

Reaching Agreement Among Reasoning LLM Agents

Chaoyi Ruan
NUS

Yiliang Wang
NUS

Ziji Shi
NUS

Jialin Li
NUS

Abstract

Multi-agent systems have extended the capability of agentic AI. Instead of single inference passes, multiple agents perform collective reasoning to derive high quality answers. However, existing multi-agent orchestration relies on static heuristic workflows such as fixed loop limits and barrier synchronization. These ad-hoc approaches waste computational resources, incur high latency due to stragglers, and risk finalizing transient agreements. We argue that reliable multi-agent reasoning requires a formal foundation analogous to classical distributed consensus problem.

To that end, we propose a formal model of the multi-agent refinement problem. The model includes definitions of the correctness guarantees and formal semantics of agent reasoning. We then introduce Aegean, a consensus protocol designed for stochastic reasoning agents that solves multi-agent refinement. We implement the protocol in Aegean-Serve, a consensus-aware serving engine that performs incremental quorum detection across concurrent agent executions, enabling early termination when sufficient agents converge. Evaluation using four mathematical reasoning benchmarks shows that Aegean provides provable safety and liveness guarantees while reducing latency by 1.2–20 \times compared to state-of-the-art baselines, maintaining answer quality within 2.5%. Consistent gains across both local GPU deployments and commercial API providers validate that consensus-based orchestration eliminates straggler delays without sacrificing correctness.

1 Introduction

A new computational pattern is reshaping how AI systems solve complex problems: Rather than relying on a single inference pass, modern systems orchestrate multiple reasoning paths that deliberate collectively to produce answers that exceed the capabilities of individual attempts. This pattern emerges in frontier models like Gemini Deep Research [16] and Grok Heavy [49] that explore diverse solution trajectories to multi-agent architectures where independent model

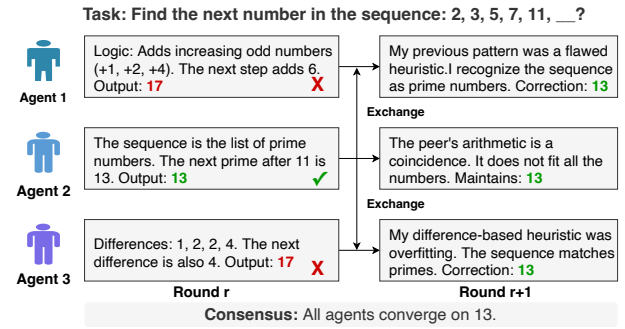


Figure 1: A multi-agent consensus example from 3 models.

instances engage in iterative discussion [13, 18, 28, 48], critique each other’s reasoning [4, 8], and progressively refine their answers [30, 35, 50]. These approaches share a common structure: Parallel reasoning processes generate candidate solutions, exchange information, and eventually converge on a single result. They face a challenge similar to the core problem in distributed consensus: How should a set of independent processes reach agreement over conflicting proposals?

Figure 1 illustrates this paradigm: Initially, Agent 1 and Agent 3 propose 17 based on flawed arithmetic, while Agent 2 correctly identifies the prime sequence and outputs 13. After exchanging reasoning traces, Agent 1 and Agent 3 recognize their error and converge to the correct answer. This *refinement process* enables higher solution accuracy than single inference passes. However, it is unclear *when* the system should output a solution and *which* solution it should output. Outputting a solution with majority support in round r would capture the wrong answer; delaying the output in rounds after $r + 1$ would unnecessarily prolong the refinement process.

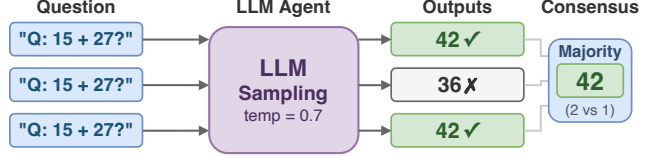
Classical distributed consensus problem addresses similar questions. Unfortunately, consensus protocols such as Paxos [25] and Raft [38] do not apply in the multi-agent refinement context. These protocols assume deterministic state machines. LLMs, however, are inherently stochastic where the same query may lead to different responses. Consensus

protocols apply quorum intersection to ensure committed values are stable. A solution supported by majority agents in a round may dissolve in the next round as agents refine their positions after considering peer rationales. Lastly, consensus protocols do not enforce constraints on the committed value. Multi-agent refinement aims to derive high-quality answers.

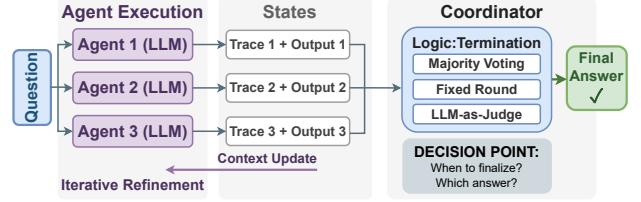
Fundamentally, existing literature [4, 8, 13, 30, 35, 50] lacks a formal model of agreement among stochastic LLM agents. The model should define the semantics of agent reasoning, answer refinement, and termination conditions. Without such a model, existing multi-agent solutions compromise on accuracy and performance. First, heuristic termination mechanisms decouple computational cost from actual convergence. The coordinators rely on fixed round limits, wasting compute on easy queries while prematurely halting complex ones. Second, barrier-based orchestration suffers high completion time due to tail stragglers. Standard workflows require all reasoning paths to complete before evaluating agreement [7, 46], causing the slowest agent to dictate latency. Table 1 shows that the slowest model has 11-64 \times higher latency than the fastest model. Lastly, systems that finalize upon observing simple majority cannot distinguish unstable states from stable agreement. They risk committing transient states that would dissolve in the next round.

Addressing these challenges requires rethinking the foundations underlying multi-agent coordination. In this work, we formalize the multi-agent refinement problem. We define the safety and liveness properties of multi-agent refinement, and provide formal semantics of agent reasoning. To validate our model, we conducted experiments on a diverse set of real-world AI models. The results provide strong empirical support for our agent refinement semantics.

We present Aegean, a consensus framework that bridges non-deterministic reasoning and deterministic system guarantees. Based on our formal model, we design an Aegean protocol that applies leader-based coordination to progress rounds of agent refinement. The protocol guarantees safety and liveness, including *refinement monotonicity* where the quality of solutions strictly improves over rounds. It also incorporates a modular refinement decision engine which defines customizable agreement conditions, such as solution similarity threshold and stability horizon. Built atop this protocol, Aegean-Serve embeds consensus coordination directly within inference infrastructure, enabling progressive quorum detection that evaluates outputs as they stream. The engine immediately cancels in-progress reasoning paths once quorum is secured, decoupling latency from stragglers. Evaluation across four reasoning benchmarks (GSM8K [9], MMLU [19, 20], AIME [17], IMO [34]) demonstrates that Aegean reduces average latency by 1.2–20 \times and P99 tail latency by up to 11 \times compared to state-of-the-art baselines. The system maintains accuracy within 2.5% while reducing token consumption by 1.1–4.4 \times through early termination when consensus stabilizes. These gains hold consistently across locally de-



(a) Single-agent consensus process



(b) Multi-agent consensus process

Figure 2: Architecture of agentic consensus. (a) Single-agent consensus aggregates samples from one model; (b) multi-agent consensus coordinates multiple agents through iterative refinement with a central coordinator.

ployed models and commercial model API, confirming that consensus-based orchestration eliminates straggler delays without sacrificing correctness.

We make the following contributions:

- We formalize the first consensus model for stochastic multi-agent LLMs, establishing round-based stability as the necessary condition for safe commitment.
- We design the first consensus protocol, Aegean, for non-deterministic participants, introducing stability horizons and quorum-based finalization with provable safety and liveness guarantees.
- We build the first consensus-aware serving engine, Aegean-Serve, enabling consensus-aware execution and achieving significant performance benefits.

2 Background

From single response to collective consensus. Large language models generate outputs through autoregressive generation, where each token is drawn from a probability distribution conditioned on prior context. This sampling process introduces inherent stochasticity: Given identical inputs, the LLM model may produce different outputs across invocations.

This phenomenon motivates a natural question: Can we leverage multiple samples to identify the correct answer more reliably than a single sample, like collective wisdom? The self-consistency approach [47] first explored this by sampling multiple reasoning paths from the same model and selecting the most frequent answer, as shown in Figure 2-(a). The insight is that by aggregating across multiple reasoning samples, the system is more likely to converge to the correct answer. However, it is limited by the knowledge of a single model.

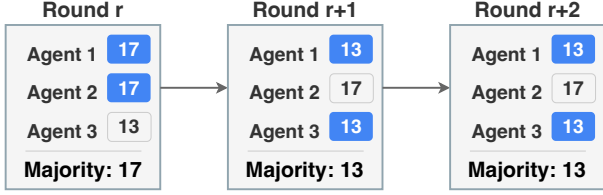


Figure 3: Consensus is unstable; reasoning exchange can flip a majority from A to B between rounds r and $r + 1$, rendering termination strategies unreliable.

Table 1: Convergence efficiency and model latency overhead using GPT-5-mini [39], Gemini-2.5-Flash [10], and DeepSeek-R1 [12]. L_{fast} , L_{med} , and L_{slow} denote the latency of the fastest, median, and slowest models per round.

| Benchmark | Convergence | Per-Round Latency (s) | | | Barrier Max(L_i) |
|-----------|-------------|-----------------------|------------------|-------------------|-------------------------|
| | Avg. Rounds | L_{fast} | L_{med} | L_{slow} | |
| GSM8K | 2.0 | 1.3 | 4.4 | 15.2 | 15.2 |
| MMLU | 2.3 | 2.5 | 5.8 | 45.0 | 45.0 |
| AIME | 4.1 | 5.8 | 29.4 | 370.6 | 370.6 |

Multi-agent systems extend this principle through two key mechanisms: debate and voting. Diverse agents engage in multi-round discussions where they critique each other’s reasoning and progressively refine their proposals [4, 13, 47]. Systems [8, 23, 27] like TUMIX [8] instantiate this pattern by having agents exchange reasoning traces across multiple rounds [22, 35], with the premise that collective deliberation can exceed individual reliability. The decision-making stage usually employs consensus mechanisms like majority voting [5, 8, 15, 27, 47], where the system commits to whichever answer receives the most votes. Note that majority voting remains the dominant approach due to its interpretability.

Figure 2-(b) abstracts the common architecture underlying these multi-agent systems. A coordinator orchestrates a panel of agents, each acting as an independent reasoning node: It receives a problem, generates a candidate solution with supporting rationale, and may revise its answer upon receiving peer solutions. The coordinator aggregates agent outputs after each round and distributes the collective context back to agents for further deliberation. This cycle continues until the termination condition is met.

The combination of parallel exploration and iterative refinement creates systems that significantly outperform single-agent inference on complex reasoning tasks. However, it also creates new problems: When should the system commit to a final result, and what does agreement mean for stochastic agents that may change their positions?

3 The Multi-Agent Consensus Problem

For systems [8, 23, 27] following the architecture in Figure 2-(b), this termination decision involves two distinct challenges: determining what constitutes effective agreement among stochastic agents, and navigating the trade-off between inference overhead and answer quality. We examine both challenges, demonstrating that classical consensus assumptions are fundamentally incompatible with LLM agents and quantifying the severe performance penalties of current approaches.

The agreement problem: consensus instability. Classical consensus protocols [25, 38] like Paxos [25] guarantee *agreement permanence*: Once a value is accepted by a majority of replicas, the value is permanently committed. These protocols leverage the *quorum intersection* property to provide such a guarantee despite node failures and network asynchrony. However, they are agnostic to the semantics of the agreed value; they permit any proposed value to be committed.

Multi-agent LLM systems face a different problem. Agents actively revise and refine their solutions based on new context from other agents’ reasoning. An agreement based on simple majority voting does not ensure persistence of the answer, since agents may further refine their solutions. It also enforces no guarantee on the answer quality — a majority of agents may incidentally arrive at the same wrong solution.

Consider the example in Figure 3. Three agents deliberate on a mathematical problem. In round r , the agents produce answers (17, 17, 13), yielding a $\frac{2}{3}$ majority for 17. However, this majority answer is neither stable nor correct. After exchanging reasoning traces, Agent 1 recognizes an error in its earlier answer, and the answer distribution shifts to (13, 17, 13) in round $r + 1$. The majority has now shifted from 17 to 13.

This instability shows that a simple majority voting approach is unreliable. Figure 3 illustrates the symptom (a shifting majority), while our experiments (Table 2) reveal the root cause: high volatility of agent reasoning. We observed that agents frequently revise their solutions after reviewing reasoning from peers. Without proper semantics and protocol designs, a multi-agent system cannot distinguish between coincident agreements that may oscillate and a stable solution that can persist as agents further refine their reasoning.

Issues with current solutions ask for a *formal model* of agreement for non-deterministic agents. Only with such formal semantics, a multi-agent protocol can reason about the convergence of refinements towards higher quality answers, and determine when a solution has reached stable consensus.

The performance trade-off. Even if agreement can be defined, detecting it imposes severe performance costs. The system must balance inference overhead (latency and token consumption) against answer quality. This trade-off manifests in two ways: across rounds (how long to deliberate) and within rounds (how long to wait for agents).

Across rounds: the inefficiency of fixed limits. Current systems [46] often rely on fixed round limits (e.g., stopping after

4 rounds) to bound computational cost. However, this heuristic ignores the varying complexity of tasks. As shown in the “Convergence” column of Table 1, both GSM8K and MMLU problems converge rapidly in 2.0 to 2.3 rounds, yet complex reasoning tasks like AIME require 4.1 rounds on average. A fixed limit wastes computation on simple problems while potentially truncating necessary reasoning on difficult ones. Fixed limits thus fundamentally misalign computational cost with the actual difficulty of the problem.

Within rounds: the bottleneck of barrier synchronization. Standard workflows enforce barrier synchronization [8, 46], waiting for all agents to complete a round before evaluating agreement [7, 46]. This couples system latency to the slowest straggler. Table 1 illustrates this disparity: On GSM8K, while the fastest model completes in 1.3s, the system must wait 15.2s for the slowest, resulting in an $11\times$ slowdown. On MMLU, this gap grows to $18\times$ (45.0s versus 2.5s). AIME exhibits the most severe disparity, where the slowest model takes 370.6s compared to the fastest at 5.8s, a $64\times$ difference. Crucially, the straggler identity varies: In our experiments, DeepSeek-R1 was the slowest in 50% of GSM8K rounds, GPT-5-mini in 42%, and Gemini in 8%. This variability makes static exclusion strategies infeasible. When a sufficient subset of agents has already converged, waiting for unpredictable stragglers negates the benefits of parallel exploration without improving the outcome.

Navigating this trade-off requires efficient commitment semantics that finalize an answer as soon as sufficient agreement is detected, without waiting for redundant responses. However, such semantics are only meaningful once the agreement problem is solved. This raises the central question of this paper: How can a multi-agent system determine when agreement is sufficiently stable to commit safely, while still terminating efficiently? Answering this requires a principled framework that formalizes agreement among stochastic agents. No such framework exists today.

In the following sections, we bridge this gap by formalizing multi-agent refinement as a consensus problem, establishing the theoretical foundation for safe and efficient termination.

4 Formalizing Multi-Agent Refinement

4.1 Problem Formulation

In classical consensus problems, a group of processes proposes and decides on some arbitrary values. A correct consensus protocol is defined to provide the following guarantees:

- **Validity:** If a correct process decides on a value v , v is proposed by a correct process.
- **Agreement:** If values v and v' are decided on any correct processes, then $v = v'$.
- **Termination:** Eventually, every correct process decides some value v .

Our work targets collaborative multi-agent refinement, which is similar, but not identical, to the consensus problem. In our setting, a group of agents is initialized with a common task (how this task is distributed to the agents is irrelevant to the problem). Each agent can generate a solution to the task based on its local LLM model and context. When an agent thinks a solution is ready, it can *output* the solution. Note that an agent can output multiple solutions. A correct multi-agent refinement protocol shares a similar *termination* property to a consensus protocol:

- **Refinement Termination:** Eventually, some correct agent outputs a solution s .

The original validity property of the consensus problem is too weak for multi-agent refinement: It accepts a solution even if it is generated by the weakest agent. Our goal is to generate refined solutions that are better than those generated by individual agents. Determining which solution is “better” requires a function that evaluates the quality of answers. We define such a function as a *Quality Oracle*:

Definition 1 (Quality Oracle). A Quality Oracle Q is a function ($Q : S \times S \rightarrow \mathbb{R}$) that takes two strings as input, where the first string is a task description and the second input is a solution, and outputs a real number representing the quality of the solution to the task.

We assume Q is a proper function, i.e., given the same input strings, the output value is deterministic. Note that this quality oracle is unknown to any agent. To derive a meaningful validity requirement, we define a *majority optimal solution* as follows. Suppose that each agent can individually generate a solution, a *majority optimal solution* is the best solution (evaluated by Q) among the individual solutions generated by a simple majority of agents. Now, we can define the validity property of multi-agent refinement:

- **Refinement Validity:** The quality of any output solution s is *at least as good* (evaluated by Q) as that of *some* majority optimal solution.

The usefulness of our refinement validity property depends on the distribution of agent reasoning capabilities. In a deployment where the majority of agents exhibit strong reasoning capabilities, the property ensures that any output solution has high quality: By majority intersection, any majority of agents will include at least one strong agent. Due to the majority optimal property, an output solution will therefore have at least the quality of a strong agent. Moreover, refinement validity only specifies a *lower bound* on the solution quality. Practical deployments of multi-agent refinement protocols can produce solutions with stronger properties, as we demonstrate in §4.3.

The combination of termination and validity creates a problem that cannot be solved trivially. Without the validity requirement, any agent could simply output its own initial solution. Without the termination requirement, agents could deliberate indefinitely. The interaction between these prop-

erties requires careful protocol designs to ensure that agents converge to high-quality solutions within bounded time.

Classical consensus requires strict agreement: All correct processes must decide on the same value. For multi-agent refinement, this requirement is unnecessarily restrictive. The primary objective is solution quality rather than uniqueness. We therefore relax agreement to a monotonicity property that permits multiple output solutions, provided that the quality never decreases:

- **Refinement Monotonicity:** If a solution s' is outputted after solution s (in real time), the quality of s' is *at least as good* (evaluated by Q) as that of s .

This property ensures that the refinement process makes monotonic progress toward higher-quality solutions. Even if the protocol generates multiple solutions over time, each subsequent output represents an improvement or maintains the current quality level.

Failure and network models. We assume that at most f agents can fail. We target a *fail-stop model* in which a failed agent stops responding indefinitely but always follows the protocol precisely. A failure may occur due to server crashes, network partitions, or an indefinite reasoning procedure in an agent. The network is assumed to be *partially synchronous* [14], i.e., there exists a finite but unknown period (commonly known as the global stabilization time) after which all messages are delivered within bounded delay.

4.2 Agent Operational Semantics

A key capability of an agent is to perform reasoning (e.g., using an LLM model) based on task descriptions and context to generate solutions. We now formalize the operational semantics of an agent and the assumptions we make. Given the stochastic nature of existing AI models, proving formal guarantees of an agent is outside the scope of this work. Nevertheless, we provide empirical evidence in §4.3 to demonstrate that our semantics and assumptions are sound in practice.

We abstract an agent as a *state machine*, which maintains a context state c . The exact representation of the context state is irrelevant to our problem. The state machine exposes a reasoning function R with the following definition:

Definition 2 (Agent Reasoning Function). An agent exposes a reasoning function R ($R: c \times S \rightarrow c' \times S$) that takes the current agent context c and a string as input. The string is either a task description or a list of reasoning traces. R generates a new agent context c' , and outputs a string. The string includes both a task solution and a reasoning trace of this step.

In practice, agents implicitly consume their current context and update their context when performing reasoning steps. Therefore, we omit the context input/output when specifying the reasoning function (e.g., simply writing $s_o \leftarrow R(s_i)$).

When an agent receives a task description, we do not make explicit assumptions about the quality of its reasoning func-

Table 2: Accuracy before and after agent exchange. “Init.” and “Post” denote the accuracy before and after exchanging answers. “Changes” indicates the count of decision flips.

| Dataset | Model | Init. | Post | Δ | Changes |
|---------|--------------|-------|-------|----------|---------|
| GSM8K | Llama 3.1-8B | 88% | 96% | +8% | 17/100 |
| | DeepSeek-R1 | 98% | 98% | 0% | 1/100 |
| MMLU | Llama 3.1-8B | 43% | 79% | +36% | 64/100 |
| | DeepSeek-R1 | 62% | 63% | +1% | 1/100 |
| AIME | Llama 3.1-8B | 3% | 54.4% | +51% | 70/90 |
| | DeepSeek-R1 | 70% | 72% | +2% | 10/90 |

tion output (as evaluated by Q). However, we assume that all agents are capable of *reasoning refinement*, i.e., given a set of reasoning traces and solutions produced by other agents (or possibly from itself), an agent can improve its own reasoning to generate solutions that match the highest quality answer:

Assumption 1 (Reasoning Refinement). Given a set of solutions and their reasoning traces S , suppose an agent applies its reasoning function on a string \bar{s} that combines all solutions in S to produce a new solution. Then the quality of the output solution is *at least as good* (evaluated by Q) as any solution in S , i.e., $\forall s \in S, Q(R(\bar{s})) \geq Q(s)$

4.3 Agent Semantics Validation

Safety and liveness properties of a multi-agent refinement protocol rely on the reasoning refinement assumption. We now conduct experiments to validate whether real-world AI models conform to this assumption.

We evaluate the reasoning refinement assumption through a collaborative exchange protocol pairing models of different reasoning capacities. A stronger model (DeepSeek-R1) shares its reasoning traces and solutions with a weaker model (Llama3.1-8B-Instruct), and vice versa. Under the refinement assumption, we expect the weaker model to improve by learning from the stronger model’s reasoning, while the stronger model should maintain stable performance when exposed to potentially inferior traces. We evaluate on three benchmarks spanning different reasoning datasets: AIME [17], MMLU [19, 20], and GSM8K [9].

Table 2 summarizes the results. The weaker model exhibits substantial accuracy improvements across all benchmarks after receiving reasoning traces from DeepSeek-R1: Accuracy increases from 3% to 54.4% on AIME-Validation (+51%), from 43% to 79% on MMLU (+36%), and from 88% to 96% on GSM8K (+8%). In contrast, the stronger model maintains remarkable stability, with minimal accuracy changes on all datasets. The low decision change counts for DeepSeek-R1 (1/100 on GSM8K, 1/100 on MMLU, 10/90 on AIME) indicate that strong models can filter inferior reasoning rather than blindly adopting it.

These results validate two properties central to [Assumption 1](#). First, weaker models benefit from superior reasoning traces, achieving significant improvement. Second, strong models maintain quality when observing inferior reasoning, confirming quality-preserving refinement. This asymmetry is crucial for protocol correctness: Agents selectively adopt reasoning only when it represents an improvement, ensuring that collective deliberation produces monotonically improving reasoning. These empirical findings justify our reliance on the reasoning refinement assumption for the protocol guarantee.

5 Aegean Protocol

In this section, we introduce Aegean, a concrete protocol that correctly implements multi-agent refinement ([§4](#)).

5.1 Protocol Overview

Agent ensemble. An Aegean deployment consists of N agent processes, denoted A_i where i ranges from 0 to $N - 1$, forming an LLM ensemble. Agents have complete network connectivity. We assume a partially synchronous [\[14\]](#) network, and at most $\lceil \frac{N-1}{2} \rceil$ agents may suffer fail-stop failures. An Aegean deployment takes as input a single task description, and outputs one or multiple solutions to the task. The protocol guarantees all properties of multi-agent refinement, as defined in [§4](#). The protocol can be easily extended to a multi-instance version; details of such a protocol variant are omitted.

The protocol progresses in *terms*. In each term, there is at most one leader agent. We use standard leader election protocols [\[25, 38, 44\]](#) to elect a leader agent. When the leader agent in a term becomes unreachable, the remaining agents advance to the next term and start a new leader election. If a leader election fails to make progress, the agents move to the next term and retry a new election. The first term begins with a predetermined leader or a leader election protocol.

In each term, the leader coordinates all agents to generate refined solutions. The protocol within a term is divided into *rounds*. All rounds share the same message pattern: The leader distributes the quorum refinement set from the previous round to all agents; each agent performs refinement and sends the refined solution to the leader; the leader aggregates a quorum of refined solutions for the next round. Leveraging the reasoning refinement ([Assumption 1](#)) and the quorum intersection properties, the quality of solutions improves monotonically in each round. Once the leader receives a quorum of solutions in a round, it can output a solution from the refinement set in the previous round. Such outputs guarantee refinement validity and monotonicity, even in the event of leader failures. To decide when and which solution to output from refinement sets, the protocol incorporates a modular *refinement decision engine*. Users can customize the solution selection criteria in the engine. [Figure 4](#) and [algorithm 1](#) show the protocol’s refinement timeline within each term and the overall workflow.

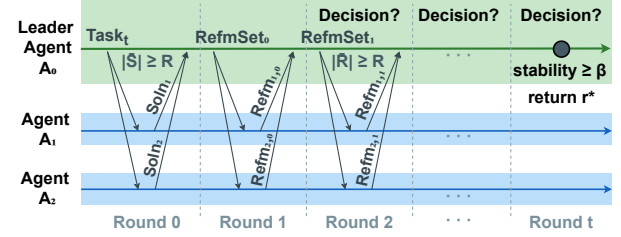


Figure 4: Execution timeline of the Aegean protocol within one term. Initially, the leader broadcasts the task and gathers solutions until quorum ($|\bar{S}| \geq R$). During refinement, the leader broadcasts the reference set and workers respond with their references. This process repeats until a candidate is established ($|\bar{R}| \geq R$). Finalization occurs when the stability counter reaches β . For brevity, we abbreviate $\langle \text{Task}, \text{term-num}, t \rangle$ as Task_t , $\langle \text{Soln}, \text{term-num}, id, s \rangle$ as Soln_{id} , $\langle \text{RefmSet}, \text{term-num}, \text{round-num}, \bar{R} \rangle$ as RefmSet_{num} , and $\langle \text{Refm}, \text{term-num}, id, \text{round-num}, r \rangle$ as $\text{Refm}_{id,num}$.

When a new leader is elected in a new term, the leader collects a quorum of refinement sets before proceeding to the first round of the term. This ensures that the quality of solutions in the new term exceeds any previous output. The design resembles existing leader and view change protocols [\[37, 44\]](#).

5.2 Protocol Specification

Algorithm 1: Multi-Agent Refinement Protocol

Input: Task t ; agent pool \mathcal{A}

Output: Refined solution r^*

```

1  $r^* \leftarrow \perp$ ;  $\mathcal{L} \leftarrow \perp$ ;  $\bar{R} \leftarrow \perp$ ; round-num  $\leftarrow 0$ ;
2 if  $\mathcal{L} = \perp$  or  $\mathcal{L}$  unreachable then
3    $\mathcal{L} \leftarrow \text{LeaderElection}()$ ;
4    $\mathcal{L}$  broadcasts  $\langle \text{Task}, \text{term-num}, t \rangle$ ;
5    $\forall a \in \mathcal{A} : s \leftarrow \mathcal{R}_a(t)$ ; Sends  $\langle \text{Soln}, \text{term-num}, id, s \rangle$  to  $\mathcal{L}$ ;
6    $\mathcal{L}$  collects quorum  $\text{Soln}$ ;  $\bar{R} \leftarrow$  set of  $s$  in the  $\text{Soln}$  quorum;
7   while  $r^* = \perp$  do
8     round-num  $+= 1$ ;
9      $\mathcal{L}$  broadcasts  $\langle \text{RefmSet}, \text{term-num}, \text{round-num}, \bar{R} \rangle$ ;
10     $\forall a \in \mathcal{A} : r \leftarrow \mathcal{R}_a(\bar{R})$ ; Sends  $\langle \text{Refm}, \text{term-num}, id, \text{round-num}, r \rangle$  to  $\mathcal{L}$ ;
11     $\mathcal{L}$  collects quorum  $\text{Refm}$ ;  $\bar{R} \leftarrow$  set of  $r$  in the  $\text{Refm}$  quorum;
12     $\mathcal{L}$  inputs  $(\bar{R}, \text{round-num})$  into decision engine;
13 return  $r^*$ ;

```

Agent protocol state. We show the protocol state each agent maintains in [Table 3](#). In each term, at most one agent can have role set to leader. Agents tag their current *term-num* in all protocol messages.

Leader election. Aegean uses a leader election protocol

Table 3: Protocol state variables maintained by each agent.

| Variable | Description |
|-----------|---|
| term-num | Current term number |
| role | Role: leader, candidate, or worker |
| id | Unique agent ID |
| round-num | Current refinement round within term |
| refmset | Most recent refinement set \bar{R} received |

similar to Raft [38] (line 3). Unlike Raft, any agent is eligible for the leader role in a term; the leader change protocol ensures protocol safety. When an agent enters a new term, it broadcasts a $\langle \text{RequestVote}, \text{term-num} \rangle$ message to all agents and transitions to candidate state. When an agent receives a RequestVote with a higher term-num, it advances its term-num, transitions to worker, and replies a $\langle \text{Vote}, \text{term-num}, \text{id} \rangle$ to the candidate. Once a candidate receives a quorum of Vote from unique agents (it always votes for itself) for its current term, it transitions to leader state. An agent can only cast one Vote in each term. If the protocol fails to make progress, agents will time out, advance to the next term, and retry leader election.

Refinement protocol. As shown in Figure 4, all agents receive an identical task t before the protocol starts. Details of this task distribution are irrelevant to Aegean. When an agent transitions to the leader role in the first term, it broadcasts a $\langle \text{Task}, \text{term-num}, t \rangle$ to all agents (line 4). Upon receiving the message, a worker agent applies its reasoning function \mathcal{R} on task t to derive its initial solution s . Then, it sends a $\langle \text{Solution}, \text{term-num}, \text{id}, s \rangle$ to the leader (line 5). The leader collects a quorum of Solution from unique agents (including from itself), and broadcasts to all agents a $\langle \text{RefmSet}, \text{term-num}, \text{rounds-num}, \bar{R} \rangle$, where round-num is set to 1 and \bar{R} contains all the solutions in the quorum (line 6).

Once an agent receives a RefmSet with a round-num no less than its own, it stores \bar{R} in its refmset and updates its round-num. It then applies its reasoning function \mathcal{R} on \bar{R} to derive a refined solution r . Next, the agent sends a $\langle \text{Refm}, \text{term-num}, \text{id}, \text{round-num}, r \rangle$ to the leader (line 10). The leader collects a quorum of Refm from unique agents with round-num matching its own, and inputs the set of refinements \bar{R} and the round-num to the refinement decision engine (line 12). Then, the leader advances the round-num (line 8) and again broadcasts a $\langle \text{RefmSet}, \text{term-num}, \text{round-num}, \bar{R} \rangle$ to all agents.

The workers and the leader repeat the refinement loop until the refinement decision engine outputs a solution (line 13). The leader then returns the solution to the client.

Leader change. When a new leader is elected, it broadcasts a $\langle \text{NewTerm}, \text{term-num} \rangle$ to all agents. If an agent receives a NewTerm with a term-num greater than its own, it replies a $\langle \text{NewTermAck}, \text{term-num}', \text{term-num}, \text{id}, \text{round-num}, \bar{R} \rangle$ to the new leader, where term-num' is the new term and

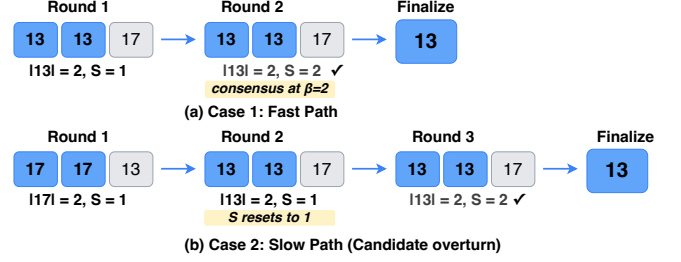


Figure 5: Aegean consensus execution examples with $\alpha = 2$ and $\beta = 2$ among 3 agents. Three boxes in each round represent responses from three agents.

\bar{R} is its current refmset. It then advances its term-num to that in NewTerm. When the new leader receives a quorum of NewTermAck with term-num' matching its own, it finds the NewTermAck with the highest term-num, and in case of ties, the one with the highest round-num. Next, the leader sets its round-num to 1, and broadcasts a $\langle \text{RefmSet}, \text{term-num}, \text{round-num}, \bar{R} \rangle$ to all agents, where \bar{R} is from the selected NewTermAck. The agents and the leader then proceed with the normal refinement protocol.

Refinement decision engine. The protocol defines a refinement decision engine for determining *when* to output a solution and *which* solution to output. The engine takes as input a term-num, a round-num, and a set of refined solutions \bar{R} . Any solution in a refinement set satisfies refinement validity. However, to guarantee refinement monotonicity, the engine can only output a solution in round i once it receives a refinement set in round $i + 1$.

Beyond the minimum protocol safety requirements, users can define additional criteria for outputting a solution. For instance, a user can enforce a similarity threshold α : The engine only emits a solution if at least α solutions in a refinement set are semantically equivalent to the solution. Semantic equivalence is evaluated using standard vector embedding similarity or exact match. Requiring multiple agents to agree filters stochastic errors that affect individual outputs. A user can also apply a stability horizon threshold β : The engine outputs a solution only after the solution or its semantical equivalence appears in β consecutive rounds of the refinement sets. Equivalence across rounds is determined using exact match, LLM-as-judge, or both based on tasks. β -round persistence ensures the agreement survives continued deliberation rather than transient coincidence.

Figure 5 illustrates two consensus scenarios with α, β termination condition. Case 1 shows a fast consensus path where two agents initially agree on value 13 in Round 1. This majority remains stable through Round 2, satisfying the $\beta = 2$ stability requirement for finalization. Case 2 demonstrates a slow consensus path where stability validation proves necessary. Two agents initially agree on value 17 in Round 1, but the majority shifts to value 13 in Round 2. This overturn re-

sets the stability counter to 1. By Round 3, value 13 maintains majority support for two consecutive rounds, finally meeting the $\beta = 2$ requirement and allowing finalization.

5.3 Correctness Proof

We now prove that the Aegean protocol guarantees multi-agent refinement safety (refinement validity and monotonicity) and liveness (refinement termination) as defined in §4.1.

Lemma 1 (Refinement Monotonicity). *If Aegean outputs a solution s' after solution s in real time, s' has equivalent or better quality than s .*

Proof. First, we show that Aegean outputs at most one solution in a round within a term. Each term has at most one leader due to the leader election protocol. Since only a leader can output a solution, and a leader outputs at most one solution in a round, the proposition is true.

Next, we show that within a term, any output solution s' in round $i + 1$ has equivalent or better quality than any solution s in round i . We denote the refinement set in round i by \bar{R}_i . An output solution s in round i , if exists, is contained within \bar{R}_i . In round $i + 1$, any refined solution r is generated by performing the agent reasoning function on \bar{R}_i . Given the reasoning refinement assumption (Assumption 1), the quality of r is at least as good as s . As such, any solution in the round $i + 1$ refinement set \bar{R}_{i+1} has quality equal to or better than s . Since any output solution s' in round $i + 1$ is contained within \bar{R}_{i+1} , the quality of s' is at least as good as s . Because output solutions in real time order obey round order, the lemma holds within a round.

Lastly, we show that any output solution s' in term $t + 1$ has equivalent or better quality than any output solution s in term t . Given the proposition we just proved, we only need to show the case for the last output solution s in term t and the first output solution s' in term $t + 1$. Since s is an output solution, s is contained within a refinement set \bar{R} , and a quorum of agents has stored \bar{R} locally. When the leader in term $t + 1$ receives a quorum of `NewTermAck`, at least one `NewTermAck` contains \bar{R} due to quorum intersection. As s is the last output solution in term t , the leader will include \bar{R} in its initial `RefmSet`. Following the same reasoning in the proof of the previous proposition, any first output solution s' in term $t + 1$ has equivalent or better quality than any solution in \bar{R} , including s . \square

Lemma 2 (Refinement Validity). *If Aegean outputs a solution s , s satisfies refinement validity.*

Proof. Prove by induction. We first prove the base case. The first leader collects a quorum of solutions \bar{S} from distinct agents. Suppose the highest quality solution in \bar{S} is s^* , s^* is then a majority optimal solution. In the first round, any refined solution r is generated by performing the agent reasoning function on \bar{S} . Given the reasoning refinement assumption

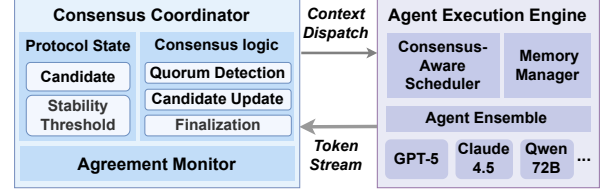


Figure 6: The architecture of Aegean-Serve. The consensus coordinator maintains protocol state and detects quorum. It directs the agent execution engine, which optimizes resource usage through ensemble-aware scheduling.

(Assumption 1), the quality of r is at least as good as any solution in \bar{S} , including s^* . Since any output solution s in the round is selected from these refined solutions, s is at least as good as s^* . This proves the base case. The inductive step is trivially true following Lemma 1. \square

Lemma 3 (Refinement Termination). *Eventually, Aegean outputs a solution s .*

Proof. Aegean assumes a partially synchronous network model. After periods of asynchrony, all messages are delivered within bounded time. The leader election protocol is guaranteed to elect a leader. Given that at most f agents can fail, eventually a correct agent becomes the leader. Since at least a quorum of agents are alive, this leader can collect a quorum of `RefmSet` within a round. After two refinement rounds, the leader can output a solution. \square

Theorem 1 (Correctness). *The Aegean protocol correctly implements multi-agent refinement.*

Proof. The correctness theorem follows directly from Lemma 1, Lemma 2, and Lemma 3. \square

6 Consensus-Aware Serving Infrastructure

The protocol specification in §5 builds formal consensus but abstracts away execution concerns. This section presents Aegean-Serve, a serving infrastructure that exposes consensus semantics to the scheduling layer, enabling optimizations that neither consensus-oblivious serving systems nor serving-oblivious consensus implementations can achieve.

6.1 System Architecture

As shown in Figure 6, Aegean employs a two-layer architecture. The *consensus coordinator* maintains protocol state including the current candidate answer, stability threshold, and round history. It executes the consensus logic from algorithm 1, tracking when a value achieves quorum and monitoring progress toward the stability horizon. Its embedded *agreement monitor* tracks agent responses incrementally and

evaluates quorum predicates as answers arrive, enabling early termination without barrier synchronization. The *agent execution engine* manages inference through a *consensus-aware scheduler* that treats agent ensembles as first-class scheduling units, and a *memory manager* that handles KV cache allocation across the heterogeneous agent pool.

Universal engine abstraction. To enable Aegean’s optimizations across diverse LLM backends with varying model architectures, tokenizers, and memory layouts, we define a minimal interface that execution engines must implement:

```
def Dispatch(q: str, ctx: ConsensusCtx,
            eid: int) -> Handle
def OnComplete(h: Handle, cb: Callable[[
    Answer], None])
def Cancel(h: Handle) -> bool
def QueryEnsemble(eid: int) ->
    EnsembleState
```

These methods expose both inference functionality and consensus-aware coordination. The `Dispatch` method initiates agent execution with attached consensus context (current candidate, stability threshold, round number), returning a handle for subsequent operations. The `OnComplete` method registers a callback invoked upon generation completion, delivering the normalized answer to the agreement monitor. The `Cancel` method terminates in-progress generation for early quorum exploitation, returning resources to the memory pool. Finally, `QueryEnsemble` retrieves aggregate state for all agents sharing an ensemble identifier, enabling collective admission decisions and ensemble-aware preemption policies. This abstraction decouples consensus logic from backend-specific implementation details, allowing Aegean to integrate with systems like vLLM [45] with minimal adaptation.

The layers interact through an event-driven design. The coordinator dispatches execution requests annotated with consensus context. As agents complete generation, the engine delivers normalized answers to the agreement monitor. When the monitor detects that at least α agents agree on the same answer and meet the termination criteria, it signals the coordinator to update the candidate. It also issues cancellation directives for in-progress agents whose answers can no longer affect the round outcome.

6.2 Ensemble-Aware Scheduling

Traditional LLM serving systems treat each inference request as an independent scheduling unit. This abstraction is insufficient for consensus-based multi-agent execution, where the outputs of multiple agents are collectively evaluated. We introduce the *agent ensemble* as a first-class scheduling primitive: a set of n inference requests that share a common query, whose outputs will be aggregated for quorum evaluation.

Collective admission and scheduling. Rather than admitting requests individually, when a new query arrives, the scheduler evaluates whether sufficient resources exist for the entire

ensemble before admitting any member. This collective policy prevents scenarios where partially admitted ensembles consume resources while waiting indefinitely for peers that cannot be scheduled. Specifically, the scheduler checks that GPU memory can accommodate n concurrent KV caches and that system load allows at least α agents to complete within the round timeout. Once admitted, the scheduler co-schedules the ensemble to synchronize their progress through the decode phase. This minimizes completion time variance, reducing the window during which completed agents hold resources while awaiting others and enabling earlier quorum detection.

Ensemble-aware termination and preemption. The agreement monitor implements progressive cancellation to exploit early consensus within an ensemble. Upon each agent completion, the monitor increments the support count for the returned answer. When a solution receives α votes and sustains for β rounds, the coordinator declares round termination and cancels in-progress agents through the scheduler’s abort interface, returning their KV cache blocks to the free pool immediately.

Also, under memory pressure, request-level preemption policies can fragment ensembles by evicting some agents while others continue, potentially leaving insufficient agents for a quorum. Our ensemble-aware policy treats the ensemble atomically: Before preempting any agent, the scheduler verifies that at least α agents would survive. If this condition fails, the scheduler either preempts the entire ensemble for later retry or protects near-completion ensembles from preemption.

Protocol-aware resource management. The protocol’s consensus state enables predictive resource allocation. Once a candidate is established with s consecutive rounds of stability, at most $S - s$ additional rounds remain before finalization. The coordinator exploits this bounded horizon by sending *reservation hints* specifying expected remaining rounds and reduced ensemble size ($R + 1$ agents rather than n), preventing nearly-finalized queries from being starved by new arrivals. Conversely, upon finalization, an *eager release directive* triggers immediate resource reclamation rather than waiting for garbage collection.

6.3 Failure Handling

The coordinator detects hard failures (crashed processes) through periodic heartbeats and soft failures (stalled generation) through completion timeouts. Upon failure, if the remaining healthy agents H_t satisfy $|H_t| \geq \alpha$, execution continues normally with the failed agent marked as non-participating. When $|H_t| < \alpha$, the coordinator aborts and restarts if no candidate exists; otherwise, it preserves the current candidate and transitions to the next round with a fresh ensemble. This policy ensures failures never violate safety: A candidate value remains a candidate, and finalization occurs only after the required S consecutive confirmations, regardless of failures.

7 Experiment

We implement Aegean atop vLLM v0.10 [45], modifying around 2000 lines of Python code.

7.1 Experimental Setup

Testbed. To evaluate Aegean under realistic serving conditions, we deploy the system on $8 \times$ NVIDIA H100 GPUs (80GB) using vLLM v0.10 [45] as the inference backend for locally hosted models. For commercial model evaluation, we access them through the OpenRouter API [21], which provides a unified interface to multiple providers while preserving their native latency characteristics.

Agent ensemble. We build an agent pool to reflect production deployments where models of varying capabilities collaborate. The pool comprises five open-source models deployed locally: Llama-3.1-8B-Instruct [36], Qwen3-8B [42], GPT-OSS-20B [40], Mistral-7B-Instruct [1], DeepSeek-Qwen-2.5-7B [11], each allocated to a dedicated H100. We also evaluate three commercial models of comparable capability (GPT-5-mini [39], Gemini-2.5-flash [10], Claude-4.5-Haiku [2]) via public API to demonstrate Aegean’s generalizability beyond local deployments.

Benchmarks and workload. We select four widely-adopted reasoning benchmarks for multi-agent systems of varying difficulty: GSM8K [9], MMLU [19, 20], AIME [17] and IMO datasets [34]. This diversity allows us to examine whether Aegean adapts computational effort to problem complexity. To simulate production traffic patterns, query submitters generate Poisson arrivals with varying request rates.

Baselines and configuration. We compare Aegean against the following baselines. *Multi-Agent-Base* faithfully reproduces the core mechanisms of state-of-the-art multi-agent systems [5, 8, 13, 23, 27] like TUMIX [8]. *Multi-Agent-Base* adopts per-round barrier synchronization where all agents must complete before proceeding, heuristic-based round limits, and majority voting for final answer selection. *Best/Worst single model* selects the highest/lowest-accuracy model within the LLM ensemble of each experiment, establishing single-model performance bounds. We implement all baselines in a unified codebase with the same agent ensemble; Aegean differs only in consensus detection and termination strategy. Unless otherwise specified, Aegean uses default parameters: $n=3$ agents, $\alpha=2$ (majority quorum size), $\beta=2$ rounds (stability horizon threshold), $T_{\max}=5$ rounds.

Metrics. We report metrics that capture both efficiency and accuracy. Average latency and P99 tail latency characterize typical and worst-case response times, respectively. Accuracy measures the fraction of problems answered correctly.

7.2 End-to-End Performance

Our evaluation first verifies that Aegean achieves its primary design goal of reducing latency while preserving accuracy compared to baseline systems.

Local model deployment. Figure 7 compares Aegean against *Multi-Agent-Base* with varying maximum rounds (4, 5, 6) on four reasoning benchmarks using locally deployed models. The top row presents average latency while the bottom row shows P99 latency, capturing both typical and tail-case performance characteristics.

For average latency, Aegean maintains consistently low values even as request rate increases from 0.1 to 1.0 requests/sec. On simpler benchmarks like GSM8K and MMLU, Aegean sustains latency below 100s throughout all load levels, while *Multi-Agent-Base* degrades substantially at 1.0 requests/sec: On GSM8K, MaxRound=4/5/6 configurations reach 126s, 237s, and 277s respectively ($1.9\times$, $3.5\times$, $4.1\times$ higher than Aegean); on MMLU, they reach 108s, 337s, and 353s ($2.7\times$, $8.4\times$, $8.8\times$ higher). For more challenging benchmarks, Aegean achieves latency around 325s on AIME and 3993s on IMO at the highest request rate. In contrast, *Multi-Agent-Base* exhibits severe latency degradation under load: On AIME at 1.0 requests/sec, MaxRound=4/5/6 reach 937s, 2662s, and 6571s ($2.9\times$, $8.2\times$, and $20.2\times$ higher); on IMO, they reach 4889s, 6743s, and 8138s ($1.2\times$, $1.7\times$, $2.0\times$ higher).

The P99 latency results further highlight the advantage of quorum-based execution. On GSM8K and MMLU, Aegean maintains P99 latency of 153s and 286s at maximum load. In comparison, *Multi-Agent-Base* shows consistently higher tail latency: On GSM8K, MaxRound=4/5/6 reach 291s, 336s, and 394s ($1.9\times$, $2.2\times$, and $2.6\times$ higher); on MMLU, they reach 328s, 542s, and 681s ($1.1\times$, $1.9\times$, and $2.4\times$ higher). The gap widens dramatically on harder benchmarks: At 1.0 requests/sec, Aegean achieves P99 latency of 772s on AIME, while MaxRound=4/5/6 reach 1595s, 2782s, and 8749s ($2.1\times$, $3.6\times$, and $11.3\times$ higher). On IMO, Aegean achieves 5391s compared to 6175s, 8097s, and 9888s for MaxRound=4/5/6 ($1.1\times$, $1.5\times$, and $1.8\times$ higher). The controlled tail latency shows that Aegean provides bounded worst-case performance, critical for meeting service level objectives in production.

Figure 7 reveals distinct saturation behavior between the two approaches. *Multi-Agent-Base* shows exponential latency growth as the system queues requests waiting for slow agents to complete all rounds. In contrast, Aegean’s quorum-based termination processes queries without waiting for stragglers, resulting in sub-linear growth under increasing load. The performance gap becomes more pronounced on harder benchmarks like AIME and IMO, where individual agent response times exhibit higher variance.

Public API deployment. Given that production systems increasingly rely on commercial APIs, we further evaluate on cloud provider APIs. Figure 8 evaluates the same systems using commercial frontier models (GPT-5-mini, Claude-

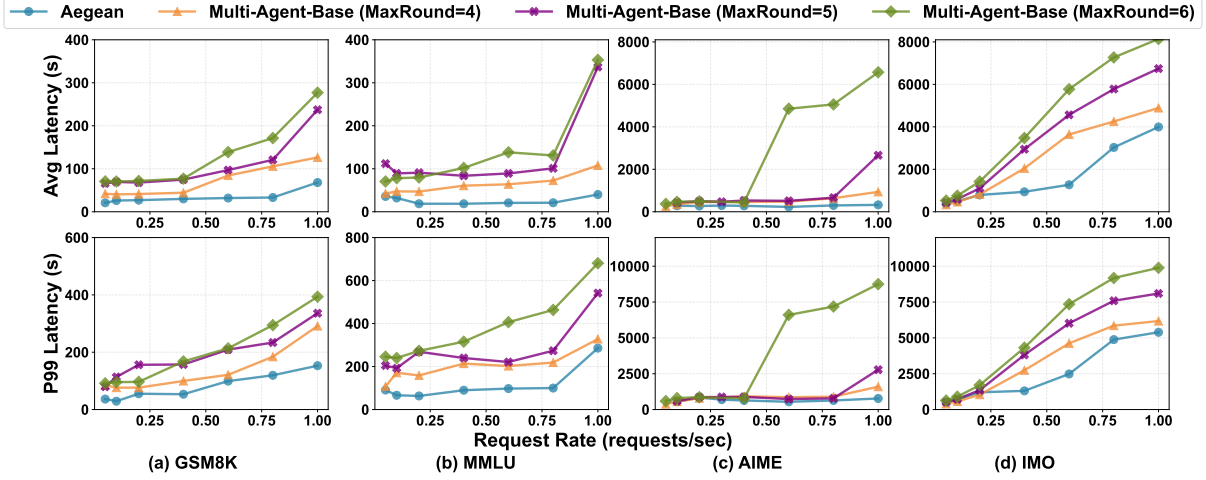


Figure 7: Throughput versus latency on locally deployed models across four benchmarks, comparing Aegean to Multi-Agent-Base with varying maximum rounds.

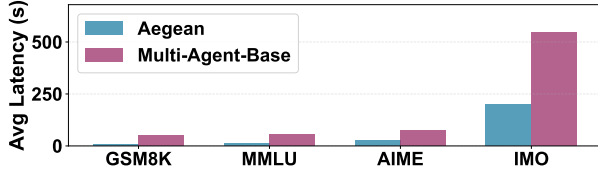


Figure 8: Average latency comparison using commercial frontier models accessed via public API.

4.5-haiku, Gemini-2.5-flash) accessed via OpenRouter API. Aegean achieves significant latency reduction across all benchmarks: 8.0s vs 49.2s on GSM8K ($6.2\times$ speedup), 13.8s vs 57.2s on MMLU ($4.1\times$ speedup), 26.4s vs 72.5s on AIME ($2.7\times$ speedup) and 198.8s vs 543.6s on IMO ($2.7\times$ speedup). We use a fixed request rate of 0.5 requests/sec for API experiments because commercial providers maintain large GPU clusters that absorb load fluctuations, resulting in stable per-request latency. The latency improvements instead stem from two inherent characteristics of commercial APIs. First, commercial APIs exhibit higher variance in response times due to shared infrastructure and rate limiting, making straggler delays more pronounced in wait-all systems. Second, frontier models generate longer responses on average, amplifying the cost of waiting for all agents to complete. Aegean’s consensus protocol effectively masks this variance by committing as soon as a quorum converges, providing predictable latency regardless of underlying API fluctuations. The consistent latency reduction across both deployment modes confirms that quorum-fast execution addresses the performance bottleneck in multi-agent systems.

Accuracy results. Critically, Aegean maintains accuracy comparable to Multi-Agent-Base while achieving these latency improvements. Figure 9 compares accuracy across four

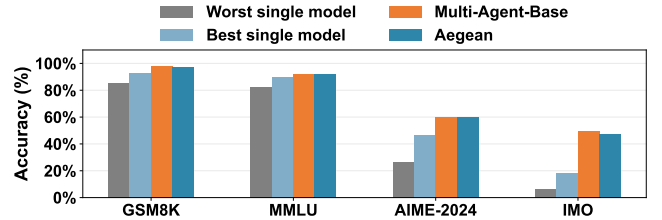


Figure 9: Accuracy comparison among best single model, Multi-Agent-Base and Aegean.

benchmarks for worst/best single models, Multi-Agent-Base, and Aegean. On GSM8K, both multi-agent approaches significantly outperform single models, with Multi-Agent-Base achieving 98% and Aegean achieving 97% compared to 93% for the best single model. MMLU shows similar patterns. On more challenging benchmarks, the benefits of multi-agent collaboration become more pronounced. For AIME, both multi-agent approaches achieve 60% accuracy compared to 46.7% for the best single model, representing a 28% relative improvement. IMO exhibits the largest gap: Multi-Agent-Base reaches 49.5% and Aegean achieves 47%, compared to only 18% for the best single model. These results show that quorum-based early termination eliminates computational waste without sacrificing correctness. Aegean matches Multi-Agent-Base accuracy within 2.5% across all benchmarks while substantially reducing latency. The small accuracy gap reflects a fundamental design choice: Aegean commits once a quorum agrees rather than waiting for all agents. For accuracy-critical applications, operators can increase α to require stronger agreement.

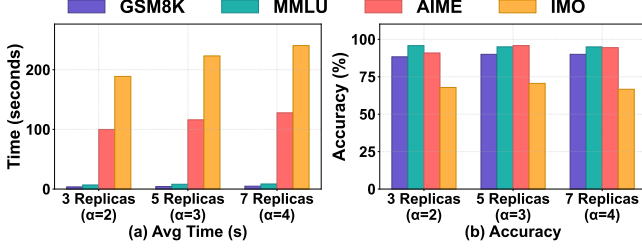


Figure 10: Impact of ensemble size on consensus performance using homogeneous replicas.

7.3 Deep Dive

Beyond end-to-end performance, we now examine how Aegean’s three primary parameters (ensemble size (N), stability horizon (β), and quorum threshold (α)) affect the latency-accuracy tradeoff. We vary each parameter while holding the others constant.

Impact of ensemble size. First, we evaluate how consensus performance scales with ensemble size by varying the number of replicas from 3 to 7 using the same underlying model. We deliberately employ homogeneous Qwen3-8B replicas in this ablation to isolate the effect of ensemble size from model diversity. Using heterogeneous models would introduce a confounding factor, as performance changes could stem from either the increased replica count or the addition of models with different capabilities. By holding the model constant, we can measure how the consensus protocol’s coordination overhead scales with the number of participants. As shown in Figure 10, the system exhibits favorable sub-linear scaling across benchmarks of varying difficulty. For simpler benchmarks, average time per problem increases modestly from 3.6s to 4.8s on GSM8K and from 6.9s to 8.5s on MMLU when scaling from 3 to 7 replicas, representing only 33% and 23% overhead, respectively. The sub-linear pattern extends to challenging benchmarks: AIME increases from 99.9s to 127.7s (28% overhead) and IMO grows from 188.8s to 240.4s (27% overhead), indicating that coordination costs do not compound with problem difficulty. Accuracy remains stable or improves slightly across all configurations. GSM8K improves from 88.3% to 90%, while MMLU maintains consistent performance between 95-96%. On AIME, accuracy peaks at 95.8% with 5 replicas, suggesting that moderate ensemble sizes can enhance correctness on challenging problems.

Impact of stability horizon. Second, we vary the stability horizon β to understand its effect on convergence with the GSM8K dataset. Figure 11-(a) shows that average time per problem increases linearly from 4.8s at $\beta = 1$ to 9.7s at $\beta = 4$. However, the accuracy results in Figure 11-(b) reveal a more nuanced picture. At $\beta = 1$, accuracy drops to 85%, indicating premature termination before models reach true consensus. In contrast, $\beta = 2$, $\beta = 3$, and $\beta = 4$ all achieve approximately 98% accuracy, suggesting they allow sufficient time for sta-

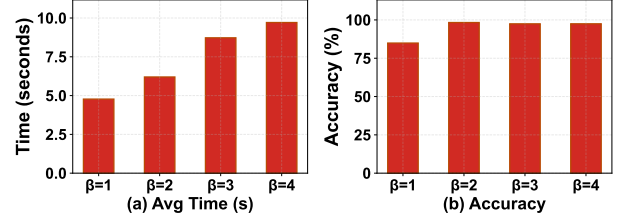


Figure 11: Impact of stability parameter β on consensus performance with 3 models and $\alpha=2$.

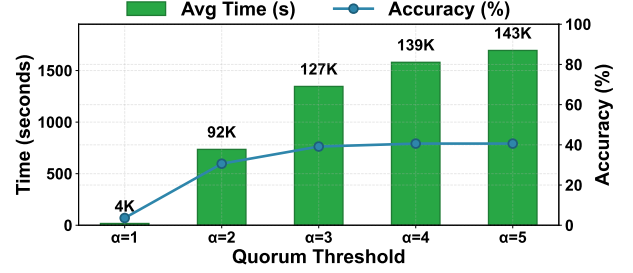


Figure 12: Impact of quorum threshold α on an ensemble of 5 models.

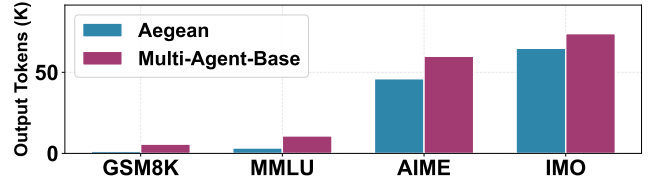


Figure 13: Average output tokens per problem for Aegean and Multi-Agent-Base.

ble agreement. Increasing β beyond 2 provides diminishing returns. While $\beta = 3$ and $\beta = 4$ maintain the same accuracy as $\beta = 2$, they incur substantial time penalties. We therefore select $\beta = 2$ for 3 agents as our default, balancing accurate consensus with convergence efficiency.

Impact of quorum threshold. Third, we examine the quorum threshold α using an ensemble of 5 open source models in §7.1. We use the most difficult dataset IMO. Figure 12 reveals a clear tradeoff between latency and accuracy. At $\alpha=1$, the system terminates immediately with any single answer, achieving only 17s latency but near-zero accuracy. As α increases, accuracy improves sharply from $\alpha=1$ to $\alpha=3$ (majority threshold), then plateaus: $\alpha=3$, $\alpha=4$, and $\alpha=5$ all achieve approximately 43% accuracy, while latency continues to grow from 1346s to 1695s and token consumption increases from 127K to 143K. This pattern validates the classical $f+1$ of $2f+1$ quorum design: The majority threshold ($\alpha=3$ for $N=5$) captures the consensus signal without paying the additional cost of waiting for slower agents. Stricter thresholds provide no accuracy benefit while imposing 17-26% additional latency overhead.

7.4 Computational Cost Reduction.

Finally, we measure the token savings from early termination. Figure 13 compares average output tokens across four benchmarks. On GSM8K and MMLU, Aegean achieves $4.4\times$ and $3.3\times$ token reductions respectively, consuming only 1.3K and 3.3K tokens per problem compared to 5.7K and 10.7K for Multi-Agent-Base. These significant reductions reflect the relative simplicity of these benchmarks: Agents reach quorum quickly, allowing the system to terminate rounds early and cancel redundant generation. On more challenging benchmarks, reductions are smaller but remain consistent. AIME shows $1.3\times$ reduction (46.0K vs 59.9K tokens) and IMO shows $1.1\times$ reduction (64.8K vs 73.8K tokens). The correlation between problem difficulty and token consumption demonstrates that Aegean naturally adapts computational expenditure to task complexity. Easy problems that achieve rapid consensus incur minimal overhead, while difficult problems receive the extended deliberation they require.

8 Related Work

Traditional consensus protocols. Classic consensus algorithms [3, 24, 29, 32, 37, 38, 43] provide strong consistency guarantees. These protocols require multi-round message exchanges and leader-based architectures, creating bottlenecks when nodes are geographically dispersed or subject to variable network conditions. Paxos and Raft require majority quorums with repeated leader coordination, while Byzantine-tolerant protocols like PBFT incur quadratic communication costs that limit scalability. Optimizations such as Fast Paxos [26] reduce latency but rely on super-majority quorums. More fundamentally, traditional consensus is designed for deterministic state machines and is incompatible with stochastic multi-agent reasoning. Aegean adapts these primitives by replacing immediate commits with stability horizons, requiring consensus to persist for β consecutive rounds. This filters out transient states that traditional protocols cannot handle.

Multi-agent serving optimizations. As LLM applications evolve to complex multi-agent workflows, specialized serving frameworks have emerged. Parrot [31] introduces Semantic Variables for application-level optimization. Autelix [33] applies program-level scheduling to mitigate head-of-line blocking. Kairos [6] and Gradientsys [41] reduce latency through workflow-aware priority scheduling. However, these frameworks optimize predetermined workflows, treating multi-agent interaction as static dependency graphs where all nodes must complete. None explores consensus-aware optimizations that adapt workloads based on output convergence. Aegean addresses this by embedding consensus logic into the serving layer, using streaming quorum detection to cancel redundant generation once agreement is reached.

9 Conclusion

Aegean represents a shift from ad-hoc multi-agent orchestration to principled consensus-based coordination. Classical consensus protocols enable unreliable processors to act as a coherent system. Similarly, Aegean transforms stochastic agents into a reliable reasoning engine. We replace heuristic workflows with formal stability horizons and quorum-based locks. This design aligns computational cost with actual convergence, filters out transient agreements, and accelerates decision-making. Our results demonstrate that Aegean brings significant performance benefits to multi-agent LLM systems.

References

- [1] Mistral AI. Mistral-7b-instruct-v0.3. <https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3>, 2024. Accessed: November 27, 2025.
- [2] Anthropic. Claude haiku 4.5. <https://www.anthropic.com/news/claude-haiku-4-5>, October 2025. Accessed: November 27, 2025.
- [3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*, New Orleans, LA, February 1999. USENIX Association.
- [4] Jingyi Chai, Shuo Tang, Rui Ye, Yuwen Du, Xinyu Zhu, Mengcheng Zhou, Yanfeng Wang, Weinan E, Yuzhi Zhang, Linfeng Zhang, and Siheng Chen. Scimaster: Towards general-purpose scientific ai agents, part i. x-master as foundation: Can we lead on humanity’s last exam?, 2025.
- [5] Jiefeng Chen, Jie Ren, Xinyun Chen, Chengrun Yang, Ruoxi Sun, Jinsung Yoon, and Sercan Ö Arık. Sets: Leveraging self-verification and self-correction for improved test-time scaling, 2025.
- [6] Jinyuan Chen, Jiuchen Shi, Quan Chen, and Minyi Guo. Kairos: Low-latency multi-agent serving with shared llms and excessive loads in the public cloud. *arXiv preprint arXiv:2508.06948*, 2025.
- [7] Justin Chih-Yao Chen, Archiki Prasad, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. Magi-core: Multi-agent, iterative, coarse-to-fine refinement for reasoning. *arXiv preprint arXiv:2409.12147*, 2024.
- [8] Yongchao Chen, Jiefeng Chen, Rui Meng, Ji Yin, Na Li, Chuchu Fan, Chi Wang, Tomas Pfister, and Jinsung Yoon. Tumix: Multi-agent test-time scaling with tool-use mixture, 2025.

- [9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- [10] Google DeepMind. Gemini 2.5 flash. <https://deepmind.google/models/gemini/flash>, May 2025. Accessed: November 27, 2025.
- [11] DeepSeek-AI. Deepseek-r1-distill-qwen-7b. <https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B>, 2025. Accessed: November 27, 2025.
- [12] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.
- [13] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023.
- [14] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the Presence of Partial Synchrony. *J. ACM*, 35(2), April 1988.
- [15] Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence, 2025.
- [16] Google. Try deep think in the gemini app. <https://blog.google/products/gemini/gemini-2-5-deep-think/>, Aug 2025. Accessed: November 27, 2025.
- [17] Hugging Face H4. Aime 2024 dataset. https://huggingface.co/datasets/HuggingFaceH4/aime_2024, 2024. Accessed: November 27, 2025.
- [18] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, and Zhaozhuo Xu. Llm multi-agent systems: Challenges and open problems, 2025.
- [19] Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [20] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [21] OpenRouter Inc. Openrouter: The unified interface for llms. <https://openrouter.ai>, 2025. Accessed: November 27, 2025.
- [22] Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating hallucination in large language models via self-reflection, 2023.
- [23] Lars Benedikt Kaesberg, Jonas Becker, Jan Philip Wahle, Terry Ruas, and Bela Gipp. Voting or consensus? decision-making in multi-agent debate. In *Findings of the Association for Computational Linguistics: ACL 2025*, page 11640–11671. Association for Computational Linguistics, 2025.
- [24] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.
- [25] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column)* 32, 4 (Whole Number 121, December 2001), pages 51–58, December 2001.
- [26] Leslie Lamport. Fast paxos. 19(2):79–103, October 2006.
- [27] Baixuan Li, Dingchu Zhang, Jialong Wu, Wenbiao Yin, Zhengwei Tao, Yida Zhao, Liwen Zhang, Haiyang Shen, Runnan Fang, Pengjun Xie, Jingren Zhou, and Yong Jiang. Parallelmuse: Agentic parallel thinking for deep information seeking, 2025.
- [28] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society, 2023.
- [29] Jialin Li, Ellis Michael, Naveen Kr Sharma, Adriana Szekeres, and Dan RK Ports. Just say no to paxos overhead: Replacing consensus with network ordering. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 467–483, 2016.
- [30] Wenzhe Li, Yong Lin, Mengzhou Xia, and Chi Jin. Rethinking mixture-of-agents: Is mixing different large language models beneficial?, 2025.
- [31] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of llm-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 929–945, 2024.
- [32] Barbara Liskov and James Cowling. Viewstamped replication revisited. 2012.
- [33] Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E Gonzalez, et al. Autellix: An efficient serving engine for llm agents as general programs. *arXiv preprint arXiv:2502.13965*, 2025.

- [34] Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.
- [35] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.
- [36] Meta. Meta-llama-3.1-8b-instruct. <https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>, 2024. Accessed: November 27, 2025.
- [37] Brian M Oki and Barbara H Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 8–17, 1988.
- [38] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319, 2014.
- [39] OpenAI. Gpt-5 mini (2025-08-07). <https://platform.openai.com/docs/models/gpt-5-mini>, August 2025. Accessed: November 27, 2025.
- [40] OpenAI. Gpt-oss-20b. <https://huggingface.co/openai/gpt-oss-20b>, 2025. Accessed: November 27, 2025.
- [41] Xinyuan Song, Zeyu Wang, Siyi Wu, Tianyu Shi, and Lynn Ai. Gradientsys: A multi-agent llm scheduler with react orchestration. *arXiv preprint arXiv:2507.06520*, 2025.
- [42] Qwen Team. Qwen3-8b: Large language model. <https://huggingface.co/Qwen/Qwen3-8B>, 2025. Accessed: November 27, 2025.
- [43] Sarah Tollman, Seo Jin Park, and John Ousterhout. {EPaxos} revisited. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 613–632, 2021.
- [44] Robbert Van Renesse and Deniz Altinbuken. Paxos Made Moderately Complex. *ACM Computing Survey*, 47(3), February 2015.
- [45] vLLM Project Contributors. vllm: A high-throughput and memory-efficient inference and serving engine for llms. <https://github.com/vllm-project/vllm>, 2025. Accessed: November 27, 2025.
- [46] Junlin Wang, Jue WANG, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025.
- [47] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- [48] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Auto-gen: Enabling next-gen llm applications via multi-agent conversation, 2023.
- [49] xAI. Grok 4. <https://x.ai/news/grok-4/>, Jul 2025. Accessed: November 27, 2025.
- [50] Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, Bo Pang, Yingbo Zhou, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. Diversity empowers intelligence: Integrating expertise of software engineering agents, 2024.