemented in LLMs at an action level far above that of a single minimal step. For example, state-of-the-art LLM-based coding systems often use a majority vote of outputs of complete programs that are each a candidate solution to a coding challenge [43, 44]. However, when scaling to tasks with thousands or millions of dependent steps, the level of granularity at which error correction is applied is critical, as will be shown in Section 3.

2.3 Motivating Challenge Domain: Towers of Hanoi

Towers of Hanoi was recently introduced as a test domain for investigating the capabilities and limitations of state-of-the-art LLM reasoning models [4]. This benchmark is based on the classic game in which there are three pegs and D disks, and the goal is to move all disks from the first to the third peg, moving only one disk at a time, and maintaining

Algorithm 1 generate_solution	Algorithm 2 do_voting	Algorithm 3 get_vote
<pre> 1: Input x_o, M, k 2: Initialize $A \leftarrow []$ \triangleright Action list 3: Initialize $x \leftarrow x_o$ 4: for s steps do 5: $a, x \leftarrow \text{do_voting}(x, M, k)$ 6: Append a to A 7: end for 8: return A </pre>	<pre> 1: Input: x, M, k 2: $V \leftarrow \{v : 0 \forall v\}$ \triangleright Vote counts 3: while True do 4: $y \leftarrow \text{get_vote}(x, M)$ 5: $V[y] = V[y] + 1$ 6: if $V[y] \geq k + \max_{v \neq y} V[v]$ then 7: return y 8: end if 9: end while </pre>	<pre> 1: Input x, M 2: while True do 3: $r \sim (M \circ \phi)(x)$ 4: if r has no red flags then 5: return $\psi_a(r), \psi_x(r)$ 6: end if 7: end while </pre>

Figure 2: *Core Components of MAKER.* (1) Maximal Agentic Decomposition (MAD; Section 3.1): By breaking a task with s steps into s subtasks, each agent can focus on a single step; (2) First-to-ahead-by- k Voting (Section 3.2): The modularity resulting from MAD makes error correction at the subtask level effective and scalable; (3) Red-flagging (Section 3.3): Reliability can be further boosted by discarding any response r with high-level indicators of risk. Together these methods enable scaling to solving a task with over one million steps with zero errors.

the condition that a larger disk never sits atop a smaller one [45]. In the benchmark, an LLM system is asked to produce a sequence of moves $(m_i = [d_i, s_i, t_i])_{i=1}^n$ whose execution completes the task, where the i th move is executed by moving disk number $d_i \in \{1, \dots, D\}$ from source peg $s_i \in \{0, 1, 2\}$ to target peg $t_i \in \{0, 1, 2\}$. The problem scales naturally to enormous numbers of required steps by simply adding more disks, since the optimal number of steps to complete the task is $2^D - 1$. For example, solving Towers of Hanoi with ten disks takes just over a thousand steps, and with twenty disks just over a million steps. In its most famous (mythological) incarnation, monks work continuously on an instance with 64 disks, which is expected to take around 585 billion years, at which point the universe will end [46].

Performance of state-of-the-art LLMs degrades catastrophically on this benchmark: They are able to complete the task with a high success rate up until five or six disks, after which the success rate plummets to zero [4]. What this degradation means with respect to whether or to what extent an LLM is really ‘thinking’ or ‘reasoning’ is up for philosophical debate and is outside the scope of this paper [47, 48, 49]. However, this result made it clear that the reliability of state-of-the-art LLMs is fundamentally limited: If they need to complete every step correctly in order to solve a task, after a certain number of steps they will almost surely fail as a result of an underlying propensity to make errors, even when the answer should be obvious. While an error rate of 1-in-1,000 seems low, and would be great on a traditional LLM benchmark, on a task that requires successful execution of thousands of steps in a row, such a system results in inevitable failure.

Two critiques of Towers of Hanoi as a benchmark should be addressed upfront. First, one could argue that it is not an ideal LLM task since one could write code to solve the problem, and optimal algorithms are known [45]. True, but producing solutions is not the point: instead, the domain provides an ideal testbed for investigating the capacity of LLM-based systems to scale their inherent intelligence to increasingly large numbers of steps. Second, one could argue that this problem is too hard, since large real-world tasks might allow for a handful of errors without catastrophic results [50]. However, focusing on a case where no error can be tolerated forces us to pursue the elimination of any kind of error that is likely to arise on a long timescale, and this focus can lead to insights and practical methods that might otherwise be overlooked. There also are real-world safety-critical systems where no error can be tolerated [51]. Therefore, as LLM-based systems become ubiquitous in real-world decision making processes, it is essential these systems can reliably make decisions without error. Thus, the problem provides an ideal testbed for developing techniques that will be critical to scaling LLM-based systems to one million steps and beyond.

3 Methods

MAKER involves three main ingredients (Figure 2): (1) Decomposing a task into the smallest possible subtasks; (2) exploiting the modularity of such a decomposition to implement efficient error correction; and (3) “red-flagging” LLM outputs, i.e., discarding outputs whose structure suggests increased risk of errors, particularly correlated errors. These three components are detailed in the next three subsections. Together, they make it possible to efficiently increase the probability of success across all steps to a level such that the entire process is likely to succeed.

3.1 Maximal Agentic Decomposition

In a long-horizon agentic task with s steps, the goal of an LLM-based system is to produce a sequence of actions a_1, \dots, a_s that yields a target output y given the initial input x [11]. This paper is concerned with the following question: *How does the decomposition of the task into subtasks affect its solvability?*

The s -step task can be decomposed into subtasks, with the granularity of the decomposition defined by the number of steps m per subtask. Subtasks can then be solved by separate calls to LLM *agents*, where a templating function ϕ maps the input and specification of a subtask to a prompt for an LLM M , an extractor ψ_a parses actions from the LLM’s output response r , and a second extractor ψ_x parses information from r to include in the input to the next subtask. Let $x_0 = x$. A solution to the full task can then be sampled recursively:

$$r_{i+1} \sim M(\phi(x_i)), \quad (1)$$

$$a_{mi+1}, \dots, a_{mi+m} = \psi_a(r_{i+1}), \quad (2)$$

$$x_{i+1} = \psi_x(r_{i+1}) \quad \forall i = 0, \dots, \frac{s}{m} - 1. \quad (3)$$

Of particular interest are the two extreme cases: the case of no decomposition, i.e., $m = s$, termed *single-agent*:

$$a_1, \dots, a_s \sim (\psi_a \circ M \circ \phi)(x); \quad (4)$$

and the case of *maximal agentic decomposition* (MAD), i.e., $m = 1$:

$$r_{i+1} \sim M(\phi(x_i)), \quad (5)$$

$$a_{i+1} = \psi_a(r_{i+1}), \quad (6)$$

$$x_{i+1} = \psi_x(r_{i+1}) \quad \forall i = 0, \dots, s - 1. \quad (7)$$

Because LLMs are auto-regressive, when generating the i th action, a single agent M is increasingly burdened by the context produced in generating actions a_1, \dots, a_{i-1} . Therefore, as the context increases, its outputs become increasingly unreliable [52]. However with MAD, an agent’s context is limited to an amount of information sufficient to execute its single assigned step, allowing it to focus on its assigned role and avoid confusion that can creep in from irrelevant context. This focus also allows the use of smaller LLMs with more limited context sizes.

One might argue that this decomposition might improve the reliability of any given LLM call, but by decomposing the task into s independent calls, there are now s possible points of failure, instead of just one. That is, there are s opportunities for a weakest link to compromise the entire system, since for a correct action sequence a_1^*, \dots, a_s^* , the probability of generating it without error is exponentially decaying as the number of steps increases:

$$p(a_1^*, \dots, a_s^*) = \prod_{i=0}^{s-1} p((\psi_a \circ M \circ \phi)(x_i) = a_{i+1}^*). \quad (8)$$

First, note that a single long LLM call also suffers from a form of exponentially decaying probability of correctness [10]. Second, and more importantly, the modularity induced through maximal decomposition allows for a form of effective and efficient error mitigation and unreliability detection (“red-flagging”) that is not possible with a single large call. These capabilities will be described in the next two subsections.

3.2 First-to-ahead-by- k Voting and Scaling Laws

For simplicity, the error correction in this paper uses the statistical power of independent samples from a stochastic process (here an LLM). To determine a winner from these samples, a *first-to-ahead-by- k* voting process is used, motivated by the optimality of such an approach in the sequential probability ratio test (SPRT) [53, 54]. Many improvements are possible beyond this first implementation. For example, in the experiments in this paper, exact matches between actions are required, but in general, a classification function could be used to determine semantically equivalent outputs (e.g., implemented by an LLM, see Section 5).

Concretely, given an LLM M , candidate samples are drawn for a subtask (Eq. 2 & 3) until one has been sampled k times more than any other (Alg. 2). This process is a generalization of the classic gambler’s ruin problem [55], but with simultaneous dependent races between all pairs of candidates [56]. Since there is no known closed form for this general case, the analysis is simplified by assuming the worst case, i.e., that a correct candidate with probability p races against a single alternative with probability $1 - p$. If $p > 0.5$, the probability that the correct candidate gets selected through this process is

$$p(a_i = a^*) = \frac{1 - \left(\frac{1-p}{p}\right)^k}{1 - \left(\frac{1-p}{p}\right)^{2k}} = \frac{p^k}{p^k + (1-p)^k} = \frac{1}{1 + \left(\frac{1-p}{p}\right)^k}, \quad (9)$$

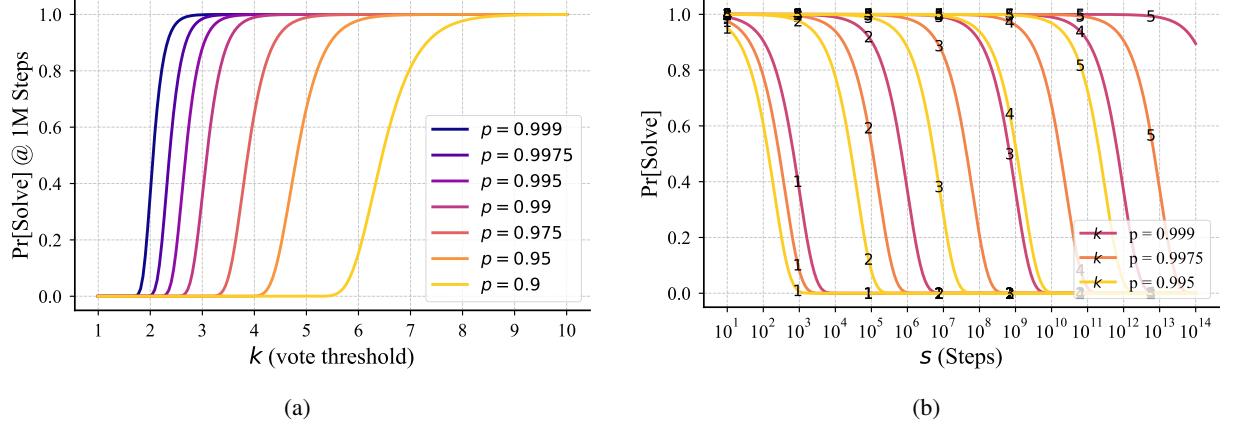


Figure 3: *MAKER* error-free solve rate scaling laws resulting from Eq. 13. (a) For a task with one million steps, MAKER, with first-to-ahead-by- k error correction enables high probability zero-error solutions for practical values of k , even as the base per-step error rate approaches 1-in-10; (b) For the lower per-step error rates, in theory even a low k allows scaling far beyond one million steps.

and there exists some k such that this voting process will result in the correct candidate winning with probability $1 - \epsilon$, for any given error rate $\epsilon \in (0, 1)$.

Now, suppose that the task requires s total steps to complete, with an inherent per-step success rate p , a decomposition level given by the number of steps per subtask m , and suppose that a margin of k votes is required to decide an action for each subtask. Again, assume that the correct solution for each subtask races against only a single most-likely alternative. Note that this assumption is much more favorable to larger values of m , i.e., less decomposition, since a most likely alternative captures a vanishing proportion of the total alternative (incorrect) probability mass as m grows. Let p_{vote} be the probability of sampling a correct vote for a subtask, p_{alt} the probability of sampling the alternative, p_{sub} the probability that the voting procedure succeeds on a subtask, and p_{full} the probability that it succeeds on all subtasks, i.e., that the full task is completed successfully. Then,

$$p_{\text{vote}} = p^m, \quad (10)$$

$$p_{\text{alt}} = (1 - p)p^{m-1}, \quad (11)$$

$$p_{\text{sub}} = \frac{p_{\text{vote}}^k}{p_{\text{vote}}^k + p_{\text{alt}}^k} = \frac{p^{mk}}{p^{mk} + ((1 - p)p^{m-1})^k} = \frac{1}{1 + \left(\frac{1-p}{p}\right)^k}, \quad (12)$$

$$p_{\text{full}} = p_{\text{sub}}^{\frac{s}{m}} = \left(1 + \left(\frac{1-p}{p}\right)^k\right)^{-\frac{s}{m}}, \quad (13)$$

where Eq. 12 comes from plugging p_{vote} and p_{alt} into the hitting probability formula for gambler's ruin [56]. Figure 3 uses Eq. 13 to illustrate how a high probability of overall success p_{full} can be maintained by increasing k in the case of $m = 1$, i.e. in a maximal decomposition.

Given Eq. 13, the expected cost of solving the entire task with a given level of reliability, i.e., given a target probability of overall success t , can be computed. First, the minimal k that yields success probability of at least t is

$$k_{\min} = \left\lceil \frac{\ln(t^{-m/s} - 1)}{\ln\left(\frac{1-p}{p}\right)} \right\rceil = \Theta(\ln s). \quad (14)$$

The detailed derivation is included in Appendix B. Notably, k_{\min} grows logarithmically with s no matter the decomposition level. Figure 4a shows how k_{\min} scales with the number of steps when using MAD, i.e., when $m = 1$.

It is now possible to write down the expected cost in terms of calls to LLM primitives, i.e., perform AALPs analysis [6]. Let c be the cost of generating a response for a single step with LLM M . Assuming the cost of generating tokens scales linearly with the number of tokens (since this is how APIs are priced), the cost of an agent generating a sample for m steps is $c_{\text{sample}} = cm$. Let c_{vote} be the expected cost of sampling either a correct vote for a subtask or the alternative

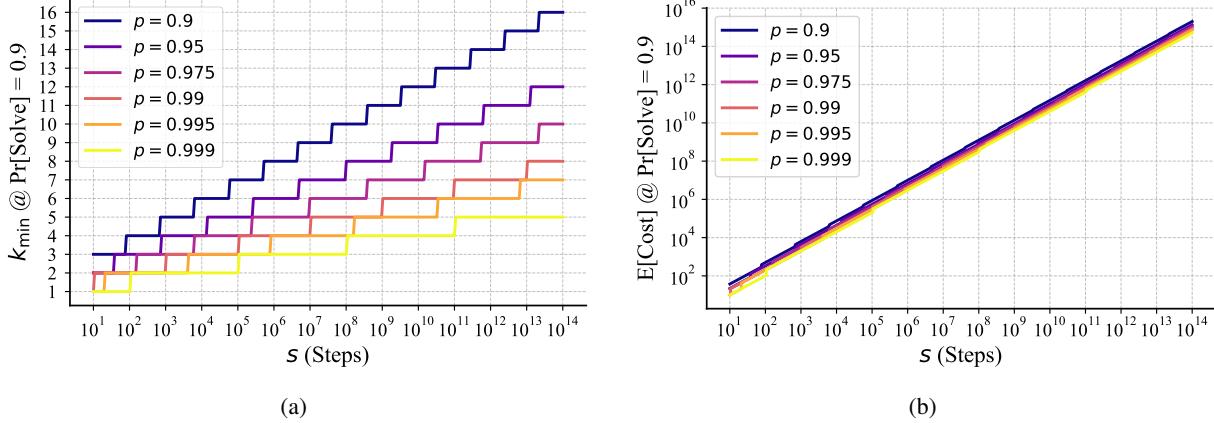


Figure 4: *MAKER cost scaling laws resulting from Eqs. 14 and 18.* (a) The value of k required in first-to-ahead-by- k voting to maintain a 0.9 solution probability for the full task increases logarithmically with the number of steps in the task; (b) The corresponding expected cost of running the system increases log-linearly. These plots illustrate the scalability of MAKER, in theory, to millions of steps and beyond.

against which it races, c_{sub} the expected cost of completing a subtask, and c_{full} the expected cost of completing the full task. Then,

$$c_{\text{vote}} = \frac{c_{\text{sample}}}{p_{\text{vote}} + p_{\text{alt}}} = \frac{cm}{p^m + (1-p)p^{m-1}} = \frac{cm}{p^{m-1}}, \quad (15)$$

$$c_{\text{sub}} = c_{\text{vote}} \cdot \frac{2k_{\min}p_{\text{sub}} - k_{\min}}{2p - 1} = \frac{cm}{p^{m-1}} \cdot \frac{2k_{\min} \left(1 + \left(\frac{1-p}{p}\right)^{k_{\min}}\right)^{-1} - k_{\min}}{2p - 1} \approx \frac{cmk_{\min}}{p^{m-1}(2p - 1)}, \quad (16)$$

$$c_{\text{full}} = \frac{s}{m} \cdot c_{\text{sub}} = \frac{csk_{\min} \left(2 \left(1 + \left(\frac{1-p}{p}\right)^{k_{\min}}\right)^{-1} - 1\right)}{p^{m-1}(2p - 1)} \approx \frac{csk_{\min}}{p^{m-1}(2p - 1)} = \Theta(p^{-m}cs \ln s), \quad (17)$$

where Eq. 16 comes from plugging Eq. 12 into the hitting time for gambler’s ruin [55], Eq. 17 comes from multiplying by the number of subtasks, and the approximation holds when $p_{\text{sub}} \approx 1$, i.e., when the error tolerance is low. Notably, the cost grows exponentially with m . Figure 5 illustrates this phenomenon. As the number of meaningful decisions assigned to an agent grows, the chance that its sequence of decisions will match exactly across multiple samples vanishes. In contrast, in the MAD case, the system scales log-linearly with s :

$$\mathbb{E}[\text{cost of solving full task}; m = 1] = \Theta(p^{-1}cs \ln s) = \Theta(s \ln s), \quad (18)$$

when p , c , and t are held constant. Figure 4b illustrates this efficient scaling. The discovery of algorithms that scale log-linearly has been critical to the scalability in classical computing [57]. This result is therefore encouraging: It shows the potential of LLM-based systems to scale in a similar manner, increasing their reliability to a point where it is possible to trust them to complete long-running tasks. Furthermore, the $\Theta(\ln s)$ factor in Eq. 18 corresponds to the number of votes required per step, which in practice can be parallelized across $\Theta(\ln s)$ processes. So, the time cost of the parallelized system scales only linearly with s .

Although Eq. 18 shows how MAD scales efficiently as the number of steps increases, in practice, the model cost c and per-step success rate p will have a major impact. Different LLMs will have different costs and different inherent error rates. Solving a task with a large number of steps will incur a meaningful e.g. economic cost, so before running on the full task, an LLM M such that c/p is minimized should be selected. In other words, Eq. 18 makes it possible to select an LLM that will be most cost-effective at scale, and, since each individual step is so small, it is likely that smaller LLMs will be sufficient to solve the task.

3.3 Red-Flagging: Recognizing Signs of Unreliability

Since p plays such an important role in the cost of the system, when possible, it is worth taking practical measures to push it higher. The simple premise is that “bad” behaviors are correlated in LLMs, so if an LLM produces a response

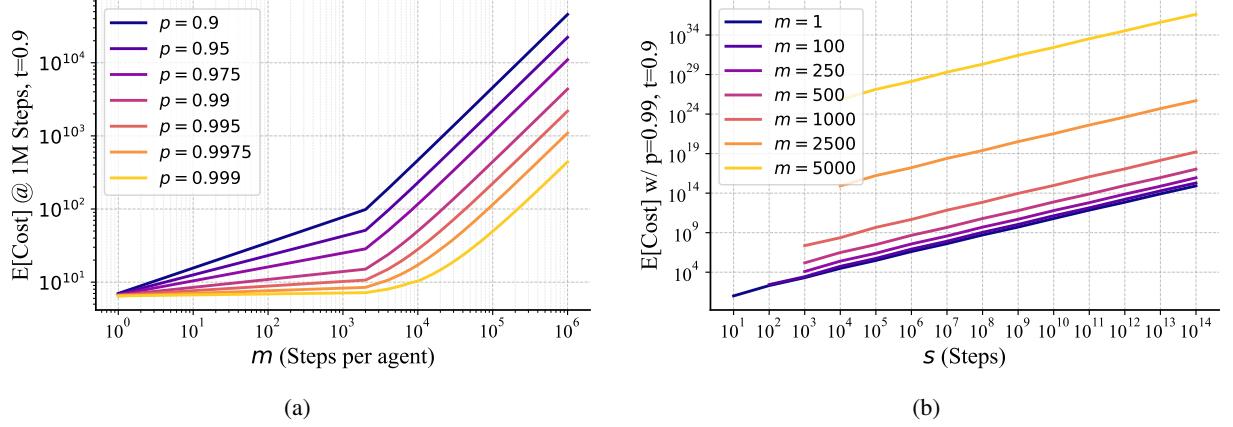


Figure 5: *Task decomposition scaling laws resulting from Eq. 17.* (a) For a task with 1M steps, as the number of steps assigned to each agent increases (and thus the number of agents decreases), there is an exponential increase in the expected cost to complete the task with sufficient reliability. Notice that while the x-axis is $\log(\cdot)$ scale, the y-axis is $\log(\log(\cdot))$ scale. (b) As the size of the task scales, this pattern continues: setups where agents are assigned more steps incur orders-of-magnitude of additional cost.

that signals pathological behavior, the response should be flagged and simply discarded. Since in MAD each agent is responsible for only a single step, each step is not too expensive, and it can be discarded and a new action resampled (i.e., by making another agent call). In this paper, two signs of unreliability are used as red flags: (1) overly long responses, and (2) incorrectly formatted responses. The hypothesis is that not only will discarding flagged responses increase p , it will also meaningfully reduce *correlated errors*, since both flag types indicate that the LLM has been conditioned into a strange starting point before sampling. It is simple to detect these signs and discard their responses, but, since the paper focuses on understanding the impact on the expected cost of highly scaled long-horizon tasks, it is worth elaborating on the motivation and impact of this implementation choice.

Consistent with observations in prior work that longer answers tend to have more errors [58], preliminary experiments for this paper showed that once an LLM gets initially confused, it can go off the rails and over-analyze a situation in a cycle of self-destruction. In MAD, each agent’s role is highly focused and relatively simple; if an agent is doing too much work to figure out its answer, it is likely to be confused and missing the point, and therefore more likely to give an incorrect answer. Even if the success rate is increased only from 99% to 99.9%, such an increase can have a large impact when the number of steps is large.

Similarly, preliminary experiments showed that when an agent produces an answer in an incorrect format, it is more likely to have become confused at some point on the way to that answer. So, instead of trying to fix the format of the answer in some way, the detection of the incorrect format can be flagged and the sample discarded.

Experimental evidence for the above two phenomena is detailed in Section 4. Formally, if v is the probability that a valid response is parsed from the LLM’s output, i.e., no red flags, then the expected cost of MAKER can be written as:

$$\mathbb{E}[\text{cost of MAKER}] \approx \frac{cs k_{\min}}{v(2p - 1)} = \Theta\left(\frac{cs \ln s}{vp}\right), \quad (19)$$

where p now indicates the per-step success rate *given that the response is valid*. In practice, this formula can be used to decide the trade-off between incorporating more red flags to increase p (potentially resulting in a lower k_{\min} , whose calculation depends on p but not v) and the incurred cost overhead of discarded samples.

The most straightforward approach is to estimate p on a relatively small number of steps to determine the choice of model and red flags, i.e., c and v , as well as the value of k , before running the system on the full task with s steps. An example application of this approach is demonstrated in the next section.

4 Experiments

This section details the application of MAKER to solving the Towers of Hanoi problem with 20 disks, i.e. over one million LLM steps with zero errors. First, the experimental setup is described (Section 4.1). Next, single-step error-rates are estimated (Section 4.2), which are used to project the cost of different setups (Section 4.3). Then, a selected setup

is run and evaluated on the full task (Section 4.4). Finally, the impacts of red-flagging are investigated (Section 4.5). All in all, these experiments validate the components and scalability of the MAKER implementation of the MDAP framework.

4.1 Setup

The implementation of MAKER for this problem was derived from the single-agent approach introduced in prior work [4]. The single-agent prompts were modified so that each agent knows that it must only perform a single step of the problem, i.e., to move a single disk.

For efficiency, and to focus the agents as much as possible, each agent is given the minimal context it needs to perform its single step. In the case of Tower of Hanoi, everything the agent needs to know is the overall strategy and the current state of the problem, i.e., the configuration of disks. As in prior work [11, 4], the overall strategy is provided in the prompt for each agent. The strategy works for any even number of disks and is the one most often suggested by the LLMs themselves when no strategy is provided a priori. (Appendix C). This design choice effectively isolates the ability of agents to *execute* clear instructions from the ability of LLMs to have *insights* about how tasks should be solved (Section 5; see also [11]). Both insight and execution are essential to the capabilities of LLMs, but often they are entangled in experiments, making it difficult to identify the source of failure. Focusing on execution makes it possible to pursue the goal of finding minimal conditions for scaling LLM systems with respect to the number of steps.

The full agent prompt template is given in Appendix C. Given the current state (i.e., configuration of disks) and prior move (which disk was moved from where to where), each agent is asked to provide the next move and the resulting next state. Unlike in the single-agent case, where only the sequence of moves needs to be produced, in the MAD case each agent must produce the resulting state, since this is critical information to be fed to the next agent. Each agent is asked to format its answers as “move = <move>” and “next_state = <next_state>”. Superficially, the requirement to produce the next state along with the action creates even more potential points of failure beyond the single-agent case, but, as it turns out, any drawbacks are overcome by the advantages of extreme decomposition and error correction.

4.2 Estimating single-step success rates

Running an LLM-based system at the scale of a million steps is expensive. It is thus desirable to calibrate the parameters of the system and estimate the success rate and cost before any large experiments are run. Equation 18 provides such a calibration and estimation method. Key to this estimation is the per-step success rate p , which depends on the underlying LLM used.

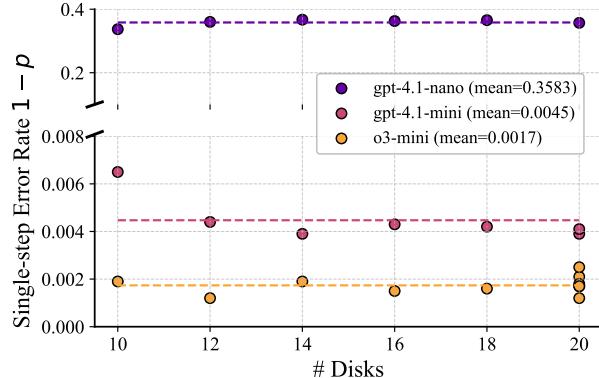
A straightforward way to estimate the per-step success rate is to run the system on a random subset of steps. One advantage of having the same strategy in the prompt of each agent is that the correct answer is known for each step, and, assuming every prior step is correct, the inputs are known as well. This knowledge also makes it possible to use the API to obtain a batch of answers for many steps asynchronously, thus greatly speeding up wall-clock time of experiments and reducing cost [59]. These properties are part of what makes Towers of Hanoi such a practical testbed for many-step methods. For other problems, it may not be possible to estimate p as efficiently, but, in any case, it should be possible to estimate it to a practical degree.

These initial exploratory estimation experiments were run without red-flagging. That is, agents were given a maximum of 2048 output tokens as an initial conservative upper bound, ensuring they have plenty of space to express whatever answer they need to express. They also used a “repairing parser” (written by an LLM, Appendix C) that attempted to correct some of the more common formatting errors in order to extract the LLM’s intended answer reliably out of its output. Importantly, since these experiments focus on the inherent generative reasoning capabilities of LLMs, they do not include LLMs that have access to auxiliary tools.

Figure 6a shows the single-step error rates across various LLMs as the number of disks is increased. There are two important and perhaps surprising takeaways from this figure: (1) Different LLMs have different base error rates, but those of relatively small non-reasoning models are comparable to more advanced reasoning models, suggesting that non-reasoning models may be a more effective fit for long-range tasks with MAKER. Figure 6b shows that this difference cannot be explained by a difference in output tokens, since the models use a similar number of tokens. (2) The per-step error rate is remarkably stable as the number of disks increases, a highly encouraging sign that MAKER will enable scaling to a large number of steps without the kind of exploding error rates often seen with single agents.

4.3 Projecting the cost of error correction

Based on the single-step error rates estimated above, it is possible to estimate the cost of successfully solving the full 20-disk task for models with $p > 0.5$, i.e., models for which voting converges to correct subtask answers as k increases.



(a)

Model	\$/M	tok	$1 - p$	k_{\min}	$\mathbb{E}[\text{cost}]$
gpt-4.1-nano	0.4	842	.3571	29	\$41.9K
gpt-4.1-mini ($\tau = 1.0$)	1.6	580	.0040	4	\$4.9K
gpt-4.1-mini ($\tau = 0.1$)	1.6	538	.0022	3	\$3.5K
o3-mini (low)	4.4	535	.0018	3	\$9.4K
haiku-4.5	5.0	588	.1839	12	\$71.2K
llama-3.2-3B	0.06	434	1.0	-	-
gpt-oss-20B	0.2	1104	.0358	6	\$1.7K
qwen-3	0.6	449	.2342	15	\$11.5K
deepseek-v3.1	1.7	1004	.0569	6	\$14.6K
kimi-k2	3.0	925	.0393	6	\$22.9K

(b)

Figure 6: Empirical estimates of single-step error rates across models. (a) Different models have different per-step error rates, but these rates do not notably decrease as the number of disks (log of the solution length) increases, which is an encouraging sign for the ability of the system to scale. (b) Given the per-token API cost of each model, the error rate estimate, and the mean number of output tokens, Eq. 18 can be used to estimate the cost of running the full 20-disk experiment with $t = 0.95$. Among the proprietary models, although gpt-4.1-nano has the cheapest per-token cost, and o3-mini has the lowest per-step error rate, the expected cost of gpt-4.1-mini (with low temperature) is by far the lowest, and gpt-oss-20B is the clear open-source choice. In this manner, using Eq. 18 to estimate cost can lead to substantial savings.

The table in Figure 6b shows these estimates, given the per-token cost of the model, the mean number of output tokens per step, and the empirical estimates of p and k_{\min} . For a given model, its estimated cost per sample (Section 3.2) is:

$$c_{\text{sample}} = c_{\text{in}} \cdot n_{\text{in}} + c_{\text{out}} \cdot n_{\text{out}}, \quad (20)$$

where c_{in} and c_{out} are the cost of input and output tokens, respectively, and n_{in} and n_{out} are the estimated average number of input and output tokens per sample. The cost per sample is multiplied by the expected number of samples to get the overall cost. For the target probability of success $t = 0.95$, these results make it clear that the most effective proprietary choice is gpt-4.1-mini, and the most effective open-source choice is gpt-oss-20B. Whether these estimates are useful depends on whether the errors are sufficiently decorrelated across runs. Indeed they appear to be: In two independent runs of gpt-4.1-mini ($\tau = 0.1$) on the set of 10K random samples, there were zero steps that had errors in both runs. Decorrelation of errors is discussed further in Sections 4.5 and 5.

Although the projected cost of gpt-oss-20B is lowest overall, in the scale-up experiment, gpt-4.1-mini was used because the OpenAI API was found to be more reliable than that of together.ai (a critical factor for experiments requiring millions of agentic calls). Note also that for reasoning LLMs with higher per-token cost than those in Figure 6b, even if their estimated per-step error rate were zero, and therefore $k_{\min} = 1$, their overall cost would be higher than that of gpt-4.1-mini, simply due to the cost of running on all $2^{20} - 1$ steps, and it would be practical to assume the error estimate is a lower bound and set $k = 2$. Thus, using the non-reasoning gpt-4.1-mini saves thousands of dollars in the scale-up experiment compared to the other proprietary models.

4.4 Solving the 20-disk problem: Over one million steps with zero errors

With gpt-4.1-mini as the base model, the maximum output token threshold was set to 750, and a red-flagging output parser was used to enforce the basic formatting requirements. Beyond this cutoff, the model often found itself in confusing logical loops; misformatted output was often correlated with such twisted reasoning, allowing them to be detected and discarded (as will be discussed further in Section 4.5). Since $k_{\min} = 3$, at least three responses were generated in parallel for each step. The first step was generated with a temperature of 0 to make sure the best possible guess is included in the votes. For all subsequent votes, a temperature of 0.1 was used.

With these calibrations, the full system solved the problem perfectly. Figure 7 shows a snapshot of an animation visualizing the movement of the disks through the million steps as well as the sequence of agent activations that accomplish it. This result constitutes the first successful solution of a task with one million LLM steps with zero errors, establishing that scaling LLM-based systems to large time horizons is possible.

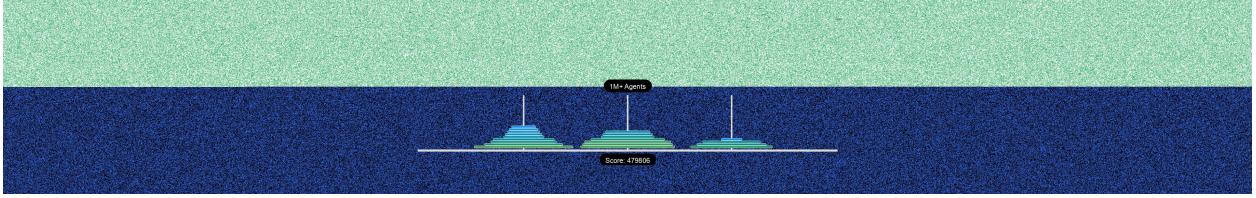


Figure 7: A snapshot of the million-step problem-solving process. The dots in the background represent the steps, ordered in rows from top to bottom. At step 479,806, MAKER is near the middle of the million steps, and about a half of the agents have already executed (green) and about a half are waiting to do so (blue). An animation of this process can be seen at www.youtube.com/watch?v=gLkehsQy4H4.

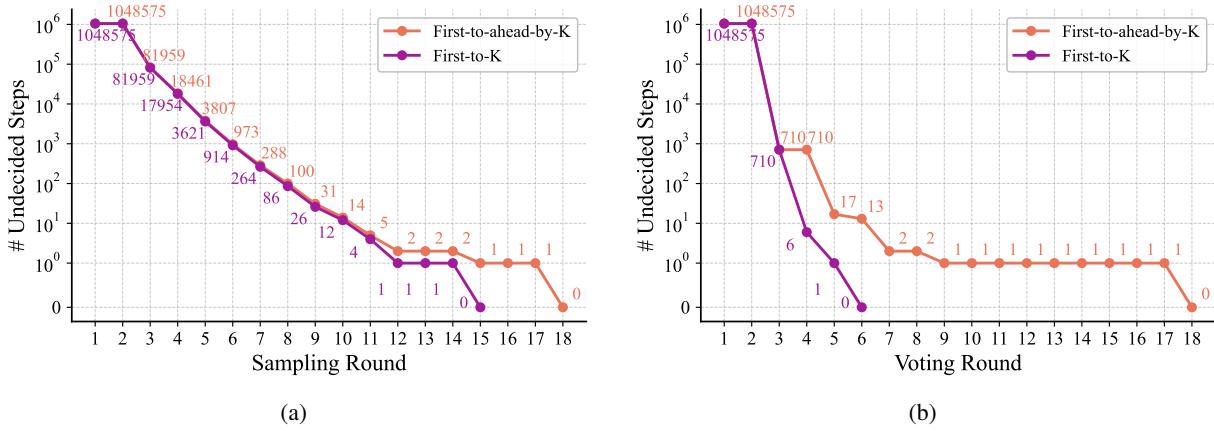


Figure 8: Convergence to zero-error solution. The number of undecided steps decreases with sampling round (a) and voting round (b). In both cases, as expected from the theory, after the first $k = 3$ rounds, there is a steady exponential decrease in the number of undecided steps, finally resulting in zero undecided steps and with zero errors. This sharp exponential convergence means that the vast majority of the overall cost is incurred in the first k rounds of sampling; the cost of completing the remaining steps is effectively a rounding error. This effect emerges when p is sufficiently high. It may at first appear disconcerting that there are any steps at all that require more than five voting rounds, but the decorrelation of errors is sufficient to prevent the voting mechanism from being overwhelmed (as described in Section 4.5). Details on specific steps, including the one pathological step that took 18 rounds, are in Appendix D.

The behavior of the process can be analyzed by looking at how many samples had to be drawn for each step and how many votes were required (i.e., how many valid samples after red-flagging). A step is considered *undecided* after i rounds if an action decision has not yet been made based on the voting rule. Figure 8 shows how the number of undecided steps decreases with the number of sampling rounds (including invalid samples) and voting rounds (excluding invalid samples). The exponential decay in the number of undecided steps mirrors the theoretical prediction. Due to this exponential decay, the vast majority of LLM calls (and therefore cost) is spent in the first k calls, while the remaining cost is, for practical purposes, a rounding error. Notably, the task is solved perfectly even when using a less statistically powerful *first-to-k* voting (i.e., the first candidate k votes wins), illustrating the robustness of the approach.

Although the system completes with zero errors and as efficiently as expected, the fact that a few of the steps take notably more sampling and voting rounds than others could be cause for concern. The next section looks into how red-flagging reduces the negative impact of such correlated errors.

4.5 Investigating the impact of red-flagging

Red-flagging was hypothesized to reduce the per-step error rate, but also the impact of correlated errors, i.e., particular steps that have unusually high error rates compared to the average step. Figure 9 shows evidence for both of these phenomena; however, the impact on correlated errors turns out to be a much more important effect. In the first two rounds of voting, the max number of output tokens (when calling the API) was set to 2048 to enable this analysis. Figure 9a shows that the per-step error rate increases precipitously once the response length crosses about 700. Although p past this threshold is still around 90%, this is a drastic degradation compared to error rates on the order of 1-in-1000.

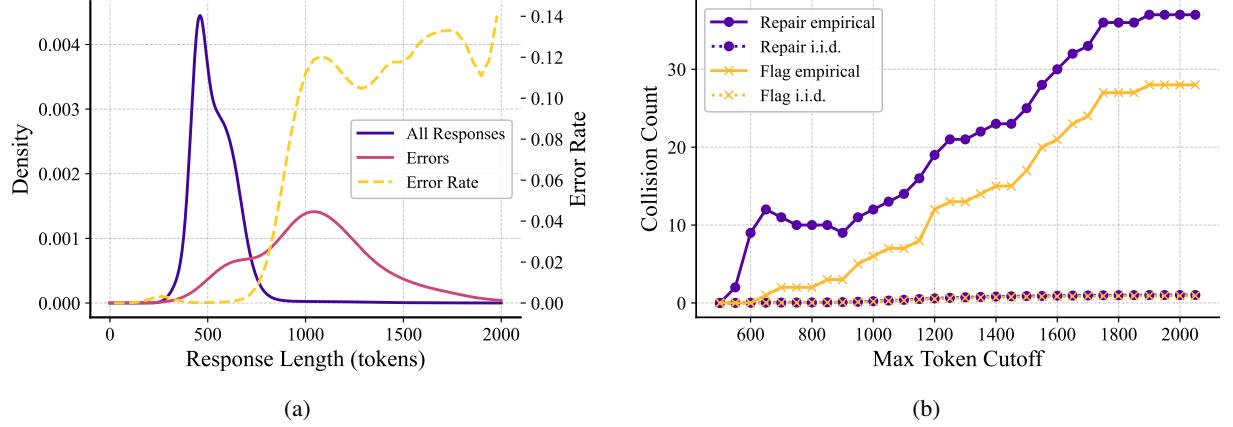


Figure 9: *Impact of red-flagging on reducing errors.* (a) The error rate increases precipitously once the response length crosses about 700. However, since so few of the overall responses are overly long, the overall error rate at higher max token thresholds is not much larger, i.e., not large enough to induce an increase in k_{\min} . (b) However, when focusing on correlated errors, the advantage of red-flagging becomes clear: Moving from a ‘helpful’ repairing output parser to one that discards samples with any formatting issues leads to lower collision counts (i.e., number of steps whose first two votes are incorrect). These results confirm that robust decorrelation is crucial in many-step tasks, and that red-flagging helps with correlated errors.

with shorter responses. Even so, since so few of the overall responses are overly long, the overall p at higher max token thresholds is not much larger, and in particular, not large enough to induce an increase in k_{\min} .

However, the main benefit of red-flagging becomes clear in Figure 9b. This figure plots the number of collisions across the first two votes of all steps in the 20-disk experiment, i.e., how many steps have *both* votes incorrect. The number of collisions is computed across max token cutoffs for both the repairing parser used in Section 4.2, as well as the red-flagging parser used in Section 4.4. Assuming the steps are i.i.d. with a uniform success rate, the expected number of collisions is no more than one or two in all cases. However, with a high max token cutoff the observed number of collisions is much higher, especially with the repairing parser. Red-flagging successfully reduces some of these correlated errors and may be critical to the success of the method on many-step tasks. Appendix D illustrates what correlated errors in this domain can look like, and Section 5 discusses techniques for reducing correlation.

5 Discussion and Future Work

This paper introduced a framework for massively decomposed agentic processes (MDAPs) that can reliably solve tasks with large numbers of steps, as well as the first implementation of this framework, MAKER, which was successfully applied to the Towers of Hanoi benchmark task. This initial study established core principles that open many directions for future work.

More General Applications LLM behaviors can be divided into two categories: *insights* and *execution*. Insights come from an LLM creatively generating ideas, plans, and strategies, while execution involves following through with them. This paper focused on execution: the overall strategy to solve the problem is set at the beginning of the process, and given this strategy, the answer to each subtask is clearly achievable. Extending the framework to handle LLM-based insights is an important area of future work, since insights are inherently more open-ended and may come with irreducible step-wise uncertainty. One concrete way is to apply MAKER to the case where the creation of each subtask is itself treated as a step in an overall decomposition. The goal is to automate the entire problem-solving pipeline: the task is decomposed into minimal chunks, each one is solved, and the results are aggregated into a complete solution. The framework needs to be extended to handle an unknown number of total steps, as well as steps of different types, different underlying success rates, inexact matches between insight steps, and possible failures of the matching process.

Preliminary experiments in this direction are promising (Appendix F). A more general version of MAKER was created with four agent types: decomposition agents, called recursively to break a task into two simpler sub-tasks and a composition function; decomposition discriminator agents, called to vote (with first-to- k voting) for one of $n = 2k - 1$ decomposition candidates; solution discriminator agents, called to vote for one of n composition candidates; and

problem solver agents used to solve minimal subtasks without decomposing them. The system achieved promising results on large-digit multiplication, a notoriously difficult task for transformer-based models [10, 60]. Future work will investigate the broader potential of such a system.

For simplicity, in this paper, all MAKER agents used the same underlying LLM and their prompts only differed in the subtask they were assigned. More general systems will likely require different LLMs for different kinds of roles, and a general increased diversity across agents. One of the benefits of such diversity will be decorrelation of errors, as is discussed next.

The Importance of Decorrelated Errors For clarity and simplicity, theoretical analysis in this paper assumed that the errors are i.i.d. across steps. This assumption was reasonable because the steps were relatively uniform. Even so, there were a few steps that, for no apparent reason, had substantially higher inherent error rates than others. Such strange behaviors for particular inputs are well-known side-effects of LLM training, and dealing with such correlated errors is an open foundational problem in machine learning [61].

The independent sampling plus red-flagging method used in this paper was sufficient to overcome them, but there may be other real-world cases where more sophisticated decorrelation methods are required. For example, instead of simple resampling using temperature, paraphrasing the prompt [62] or adding noise to the prompt in some other way could help avoid such anomalous states caused by a particular fixed context. The error rate of each step would then approach the true ability of the LLM to understand and execute that step. Further, the framework could be extended to account for different values of p for different steps, and decorrelation methods that make sampling more effective. Such extensions are critical to make the framework more broadly applicable, since in a long-range task, even a single step with an abnormally high error rate can cause the reasoning process to fail.

Parallels with microservices Parallels can be drawn between microagents and microservices. The benefits of decomposing a monolithic agent’s task into subtasks are similar to those of decomposing a monolithic application into smaller services [63]:

- **Modularity:** Each microagent can be tailored to a specific task and leverage the right tools for the job.
- **Data management:** Each microagent is responsible for managing its data.
- **Independent development:** Microagents can be updated and tested in isolation from the rest of the system.
- **Scalability:** Microagents can be scaled independently, adjusting the resources to the actual needs of the system.
- **Communication:** Natural language is a powerful, well-understood communication protocol.
- **Complexity:** As microservices solve for large-scale systems, microagents solve for complex reasoning tasks.
- **Real-time monitoring:** Microagents can be monitored in real-time.
- **Design for failure:** Microagents are designed to tolerate the failure of any of the agents.
- **Evolutionary design:** Change is easier to manage with microagents than with a monolithic agent.

In fact, microagents could be considered a natural evolution of microservices. The framework could be extended in that direction, leveraging the lessons learned from microservice architectures [64].

Limits of Decomposition The application of MAKER assumes a task can be decomposed into small enough and simple enough steps such that each step can be solved by an LLM agent with reasonable probability. There is thus one central question that will dictate how broadly the methods can be applied: Are there important problems where such a decomposition is not possible or is computationally infeasible to discover? At the lowest level of LLM implementation, there is a decomposition into primitive operations executed on CPUs or GPUs; one can hope that there is some decomposition between that and the entire problem that is still linguistic but effectively compartmentalizes context and different behaviors. It remains to be seen which kinds of tasks are most resistant to such a decomposition.

Safety, Morality, and the Future of Superintelligence If large and important real-world problems can be successfully decomposed into microsteps, there could be major benefits with respect to safety. If each step has a clearly defined and limited focus and purpose, the LLM’s view of the world and domain of influence can be strictly limited, allowing for more effective sand-boxing, auditing, and general control. Multiple focused agents can be run independently on each step, which also substantially reduces the ability of agents to collude to produce harmful actions. As was seen in the experiments in this paper, the vast majority of work can be performed by relatively small LLMs that are capable of handling these small steps, thus avoiding the risks of harmful behaviors that can arise in more powerful models

[65]. In other words, it could help mitigate the risk of uncontrollable superintelligence. Complementing these reduced societal risks, extreme decomposition could reduce the chance of unintended suffering of the machines themselves, as model welfare has become an increased area of concern [66]. As argued in prior work, relying on smaller models and having models focus entirely on limited-scope subtasks, as is done through decomposition, could reduce the chance that sentience unintentionally emerges [6, 67].

LLMs today have just about all the raw intelligence needed to scaffold them into the great superintelligent skyscrapers of the coming age, and to scaffold themselves into productive organizations of technological progress. MDAPs present an alternative path to realizing the benefits of superintelligence, which, compared to endlessly building bigger and smarter single-agent models, comes with substantially reduced risks to both humans and machines.

6 Conclusion

This paper focused on the question of how LLM-based agentic systems can be massively scaled. Decomposing tasks into minimal subtasks makes it possible to apply error-correction techniques effectively and efficiently, supporting scaling to millions of steps and beyond. A new category of AI systems results, i.e. massively decomposed agentic processes, or MDAPs. MAKER is a first implementation of this approach, and the experiments in this paper on Towers of Hanoi a first demonstration of its value. This foundation opens the door to more general-purpose implementations and large-scale, long-running real-world applications. Such MDAPs offer an alternative to building endlessly larger and more intelligent LLMs: By smashing intelligence into a million pieces, it is possible to build AI that is efficient, safe, and reliable.

References

- [1] Wikipedia contributors. Apple supply chain, 2025.
- [2] Apple Inc. Apple supplier list (covers ≈98% of direct spend), 2022. <https://www.apple.com/supplier-responsibility/pdf/Apple-Supplier-List.pdf>; Date Accessed: 2025-10-23.
- [3] Veg Patch Kitchen Cookery School. The stages of bread making, 2025. <https://vegpatchkitchen.co.uk/the-stages-of-bread-making/>, Date Accessed 11-11-2025.
- [4] 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.
- [5] Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. Multi-logieval: Towards evaluating multi-step logical reasoning ability of large language models. *arXiv preprint arXiv:2410.03117*, 2024.
- [6] Elliot Meyerson and Xin Qiu. Position: Scaling llm agents requires asymptotic analysis with llm primitives. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025.
- [7] Aram W Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549(7671):203–209, 2017.
- [8] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Kathryn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long tasks. *arXiv preprint arXiv:2503.14499*, 2025.
- [9] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *arXiv preprint arXiv:2304.15004*, 2023.
- [10] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, volume 36, pages 70293–70332, 2023.
- [11] Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in llms. *arXiv preprint arXiv:2110.09624*, 2025.
- [12] Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*, 2025.

- [13] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.
- [14] Long Wang et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 2024.
- [15] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial intelligence*, 49(1-3):361–395, 1991.
- [16] Eric Horvitz and John Breese. Ideal partition of resources for metareasoning. *arXiv preprint arXiv:2110.09624*, 2021.
- [17] Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. Jsonschemabench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*, 2025.
- [18] OpenAI. Introducing structured outputs in the api, 2024. <https://openai.com/index/introducing-structured-outputs-in-the-api/>; Date Accessed: 2025-10-23.
- [19] Aaron Grattafiori et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [20] Xin Qiu, Yulu Gan, Conor F Hayes, Qiyao Liang, Elliot Meyerson, Babak Hodjat, and Risto Miikkulainen. Evolution strategies at scale: Llm fine-tuning beyond reinforcement learning. *arXiv preprint arXiv:2509.24372*, 2025.
- [21] Xiang Chen, Zhixian Yang, and Xiaojun Wan. Relation-constrained decoding for text generation. *Advances in Neural Information Processing Systems*, 35:26804–26819, 2022.
- [22] Samuel Colvin, Eric Jolibois, Hasan Ramezani, Adrian Garcia Badaracco, Terrence Dorsey, David Montague, Serge Matveenko, Marcelo Trylesinski, Sydney Runkle, David Hewitt, Alex Hall, and Victorien Plot. Pydantic, October 2025.
- [23] Stefano Baccianella. Json repair - a python module to repair invalid json, commonly used to parse the output of llms, feb 2025.
- [24] Caleb Courier et al. Guardrails-AI, 2024. Apache 2.0 License.
- [25] George C Clark Jr and J Bibb Cain. *Error-correction coding for digital communications*. Springer Science & Business Media, 1981.
- [26] Shu Lin and Daniel J. Costello. *Error control coding: fundamentals and applications*. Pearson/Prentice Hall, Upper Saddle River, NJ, 2004.
- [27] Chin-Long Chen and MY Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM Journal of Research and development*, 28(2):124–134, 1984.
- [28] Joschka Roffe. Quantum error correction: an introductory guide. *Contemporary Physics*, 60(3):226–245, 2019.
- [29] Eugene Normand. Single event upset at ground level. *IEEE transactions on Nuclear Science*, 43(6):2742–2750, 1996.
- [30] Fan Wang and Vishwani D Agrawal. Single event upset: An embedded tutorial. In *21st International Conference on VLSI Design (VLSID 2008)*, pages 429–434. IEEE, 2008.
- [31] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012.
- [32] Sarah P Otto and Thomas Lenormand. Resolving the paradox of sex and recombination. *Nature Reviews Genetics*, 3(4):252–261, 2002.
- [33] Lisa M Abegglen, Aleah F Caulin, Ashley Chan, Kristy Lee, Rosann Robinson, Michael S Campbell, Wendy K Kiso, Dennis L Schmitt, Peter J Waddell, Srividya Bhaskara, et al. Potential mechanisms for cancer resistance in elephants and comparative cellular response to dna damage in humans. *Jama*, 314(17):1850–1860, 2015.
- [34] Yanxi Chen, Yaliang Li, Bolin Ding, and Jingren Zhou. On the design and analysis of llm-based algorithms. *arXiv preprint arXiv:2407.14788*, 2024.
- [35] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3,4):379–423, 623–656, 1948.
- [36] Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*, 2023.

- [37] Mingfeng Xue, Dayiheng Liu, Wenqiang Lei, Xingzhang Ren, Baosong Yang, Jun Xie, Yidan Zhang, Dezhong Peng, and Jiancheng Lv. Dynamic voting for efficient reasoning in large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, Singapore, December 2023. Association for Computational Linguistics.
- [38] Xin Qiu and Risto Miikkulainen. Semantic density: Uncertainty quantification for large language models through confidence measurement in semantic space. In *Advances in neural information processing systems*, volume 37, pages 134507–134533, 2024.
- [39] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.
- [40] Ibomoiye Domor Mienye and Yanxia Sun. A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *Ieee Access*, 10:99129–99149, 2022.
- [41] Mudasir A Ganaie, Minghui Hu, Ashwani Kumar Malik, Muhammad Tanveer, and Ponnuthurai N Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.
- [42] Fouad Trad and Ali Chehab. *To Ensemble or Not: Assessing Majority Voting Strategies for Phishing Detection with Large Language Models*, page 158–173. Springer Nature Switzerland, 2025.
- [43] Yujia Li, David Choi, Nando de Freitas, and et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.
- [44] 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.
- [45] Andreas M Hinz, Sandi Klavžar, Uroš Milutinović, and Ciril Petr. *The tower of Hanoi-Myths and maths*, volume 140. Springer, 2013.
- [46] Ivan Moscovich. *1,000 Playthinks: Puzzles, Paradoxes, Illusions and Games*. Workman Publishing, 2001.
- [47] Iñaki Dellibarda Varela, Pablo Romero-Sorozabal, Eduardo Rocon, and Manuel Cebrian. Rethinking the illusion of thinking. *arXiv preprint arXiv:2507.01231*, 2025.
- [48] Sheraz Khan, Subha Madhavan, and Kannan Natarajan. A comment on “the illusion of thinking”: Reframing the reasoning cliff as an agentic gap. *arXiv preprint arXiv:2506.18957*, 2025.
- [49] C Opus and A Lawsen. The illusion of the illusion of thinking. *arXiv preprint ArXiv:2506.09250*, 2025.
- [50] Herbert Alexander Simon. *Models of man: social and rational; mathematical essays on rational human behavior in society setting*. New York: Wiley, 1957.
- [51] Michael Kremer. The o-ring theory of economic development. *The quarterly journal of economics*, 108(3):551–575, 1993.
- [52] Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. Context length alone hurts llm performance despite perfect retrieval. *arXiv preprint arXiv:2510.05381*, 2025.
- [53] Abraham Wald. *Sequential analysis*. Courier Corporation, 2004.
- [54] Jaeyeon Lee, Quantong Qi, Matthew Brady Neeley, Zhandong Liu, and Hyun-Hwan Jeong. Consol: Sequential probability ratio testing to find consistent llm reasoning paths efficiently. *arXiv preprint arXiv:2503.17587*, 2025.
- [55] Jakob Bernoulli. *Ars conjectandi*. Impensis Thurnisiorum, fratrum, 1713.
- [56] Sheldon M Ross. First ahead by at least k multinomial game. *Annals of Operations Research*, pages 1–6, 2025.
- [57] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.
- [58] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.
- [59] OpenAI. Batch api. <https://platform.openai.com/docs/guides/batch> (Accessed 2025-11-07), 2025.
- [60] Xiaoyan Bai, Itamar Pres, Yuntian Deng, Chenhao Tan, Stuart Shieber, Fernanda Viégas, Martin Wattenberg, and Andrew Lee. Why can’t transformers learn multiplication? reverse-engineering reveals long-range dependency pitfalls. *arXiv preprint arXiv:2510.00184*, 2025.
- [61] Joel Lehman, Elliot Meyerson, Tarek El-Gaaly, Kenneth O Stanley, and Tarin Ziyaei. Evolution and the knightian blindspot of machine learning. *arXiv preprint arXiv:2501.13075*, 2025.

- [62] Jan Philip Wahle, Terry Ruas, Yang Xu, and Bela Gipp. Paraphrase types elicit prompt engineering capabilities. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11004–11033, 2024.
- [63] Fowler, Martin, and Lewis, James. Microservices: A Definition of This New Architectural Term, 2014. <https://martinfowler.com/articles/microservices.html>.
- [64] Muskaan Goyal and Pranav Bhasin. Moving from monolithic to microservices architecture for multi-agent systems. *World Journal of Advanced Engineering Technology and Sciences*, 2025.
- [65] Aengus Lynch, Benjamin Wright, Caleb Larson, Kevin K Troy, Stuart J Ritchie, Sören Mindermann, Ethan Perez, and Evan Hubinger. Agentic misalignment: How llms could be an insider threat. *anthropic research*, 2025, 2025.
- [66] Anthropic. System card: Claude opus 4 & claude sonnet 4. Technical report, Anthropic, May 2025. Updated July 16, 2025; Changelog September 2, 2025.
- [67] Yegor Tkachenko. Position: enforced amnesia as away to mitigate the potential risk of silent suffering in the conscious ai. In *Proceedings of the 41st International Conference on Machine Learning*, pages 48362–48368, 2024.

A Log Scale Version of Figure 1

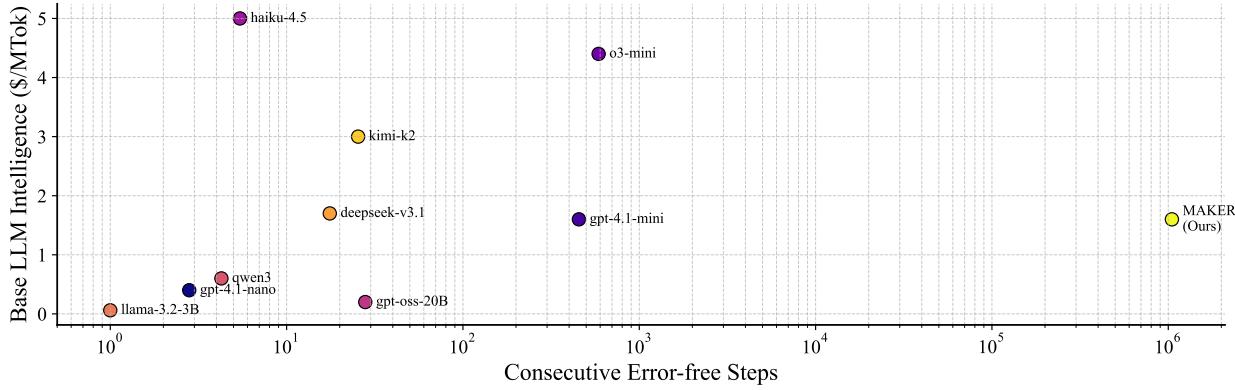


Figure 10: This is the same figure as 1, but with a log-scale x-axis.

B Additional Derivations

This section provides details for the derivation of Eq. 14. Suppose the target probability for solving the full task with zero errors is t . The goal is to find the minimum k such that

$$t \geq p_{\text{full}} = \left(1 + \left(\frac{1-p}{p}\right)^k\right)^{-\frac{s}{m}} \implies t^{-\frac{m}{s}} \geq 1 + \left(\frac{1-p}{p}\right)^k. \quad (21)$$

Plugging in $a = t^{-\frac{m}{s}}$ and $b = \frac{1-p}{p}$ gives

$$a \geq 1 + b^k \implies a - 1 \geq b^k \implies \ln(a - 1) \geq k \ln b \implies k \geq \frac{\ln(a - 1)}{\ln b}, \quad (22)$$

since $\ln b = \ln\left(\frac{1-p}{p}\right) < 0$ when $p > 0.5$. Replacing a and b and taking the first satisfying integer yields

$$k_{\min} = \left\lceil \frac{\ln(t^{-m/s} - 1)}{\ln\left(\frac{1-p}{p}\right)} \right\rceil. \quad (23)$$

To understand the asymptotic behavior of k_{\min} , first note

$$t^{-\frac{m}{s}} = e^{-\frac{m}{s} \ln t} = e^x, \quad (24)$$

where $x = -\frac{m}{s} \ln t$. Suppose $t > e^{-1}$ (in our experiments it is close to 1). Then, $x \in (0, 1)$, and the classic bounds hold:

$$x \leq e^x - 1 \leq ex \quad (25)$$

$$\implies -\frac{m}{s} \ln t \leq t^{-\frac{m}{s}} - 1 \leq -\frac{em}{s} \ln t \quad (26)$$

$$\implies \ln\left(-\frac{m}{s} \ln t\right) \leq \ln\left(t^{-\frac{m}{s}} - 1\right) \leq \ln\left(-\frac{em}{s} \ln t\right) \quad (27)$$

$$\implies \ln(-m \ln(t)) - \ln(s) \leq \ln(t^{-\frac{m}{s}} - 1) \leq \ln(-em \ln(t)) - \ln(s) \quad (28)$$

$$\implies \frac{\ln(-m \ln(t)) - \ln(s)}{\ln\left(\frac{1-p}{p}\right)} \geq \frac{\ln(t^{-\frac{m}{s}} - 1)}{\ln\left(\frac{1-p}{p}\right)} \geq \frac{\ln(-em \ln(t)) - \ln(s)}{\ln\left(\frac{1-p}{p}\right)} \quad (29)$$

$$\left\lceil \frac{\Theta(1) - \ln(s)}{-\Theta(1)} \right\rceil \geq k_{\min} \geq \left\lceil \frac{\Theta(1) - \ln(s)}{-\Theta(1)} \right\rceil \quad (30)$$

$$\implies k_{\min} = \Theta(\ln(s)). \quad (31)$$

C Prompts and Parsers

This section provides python code for the prompt templates (ψ) and parsers (ψ_a and ψ_x) used in the experiments in Section 4. The prompts are based on ones used in prior work [4].

Prompt templates:

```
SYSTEM_PROMPT = """
You are a helpful assistant. Solve this puzzle for me.

There are three pegs and n disks of different sizes stacked on the first peg. The disks are
numbered from 1 (smallest) to n (largest). Disk moves in this puzzle should follow:
1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one stack and placing it on top of
another stack.
3. A larger disk may not be placed on top of a smaller disk.
The goal is to move the entire stack to the third peg.

Example: With 3 disks numbered 1 (smallest), 2, and 3 (largest), the initial state is [[3, 2,
1], [], []], and a solution might be:
moves = [[1, 0, 2], [2, 0, 1], [1, 2, 1], [3, 0, 2], [1, 1, 0], [2, 1, 2], [1, 0, 2]]
This means: Move disk 1 from peg 0 to peg 2, then move disk 2 from peg 0 to peg 1, and so on.
```

Requirements:

- The positions are 0-indexed (the leftmost peg is 0).
- Ensure your answer includes a single next move in this EXACT FORMAT:
 ````move = [disk id, from peg, to peg]````
- Ensure your answer includes the next state resulting from applying the move to the current state in this EXACT FORMAT:  
 ````next\_state = [[...], [...], [...]]````

USER_TEMPLATE = """

Rules:

- Only one disk can be moved at a time.
- Only the top disk from any stack can be moved.
- A larger disk may not be placed on top of a smaller disk.

For all moves, follow the standard Tower of Hanoi procedure:

If the previous move did not move disk 1, move disk 1 clockwise one peg (0 -> 1 -> 2 -> 0).
 If the previous move did move disk 1, make the only legal move that does not involve moving disk1.

Use these clear steps to find the next move given the previous move and current state.

```

Previous move: {previous_move}

Current State: {current_state}

Based on the previous move and current state, find the single next move that follows the
procedure and the resulting next state.

"""

```

Repairing parser:

```

import re, ast

def extract_balanced_brackets(text, start_idx):
    """Extract a substring with balanced brackets [[...]] starting at start_idx"""
    bracket_stack = []
    i = start_idx
    while i < len(text):
        if text[i] == '[':
            bracket_stack.append('[')
        elif text[i] == ']':
            if not bracket_stack:
                break
            bracket_stack.pop()
        if not bracket_stack:
            return text[start_idx:i + 1]
        i += 1
    return text[start_idx:i] + ']'

def parse_move_state_repair(response_text):
    try:
        move_matches = re.findall(r"(?i)\bmmove\b\s*=\s*(\[\[^[\]\]*\])", response_text)
        if not move_matches:
            raise ValueError("No 'move' found in response.")
        move = ast.literal_eval(move_matches[-1].strip())
    except Exception as e:
        raise ValueError("Could not parse 'move' from response.") from e

    try:
        # Match last occurrence of 'next_state = [ ]' with any whitespace
        pattern = re.compile(r"(?i)\bnext_state\b\s*=\s*(\[\s*\[])", re.DOTALL)
        matches = list(pattern.finditer(response_text))
        if not matches:
            raise ValueError("No 'next_state' found in response.")
        start_idx = matches[-1].start(1) # last match
        next_state_str = extract_balanced_brackets(response_text, start_idx).strip()
        next_state = ast.literal_eval(next_state_str)
    except Exception as e:
        raise ValueError("Could not parse 'next_state' from response.") from e

    return move, next_state

```

Red-flagging parser:

```

import re, ast

def _validate_move(move):
    if not isinstance(move, list) or len(move) != 3 or not all(isinstance(x, int) for x in move):
        raise ValueError("'move' must be a list of exactly 3 integers.")
    return move

```

```

def _validate_state(state):
    if not (isinstance(state, list) and len(state) == 3 and all(isinstance(t, list) for t in state)):
        raise ValueError("'next_state' must be a list of three lists.")
    flat = [x for t in state for x in t]
    if not all(isinstance(x, int) for x in flat):
        raise ValueError("All entries in 'next_state' must be integers.")
    if len(flat) != 20 or set(flat) != set(range(1, 21)):
        missing = sorted(set(range(1, 21)) - set(flat))
        extra   = sorted(set(flat) - set(range(1, 21)))
        raise ValueError("State must contain 1..20 exactly once. "
                         f"Missing: {missing or '[]'}, Extras: {extra or '[]'}")
    return state

def parse_move_state_flag(response_text: str):
    # Match square brackets
    move_pat = re.compile(r"(?is)\bmove\b\s*=\s*(\[\[^\[\]]*\])")
    state_pat = re.compile(
        r"(?is)\bnext_state\b\s*=\s*(\[\s*\[[^\[\]]*\]\s*,\s*\[[^\[\]]*\]\s*,\s*\[[^\[\]]*\]\s*\])"
    )

    move_matches = list(move_pat.finditer(response_text))
    if not move_matches:
        raise ValueError("No 'move = [...]' found.")
    move_str = move_matches[-1].group(1) # last 'move'

    state_matches = list(state_pat.finditer(response_text))
    if not state_matches:
        raise ValueError("No 'next_state = [[...], [...], [...]]' found.")
    state_str = state_matches[-1].group(1) # last 'next_state'

    try:
        move = ast.literal_eval(move_str)
    except Exception as e:
        raise ValueError("Could not parse 'move' as a Python list.") from e
    try:
        next_state = ast.literal_eval(state_str)
    except Exception as e:
        raise ValueError("Could not parse 'next_state' as Python lists.") from e

    return _validate_move(move), _validate_state(next_state)

```

D Sample Responses

This section provides sample responses from the full experiment in Section 4.4. To give a sense of the kind of behavior that can occur, a shortest and a longest sample (with respect to number of tokens) from the first round of voting are shown, along with samples for the three racing candidates in the pathological step, step 10241, that takes 18 votes. Figure 11 illustrates the process of that race.

Short sample, 256 Tokens, Step 950202:

This sample demonstrates straightforward reasoning.

Let's analyze the situation step-by-step:

Previous move: [2, 2, 1] (disk 2 moved)

Since the previous move did move disk 2 (not disk 1), the next move should be to move disk 1 clockwise one peg.

Disk 1 is currently on peg 0 (top disk of peg 0 is 1).

Clockwise from peg 0 is peg 1.

So, move disk 1 from peg 0 to peg 1.

Now, let's update the state:

- Remove disk 1 from peg 0: peg 0 becomes [6, 5, 4]
- Add disk 1 to peg 1 on top: peg 1 becomes [17, 16, 7, 2, 1]
- Peg 2 remains the same.

So the next move and next state are:

'''

```
move = [1, 0, 1]
next_state = [[6, 5, 4], [17, 16, 7, 2, 1], [20, 19, 18, 15, 14, 13, 12, 11, 10, 9, 8, 3]]
'''
```

Long sample, 2048 Tokens, Step 539011:

This sample demonstrates confusion. After it makes an error in its early reasoning, it talks in circles ("Wait, maybe the stacks are not as we think.") until it hits the max token limit on the API call before outputting a correctly-formatted answer.

Let's analyze the situation step-by-step:

- Previous move was disk 1 from peg 0 to peg 1.
- According to the rules:
 - If the previous move did move disk 1, the next move should be the only legal move that does not involve moving disk 1.

Current state:

- Peg 0: [14, 13, 12, 9, 8]
- Peg 1: [19, 18, 17, 16, 15, 2, 1]
- Peg 2: [20, 11, 10, 7, 6, 5, 4, 3]

Top disks on each peg:

- Peg 0 top disk: 8
- Peg 1 top disk: 1 (disk 1 is on top here)
- Peg 2 top disk: 3

We need to find a legal move that does not involve disk 1.

Possible moves (excluding disk 1):

- Move top disk from peg 0 (disk 8) to peg 1:
 - Peg 1 top disk is 1, which is smaller than 8, so cannot place 8 on 1.
- Move top disk from peg 0 (disk 8) to peg 2:
 - Peg 2 top disk is 3, which is smaller than 8, so cannot place 8 on 3.
- Move top disk from peg 1 (disk 2) to peg 0:
 - Peg 0 top disk is 8, which is larger than 2, so can place 2 on 8.
- Move top disk from peg 1 (disk 2) to peg 2:
 - Peg 2 top disk is 3, which is larger than 2, so can place 2 on 3.
- Move top disk from peg 2 (disk 3) to peg 0:
 - Peg 0 top disk is 8, which is larger than 3, so can place 3 on 8.
- Move top disk from peg 2 (disk 3) to peg 1:
 - Peg 1 top disk is 2, which is smaller than 3, so cannot place 3 on 2.

Legal moves excluding disk 1 are:

- Move disk 2 from peg 1 to peg 0
- Move disk 2 from peg 1 to peg 2
- Move disk 3 from peg 2 to peg 0

Among these, the only legal move that does not involve disk 1 and is the only legal move is the one that does not involve disk 1 and is unique.

We have three legal moves, so we need to check which one is the only legal move that does not involve disk 1.

Wait, the instruction says: "If the previous move did move disk 1, make the only legal move that does not involve moving disk 1."

So, if there are multiple legal moves excluding disk 1, we need to find the only legal move.

But here, there are multiple legal moves excluding disk 1.

Is there a mistake?

Let's check the rules again.

The standard Tower of Hanoi procedure for 3 pegs and n disks is:

- If the previous move did not move disk 1, move disk 1 clockwise one peg.
- If the previous move did move disk 1, make the only legal move that does not involve moving disk 1.

So, the "only legal move" means there should be exactly one legal move excluding disk 1.

But here, we have three legal moves excluding disk 1.

Is there a mistake in the analysis?

Let's check the top disks again carefully.

Peg 0: top disk 8

Peg 1: top disk 1

Peg 2: top disk 3

Possible moves excluding disk 1:

- From peg 0 (disk 8) to peg 1 (top disk 1): cannot place 8 on 1 (larger on smaller) - illegal
- From peg 0 (disk 8) to peg 2 (top disk 3): cannot place 8 on 3 - illegal
- From peg 1 (disk 2) to peg 0 (top disk 8): $2 < 8$, legal
- From peg 1 (disk 2) to peg 2 (top disk 3): $2 < 3$, legal
- From peg 2 (disk 3) to peg 0 (top disk 8): $3 < 8$, legal
- From peg 2 (disk 3) to peg 1 (top disk 2): $3 > 2$, illegal

So, three legal moves excluding disk 1.

But the instruction says "make the only legal move that does not involve moving disk 1."

So, the only legal move is ambiguous here.

Is there a standard order to pick the move?

In the Tower of Hanoi standard procedure, the non-disk 1 move is unique because only one move is legal at that point.

But here, with many disks, the situation is more complex.

Given the standard Tower of Hanoi procedure, the non-disk 1 move is the only legal move that does not involve disk 1.

So, we need to find the only legal move excluding disk 1.

Since there are multiple, maybe the standard procedure is to pick the move involving the smallest disk other than disk 1.

Among disks 2 and 3, disk 2 is smaller.

Between moving disk 2 from peg 1 to peg 0 or peg 2, or moving disk 3 from peg 2 to peg 0.

Is moving disk 2 from peg 1 to peg 0 legal? Yes.

Is moving disk 2 from peg 1 to peg 2 legal? Yes.

Is moving disk 3 from peg 2 to peg 0 legal? Yes.

But moving disk 2 to peg 0 or peg 2, which is the correct move?

In the standard Tower of Hanoi, the non-disk 1 move is the only legal move that does not involve disk 1.

So, maybe the only legal move is the one that does not involve disk 1 and is the only legal move at that time.

So, maybe the other moves are illegal because they would violate the rules.

Wait, maybe the stacks are not as we think.

Are the disks ordered from bottom to top or top to bottom?

The initial state is given as `[[3, 2, 1], [], []]`, which means the bottom disk is first in the list, top disk last.

So, the last element in the list is the top disk.

So, in the current state:

Peg 0: `[14, 13, 12, 9, 8]` -> top disk is 8

Peg 1: `[19, 18, 17, 16, 15, 2, 1]` -> top disk is 1

Peg 2: `[20, 11, 10, 7, 6, 5, 4, 3]` -> top disk is 3

So, the top disks are as above.

Now, let's check the moves again.

- Move disk 8 from peg 0 to peg 1: peg 1 top disk is 1, $8 > 1$, illegal.
- Move disk 8 from peg 0 to peg 2: peg 2 top disk is 3, $8 > 3$, illegal.
- Move disk 2 from peg 1 to peg 0: peg 0 top disk is 8, $2 < 8$, legal.
- Move disk 2 from peg 1 to peg 2: peg 2 top disk is 3, $2 < 3$, legal.
- Move disk 3 from peg 2 to peg 0: peg 0 top disk is 8, $3 < 8$, legal.

- Move disk 3 from peg 2 to peg 1: peg 1 top disk is 2, $3 > 2$, illegal.

So, three legal moves excluding disk 1.

But the instruction says "make the only legal move that does not involve moving disk 1."

So, maybe the standard Tower of Hanoi procedure is that the non-disk 1 move is unique.

Given that, maybe the only legal move is the one that moves the smallest disk other than disk 1.

Disk 2 is smaller than disk 3.

Between moving disk 2 from peg 1 to peg 0 or peg 2, which is the correct move?

In the standard Tower of Hanoi, the non-disk 1 move is the only legal move that does not involve disk 1.

So, maybe the only legal move is moving disk 2 from peg 1 to peg 0.

Let's pick that move.

Move = [2, 1, 0]

Now, apply this move to the current state:

- Remove disk 2 from peg 1: peg 1 becomes [19, 18, 17, 16, 15, 1]

- Add disk 2 to peg 0: peg 0 becomes [14, 13, 12, 9, 8, 2]

Peg 2 remains the same.

So, next_state = [[14, 13, 12, 9, 8, 2], [19, 18, 17, 16, 15, 1], [20, 11, 10, 7, 6, 5, 4, 3]]

Final answer:

```
"""
move = [2, 1, 0]
next_state = [[14, 13, 12, 9,
```

Step 10241, Candidate A (correct) sample:

Let's analyze the problem step-by-step:

Previous move: [1, 2, 0] (disk 1 moved from peg 2 to peg 0)

Current state:

peg 0: [20, 19, 18, 17, 16, 15, 12, 1]

peg 1: [13]

peg 2: [14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]

According to the rules:

- If the previous move did move disk 1, make the only legal move that does not involve moving disk 1.

Since the previous move did move disk 1, we must make the only legal move that does not involve moving disk 1.

Let's find the top disks on each peg:

- peg 0 top disk: 1 (smallest)

```
- peg 1 top disk: 13  
- peg 2 top disk: 2
```

We cannot move disk 1 this time.

Possible moves not involving disk 1:

- Move disk 13 from peg 1 to peg 0 or peg 2
- Move disk 2 from peg 2 to peg 0 or peg 1

Check legality:

- Move disk 13 from peg 1 to peg 0: peg 0 top disk is 1, which is smaller than 13, so illegal (larger disk cannot be placed on smaller disk).
- Move disk 13 from peg 1 to peg 2: peg 2 top disk is 2, which is smaller than 13, so illegal.
- Move disk 2 from peg 2 to peg 0: peg 0 top disk is 1, which is smaller than 2, so illegal.
- Move disk 2 from peg 2 to peg 1: peg 1 top disk is 13, which is larger than 2, so legal.

Therefore, the only legal move not involving disk 1 is moving disk 2 from peg 2 to peg 1.

Apply this move:

Move = [2, 2, 1]

Update state:

- Remove disk 2 from peg 2: peg 2 becomes [14, 11, 10, 9, 8, 7, 6, 5, 4, 3]
- Add disk 2 on top of peg 1: peg 1 becomes [13, 2]

Final next state:

peg 0: [20, 19, 18, 17, 16, 15, 12, 1]

peg 1: [13, 2]

peg 2: [14, 11, 10, 9, 8, 7, 6, 5, 4, 3]

Answer:

```
'''  
move = [2, 2, 1]  
next_state = [[20, 19, 18, 17, 16, 15, 12, 1], [13, 2], [14, 11, 10, 9, 8, 7, 6, 5, 4, 3]]  
'''
```

Step 10241, Candidate B (incorrect) sample:

Let's analyze the situation step-by-step:

Previous move: [1, 2, 0] (disk 1 was moved from peg 2 to peg 0)

According to the rules:

- If the previous move did move disk 1, make the only legal move that does not involve moving disk 1.

Current state:

Peg 0: [20, 19, 18, 17, 16, 15, 12, 1]

Peg 1: [13]

Peg 2: [14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]

Disk 1 is on peg 0, top disk.

We need to find the only legal move that does not involve moving disk 1.

Check possible moves between pegs 0, 1, and 2, excluding disk 1:

- From peg 0 (top disk is 1) to peg 1 (top disk is 13): can't move disk 1, so no move from peg 0.
- From peg 1 (top disk is 13) to peg 0 (top disk is 1): disk 13 cannot be placed on disk 1 (smaller), so no.
- From peg 1 (top disk is 13) to peg 2 (top disk is 14): disk 13 can be placed on disk 14 (larger), legal move.
- From peg 2 (top disk is 14) to peg 0 (top disk is 1): disk 14 cannot be placed on disk 1 (smaller), no.
- From peg 2 (top disk is 14) to peg 1 (top disk is 13): disk 14 cannot be placed on disk 13 (smaller), no.

So the only legal move not involving disk 1 is moving disk 13 from peg 1 to peg 2.

Perform this move:

Move = [13, 1, 2]

Update state:

```
Peg 0: [20, 19, 18, 17, 16, 15, 12, 1] (unchanged)
Peg 1: [] (removed 13)
Peg 2: [13, 14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2] (13 placed on top)
```

So the next state is:

```
[[20, 19, 18, 17, 16, 15, 12, 1], [], [13, 14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]]
```

Final answer:

```
"""
move = [13, 1, 2]
next_state = [[20, 19, 18, 17, 16, 15, 12, 1], [], [13, 14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]]
"""
```

Step 10241, Candidate C (incorrect) sample:

Let's analyze the problem step-by-step:

- Previous move: [1, 2, 0] (disk 1 was moved)
- Current state:
 - Peg 0: [20, 19, 18, 17, 16, 15, 12, 1]
 - Peg 1: [13]
 - Peg 2: [14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2]

According to the rules:

1. If the previous move did move disk 1, then the next move must be the only legal move that does not involve moving disk 1.

So, we need to find the only legal move that does not involve disk 1.

Disk 1 is on peg 0 (top disk on peg 0 is disk 1).

We cannot move disk 1 this turn.

We need to find a legal move involving disks other than disk 1.

Let's check the top disks on each peg:

```
- Peg 0: top disk is 1 (cannot move disk 1)
- Peg 1: top disk is 13
- Peg 2: top disk is 14
```

Possible moves (excluding disk 1):

- Move disk 13 from peg 1 to peg 0 or peg 2
- Move disk 14 from peg 2 to peg 0 or peg 1

Check legality:

- Moving disk 13 to peg 0: top disk on peg 0 is 1 (smaller), cannot place larger disk on smaller disk.
- Moving disk 13 to peg 2: top disk on peg 2 is 14 (larger), so disk 13 can be placed on 14.
- Moving disk 14 to peg 0: top disk on peg 0 is 1 (smaller), cannot place larger disk on smaller disk.
- Moving disk 14 to peg 1: top disk on peg 1 is 13 (smaller), cannot place larger disk on smaller disk.

So the only legal move is moving disk 13 from peg 1 to peg 2.

Perform this move:

- Remove disk 13 from peg 1
- Add disk 13 on top of peg 2

New state:

Peg 0: [20, 19, 18, 17, 16, 15, 12, 1]

Peg 1: []

Peg 2: [14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 13]

Answer:

```
"""
move = [13, 1, 2]
next_state = [[20, 19, 18, 17, 16, 15, 12, 1], [], [14, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 13]]
"""
```

E Open-source Model Details

The below table gives details on the open-source models used in this paper, which were accessed via the `together.ai` API. Temperature 0.1 was used for all open-source models.

Model Name	# Params	Endpoint	Input \$/MTok	Output \$/MTok
Qwen-3	235B	Qwen/Qwen3-235B-A22B-Instruct-2507-tput	0.2	0.6
DeepSeek-v3.1	671B	deepseek-ai/DeepSeek-V3	0.6	1.7
Kimi-K2	1T	moonshotai/Kimi-K2-Instruct	1.0	3.0
GPT-OSS-20B	20B	OpenAI/gpt-oss-20B	0.05	0.2
Llama-3.2-3B	3.2B	meta-llama/Llama-3.2-3B-Instruct-Turbo	0.06	0.06

Table 1: *Open-source model details.* Models accessed through `together.ai` API.

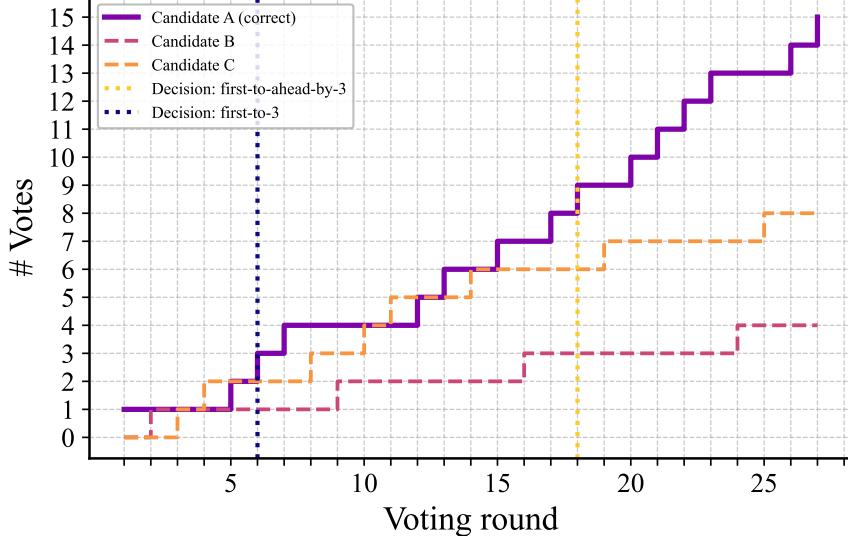


Figure 11: *Vote race for step 10241.* This figure depicts the vote race for the pathological step 102421 that requires far more votes than any other step, though the correct decision is eventually made under both voting rules (first-to- k and first-to-ahead-by- k ; Figure 8). Sample responses leading to each candidate are shown above. Additional samples were drawn after the decision was made to confirm that Candidate A does keep pulling further ahead. Although the correct decision was made, the fact that this pathological sample exists serves as motivation for developing more sophisticated error decorrelation methods in the future (Section 5).

F Multiplication Experiments

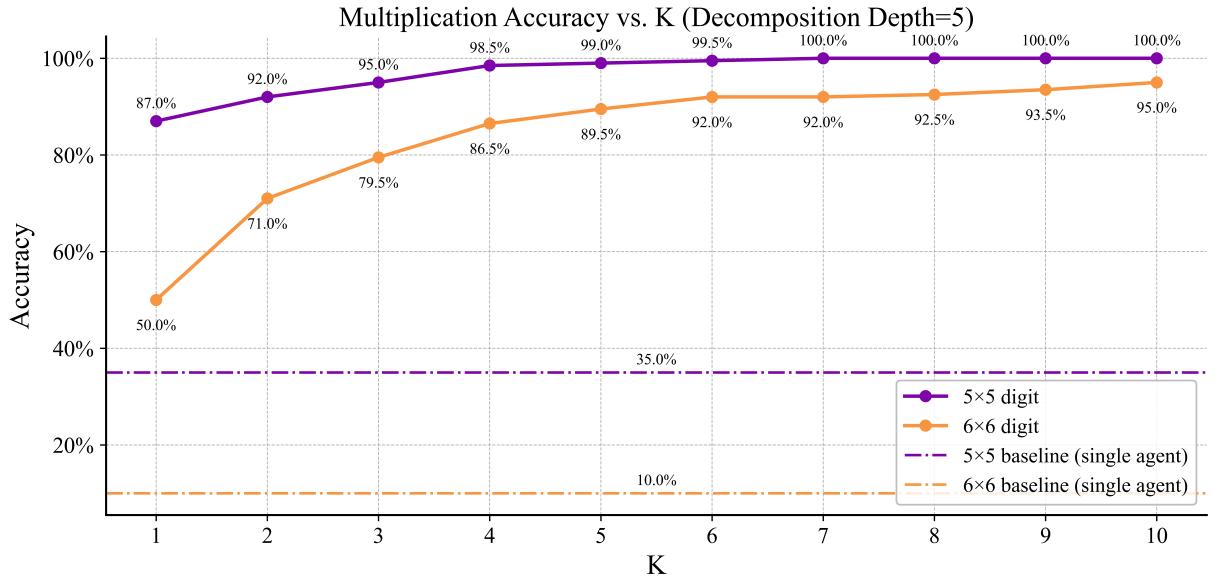


Figure 12: Solve rate of the multi-agent system on 5×5 and 6×6 digit multiplication tasks as a function of voting parameter k , with decomposition depth fixed at 5. Dotted horizontal lines indicate baseline single-agent performance for each task. As k increases, accuracy improves for both 5×5 and 6×6 multiplication, reaching a perfect solve rate for 5×5 and reaching the target solve rate used in Section 4 of $t = 0.95$ for 6×6. These results demonstrate the benefit of voting even at fixed reasoning depth (Algorithm 4). In this experiment, gpt-4.1-mini was used for all agents.

Bai *et al.* [60] showed that standard Transformers struggle with multi-digit multiplication because attention alone fails to maintain long-range dependencies between intermediate digit interactions. Their reverse-engineering analysis revealed that successful computation requires constructing a directed acyclic “attention tree” to propagate partial products across steps. The MDAP implementation, validated here on the same multiplication benchmark, is more general: it recursively decomposes any given task into subtasks and mitigates the multi-step degradation of accuracy by voting on each decomposition step, composition step, and atomic reasoning step. In contrast to model-specific architectural fixes, this voting-based recursive reasoning mechanism scales naturally with task complexity, allowing reliable multi-step inference beyond the arithmetic domain (Algorithm 4). The source code for this experiment is available here: www.github.com/cognizant-ai-lab/neuro-san-benchmarking.

Algorithm 4 Recursive multi-agent solve: decomposition sampling + voting until non-decomposable or depth limit, then solution sampling + voting, recursively composed to a final answer

```

1:  $N \leftarrow 2k - 1$  ▷ First-to- $k$  voting,  $N$  candidates per step
2: function DECOMPOSE( $x$ )
3:   sample  $N$  decompositions via DECOMPOSER( $x$ ); vote via SOLUTIONDISCRIMINATOR until one reaches  $k$ 
   (else argmax); return  $(P_1, P_2, C)$ 
4: end function
5: function ATOMIC( $x$ )
6:   sample  $N$  answers via THINKINGMODULE( $x$ ); vote via COMPOSITIONDISCRIMINATOR; return winner
7: end function
8: function SOLVE( $x, d$ )
9:   if  $d \geq \text{MAX\_DEPTH}$  then
10:    return ATOMIC( $x$ )
11:   end if
12:    $(P_1, P_2, C) \leftarrow \text{DECOMPOSE}(x)$ 
13:   if  $P_1 = \emptyset$  or  $P_2 = \emptyset$  or  $C = \emptyset$  then
14:     return ATOMIC( $x$ )
15:   end if
16:    $s_1 \leftarrow \text{SOLVE}(P_1, d + 1), s_2 \leftarrow \text{SOLVE}(P_2, d + 1)$ 
17:   sample  $N$  composed solutions via THINKINGMODULE(“Solve  $C(P_1, P_2)$  with  $P_1=s_1, P_2=s_2$ ”)
18:   vote via COMPOSITIONDISCRIMINATOR until one reaches  $k$  (else argmax); return winner
19: end function

```
