The Era of Agentic Organization: Learning to Organize with Language Models

Zewen Chi[1] **Li Dong**[1] **Qingxiu Dong**[2] **Yaru Hao**[1] **Xun Wu**[1] **Shaohan Huang**[1] **Furu Wei**[*1]
[1] Microsoft Research [2] Peking University
https://aka.ms/GeneralAI

Abstract

We envision a new era of AI, termed **agentic organization**, where agents solve complex problems by working collaboratively and concurrently, enabling outcomes beyond individual intelligence. To realize this vision, we introduce **asynchronous thinking** (AsyncThink) as a new paradigm of reasoning with large language models, which organizes the internal thinking process into concurrently executable structures. Specifically, we propose a thinking protocol where an organizer dynamically assigns sub-queries to workers, merges intermediate knowledge, and produces coherent solutions. More importantly, the thinking structure in this protocol can be further optimized through reinforcement learning. Experiments demonstrate that AsyncThink achieves 28% lower inference latency compared to parallel thinking while improving accuracy on mathematical reasoning. Moreover, AsyncThink generalizes its learned asynchronous thinking capabilities, effectively tackling unseen tasks without additional training.

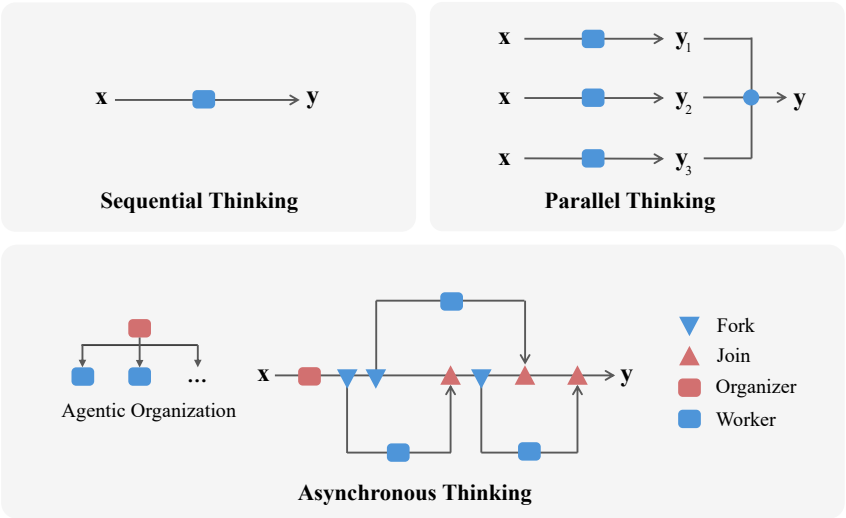

Figure 1: Comparison among our asynchronous thinking paradigm, sequential thinking, and parallel thinking. Sequential thinking employs a purely sequential decoding trajectory; parallel thinking executes multiple independent traces with an outcome aggregation. Differently, AsyncThink learns to form an agentic organization to think concurrently and collaboratively.

*Contact person: fuwei@microsoft.com.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Our vision for the next era of artificial intelligence is to realize **agentic organization**, where agents form organizational systems that collaborate to tackle complex problems beyond the limits of individual intelligence Casadei [2023], Pretel et al. [2025]. Although large language models have unlocked remarkable reasoning capabilities as individual agents El-Kishky [2024], Bai et al. [2025], realizing this vision requires reasoning models that can not only think independently but also think collaboratively as an organized system. This gap motivates research into new reasoning paradigms Zheng et al. [2026], Pan et al. [2025].

Developing AI systems that realize agentic organization introduces several key challenges Han et al. [2024]. First, organizing multiple agents to think concurrently often incurs additional latency Zhang et al. [2024]. Current parallel thinking approaches, which typically generate independent thinking traces and aggregate them afterward Yao et al. [2023], Zhao et al. [2025], Brown et al. [2024], are limited not only by the slowest thinking trace but also by the additional delay incurred during the final aggregation process. Second, adaptivity and dynamism remain difficult to achieve. Parallel thinking methods rely on manually designed, fixed workflows that cannot accommodate the diverse requirements of different queries Pan et al. [2025]. Some tasks benefit from divide-and-conquer strategies, while others require step-by-step reasoning. Third, learning effective agentic organization policies remains an open problem Guo et al. [2024], as manually designing optimal thinking structures for every possible query is intractable.

In this work, we introduce **asynchronous thinking** (AsyncThink) as a new paradigm for reasoning with large language models, with the goal of **learning to organize** the internal thinking into concurrently executable structures. Specifically, we propose a thinking protocol where an LLM plays both roles: an organizer that dynamically structures the process through `Fork` and `Join` actions, and workers that execute sub-queries and return intermediate knowledge or results. This thinking protocol provides the foundation for adaptivity and dynamic reasoning, allowing the model to explore diverse execution structures.

To train an AsyncThink model, we propose a two-stage training procedure. First, we perform a cold-start format fine-tuning on synthetic role-specific data to learn the syntax of AsyncThink actions. Then, we further train the AsyncThink model with a reinforcement learning stage, with rewards that encourage correctness, format compliance, and thinking concurrency.

We evaluate our AsyncThink models on multi-solution countdown, mathematical reasoning, and Sudoku tasks. Our experiments demonstrate that our models consistently achieve higher accuracy while reducing latency compared to sequential thinking and parallel thinking models. We further analyze its performance through ablation studies and accuracy-latency frontier comparisons, highlighting the effectiveness of our two-stage training procedure. Remarkably, our models also demonstrate strong generalization, exhibiting zero-shot AsyncThink reasoning on previously unseen tasks despite training solely on simple countdown data. Our contributions are as follows:

- We formalize the **learning-to-organize** problem, clarifying its setup and objective, building a foundation for studying how to organize agents to collaborate and operate concurrently.
- We introduce **asynchronous thinking**, a new reasoning paradigm that allows large language models to organize their internal thinking into concurrently executable structures, and learn this organization ability through reinforcement learning.
- We evaluate AsyncThink models on reasoning tasks, demonstrating improved accuracy and reduced latency. Remarkably, our models generalize asynchronous thinking to unseen tasks.

2 Organizer-Worker Thinking Protocol

Figure 2 illustrates an overview of our asynchronous thinking paradigm. The organizer-worker thinking protocol introduces two classes of roles for thinking, where an organizer manages the thinking processes and workers execute individual thinking processes for the assigned sub-queries. The two agent roles, organizer and worker, share the same LLM backbone and both perform autoregressive text decoding. Their distinction lies primarily in the set of actions each can take. This protocol provides the foundation for AsyncThink to explore diverse thinking structures.

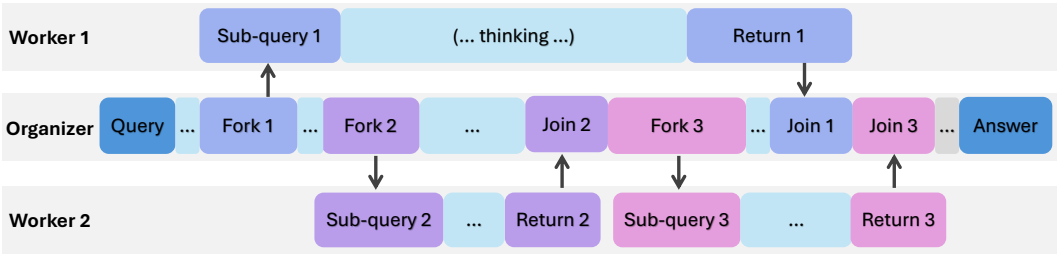

Figure 2: An example of the thinking protocol of AsyncThink. The protocol achieves asynchronous thinking through the `Fork-Join` actions, which controls the trajectory of thinking.

Table 1: Analogy between agentic-organization concepts and computer system components.

Concept	**Definition**	**Analogue (Computer Systems)**
Agent	A model performing actions sequentially.	*CPU core* that executes instructions for a single process at a time.
Agent Pool	A group of agents, determining the number of agents that can run concurrently.	*Multicore CPU* representing all available cores and defining parallel capacity.
Organization Policy	The strategy of organizing the agents to work collaboratively and concurrently.	*Multiprocess program* that organizes processes to achieve optimized execution.

2.1 Agentic Organization

Table 1 outlines the concepts of agentic organization by drawing an analogy to those of computer systems. Specifically, an *agent* refers to a model that can perform actions sequentially, and an *agent pool* represents a group of agents that run concurrently. In our setup, we specify an agent pool with a fixed capacity to ensure fair comparisons between different methods. In addition, agents may share the same model and model weights, as they can be viewed as different instances of the same model. *Organization policy* refers to the strategy of organizing agents to work collaboratively and concurrently to complete tasks. For example, a straightforward design has agents that operate independently and aggregate their outcomes in the end, as in parallel thinking Zhao et al. [2025], Wen et al. [2025]. In contrast, in this work, we propose an adaptive organization policy rather than a fixed, manually designed one.

2.2 Organizer

Organizer is the main role running through the whole thinking process, which is responsible for global coordination of asynchronous thinking. At the beginning of thinking, we concatenate the system prompt, which instructs the LLM how to take the `Fork-Join` actions in the text format, and the user query as the prompt. During the process, an organizer can take one of the following four actions sequentially:

- `Think`: Advance the current decoding process in its own decoding thread.
- `Fork`: Assign a thinking job to an available worker with a sub-query. The expected format is ... ⟨FORK-i⟩ *sub-query* ⟨/FORK-i⟩ ... , where i is a sub-query identifier used to distinguish this sub-query from the others. When a sub-query i already exists or there are already $c-1$ active workers, organizer cannot take the `Fork`-i action.
- `Join`: Request the output of a previously `Fork`-ed thinking job, with the format of ... ⟨JOIN-i⟩ . Upon recognizing this request, the generation process of the organizer may pause and wait for the corresponding worker to finish thinking on the assigned sub-query i. The returned output from the worker is then appended to the tail of the current organizer decoding context. The context format after completion is ... ⟨JOIN-i⟩ *returned text here* ⟨/JOIN-i⟩ ... .
- `Answer`: Terminate the inference process and produce the final answer.

The above actions can be generated in a pure text format, making them directly compatible with the input-output formats of existing LLMs.

2.3 Worker

Within an agent pool with capacity of c, there are $c - 1$ workers, which receive requests from the organizer and independently carry out thinking tasks for answering sub-queries. Once a worker completes the thinking for a sub-query and organizer requests the corresponding result, it sends the result back to the organizer, enabling communication. We consider a worker active when it is engaged in processing a sub-query and has not yet returned its result. Similar to the organizer, the input of a worker is a combination of the system prompt and the sub-query received from the organizer. The expected output format is *... thoughts ...* ⟨RETURN⟩ *some takeaways* ⟨RETURN⟩ *...* .

2.4 Asynchronous Thinking

Under the organizer-worker thinking protocol, the thinking procedure of AsyncThink begins with an organizer. This is the main role that receives the user query and drives the entire thinking process. Specifically, the organizer generates text tokens and action tags autoregressively, without using explicit action-selection modules.

When a `Fork` tag appears, AsyncThink proposes a sub-query and assigns it to an available worker. Upon receiving the request, the worker begins thinking for the sub-query. At the same time, the organizer continues its own execution such as thinking for `Fork`-ing another job, while workers run concurrently. Upon encountering a `Join` tag, AsyncThink synchronizes with the corresponding worker, i.e., if the worker is still running, the organizer pauses its decoding process until the worker returns results. Otherwise, the returned text will be appended to the organizer LLM decoding context, and the organizer resumes executing.

The thinking process concludes when the organizer generates a final answer ending with an end-of-sentence token. The thinking protocol operates entirely at the input-output surface of LLMs and does not require any modification to the underlying neural network architectures. Notably, it allows open-ended generation of the execution structure of thinking, providing a unified formulation under which existing thinking paradigms emerge as special cases. For example, sequential thinking can be realized when no `Fork` actions are taken, while parallel thinking arises when the organizer repeatedly assigns the original query to workers.

3 Learning to Organize

We train AsyncThink through a two-stage procedure. In cold-start format fine-tuning, we synthesize training data for the thinking protocol, curating query-response pairs that teach the model to play both roles. Subsequently, in reinforcement learning, the model explores diverse thinking structures and refines its asynchronous thinking capabilities, guided by reward signals that evaluate both reasoning quality and efficiency.

3.1 Stage 1: Cold-Start Format Fine-Tuning

This section introduces how to start training an AsyncThink model from an existing LLM with the cold-start format fine-tuning stage.

Data Synthesis Since existing corpora rarely contain organizer-worker thinking traces, we synthesize training data with GPT-4o. We use GPT-4o to analyze each query to detect conditionally independent thinking fragments. The model then generates the traces of both roles, following the organizer-worker thinking protocol formats by providing few-shot examples. Then, we verify the synthesized data with rules and filter out the data with format errors.

Random Initialization of Thinking Structure We empirically find that within an agent pool of $c > 2$, the GPT-4o model predominantly produces the organizer traces that follow two distinct thinking topologies: (1) interleaved `Fork` and `Join` operations, resulting in only one active worker at a time, and (2) a sequential pattern that first performs `Fork` $c - 1$ times, followed by `Join` $c - 1$

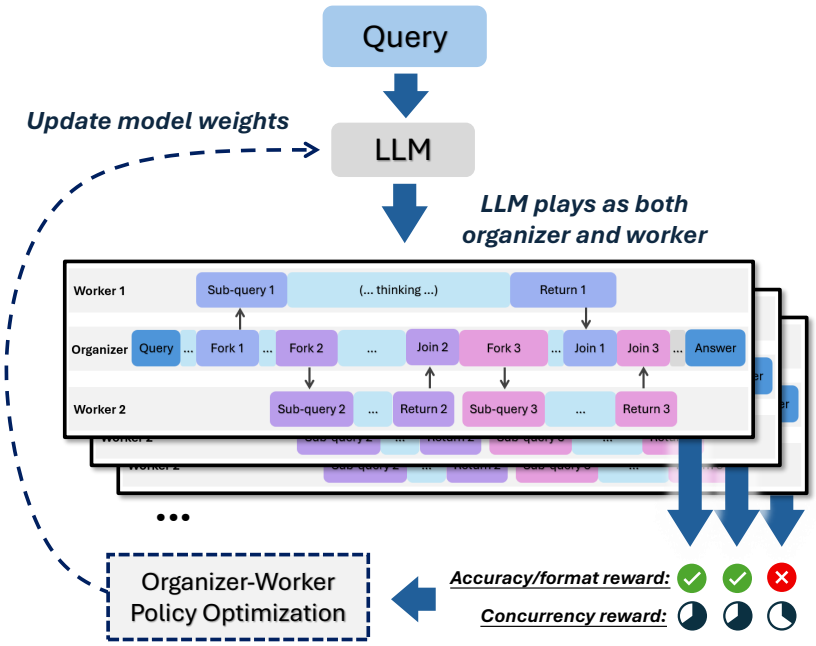

Figure 3: Illustration of the reinforcement learning framework for AsyncThink.

times. To encourage broader exploration, we randomly sample organizer action sequences and add one of the sequences to the prompt to guide the model to generate organizer outputs following the sampled structure.

Supervised Format Fine-Tuning We split the synthesized training data into query-response pairs and then fine-tune the LLM using the standard causal language modeling objective. This supervised format fine-tuning phase equips the model with the ability to emit valid organizer actions. At this stage, the model has not yet learned to produce correct answers with asynchronous thinking but only mimics the format. This limitation motivates the subsequent reinforcement learning stage.

3.2 Stage 2: Reinforcement Learning

Since the cold-start format fine-tuning stage only teaches the syntactic structure of organizer actions, the model still lacks the capability to exploit such thinking mechanism to produce answers. Thus, we further optimize the model using reinforcement learning to improve asynchronous thinking.

Reward Modeling We utilize a rule-based reward system that provides the following three types of rewards, encouraging both final-answer accuracy and thinking efficiency.

- **Accuracy Reward**: This reward measures the accuracy of the predicted final answers, typically a binary reward for single-answer questions with $R_{\text{A}} = 1$ for success and $R_{\text{A}} = 0$ for failure. For multi-solution questions, we use $R_{\text{A}} = \min(n_c, n_s)/n_s$ as the reward, where n_c and n_s represent the number of unique and correct answers and the total required solutions, respectively.
- **Format Reward**: We penalize the organizer outputs that have format errors, directly assigning a constant format error reward R_{FE} if the organizer produces the following format errors, including (1) duplicated sub-query index when the organizer redundantly `Fork`s a sub-query i when another sub-query i already exists and being processed, (2) agent pool overflow when the organizer `Fork`s a new thinking job and there are already $c - 1$ active workers, (3) `Join`-ing a non-existing sub-query, and (4) stopping without generating a final answer.
- **Thinking Concurrency Reward**: This reward encourages the model to efficiently organize the thinking processes into concurrently executable parts. Specifically, let a_t denote the number of active workers in an agent pool with capacity of c at global decoding step t, where a worker is considered active if it has been delegated a sub-query and is currently engaged in LLM decoding.

We define thinking concurrency ratio as $\eta = 1/T \cdot \sum_{t=1}^{T} a_t$, where T is the critical-path latency that will be introduced in Section 4.1. Next, we compute the thinking concurrency reward as $R_\eta = \min(\eta/c, \tau)/\tau$, where τ is a configurable threshold that prevents the model from hacking thinking concurrency, which is usually easier than predicting correct answers. The overall reward for an AsyncThink thinking trace i is computed as

$$R_i = \begin{cases} R_{\text{FE}} & \text{, trace } i \text{ has format errors} \\ R_{\text{A}} + \lambda R_\eta & \text{, otherwise} \end{cases} \tag{1}$$

Organizer-Worker Policy Optimization We extend the group relative policy optimization (GRPO) Shao et al. [2024] to handle the non-sequential thought samples in the reinforcement learning for AsyncThink. In common outcome-supervised setups, the episode is a single sequence of tokens. Differently, an episode of AsyncThink comprises multiple output traces from the organizer and its associated workers. To accommodate this structure, we treat the organizer trace together with its corresponding worker traces as a single unit when computing rewards and group-relative advantages. The resulting shared advantage is then assigned to all tokens produced by both roles. Note that tokens in the organizer trace that correspond to worker-returned outputs, i.e., segments formatted as ' *returned text here* ⟨/JOIN-i⟩ ' are masked out when computing the loss, as they are not produced by the organizer itself. The initial `Join` tag tokens, which merge the returned messages into the organizer context, are included in the loss computing for organizer.

4 Experiments

4.1 Evaluation Metrics

Different from the evaluation of sequential reasoning, asynchronous thinking evaluation requires considering both overall correctness and thinking efficiency. We use the following two metrics for asynchronous thinking: (1) *Final-Answer Accuracy* measures the correctness of the final answers. This metric reflects the overall reasoning capability of the model and enables direct comparison with other LLM reasoning methods. (2) *Critical-Path Latency* measures the minimum sequential depth required for asynchronous thinking. It represents a theoretical lower bound on inference time, abstracting away from implementation details such as the underlying inference engine implementation or hardware differences. Appendix A presents the computation of critical-path latency in detail.

4.2 Multi-Solution Countdown

The countdown task is a widely used testbed for evaluating the reasoning capabilities of LLMs, with the goal of finding the arithmetic operations $(+, -, \times, /)$ that convert the given numbers into the target number. Since the typical countdown task is not challenging for current LLMs, we extend the task to a harder version, named **multi-solution countdown** (MCD). Given a set of numbers, the goal is to find exactly four different solutions using three to six numbers from a given set of numbers. Here we define that two solutions are different when they have at least one different number or have at least one arithmetic operation used different times.

Data To construct the dataset, we randomly select 100 unique numbers ranging from 1 to $1,000$ as the target numbers of the test set. For each target number, we sample 4 unique subsets of $\{1, ..., 100\}$, which leads to 400 test examples. We ensure that the subsets have at least 4 different solutions. Similarly, we also construct training data with $22,500$ examples whose target numbers are different from those in the test set. In addition, we synthesize the cold-start data with an agent pool capacity of $c = 2$, following the procedure described in Section 3.1. The resulting training corpus contains 25.5K query-response pairs. Notice that an AsyncThink inference trace produces $2n_q + 1$ query-response pairs, where n_q is the number of sub-queries proposed by the organizer.

Training We first perform the supervised format fine-tuning (SFT) on the Qwen3-4B Yang et al. [2025a] base model for 1K steps, with a learning rate of 10^{-6} and a batch size of 128. Then, we train the SFT-ed model with reinforcement learning (RL) following the organizer-worker thinking protocol to sample asynchronous thinking trajectories with an agent pool capacity of $c = 2$. During RL, we use a learning rate of 10^{-6} and a batch of 64 MCD examples, and sample 32 trajectories

Table 2: Accuracy on the multi-solution countdown task. $\geq k$ measures whether the model finds at least k unique correct solutions. Results are averaged across 5 random seeds.

Method	All Correct	≥ 3	≥ 2	≥ 1
Sequential-Thinking	70.5	80.0	83.8	88.0
Parallel-Thinking	68.6	86.2	92.8	97.1
AsyncThink	**89.0**	**96.8**	**98.8**	**99.5**

Table 3: Math reasoning evaluation results on AIME-24 and AMC-23. We report the accuracy and critical-path latency results for each benchmark. Results are averaged across 5 random seeds.

Method	AIME-24		AMC-23	
	Acc. (↑)	Latency (↓)	Acc. (↑)	Latency (↓)
Seq-Thinking-L1K	24.7	1022.6	59.5	990.0
Seq-Thinking-L2K	35.3	2048.0	67.0	2001.1
Parallel-Thinking-L1K	24.7	1024.2	61.9	1029.5
Parallel-Thinking-L2K	**38.7**	2048.0	72.8	2031.4
AsyncThink	**38.7**	1468.0	**73.3**	1459.5

for each query to compute group-relative advantages. We use the ratio of valid solutions found by the models as the accuracy reward for the MCD task. We set the maximum response length for AsyncThink workers to 2K tokens for answering each sub-query. The maximum response length is limited to 2K tokens prior to each organizer `Join` action, after which the decoding budget resets to another 2K tokens.

Baselines We post-train the Qwen3-4B Yang et al. [2025a] base model for sequential thinking using reinforcement learning, where the model is trained to answer the query after thinking without `Fork`s or `Join`s. In addition, we also include a parallel thinking baseline that samples two responses from the sequential-thinking models with the same query separately. Then, it performs majority vote to select the first four most frequent solutions as the final answer, from around eight predicted solutions.

Results Table 2 presents the accuracy scores of the evaluated methods on the multi-solution countdown task, where '$\geq k$' means that the answer is considered correct if the model successfully finds k unique and correct solutions for a given question. The results show that AsyncThink substantially outperforms the baseline methods in all correctness thresholds. In particular, when evaluating strict correctness, i.e., 'All Correct', AsyncThink achieves 89.0% against 68.6% and 70.5% of the baseline methods. The improvement demonstrates that our asynchronous thinking method not only enhances the overall accuracy of the predicted countdown solutions but also leads to more reliable multi-solution coverage.

4.3 Math Reasoning

Data We use the DeepScaleR dataset Luo et al. [2025] as the RL training data for both AsyncThink and the baselines. We synthesize the AsyncThink cold-start data for mathematical reasoning following the procedure in Section 3.1, producing approximately 43K query-response pairs.

Training We perform format SFT on the Qwen3-4B model for 400 steps using a learning rate of 2×10^{-6} and a batch size of 128. Then, we apply reinforcement learning, where we sample asynchronous thinking trajectories with an agent pool capacity of $c = 4$. We use a learning rate of 10^{-6} and a batch of 128, and sample 8 trajectories for each query to compute group-relative advantages. We set the maximum response length of AsyncThink workers to 512 tokens when answering each sub-query, and reset it to 512 each time AsyncThink organizer finishes a `Join` action.

Baselines We compare AsyncThink against sequential and parallel thinking baselines, which train the model with DeepSeek-R1-style reinforcement learning. Specifically, sequential thinking performs reasoning within a single sequential thinking trace; parallel thinking first generates four thinking trajectories independently and then performs majority voting to determine the final answer. The suffix 'L1K' indicates that the maximum response length allowed during reinforcement learning is 1K tokens, and the same convention applies to 'L2K'.

Table 4: Ablation study results by removing key components of AsyncThink. Results are averaged across 5 random seeds.

Method	MCD			AMC-23		
	Acc. (↑)	η (↑)	Latency (↓)	Acc. (↑)	η (↑)	Latency (↓)
AsyncThink	**89.0**	64.7	4525.4	**73.3**	44.8	1459.5
$-R_\eta$ Reward	85.3	61.3	6250.6	71.3	26.1	1933.3
−Format SFT	64.8	50.0	3433.6	54.9	25.0	1007.7
−RL	0.0	49.2	1987.6	3.6	33.3	1396.9

Table 5: Generalization on an out-of-domain task, i.e., training on countdown data but evaluating on the 4×4 Sudoku task.

Method	Acc. (↑)	Latency (↓)
Sequential-Thinking	65.7	2056.4
Parallel-Thinking	84.2	3695.4
AsyncThink	**89.4**	2853.0

Results Table 3 presents the evaluation results on AMC-23 and AIME-24 MAA [2024]. AsyncThink obtains the best overall performance with substantially lower critical-path latency than the baseline methods. In particular, each AsyncThink worker is restricted to a response length of 512 tokens for each sub-query, which is substantially shorter than the 2K thinking traces used in the baselines, but overall performance remains competitive. The results indicate that short, organized thinking fragments can collectively achieve high problem-solving quality under the learned organization policy. This highlights the importance of the critical-path latency: the communication between workers and the organizer introduces non-trivial overhead (e.g., 512 worker-wise limits vs. roughly 1.5K overall latency), suggesting that future research should explicitly consider such overhead when evaluating non-sequential thinking methods.

Table 6: Comparison between theoretical critical-path latency and measured wall-clock time.

Method	Tokens	Latency	Wall-Clock Time
Sequential-Thinking	6.3K	6.3K	22.9s
AsyncThink	7.0K	4.5K	20.3s

4.4 Relation to Wall-Clock Time

While critical-path latency is a theoretical metric that abstracts away hardware and system-level implementation details, we find that it is strongly correlated with the actual wall-clock inference time in practice. Table 6 compares the decoding latency and measured end-to-end wall-clock time between Sequential-Thinking and AsyncThink on the MCD task.

Although AsyncThink generates slightly more total tokens due to the additional organizer-worker communication overhead, it substantially reduces the critical-path latency through asynchronous execution. This reduction also translates into lower real-world inference time. These results suggest that critical-path latency serves as a meaningful proxy for practical inference efficiency.

4.5 Generalization to Unseen Task

We study whether AsyncThink learns the generalized organization policy beyond the domain of training data. Specifically, we directly evaluate the AsyncThink and the baseline models, which are previously trained on the multi-solution countdown task, on the 4×4 Sudoku task. We sample 400 4×4 Sudoku examples from the Enigmata dataset as a test set Chen et al. [2025]. The evaluation results are shown in Table 5. Remarkably, AsyncThink obtains superior Sudoku performance while maintaining a much lower latency compared to parallel thinking. The results demonstrate that AsyncThink can learn general asynchronous thinking capabilities beyond the trained tasks.

Beyond evaluating the AsyncThink model on Sudoku, we further assess its generalization on two out-of-domain queries from computer science and biology. The corresponding thinking trajectories are presented in Appendix C.

4.6 Ablation Studies

Effect of Format Fine-Tuning We examine the effect of cold-start format fine-tuning in AsyncThink. For comparison, we skip the format fine-tuning stage and directly train the model with the

same reinforcement learning recipes as AsyncThink on the multi-solution countdown and math reasoning tasks, respectively. As shown in Table 4, the '$-$Format SFT' variant achieves much lower accuracy than AsyncThink for both tasks. Interestingly, we observe that the trained models can successfully perform `Fork`s but consistently remain the thinking concurrency ratio of $\eta = 1/c$, reflecting as 50.0% and 25.0% concurrency ratios in the two tasks, respectively. This behavior indicates that the cold-start format learning stage establishes a good initialization for AsyncThink to learn to explore and exploit asynchronous thinking structures.

Effect of Reinforcement Learning Figure 6 illustrates the training trajectories of accuracy, thinking concurrency, average number of `Fork` operations per query, and critical-path latency during AsyncThink reinforcement learning. Accuracy starts at zero and increases steadily, reaching 89.0% after 180 training steps. During training, the critical-path latency increases rapidly and reaches a peak at the enforced upper limit of 8K, reflecting that the model initially tends to think longer. As training progresses, the model learns to produce more concurrently-executable parts, leading to a steady increase in thinking concurrency and a decrease in latency. Meanwhile, the number of `Fork` operations per query also increases during training, indicating that the model progressively learns to distribute the thinking across the workers rather than a single thinking chain. These results highlight the benefits of modeling the thinking structure as model-predictable actions, allowing the model to adapt, explore, and optimize its thinking behavior according to task demands.

Reward Modeling To study the effects of the reward designs of AsyncThink, we conduct reinforcement learning using different rewards, and keep the other setups fixed. As shown in Table 4, '$-R_\eta$ Reward' stands for removing the thinking concurrency reward. Removing this reward reduces accuracy and leads to higher latency on both tasks, confirming its effect in encouraging parallelism among workers and the organizer. In another experiment, we design an additional leverage reward, which rewards the organizer to do `Fork` operations after a previous `Join`, building upon prior merged thinking outcomes. However, the reward introduces instability during training, sometimes resulting in a collapse to a fixed mode that exhibits near-sequential thinking with interleaved `Fork`s and `Join`s.

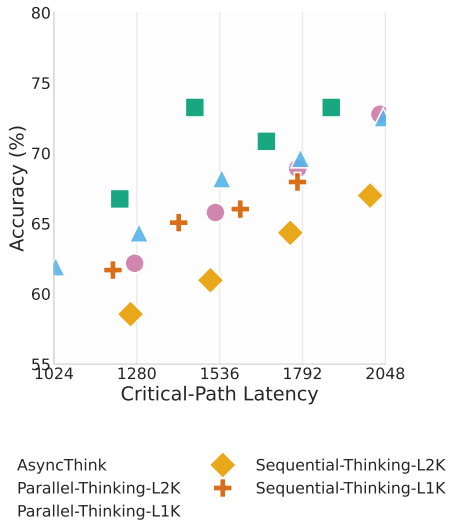

Figure 4: Accuracy-latency frontier of AsyncThink and baseline methods. The data points are collected by inference under various configurations of maximum response length.

Accuracy-Latency Frontier Figure 4 illustrates the accuracy-latency trade-off for AsyncThink and baseline methods. Each marker type represents a distinct method. We set various maximum response length limits for baseline methods to obtain the accuracy scores and their corresponding latencies. For AsyncThink, we keep the maximum response length of organizer fixed and set various maximum response lengths for workers to obtain accuracy scores with different overall latencies. Figure 4 shows that AsyncThink achieves a superior accuracy-latency frontier. In particular, AsyncThink reduces inference latency by 28% compared to parallel thinking while still achieving higher math reasoning accuracy. These results demonstrate that AsyncThink can not only successfully perform asynchronous thinking but also yields a more favorable accuracy-latency frontier.

5 Related Work

Chain-of-Thought Reasoning Chain-of-thought (CoT) reasoning Wei et al. [2022] refers to generating an explicit sequence of thinking steps before answering. Recent CoT models, such as DeepSeek-R1 Guo et al. [2025a], demonstrate that scalable verifiable supervision can substantially improve reasoning capabilities. Despite these advances, the design of verifiable reward functions

remains an open problem Zhang et al. [2025a]. Recent work has proposed model-based verifiers Ma et al. [2025], Liu et al. [2025], reasoning reward models Guo et al. [2025b], Whitehouse et al. [2026], generator-reward co-evolving Zha et al. [2025], Zhang et al. [2025b]. In addition, current research also explores a range of policy optimization algorithms, including critic-based algorithms Schulman et al. [2017], Yue et al. [2025], critic-free algorithms Shao et al. [2024], Ahmadian et al. [2024], off-policy methods Cohen et al. [2025], Roux et al. [2025], etc.

Parallel Thinking Although longer reasoning traces can improve model performance, many studies show that generating multiple reasoning paths independently and aggregating their results also improves performance Yao et al. [2023], Huang et al. [2025], Zhao et al. [2025], Brown et al. [2024], Guo et al. [2025b,a], Wen et al. [2025]. To overcome the inherent sequential nature of LLM decoding, recent work has explored parallel decoding algorithms and model architectures that allow simultaneous generation across branches or tokens Jin et al., Rodionov et al. [2025], Nie et al. [2025], Hsu et al. [2025], Yang et al. [2025b]. However, prior approaches often depend on handcrafted coordination rules or imitation from curated SFT data, and do not support reinforcement learning to further acquire reasoning capabilities. Beyond that, APR Pan et al. [2025] shows that LLMs can achieve parallel reasoning by imitating the execution traces of depth first search (DFS) and breadth first search (BFS) algorithms. However, this approach relies on predefined algorithmic traces, which limits its applicability to open-ended reasoning tasks where no universal DFS or BFS procedure can be specified. More recently, concurrent work Parallel-R1 Zheng et al. [2026] demonstrated that LLMs can learn to actively start sampling parallel thinking traces in the middle and aggregate their outcomes through model-driven decisions rather than rule-based schemes such as majority voting. Differently, we propose AsyncThink capable of learning to organize thinking processes, assigning diverse sub-queries and achieving lower latency compared to parallel thinking.

6 Future Work

Scaling Agentic Organization with Massive Agents This has two primary dimensions. First is scaling the quantity of workers. Future work should explore the scaling laws of asynchronous thinking: how accuracy-latency trade-offs evolve as the agent pool capacity grows from a few to hundreds or even thousands. Second is scaling the diversity of agents. We can move beyond a homogeneous pool to a massive organization of heterogeneous expert workers. These agents could be fine-tuned for specific domains (e.g., math, coding, data analysis) and, crucially, be equipped with different external tools (such as code interpreters, database query engines, or web search APIs). This introduces a more complex and powerful learning problem for the organizer.

Recursive Agentic Organization In this paradigm, any worker could dynamically be promoted to a sub-organizer, gaining the ability to Fork its own team of sub-workers. This would enable a flexible, hierarchical structure, naturally suited for deeply nested and complex problems that require multi-level decomposition. For instance, a top-level organizer might delegate a broad query like "*Solve the * problem*," only for the assigned worker to act as a sub-organizer, forking three new sub-workers to independently test different lemmas in parallel.

Human-AI Agentic Organization A key frontier is creating a Human-AI collaborative framework by integrating humans directly into the agentic organization. This could involve a "Human-as-Organizer" using the Fork protocol to dispatch tasks to AI workers, or a "Human-as-Worker" where the AI Forks tasks requiring human judgment (e.g., <FORK-human>Verify this conclusion</FORK-human>). Additionally, collaborative planning would allow humans and AI to co-design the asynchronous strategy before execution. This path moves beyond pure AI autonomy to enable a powerful, hybridized intelligence.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, 2024.

Kimi Team Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Haochen Ding, Meng-Xiao Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jia-Xing Guo, Hao-Xing Hu, Xiaoru Hao, Tianhong He, Weiran He, Wen He, et al. Kimi K2: Open agentic intelligence. *ArXiv*, abs/2507.20534, 2025. URL `https://api.semanticscholar.org/CorpusID:280323540`.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Roberto Casadei. Artificial collective intelligence engineering: A survey of concepts and perspectives. *Artificial Life*, 29:433–467, 2023. URL `https://api.semanticscholar.org/CorpusID:258059813`.

Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Jiaze Chen, Xuefeng Li, Qiying Yu, Hao Zhou, and Mingxuan Wang. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 3613–3661. Curran Associates, Inc., 2025. doi: 10.52202/085713-0128. URL `https://proceedings.neurips.cc/paper_files/paper/2025/file/058460c719f190500568b341bd06875f-Paper-Conference.pdf`.

Taco Cohen, David W Zhang, Kunhao Zheng, Yunhao Tang, Remi Munos, and Gabriel Synnaeve. Soft policy optimization: Online off-policy rl for sequence models. *arXiv preprint arXiv:2503.05453*, 2025.

Ahmed El-Kishky. Openai o1 system card. *ArXiv*, abs/2412.16720, 2024. URL `https://api.semanticscholar.org/CorpusID:272648256`.

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

Jiaxin Guo, Zewen Chi, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Reward reasoning models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 150477–150510. Curran Associates, Inc., 2025b. doi: 10.52202/085713-5031. URL `https://proceedings.neurips.cc/paper_files/paper/2025/file/dd35bb9efff094897fb6688a57675212-Paper-Conference.pdf`.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia V'elez, Qingyun Wu, Huazheng Wang, Thomas L. Griffiths, and Mengdi Wang. Embodied llm agents learn to cooperate in organized teams. *ArXiv*, abs/2403.12482, 2024. URL `https://api.semanticscholar.org/CorpusID:268531873`.

Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhuo Xu, and Chaoyang He. Llm multi-agent systems: Challenges and open problems. *ArXiv*, abs/2402.03578, 2024. URL `https://api.semanticscholar.org/CorpusID:267499950`.

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

Chengsong Huang, Langlin Huang, Jixuan Leng, Jiacheng Liu, and Jiaxin Huang. Efficient test-time scaling via self-calibration. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=RvMjxGpVOa.

Tian Jin, Ellie Y Cheng, Zachary Ankner, Nikunj Saunshi, Blake M Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. In *Forty-second International Conference on Machine Learning*.

Shudong Liu, Hongwei Liu, Junnan Liu, Linchen Xiao, Songyang Gao, Chengqi Lyu, Yuzhe Gu, Wenwei Zhang, Derek F. Wong, Songyang Zhang, and Kai Chen. CompassVerifier: A unified and robust verifier for LLMs evaluation and outcome reward. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33466–33494, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1698. URL https://aclanthology.org/2025.emnlp-main.1698/.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl, 2025. Notion Blog.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun MA, and Wenhu Chen. General-reasoner: Advancing llm reasoning across all domains. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 56596–56618. Curran Associates, Inc., 2025. doi: 10.52202/085713-1893. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/51c52cbb4b3faa891ca0621d13efe1be-Paper-Conference.pdf.

MAA. American invitational mathematics examination - aime 2024. *American Invitational Mathematics Examination - AIME 2024*, February 2024. URL https://maa.org/math-competitions/american-invitational-mathematics-examination-aime.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan LI. Large language diffusion models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 50608–50646. Curran Associates, Inc., 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/48b383b24230e0e6e649d9c98dae4d8c-Paper-Conference.pdf.

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

Elena Pretel, Elena Navarro, Víctor Casamayor Pujol, Schahram Dustdar, and Schahram Dustdar. Digital twins and artificial collective intelligence: Synergies for the future. *IEEE Internet Computing*, 29:75–85, 2025. URL https://api.semanticscholar.org/CorpusID:277092007.

Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm generation via concurrent attention. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 46592–46633. Curran Associates, Inc., 2025. doi: 10.52202/085713-1551. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4293727fc48c60b4adfe7a8e44f6a446-Paper-Conference.pdf.

Nicolas Le Roux, Marc G Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. Tapered off-policy reinforce: Stable and efficient reinforcement learning for llms. *arXiv preprint arXiv:2503.14286*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute. *arXiv preprint arXiv:2509.04475*, 2025.

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason E Weston, Ilia Kulikov, and Swarnadeep Saha. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *International Conference on Learning Representations*, volume 2026, pages 10397–10420, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/118c2d372db0939c7dcd716ebe6447b2-Paper-Conference.pdf.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation. *arXiv preprint arXiv:2506.09991*, 2025b.

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

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks. *arXiv preprint arXiv:2504.05118*, 2025.

Kaiwen Zha, Zhengqi Gao, Maohao Shen, Zhang-Wei Hong, Duane Boning, and Dina Katabi. Rl tango: Reinforcing generator and verifier together for language reasoning. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 119283–119313. Curran Associates, Inc., 2025. doi: 10.52202/085713-3981. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/ad06e23fe0c39b9de6e0cefe3b701f45-Paper-Conference.pdf.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. *ArXiv*, abs/2410.02506, 2024. URL https://api.semanticscholar.org/CorpusID:273098750.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, et al. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*, 2025a.

Xiaoying Zhang, Hao Sun, Yipeng Zhang, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback. *arXiv preprint arXiv:2506.03106*, 2025b.

Wenting Zhao, Pranjal Aggarwal, Swarnadeep Saha, Asli Celikyilmaz, Jason Weston, and Ilia Kulikov. The majority is not always right: Rl training for solution aggregation. *arXiv preprint arXiv:2509.06870*, 2025.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, He Xing, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and Dong Yu. Parallel-r1: Towards parallel thinking via reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *International Conference on Learning Representations*, volume 2026, pages 121144–121166, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/c4f3846c9a230cfd3373ff419b202bd5-Paper-Conference.pdf.

A Critical-Path Latency

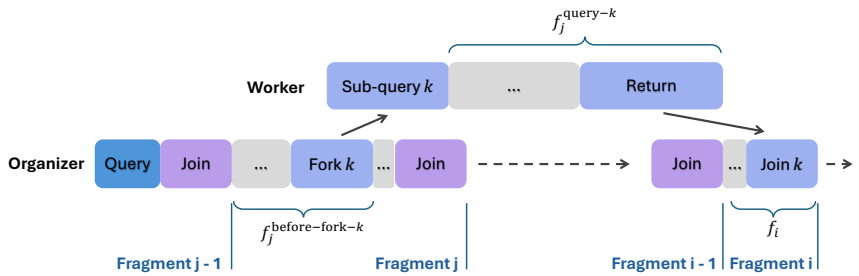

Figure 5: Illustration of the subproblem structure used in the dynamic programming formulation for computing the critical-path latency of an AsyncThink thinking trajectory.

Critical-path latency measures the minimum sequential depth required for asynchronous thinking. It represents a theoretical lower bound on inference time, abstracting away from implementation details such as the underlying inference engine implementation or hardware differences.

Figure 5 shows how to compute the critical-path latency of an AsyncThink thinking trajectory. According to the organizer-worker thinking protocol, the thinking trajectory takes the form of as a directed acyclic graph, where the organizer might wait for workers to finish at the `Join` tags, introducing extra latency beyond the latency of the organizer generating its own outputs. We present a dynamic programming method to compute the overall AsyncThink inference latency by breaking it down into a simpler subproblem, i.e., computing the latency at each `Join` position. Specifically, we split the organizer output sequence at the `Join` positions, and obtain $n_J + 1$ fragments, where n_J is the number of `Join` tags. The latency of finishing the i-th fragment l_i is computed as

$$l_i = \begin{cases} 0 & , i = 0 \\ \max(l_{i-1} + f_i, l_{j-1} + f_j^{\text{before-fork-}k} + f^{\text{query-}k}) & , 1 \le i \le n_J \\ l_{n_J} + f_{n_J+1} & , i = n_J + 1 \end{cases} \tag{2}$$

where we denote f_i the number of decoding steps of the i-th fragment; j stands for the fragment index of the `Fork` that the current `Join` is associated with; k represents the sub-query index of that `Fork`; $f_j^{\text{before-fork-}k}$ is the number of decoding steps within the fragment j to finish the `Fork` tags; $f^{\text{query-}k}$ stands for the number of decoding steps to response to the sub-query k.

B Statistical Significance

Table 7 reports the mean and standard deviation for each method on the MCD task, together with Welch's t-test comparing AsyncThink against the baselines.

The results show that AsyncThink consistently and significantly outperforms both Sequential-Thinking and Parallel-Thinking across all evaluation metrics. In particular, all improvements are statistically significant with $p < 0.01$, and most achieve $p < 0.001$.

Table 7: Statistical significance analysis on MCD over 5 random seeds.

Method	All Correct	$\geq$3	$\geq$2	$\geq$1
Sequential-Thinking	70.5 ± 1.9	80.2 ± 2.4	83.8 ± 2.7	88.0 ± 1.9
Parallel-Thinking	68.6 ± 0.7	86.2 ± 1.5	92.8 ± 1.0	97.1 ± 0.6
AsyncThink	89.0 ± 1.0	96.8 ± 1.0	98.8 ± 0.5	99.5 ± 0.2
Welch's t-test p-value				
vs. Sequential-Thinking	9×10^{-7}***	2×10^{-5}***	2×10^{-4}***	2×10^{-4}***
vs. Parallel-Thinking	8×10^{-10}***	4×10^{-6}***	2×10^{-5}***	1.6×10^{-3}**

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.

C Case Studies

To further illustrate how the learned AsyncThink models organize thinking, we present two representative examples in Figure 7 and Figure 8. These examples show how AsyncThink structures the thinking processes with novel asynchronous patterns beyond sequential thinking.

In the multi-solution countdown task (Figure 7), our AsyncThink model learns to perform multistage divide-and-merge reasoning, where the organizer adaptively invokes a worker to search for valid solutions of the input question. After analyzing the task, the organizer assigns a sub-query to a worker to first explore *multiplication-based* combinations toward the target number. Simultaneously, the organizer continues to explore other combinations. Once the worker completes, the results are merged into the organizer, and the organizer assigns new sub-queries when necessary. This iterative `Fork-Join` process enables efficient exploration and produces four valid expressions that reach the target of 888. In contrast, sequential thinking failed to find all four valid solutions in the case, highlighting the ability of AsyncThink to organize efficient thinking.

In the case of math reasoning (Figure 8), AsyncThink shows an interesting structure. With an agent pool of $c = 4$, the organizer assigns sub-queries to three workers, each exploring a distinct direction. For example, worker 3 is instructed to "assume the regular tetrahedron edge length can be set for simplicity." These workers run concurrently and their results are `Join`-ed back once the organizer verifies consistency across conclusions.

Across the case studies, AsyncThink distributes the thinking into disciplined `Fork-Join` exploration and merges them into a coherent answer. It achieves broader solution coverage and higher reliability with lower latency than sequential thinking. Moreover, it showcases organized reasoning patterns beyond reflection or self-correction, demonstrating an ability to organize multiple thinking paths and leverage intermediate results to conduct further thinking.

In addition to Figure 8 that presents an AsyncThink thinking trajectory on math reasoning, Figure 9 and Figure 10 demonstrate the capability of AsyncThink to generalize its asynchronous thinking capabilities to out-of-domain tasks, even it is trained on the multi-solution countdown data.

D Prompt Templates

We provide the prompt templates for organizer and worker of AsyncThink as follows. '{...}' represents a variable placeholder in the prompt template. For example, '{Capacity -1}' means this placeholder should be replaced with the agent pool capacity minus one, according to the experimental setup.

Prompt Template for Organizer

This is main process.

{Task-specific instruction here}

Use <FORK-*i*>subtask description</FORK-*i*> to delegate work and <JOINi> to wait for results. Integrate these results and provide final answer with <ANSWER>your final answer</ANSWER>. Notice that you can have at most {Capacity -1} subtasks running concurrently; any additional ones must wait until earlier ones finish.

{Query}

Prompt Template for Worker

This is a subprocess.

{Task-specific instruction here}

Subtask: {Sub-query assigned by the organizer}

Complete the subtask and provide results in:
<RETURN>
(Your short summary and valid expressions found)
</RETURN>

E Training Dynamics

Figure 6 shows the trajectories of accuracy, thinking concurrency ratio, average number of `Fork`, and critical-path latency throughout training.

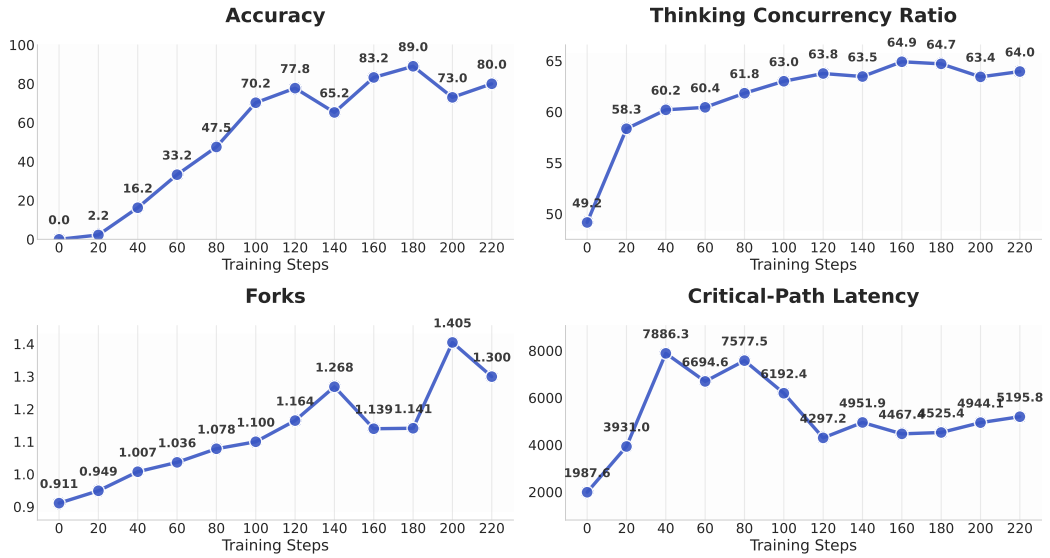

Figure 6: Trajectories of accuracy, thinking concurrency ratio, average number of `Fork`, and critical-path latency throughout training.

F Broader Impacts

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

G Limitations

Although this work demonstrates the effectiveness and achievability of AsyncThink on reasoning tasks, our experiments focus on controlled agent-pool settings that enable clear evaluation of accuracy and latency. Scaling asynchronous thinking to massive agent organizations remains an important direction for future work. In particular, studying how accuracy, latency, communication overhead, and organization policies change as the number of agents grows could reveal scaling laws for agentic organization, analogous to scaling-law studies for model size, data, and compute.

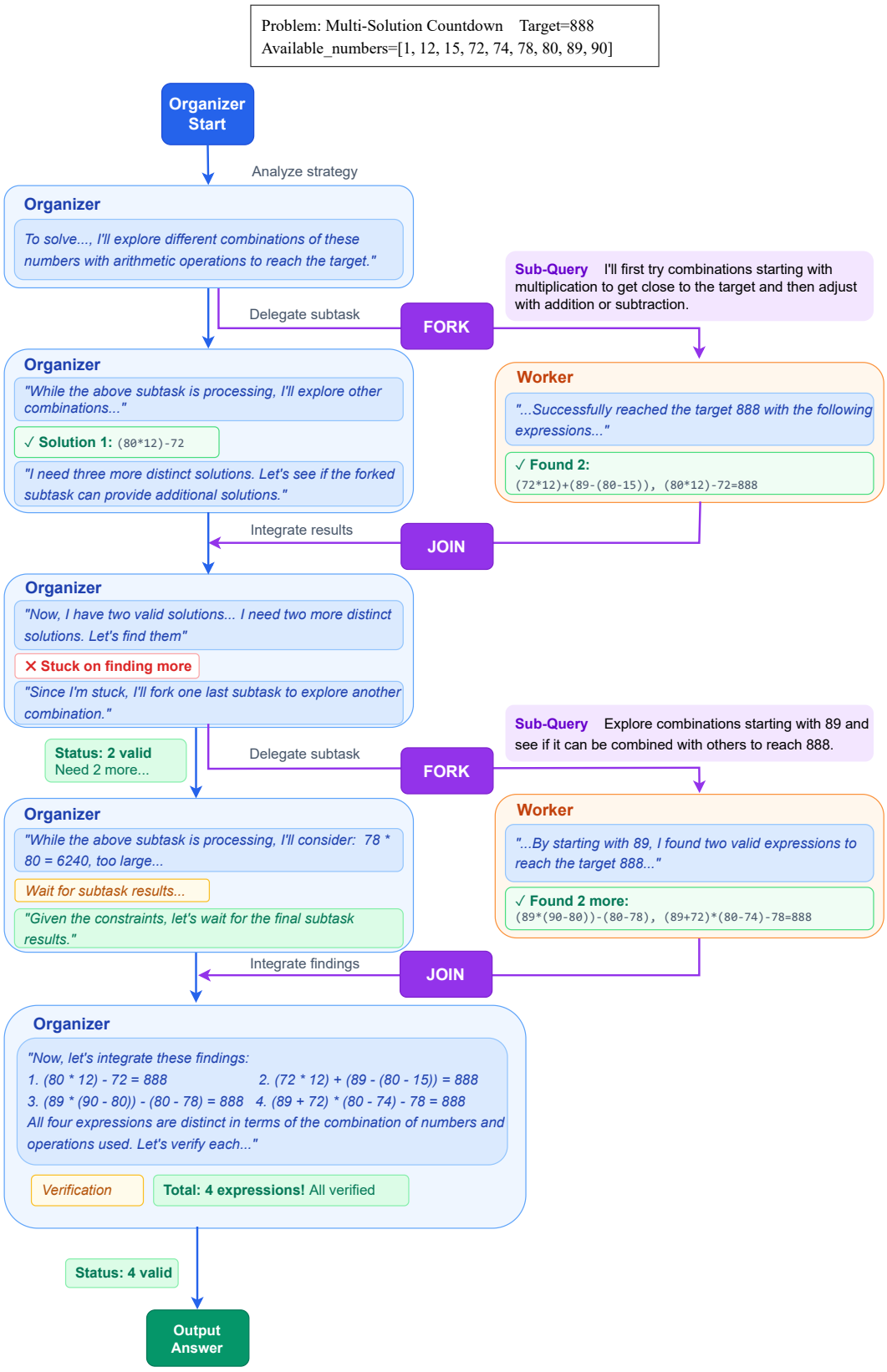

Figure 7: An AsyncThink thinking trajectory on the multi-solution countdown task with an agent pool capacity of $c = 2$. The organizer has learned to actively perform `Fork` and `Join` operations without external intervention. After the first `Join`, it checks the remaining gaps and launches a new sub-query as needed. The process finally yields four distinct valid expressions.

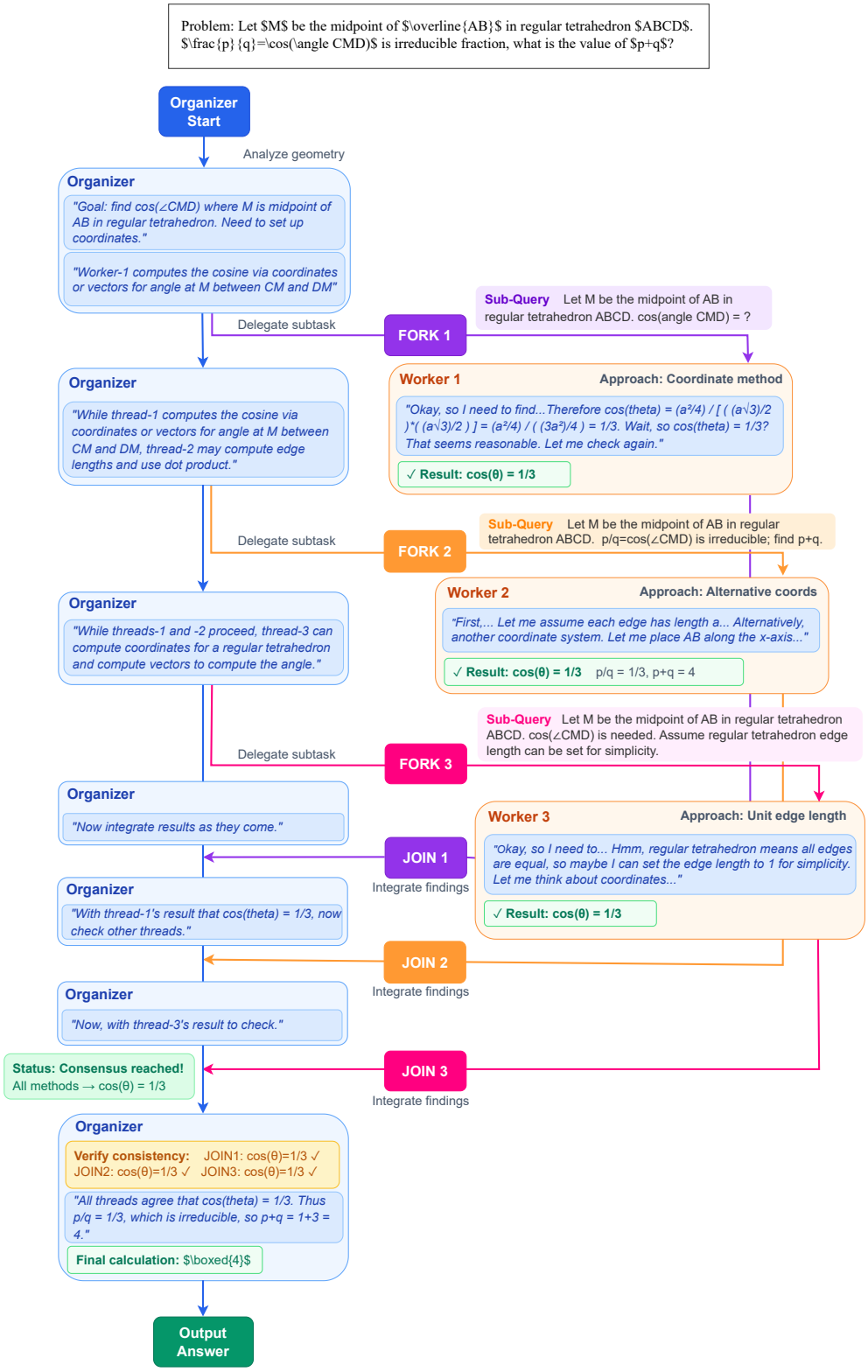

Figure 8: An AsyncThink thinking trajectory on mathematical reasoning with an agent pool capacity $c = 4$. The organizer spawns three workers, each using a different geometric formulation. These workers run in parallel and derive consistent results ($\cos\theta = 1/3$). They then rejoin for verification, converging to the final answer.

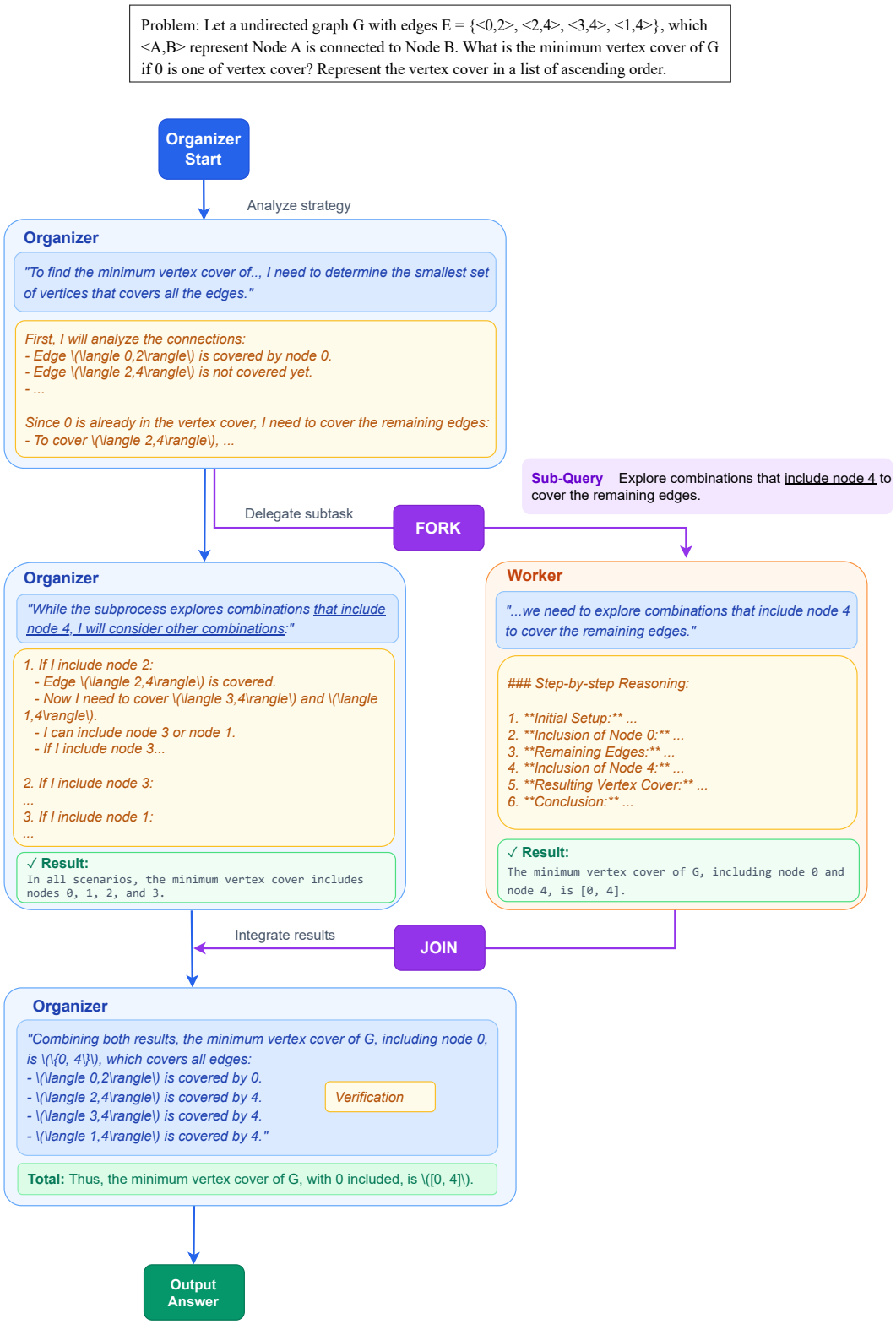

Figure 9: An example of AsyncThink on a minimum vertex cover problem from MMLU-Pro (with agent pool capacity $c = 2$). The organizer generalizes to this unseen graph theory task, performing `Fork` and `Join` operations without prior training on such problems. While a worker explores combinations with node 4, the organizer concurrently analyzes alternatives, ultimately determining the minimum vertex cover [0, 4].

Problem: Two 30" plants are crossed, resulting in progeny of the following ratio: one 22", eight 24\textquotedblright, twenty-eight 26", fifty-six 28", seventy 30", fifty-six 32", twenty-eight 34", eight 36", and one 38". Starting with A and going through the alphabet to represent different alleles, what are the most probable genotypes of the parents?
Options are:
["AaBbCcDd×AABBCCDD","AaBbCcEeFf×AaBbCcEeFf","AaBbCc×AaBbCc","AaBb×AaBb","AaBbCcDd×AaBbCcDd", "AaBbCcDdEe×AaBbCcDdEe","Aa×Aa","AaBbDd×AaBbDd","AaBbCc×AaBb","AABB×AABB"]

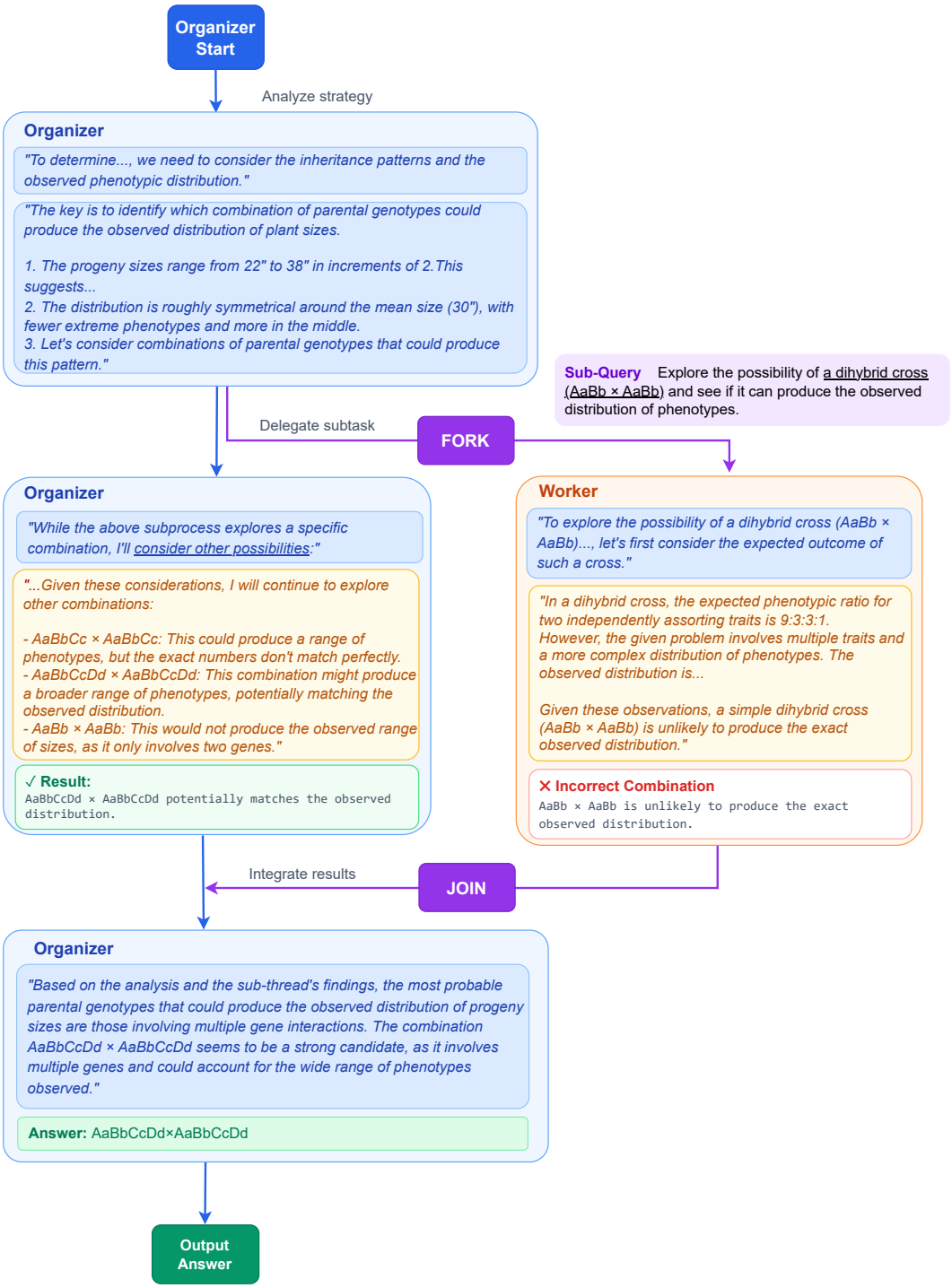

Figure 10: An example of AsyncThink on a genetics cross problem from MMLU-Pro (with agent pool capacity $c = 2$). The organizer generalizes to this biology domain, performing `Fork` and `Join` operations without domain-specific training. While a worker explores the dihybrid cross (AaBb Œ AaBb), the organizer concurrently evaluates multi-gene combinations, ultimately identifying AaBbCcDd Œ AaBbCcDd as the most probable parental genotypes.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.

- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Will release upon publication.

   Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).

- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.