

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

Zewen Chi Li Dong Qingxiu Dong Yaru Hao Xun Wu Shaohan Huang Furu Wei\*  
 Microsoft Research  
<https://aka.ms/GeneralAI>

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.

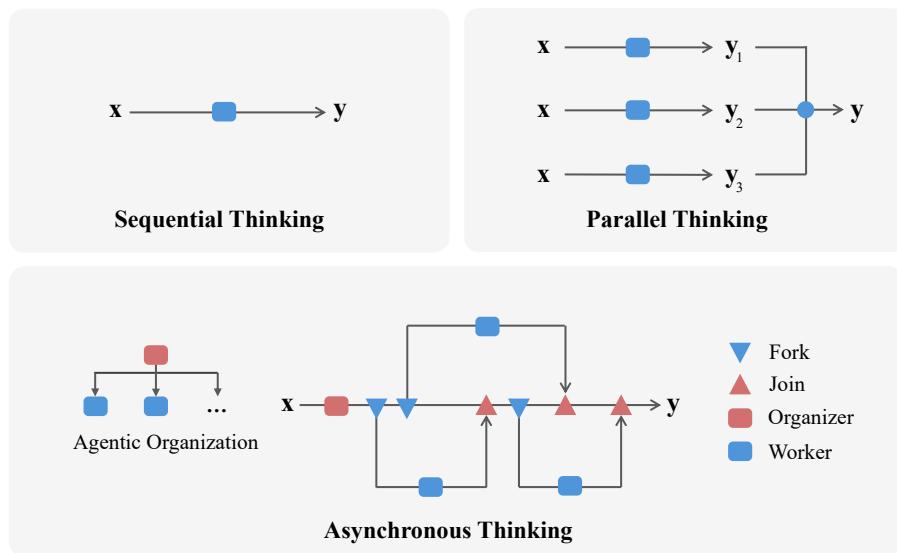


Figure 1: Comparison among our asynchronous thinking paradigm, sequential thinking, and parallel thinking. Sequential thinking employs a purely sequential decoding trajectory; parallel thinking executes multiple independent traces with an outcome aggregation. Differently, AsyncThink learns to form an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

\*Contact person: [fawei@microsoft.com](mailto:fawei@microsoft.com).

## 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 [6, 33]. Although large language models have unlocked remarkable reasoning capabilities as individual agents [12, 2], 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 [54, 32].

Developing AI systems that realize agentic organization introduces several key challenges [16]. First, organizing multiple agents to think concurrently often incurs additional latency [50]. Current parallel thinking approaches, which typically generate independent thinking traces and aggregate them afterward [45, 53, 5], 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 [32]. 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 [15], 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.

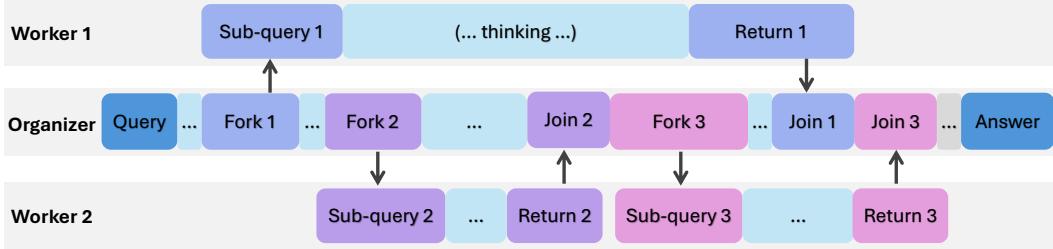


Figure 2: An example of the thinking protocol of AsyncThink. The protocol achieves asynchronous thinking through the Fork-Join actions, which controls the trajectory of thinking.

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

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

## 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 [53, 40]. 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 multiple 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 the GPT-4o model. 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`  $c - 1$  times.

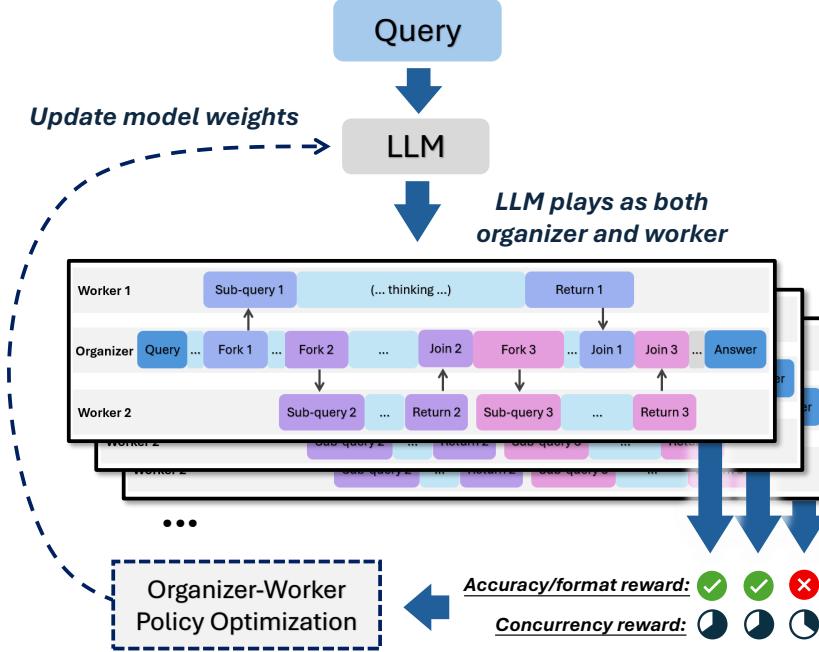


Figure 3: Illustration of the reinforcement learning framework for AsyncThink.

However, relying on a single or two thinking topologies reduces the plasticity of the model during training, and limits its ability to explore diverse organization policies. To enable 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 final 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  $R_A = 1$  for success and  $R_A = 0$  for failure. For multi-solution questions, we use  $R_A = \min(n_c, n_s)/n_s$  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 Forks a sub-query  $i$  when another sub-query  $i$  already exists and being processed, (2) agent pool overflow when the organizer Forks 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

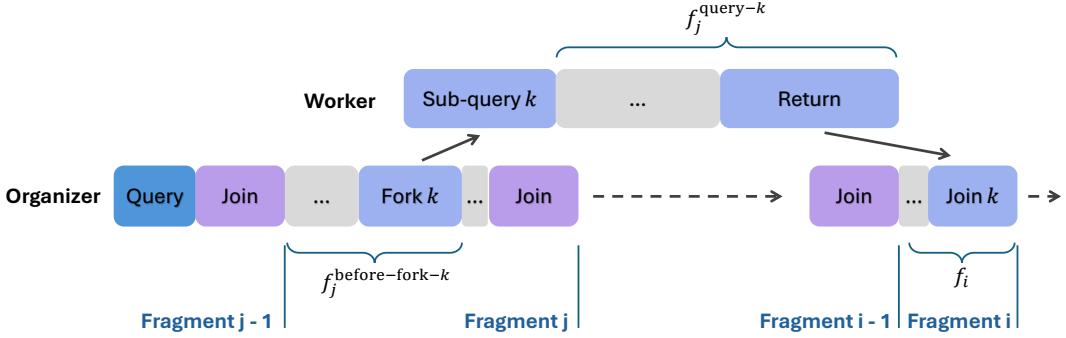


Figure 4: Illustration of the subproblem structure used in the dynamic programming formulation for computing the critical-path latency of an AsyncThink thinking trajectory.

considered active if it has been delegated a sub-query and is currently engaged in LLM decoding. We define thinking concurrency ratio as

$$\eta = \frac{1}{T} \sum_{t=1}^T a_t, \quad (1)$$

where  $T$  is the critical-path latency that will be introduced in Section 4.1. Next, we compute the thinking concurrency reward as

$$R_\eta = \min(\eta/c, \tau)/\tau, \quad (2)$$

where  $\tau$  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

$$R_i = \begin{cases} R_{FE} & , \text{trace } i \text{ has format errors} \\ R_A + \lambda R_\eta & , \text{otherwise} \end{cases} \quad (3)$$

**Organizer-Worker Policy Optimization** We extend the group relative policy optimization (GRPO) [37] 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 traces.

## 4 Experiments

To evaluate AsyncThink, we conduct experiments on three tasks, including multi-solution countdown, math reasoning, and Sudoku. In this section, we first introduce the evaluation metrics for asynchronous thinking, and then present the experimental details for each task individually.

### 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.

Figure 4 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 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

$$l_i = \begin{cases} 0 & , i = 0 \\ \max(l_{i-1} + f_i, l_{j-1} + f_j^{\text{before-fork-}k} + f^{\text{query-}k}) & , 1 \leq i \leq n_J \\ l_{n_J} + f_{n_J+1} & , i = n_J + 1 \end{cases} \quad (4)$$

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  $k$ .

## 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  $\{1, \dots, 100\}$ , 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 [43] 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 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 [43] base model for sequential thinking using reinforcement learning, where the model is trained to answer the query after thinking without `Forks` or `Joins`. 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** Figure 5 presents the accuracy scores of the evaluated methods on the multi-solution countdown task, where ‘ $\geq a$  Correct’ means that the answer is considered correct if the model

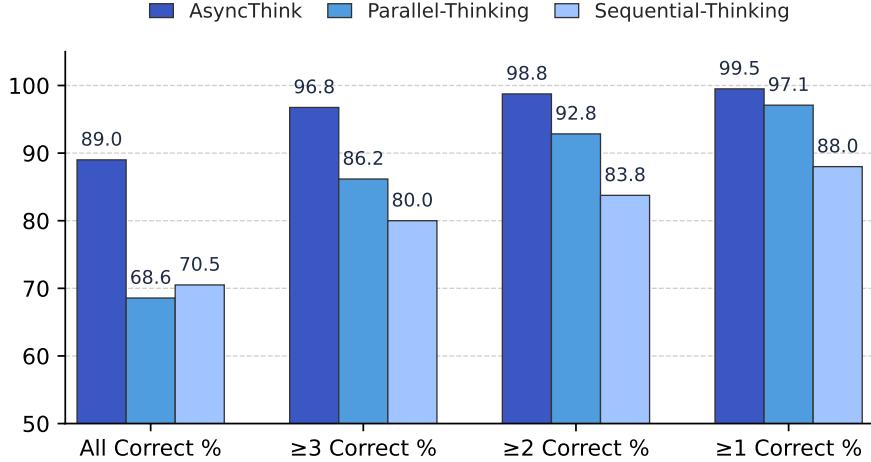


Figure 5: Evaluation results on the multi-solution countdown task. ‘ $\geq a$  Correct’ measures whether the model successfully finds  $a$  unique and correct solutions for a given question. Results are averaged across 5 random seeds.

successfully finds  $a$  unique and correct solutions for a given question. The figure shows 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

We also assess AsyncThink on math reasoning benchmarks, including AMC-23 and AIME-24 [30].

**Data** We use the DeepScaleR dataset [27] as the RL training data for both AsyncThink and the baseline methods. 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  $2 \times 10^{-6}$  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’.

**Results** Table 2 presents the evaluation results on AIME-24 and AMC-23. 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

Methods	AIME-24		AMC-23	
	Accuracy ( $\uparrow$ )	Latency ( $\downarrow$ )	Accuracy ( $\uparrow$ )	Latency ( $\downarrow$ )
Sequential-Thinking-L1K	24.7	1022.6	59.5	990.0
Sequential-Thinking-L2K	35.3	2048.0	67.0	2001.1
Parallel-Thinking-L1K	24.7	1024.2	61.9	1029.5
Parallel-Thinking-L2K	<b>38.7</b>	2048.0	72.8	2031.4
AsyncThink	<b>38.7</b>	1468.0	<b>73.3</b>	1459.5

Table 2: 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.

	MCD			AMC-23		
	Accuracy ( $\uparrow$ )	$\eta$ ( $\uparrow$ )	Latency ( $\downarrow$ )	Accuracy ( $\uparrow$ )	$\eta$ ( $\uparrow$ )	Latency ( $\downarrow$ )
AsyncThink	<b>89.0</b>	64.7	4525.4	<b>73.3</b>	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 3: Ablation study results by removing key components of AsyncThink. Results are averaged across 5 random seeds.

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.

#### 4.4 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  $4 \times 4$  Sudoku task. We sample 400  $4 \times 4$  Sudoku examples from the Enigmata dataset as a test set [8]. The evaluation results are shown in Table 4. 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 A.

#### 4.5 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 3, the ‘ $-Format SFT$ ’ variant achieves much lower accuracy than AsyncThink for both tasks. Interestingly, we observe that the trained models can successfully perform Forks 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

Methods	Accuracy ( $\uparrow$ )	Latency ( $\downarrow$ )
Sequential-Thinking	65.7	2055.5
Parallel-Thinking	84.2	3694.7
AsyncThink	<b>89.4</b>	2853.0

Table 4: Evaluation on an out-of-domain task, i.e., training on countdown data but evaluating on the  $4 \times 4$  Sudoku task. AsyncThink generalizes its asynchronous thinking capability to Sudoku, achieving higher accuracy.

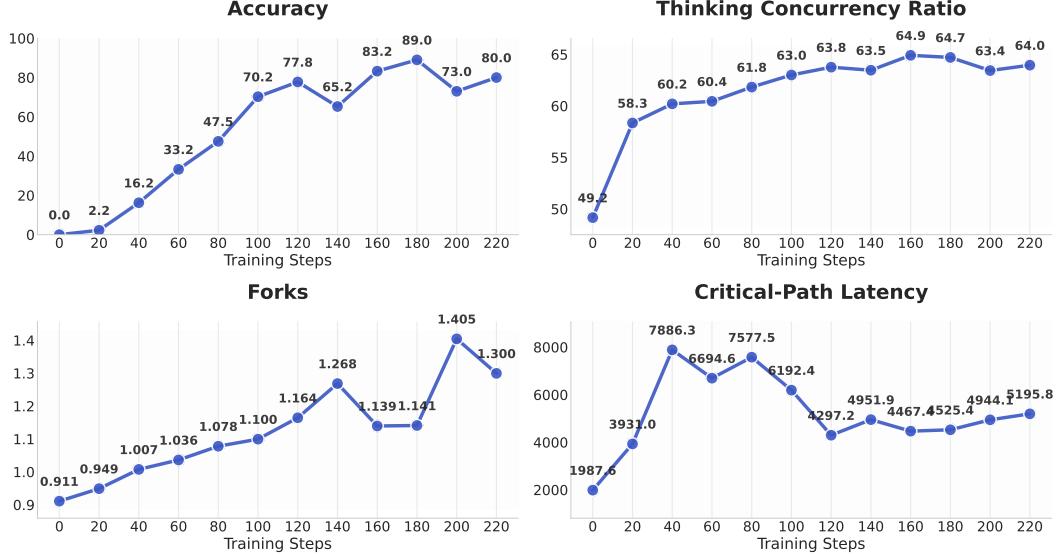


Figure 6: Trajectories of accuracy, thinking concurrency ratio, average number of Fork, and critical-path latency throughout training.

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 3, ‘ $-R_U$  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 Forks and Joins.

**Accuracy-Latency Frontier** Figure 7 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 7 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.

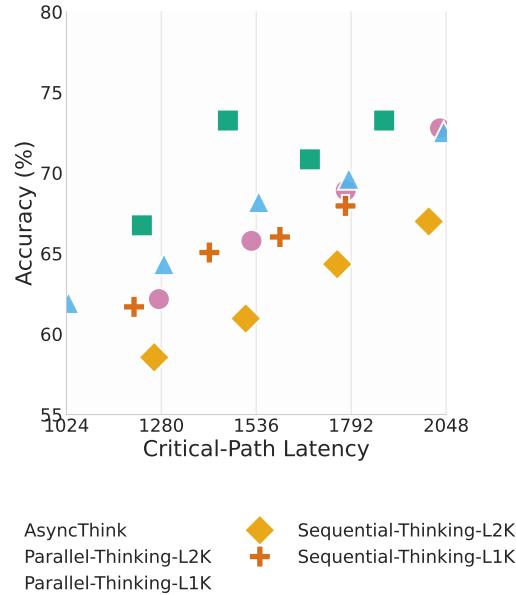


Figure 7: Accuracy-latency frontier of AsyncThink and baseline methods. The data points are collected by inference under various configurations of maximum response length.

#### 4.6 Case Studies

To further illustrate how the learned AsyncThink models organize thinking, we present two representative examples in Figure 8 and Figure 9. These examples show how AsyncThink structures the thinking processes with novel asynchronous patterns beyond sequential thinking.

In the multi-solution countdown task (Figure 8), 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 9), 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.

## 5 Related Work

**Chain-of-Thought Reasoning** Chain-of-thought (CoT) reasoning [39] refers to generating an explicit sequence of natural-language thinking steps that culminates in a final answer. To further strengthen the reasoning capability of large language models (LLMs), reinforcement learning with verifiable rewards (RLVR) has emerged as a key post-training paradigm. This framework enables models to receive learning signals from automatically verifiable outcomes rather than subjective

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

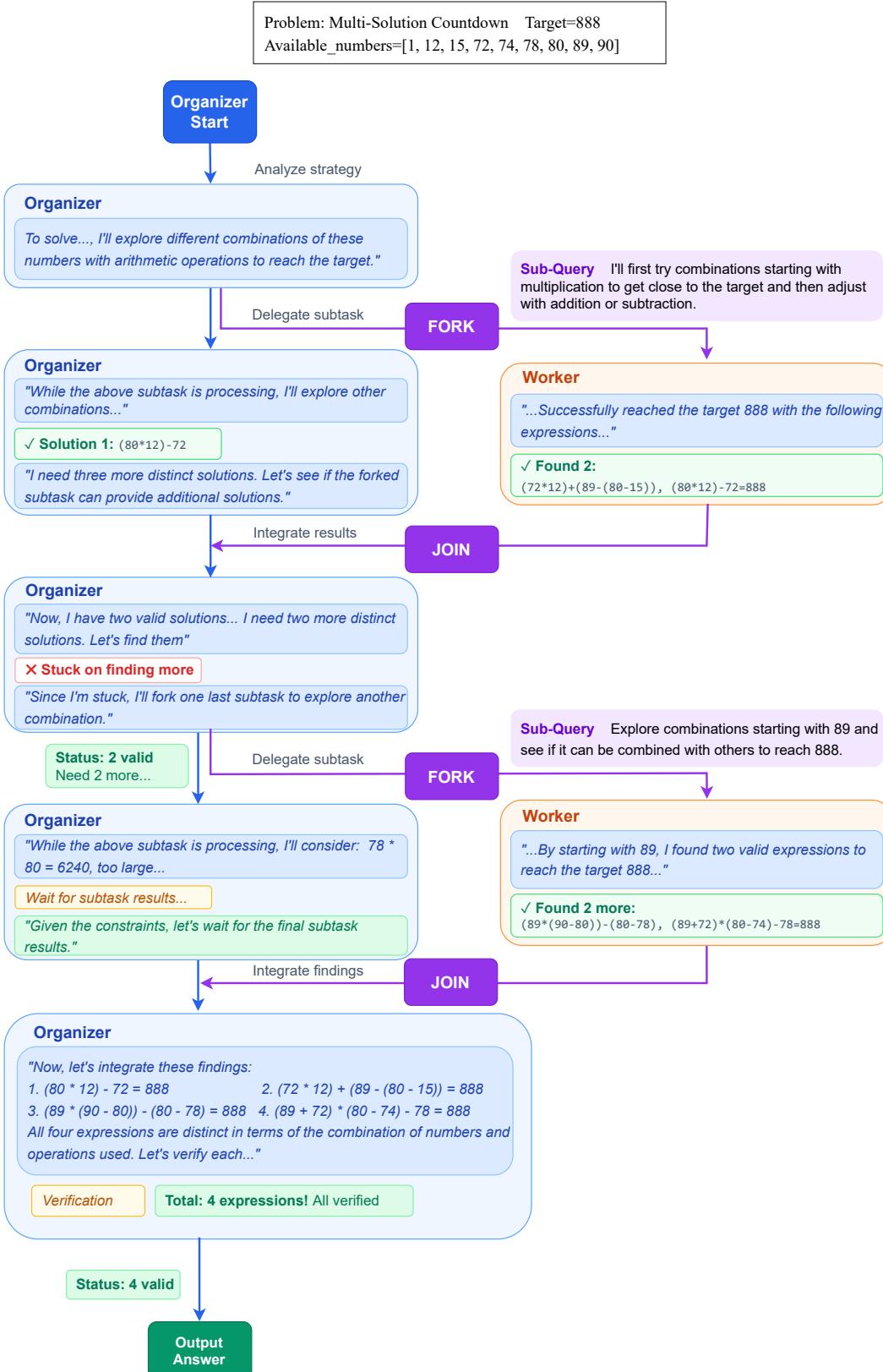


Figure 8: An AsyncThink thinking trajectory on the multi-solution countdown task with an agent pool capacity of  $c = 2$ . The organizer has learned to actively perform Fork and Join operations without external intervention. After the first Join, it checks the remaining gaps and launches a new sub-query as needed. The process finally yields four distinct valid expressions.

human feedback. Recent RLVR-based models, such as DeepSeek-R1 [13], demonstrate that scalable verifiable supervision can substantially improve reasoning capabilities. Despite these advances, the design of verifiable reward functions remains an open problem [51]. Recent work has proposed model-based verifiers [29, 26], reasoning reward models [14, 41], generator-reward co-evolving [48, 52]. In addition, current research also explores a range of policy optimization algorithms, including critic-based algorithms [36, 46], critic-free algorithms [37, 1], off-policy methods [9, 35], 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 [45, 20, 53, 5, 14, 13, 40]. 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 [21, 34, 31, 18, 44]. 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 [32] 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 [54] 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.

**Multi-Agent Systems** LLM-based multi-agent systems employ multiple agents to collaboratively solve complex tasks via specialized roles and iterative dialogue [24, 23, 3]. A wide range of multi-agent systems operate via predefined conversational roles [17, 7]. Such designs facilitate structured task execution through cooperative dialogues [42, 22] or competitive debates [11, 25]. Recent advances demonstrate that multi-agent LLMs can achieve more adaptive and efficient collaboration through dynamic belief modeling [49], role exchange [28], and agent co-evolution [19, 4]. To further enhance stability and efficiency, Zahedifar et al. [47] integrates a central controller agent for high-level planning, and Suzgun and Kalai [38] guides the meta agent in assuming distinct sub-roles through meta-prompting. A concurrent work, Puppeteer [10], further explores RL-based dynamic orchestration of LLM agents, providing additional evidence that collective intelligence can enhance model capabilities through specialized roles and cross-verification.

## 6 Conclusion and 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

- [1] 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.
- [2] 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>.
- [3] Rafael Barbarroxa, Luis Gomes, and Zita A. Vale. Benchmarking large language models for multi-agent systems: A comparative analysis of autogen, crewai, and taskweaver. In *Practical Applications of Agents and Multi-Agent Systems*, 2024. URL <https://api.semanticscholar.org/CorpusID:274515739>.
- [4] Nikolas Belle, Dakota Barnes, Alfonso Amayuelas, Ivan Bercovich, Xin Eric Wang, and William Yang Wang. Agents of change: Self-evolving llm agents for strategic planning. *ArXiv*, abs/2506.04651, 2025. URL <https://api.semanticscholar.org/CorpusID:279245109>.
- [5] 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.
- [6] 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>.
- [7] Dake Chen, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Haoyang Zhang. Gamegpt: Multi-agent collaborative framework for game development. *ArXiv*, abs/2310.08067, 2023. URL <https://api.semanticscholar.org/CorpusID:263909137>.
- [8] Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Qiying Yu, Xuefeng Li, Jiaze Chen, et al. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. *arXiv preprint arXiv:2505.19914*, 2025.
- [9] 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.
- [10] Yufan Dang, Cheng Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. *ArXiv*, abs/2505.19591, 2025. URL <https://api.semanticscholar.org/CorpusID:278905542>.
- [11] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *ArXiv*, abs/2305.14325, 2023. URL <https://api.semanticscholar.org/CorpusID:258841118>.
- [12] Ahmed El-Kishky. Openai o1 system card. *ArXiv*, abs/2412.16720, 2024. URL <https://api.semanticscholar.org/CorpusID:272648256>.
- [13] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [14] Jiaxin Guo, Zewen Chi, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Reward reasoning model. *arXiv preprint arXiv:2505.14674*, 2025.

- [15] 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>.
- [16] 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>.
- [17] Sirui Hong, Xiawu Zheng, Jonathan P. Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zi Hen Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. Metagpt: Meta programming for multi-agent collaborative framework. *ArXiv*, abs/2308.00352, 2023. URL <https://api.semanticscholar.org/CorpusID:260351380>.
- [18] 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.
- [19] Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. Self-evolving multi-agent collaboration networks for software development. *ArXiv*, abs/2410.16946, 2024. URL <https://api.semanticscholar.org/CorpusID:273507543>.
- [20] Chongsong Huang, Langlin Huang, Jixuan Leng, Jiacheng Liu, and Jiaxin Huang. Efficient test-time scaling via self-calibration. *arXiv preprint arXiv:2503.00031*, 2025.
- [21] 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*.
- [22] G. Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In *Neural Information Processing Systems*, 2023. URL <https://api.semanticscholar.org/CorpusID:268042527>.
- [23] Haoyuan Li, Hao Jiang, Tianke Zhang, Zhelun Yu, Aoxiong Yin, Hao Cheng, Siming Fu, Yuhao Zhang, and Wanggui He. Traineragent: Customizable and efficient model training through llm-powered multi-agent system. *ArXiv*, abs/2311.06622, 2023. URL <https://api.semanticscholar.org/CorpusID:265149831>.
- [24] Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 2024. URL <https://api.semanticscholar.org/CorpusID:273218743>.
- [25] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. *ArXiv*, abs/2305.19118, 2023. URL <https://api.semanticscholar.org/CorpusID:258967540>.
- [26] Shudong Liu, Hongwei Liu, Junnan Liu, Linchen Xiao, Songyang Gao, Chengqi Lyu, Yuzhe Gu, Wenwei Zhang, Derek F Wong, Songyang Zhang, et al. Compassverifier: A unified and robust verifier for llms evaluation and outcome reward. *arXiv preprint arXiv:2508.03686*, 2025.
- [27] 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.
- [28] Hao Ma, Tianyi Hu, Zhiqiang Pu, Liu Boyin, Xiaolin Ai, Yanyan Liang, and Min Chen. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 37:15497–15525, 2024.

- [29] Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhui Chen. General reasoner: Advancing llm reasoning across all domains. *arXiv preprint arXiv:2505.14652*, 2025.
- [30] 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>.
- [31] 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. *arXiv preprint arXiv:2502.09992*, 2025.
- [32] 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.
- [33] 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>.
- [34] Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitzin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm generation via concurrent attention. *arXiv preprint arXiv:2504.06261*, 2025.
- [35] 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.
- [36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [37] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Huawei 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.
- [38] Mirac Suzgun and Adam Tauman Kalai. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *ArXiv*, abs/2401.12954, 2024. URL <https://api.semanticscholar.org/CorpusID:267094917>.
- [39] 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](https://openreview.net/forum?id=_VjQlMeSB_J).
- [40] 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.
- [41] Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning. *arXiv preprint arXiv:2505.10320*, 2025.
- [42] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *ArXiv*, abs/2308.08155, 2023. URL <https://api.semanticscholar.org/CorpusID:260925901>.
- [43] 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*, 2025.

- [44] 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*, 2025.
- [45] 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.
- [46] 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.
- [47] Rasoul Zahedifar, Sayyed Ali Mirghasemi, Mahdieh Soleymani Baghshah, and Alireza Taheri. Llm-agent-controller: A universal multi-agent large language model system as a control engineer. *ArXiv*, abs/2505.19567, 2025. URL <https://api.semanticscholar.org/CorpusID:278905531>.
- [48] Kaiwen Zha, Zhengqi Gao, Maohao Shen, Zhang-Wei Hong, Duane S Boning, and Dina Katabi. Rl tango: Reinforcing generator and verifier together for language reasoning. *arXiv preprint arXiv:2505.15034*, 2025.
- [49] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yi Eve Sun, Chen Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, F. Yin, Yitao Liang, and Yaodong Yang. Proagent: Building proactive cooperative agents with large language models. In *AAAI Conference on Artificial Intelligence*, 2023. URL <https://api.semanticscholar.org/CorpusID:261064959>.
- [50] 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>.
- [51] 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*, 2025.
- [52] 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*, 2025.
- [53] Wenting Zhao, Pranjali 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.
- [54] Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Xinyu Yang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, et al. Parallel-r1: Towards parallel thinking via reinforcement learning. *arXiv preprint arXiv:2509.07980*, 2025.

## A AsyncThink Thinking Cases

We provide several AsyncThink thinking trajectories. Figure 9 presents an AsyncThink thinking trajectory on math reasoning. Figure 10 and Figure 11 present the thinking trajectories of the AsyncThink model, trained on the multi-solution countdown data, demonstrating its ability to generalize its asynchronous thinking capabilities to out-of-domain tasks.

## B 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<sub>i</sub>>subtask description</FORK<sub>i</sub>> to delegate work and <JOIN<sub>i</sub>> 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>

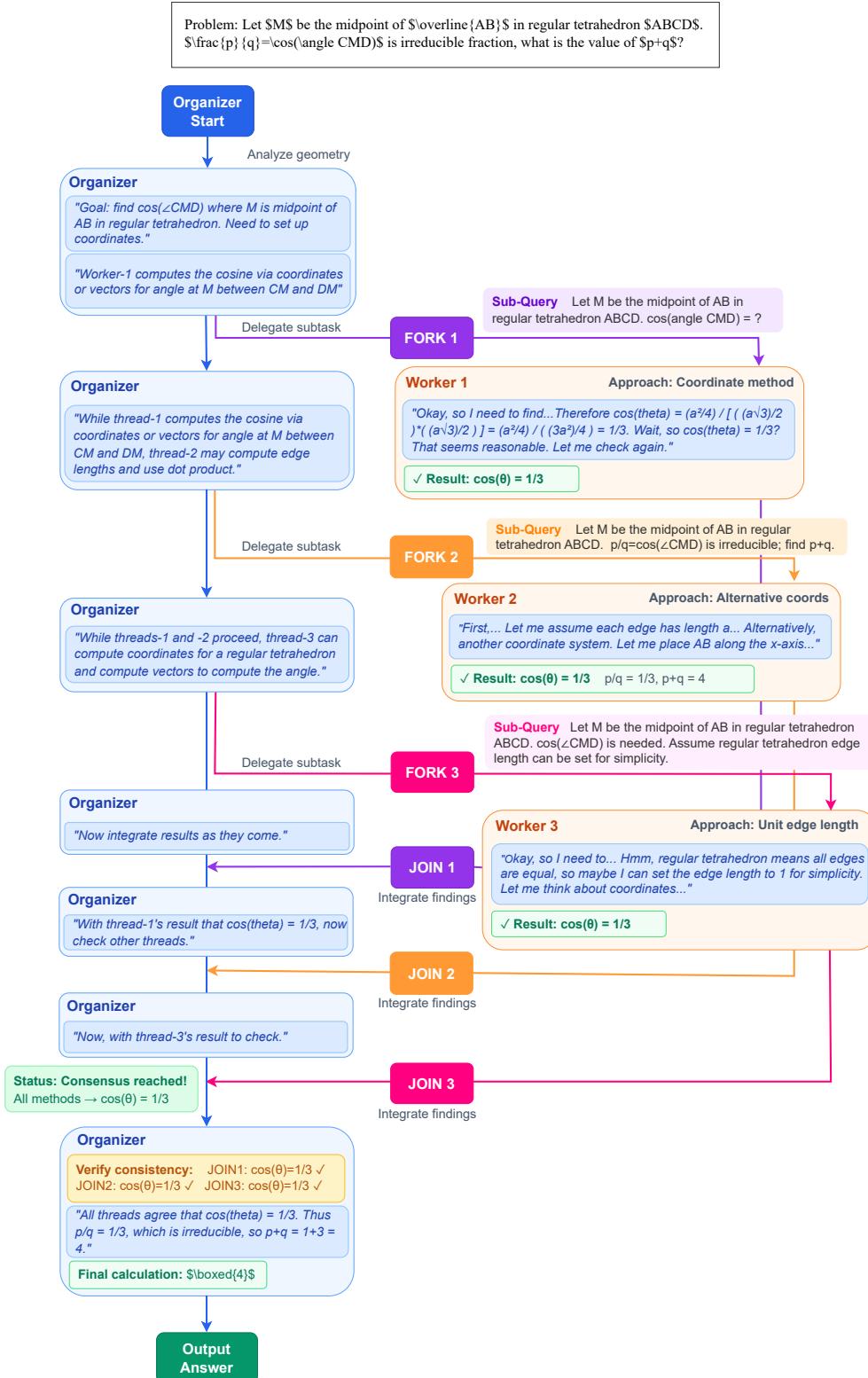


Figure 9: An AsyncThink thinking trajectory on mathematical reasoning with an agent pool capacity  $c = 4$ . The organizer spawns three workers, each using a different geometric formulation. These workers run in parallel and derive consistent results ( $\cos \theta = 1/3$ ). They then rejoin for verification, converging to the final answer.

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

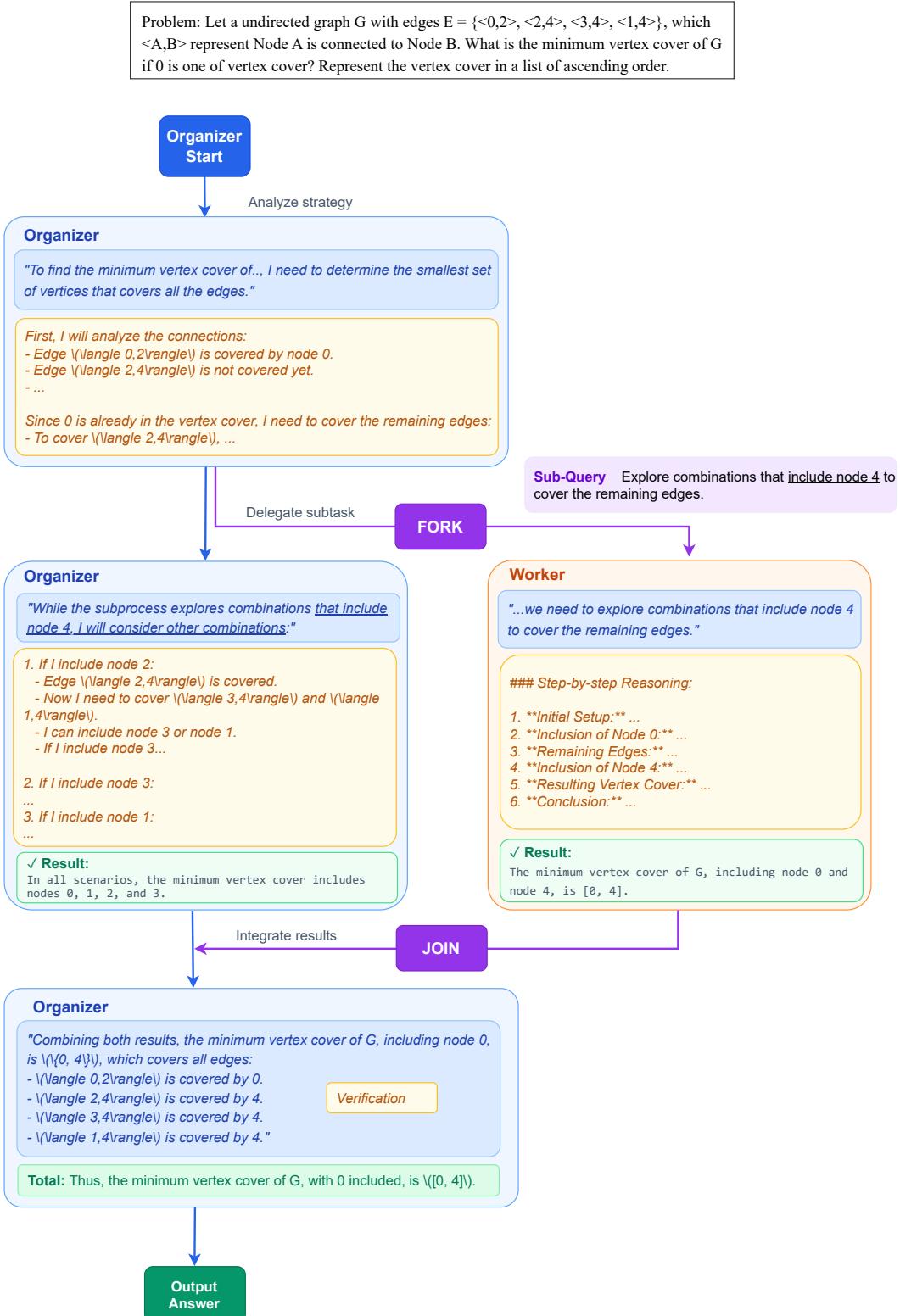


Figure 10: An example of AsyncThink on a minimum vertex cover problem from MMLU-Pro (with agent pool capacity  $c = 2$ ). The organizer generalizes to this unseen graph theory task, performing Fork and Join operations without prior training on such problems. While a worker explores combinations with node 4, the organizer concurrently analyzes alternatives, ultimately determining the minimum vertex cover  $[0, 4]$ .

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

Problem: Two 30" plants are crossed, resulting in progeny of the following ratio: one 22", eight 24", 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" ]

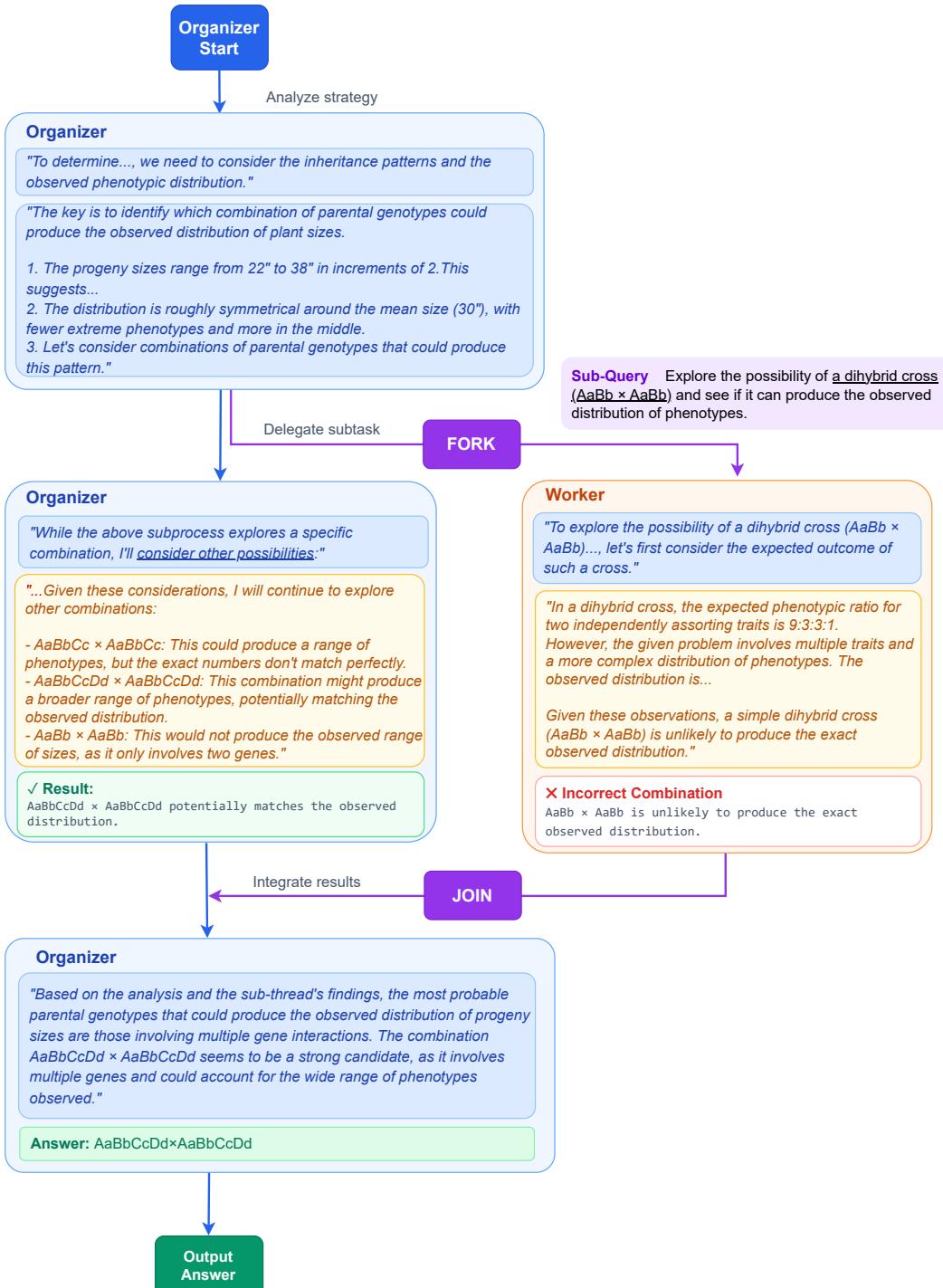


Figure 11: An example of AsyncThink on a genetics cross problem from MMLU-Pro (with agent pool capacity  $c = 2$ ). The organizer generalizes to this biology domain, performing Fork and Join operations without domain-specific training. While a worker explores the dihybrid cross ( $AaBb \times AaBb$ ), the organizer concurrently evaluates multi-gene combinations, ultimately identifying  $AaBbCcDd \times AaBbCcDd$  as the most probable parental genotypes.