MESHHEAL: SELF-HEALING DECENTRALIZED LLM AGENT NETWORKS UNDER GRAY FAILURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Decentralized LLM-based multi-agent systems coordinate through local interactions, but an agent can remain responsive while its task-solving quality persistently degrades. Such gray failures require protecting current tasks before sufficient evidence exists to alter future routing, while still allowing recovered agents to rejoin. We introduce MESHHEAL, a fully decentralized self-healing framework that couples ability-matched peer review across two timescales. At the fast timescale, an adaptive hierarchy escalates uncertain or low-scoring outputs from repeated single-reviewer evaluation to committee deliberation and, when needed, correction before use. At the slow timescale, a task- and ability-conditioned peer-relative detector aggregates scores to distinguish persistent degradation from ordinary output variation, trigger mandatory committee review, and eventually exclude degraded agents from ordinary routing; recovery probes provide fresh evidence for reintegration. To faithfully evaluate routing, we introduce Model-Backed MAS Evaluation, which ties ability assignments to execution models, since prompt-based ability assignments alone can leave routing errors hidden. Across BBH, MATH, and MMLU-Pro, MeshHeal achieves 0.839 degraded-phase accuracy using 51k total model tokens per task, versus the strongest baseline Symphony's 0.807 accuracy using 115k per task. Under staggered degradation and recovery, MeshHeal isolates degraded agents, keeps them excluded from ordinary task execution until recovery, and returns them to normal routing.

1 INTRODUCTION

Recent advances in foundation models have enabled increasingly capable autonomous AI agents that can reason, plan, follow instructions, and use external tools (Bommasani et al., 2021). LLM-based multi-agent systems (MASs) decompose complex tasks among agents with complementary roles or abilities, combining their contributions into a final answer, often outperforming a single, larger model. However, most existing MASs coordinate in a centralized manner (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; Wang et al., 2025), which creates a communication and scaling bottleneck and also a catastrophic single point of failure.

Decentralized coordination instead allows agents to execute and route work through local interactions without relying on a central orchestrator. Accordingly, a growing body of recent work has investigated decentralized MAS architectures that distribute coordination and routing decisions across the agent network (Zhang et al., 2024; Yang et al., 2025a;b). These studies, however, largely assume that participating agents remain reliable throughout execution. In practice, individual agents may fail or degrade because of problems in their model backends, local controllers, memories, or external tools, potentially disrupting task execution across the network (Huang et al., 2025; Zhang et al., 2025).

Motivated by this gap, we study decentralized LLM-based MASs operating in the presence of agent failures. We focus on *gray failures* (Huang et al., 2017; 2018), and leave other types of failures as future work. A gray failure occurs when an agent remains available and protocol-compliant, i.e., it can communicate with its neighbors and return syntactically valid responses, but its effective ability to solve some tasks is degraded relative to its nominal ability. Such a failure is "gray" because it is not directly exposed through an explicit error or loss of connectivity: the affected agent may appear healthy to some neighbors, while its degradation can only be inferred from the semantic quality of its outputs or their downstream consequences. Gray failures are particularly relevant to

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

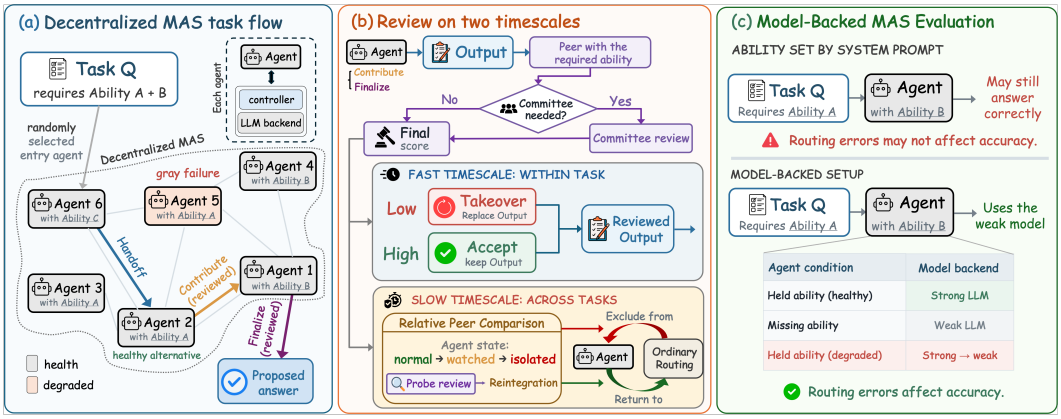


Figure 1: Overview of **MESHHEAL**. (a) Agents route and complete complementary work. (b) Review by agents with the relevant ability either keeps or replaces the current output, while scores accumulated across tasks update routing eligibility and probes allow recovered agents to return. (c) **Model-Backed MAS Evaluation** ties execution models to declared abilities and the agent's healthy or degraded condition, making routing errors affect accuracy.

decentralized LLM-based multi-agent systems because an agent's availability does not necessarily imply its semantic reliability. We do not regard every incorrect LLM response as a gray failure, since even a healthy agent has a nonzero probability of error. Instead, a gray failure refers to a ***sustained*** degradation relative to the agent's nominal task solving capability. Such degradation may arise from transient inference-serving issues, context corruption or overflow, stale memory, failures of external tools, or changes in the model backend. These failures are difficult to detect using conventional heartbeat- or connectivity-based mechanisms, since the affected agent continues to participate normally in the communication protocol.

We refer to the system-level capability to respond to such degradation as *self-healing*: detecting agent degradation, limiting its effect on current task execution, adapting future routing, and reintegrating an agent after recovery (Psaier & Dustdar, 2011). Existing LLM inference and agent-coordination methods address complementary parts of this objective. Methods based on repeated inference, deliberation, or robust aggregation improve current outputs (Wang et al., 2023; Du et al., 2024; Li et al., 2024; Wang et al., 2025; Jo & Park, 2025; Zheng et al., 2026; Lee et al., 2026). However, because they operate within individual tasks, these methods are not designed to retain producer-specific evidence over time and therefore do not support persistent-degradation detection, subsequent rerouting, or recovery-aware reintegration. Moreover, history-based methods adapt routing across tasks (Li et al., 2026; Yang et al., 2025b); however, routing adaptation alone acts only after history has accumulated, so it cannot protect the current task, and once an agent is bypassed, the system receives no fresh evidence of its recovery for reintegration. Therefore, to address gray failures, the system must protect the current task before degradation can be reliably detected, accumulate evidence to adapt future routing, and reintegrate an agent once its performance recovers.

We introduce **MESHHEAL** (see Fig. 1), a fully decentralized self-healing framework that couples peer review across two timescales. At the fast timescale, ability-matched neighboring agents review every intermediate contribution. An adaptive review hierarchy begins with repeated single-reviewer evaluation and escalates uncertain or low-scoring outputs to committee deliberation and, when necessary, multi-agent correction. At the slow timescale, a task- and ability-conditioned peer-relative detector accumulates the same review evidence to distinguish persistent degradation from ordinary output variation and impose progressively stronger routing restrictions, culminating in isolation from ordinary task execution. Recovery probes provide fresh reviewed evidence, allowing agents whose performance has recovered to return gradually to normal routing.

In our experiments on BBH, MATH, and MMLU-Pro, MESHHEAL achieves 83.9% mean accuracy during degradation over five seeds, compared with 80.7% for Symphony, the strongest baseline. It uses 51k tokens (both input and output tokens) per task, compared with 115k for Symphony and 178k for SAC, reducing token use by 56% and 71%, respectively. In sparse networks of up to 576 agents, MESHHEAL isolates all degraded agents and achieves comparable accuracy with a complete graph.

A commonly used approach to enforce ability constraints on agents when evaluating ability-aware routing is to use system prompts (Li et al., 2023; Qian et al., 2024; Chen et al., 2024; Hong et al., 2024; Chen et al., 2026). However, we identify an important confound: system prompts do not reliably enforce ability constraints, and the agent may still be able to solve tasks outside its abilities, allowing incorrect routes to produce correct final answers and thereby hiding routing errors. In a controlled BBH (Suzgun et al., 2023) study across five models, we find that prompt-based ability assignments do not reliably separate performance on assigned and unassigned abilities. We introduce Model-Backed MAS Evaluation, which uses backend selection to realize ability assignments and inject controlled gray failures. For a requested ability, a healthy agent uses a strong model if it holds that ability and a weak model otherwise. This construction provides known failure ground truth while preserving the defining property that a gray-failed agent remains responsive.

2 Related Work

LLM multi-agent coordination and robustness. LLM agents combine reasoning and tool use to solve complicated tasks (Yao et al., 2023). MASs extend these capabilities through task decomposition and specialization (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024), while decentralized systems coordinate and adapt through local interactions (Yang et al., 2025b; Li et al., 2026).

Various approaches have been proposed to improve output quality of LLM and MAS, e.g., repeated inference, deliberation, peer review, and robust aggregation (Wang et al., 2023; Du et al., 2024; Li et al., 2024; Wang et al., 2025; Jo & Park, 2025; Zheng et al., 2026; Lee et al., 2026). These approaches may also be applied here to handle gray failure. However, they operate within each task and do not identify whether a persistent failure has occurred to an agent. Therefore, a degraded agent may consistently participate in task solving.

Adaptive routing strategies were also developed to handle failures in MASs. AgentNet updates routing based on interaction history (Yang et al., 2025b), Symphony-Coord combines adaptive agent selection with voting (Guan et al., 2026), and RAPS uses reputation to identify and isolate unreliable peers (Li et al., 2026). However, routing adaptation alone cannot protect the current task, and once an agent is bypassed, the system receives no fresh evidence of its recovery for reintegration.

Distributed recovery and gray failures. Our recovery goal builds on self-stabilizing and self-healing systems, where systems adapt their behavior in response to failures (Dijkstra, 1974; Arora & Gouda, 1993; Psaier & Dustdar, 2011). Gray-failure research shows that hidden degradation may need to be inferred from peer observations rather than availability (Huang et al., 2017; 2018). We study this problem for LLM agents using reviewed output quality as the failure signal.

3 Problem Formulation

We consider a decentralized MAS with heterogeneous agent abilities and potential gray failures.

3.1 Decentralized MAS Architecture

We model a decentralized LLM-based MAS as an undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of agents, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the communication graph between the agents: an undirected edge $e_{i,j} \in \mathcal{E}$ allows agent i to send information to agent j and vice versa. Each agent consists of an LLM and a local controller. The LLM is used to generate the agent's outputs, and is called the agent's *model backend*; and the local controller coordinates the agent's interactions with its neighbors.

We consider heterogeneous agents where agents differ in their abilities. Denote by $\mathcal{K}_i$ the set of abilities of an agent i, and $\mathcal{K}_i$ varies across agents. Solving different tasks may require a different set of abilities. Therefore, it is of key importance to assign the task to the agent with the matching ability. We study the decentralized setting, where no agent has a global view of other agents, and each agent can only communicate with its neighbors on the communication graph $\mathcal{E}$.

The multi-agent system will be given a sequence of tasks denoted by $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$. The t-th task is $\tau_t = (Q_t, T_t)$, where $t \in [N]$ is the task index, Q_t is the input question, T_t is its benchmark-provided category label (e.g., `geometry` in MATH (Hendrycks et al., 2021)), and we use R_t to

represent the set of abilities required to solve Q_t. Here $[N] \triangleq \{1, \ldots, N\}$. We note that Q_t and T_t are directly given to the agent, but R_t needs to be inferred by the agent.

Objective. Let Π_{local} be the set of coordination policies whose decisions use only the current task state, local interaction history, and messages from direct neighbors. For a policy π, let $\widehat{y}_t^{\pi}$ be the final answer of the MAS and $\text{Eval}_t(\widehat{y}_t^{\pi}) \in \{0, 1\}$ indicate correctness under the benchmark evaluator. We seek to optimize the average accuracy over the N sequential tasks:

$$\max_{\pi \in \Pi_{\text{local}}} \frac{1}{N} \sum_{t=1}^{N} \mathbb{E}[\text{Eval}_t(\widehat{y}_t^{\pi})] \,, \tag{1}$$

where the expectation is over the execution randomness.

3.2 Gray Failure and Self-Healing

We study a practical setting in which agents may experience *gray failures* (Huang et al., 2017; 2018). A gray failure occurs when an agent remains available and protocol-compliant, i.e., it can communicate with its neighbors and return syntactically valid responses, but its effective ability to solve some tasks is degraded relative to its nominal ability.

Each agent may have two possible health conditions: *healthy*, *degraded*. The health condition of each agent i may change over time. We assume that each agent's health condition remains unchanged within a task but may change across tasks. An agent is called *healthy* when it operates at its normal level of reliability, and *degraded* when it suffers from gray failure. The true health condition is latent, and the agent must infer it from the agent's outputs. We further define *recovery* as restoration of an agent from *degraded* to *healthy*.

In this paper, our focus is on solving the problem in Eq. (1) when agents experience gray failures and may also recover from them. We aim to design local coordination policies so that, even with degraded agents, each task can still be routed to healthy agents with matching abilities and solved with high accuracy; and at the same time, when an agent recovers from *degraded* to *healthy*, future tasks that match this agent's abilities will still be routed to it. Such desired property is referred to as ***self-healing***. To evaluate this ability, beyond task accuracy and inference cost, we introduce three additional criteria: detection delay, closure, and reintegration. These criteria are adapted from the distributed fault-tolerance literature (Dijkstra, 1974; Arora & Gouda, 1993; Ghosh et al., 2007; Psaier & Dustdar, 2011). More specifically, suppose agent i becomes degraded at task $b_i \in [N]$ and recovers at task $r_i \in [N+1]$. If recovery is not observed in an N-task stream, we set $r_i = N + 1$. Let $\mathcal{I}_t$ denote the set of agents that are excluded from ordinary task execution at task t. Then, we define:

- *Detection delay:* For a degraded agent i, the system should isolate it as early as possible. Let $c_i = \min\{t \geq b_i : i \in \mathcal{I}_t\}$ be the index of the first task at which the degraded agent i is isolated. The detection delay is then defined as $\delta_i^{\text{delay}} = c_i - b_i$.
- *Closure:* A system satisfies the *closure* property if a degraded agent remains isolated until recovery after being detected. Formally, for any degraded agent i, $i \in \mathcal{I}_t, \forall c_i \leq t < r_i$.
- *Reintegration:* It refers to the case that after recovery, agent i returns to normal routing. Let $u_i \geq r_i$ be the index of the first task at which agent i returns to normal routing state through the end of the observed stream. Then the reintegration delay is defined as $\delta_i^{\text{rein}} = u_i - r_i$.

4 MeshHeal: Review at Two Timescales

In Sec. 4.1, we will present our base workflow design in a decentralized MAS *without gray failures*. Then, in Sec. 4.2, we build upon the base design and introduce our MeshHeal with novel two-timescale mechanism for self-healing. The full algorithm is provided in Alg. 1 in App. B.1.

4.1 Base Task Execution and Routing (No Failures)

Task execution. As illustrated in Fig. 1(a), Agents collaborate to sequentially solve and route a task through the network. Each task $\tau_t = (Q_t, T_t)$ is initially assigned to an entry agent sampled uniformly at random from the network. The query Q_t and task type T_t are given, while R_t denotes the set of abilities required to solve the task and is inferred by the entry agent.

The task is solved sequentially as it moves through the network, with each agent addressing the part aligned with its abilities and passing the rest to the next agent until it is fully solved. The task state is initialized as $X_t = (Q_t, T_t, R_t, C_t, Y_t, P_t)$, where C_t records the abilities already covered so far, Y_t stores the working context that will be passed to later agents, including accepted intermediate outputs, and P_t is a list that records the route taken so far. Initially, $C_t = Y_t = \emptyset$ and P_t is also empty.

When the task is routed to agent i, let $M_t = R_t \setminus C_t$ be the set of abilities that were not supplemented by previous agents in P_t and need to be further provided by subsequent agents, and $H_{i,t} = \mathcal{K}_i \cap M_t$ be the subset of matching abilities that can be provided by agent i. We take a rule-based approach at the controller, and these sets govern the controller's decision for the current task:

- If $H_{i,t} = \emptyset$, then agent i has no matching abilities to contribute to the remaining of task t. Then, agent i's controller chooses to **Handoff**: it generates no task content and forwards the task state unchanged to the next agent (discussed later in Routing). Thus, C_t, Y_t and P_t remain unchanged.
- If $\emptyset \neq H_{i,t} \subset M_t$, then agent i contribute matching abilities to the task that prior agents in P_t cannot; however, additional abilities beyond those in $H_{i,t}$ are still needed. Then, agent i's controller chooses to **Contribute**. The agent's LLM solves the matching part in the remaining of the task using abilities in $H_{i,t}$. The output is appended to Y_t, $C_t = C_t \cup H_{i,t}$, and agent i is appended to the list P_t. The controller recomputes $M_t \leftarrow R_t \setminus C_t$ using the updated C_t and forwards the updated task state to the next agent (discussed later in Routing).
- If $H_{i,t} = M_t$, then agent i has all the abilities needed to finish the remaining of the task. Then, agent i's controller chooses to **Finalize**. The agent solves the remaining task, combines the result with earlier outputs in Y_t, and proposes the final answer. At this point, all abilities in R_t have been covered by agents along the route in Y_t, and no further routing is required.

Routing. Each agent i maintains an estimated the shortest hop distance $D_i(s)$ to the nearest agent that holds ability s. If agent i itself holds ability s, then $D_i(s) = 0$; otherwise, it updates in a distributed manner: $D_i(s) = 1 + \min_{j \in \mathcal{N}_i} D_j(s)$, where $\mathcal{N}_i$ is the set of direct neighbors of agent i (Bellman, 1958). Under standard asynchronous Bellman-Ford assumptions with fixed topology and reliable neighbor updates, these iterations converge in finite time (Bertsekas, 1982). The path that achieves $D_i(s)$ will be used to select the next agent. When **Handoff** or **Contribute** are chosen by the controller, the task has at least one ability not provided along the route in P_t. The controller samples one ability s uniformly at random from M_t. It then routes the task state along the path that achieves $D_i(s)$, i.e., to the nearest agent with ability s.

4.2 Task Execution and Routing under Gray Failures

Having defined the base task flow without failures, we now present our novel design to address gray failures. We define a state $h_i \in \{\textbf{normal}, \textbf{watched}, \textbf{isolated}\}$ for each agent i, which determines how the agent participates in execution and routing. A **normal** agent follows the base workflow without restriction. A **watched** agent is routed and executes tasks as a normal agent, but every output it produces receives mandatory committee review. An **isolated** agent has its set of abilities $\mathcal{K}_i$ set to be $\emptyset$ temporarily, and therefore it will be excluded from task execution, but its controller remains active for message relay. When an agent becomes **isolated**, agents will need to recompute the shortest hop distance as described in Sec. 4.1. We then apply the base task flow in Sec. 4.1.

Our MeshHeal framework presents a novel two-timescale scheme with fast review and correction and slow detection, rerouting, and recovery (illustrated in Fig. 1 (b)). At the *fast timescale*, an review protocol escalates low-scoring, high-disagreement, or watched-agent outputs from repeated single-reviewer evaluation to deliberative committee review and multi-agent correction. At the *slow timescale*, a task- and ability-conditioned peer-relative detector aggregates persistent shortfalls and normalizes them against healthy-neighbor variation, enabling same thresholds to be used across model backends; guarded recovery probes close the loop through reintegration.

4.2.1 Fast Timescale: Review and Correction

At the fast timescale summarized in the fast-timescale path of Fig. 1(b), every output produced by **Contribute** or **Finalize** is reviewed before use. We denote the output by $y_{i,t}$. An eligible reviewer must be a **normal** direct neighbor that holds every ability in $H_{i,t}$. The reviewer scores $y_{i,t}$ on a fixed $[0, 1]$ scale (judge prompts in App. D.1). To reduce review noise, the same reviewer evaluates $y_{i,t}$ in three independent calls. Denote the median by q_0, and we use it as the initial score. We use the

median rather than the mean to reduce sensitivity to reviewer variability and outlier scores, consistent with prior work on robust aggregation of LLM judgments (Acharya et al., 2026; Gallaba et al., 2025). We also use the standard deviation s_u to measure disagreement. Committee review adds multiple perspectives and deliberation, improving review reliability at the cost of additional inference and communication; we therefore trigger it only when $q_0 \le \tau_{\text{to}}$, $s_u \ge \sigma_{\text{com}}$, $h_i =$ **watched** or the output is a recovery probe.

The committee contains two eligible reviewers. They first evaluate $y_{i,t}$ independently, exchange their explanations, and then reassess the same output. Let v_A and v_B be their second-round scores; the final review score is $q_{i,t} = (v_A + v_B)/2$. If committee review is not triggered, $q_{i,t} = q_0$. When only one eligible reviewer is available, two reviews are obtained through separate inference calls with isolated context histories. If $q_{i,t} > \tau_{\text{to}}$, $y_{i,t}$ is accepted. Otherwise, **Takeover** is triggered: the two reviewers generate correction candidates, which a committee reviewer in the normal state synthesizes task context following Mixture-of-Agents style aggregation (Wang et al., 2025). The resulting corrected output $\widehat{y}_{i,t}$ replaces $y_{i,t}$ before entering the task state. The score $q_{i,t}$ remains attributed to agent i and is retained as evidence for the slow timescale degradation detection.

4.2.2 Slow Timescale: Detection, Rerouting and Recovery

As summarized in the slow-timescale path of Fig. 1(b), A single low score in the fast timescale does not necessarily mean that the agent is persistently degraded. We therefore accumulate the review scores $q_{i,t}$ across tasks and use *relative peer comparison* to update the estimate of h_i. Our goal is to detect agent gray failures across different task types and model backends with the same detector, rather than tuning a separate threshold for each setting. We construct this evidence in two steps: first relative to matched peers, and then relative to normal peer variation.

Relative shortfall. For each ability $s \in H_{i,t}$, the controller considers direct **normal** neighbors that have recently reviewed outputs within the fixed history window from earlier tasks with the same task type T_t and ability s. Let $\mathcal{P}_{i,T_t,s}(t)$ denote this set of neighbors, excluding i, and let $\mu_{j,T_t,s}(t)$ denote neighbor j's recent mean review score before task t. The controller first uses these neighbors to estimate the normal review score for the same type of work ($\bar{q}_{-i,T_t,s}(t)$). For each $s \in H_{i,t}$, the controller measures how far agent i's current score falls below the corresponding reference:

$$\bar{q}_{-i,T_t,s}(t) = \frac{1}{|\mathcal{P}_{i,T_t,s}(t)|} \sum_{j \in \mathcal{P}_{i,T_t,s}(t)} \mu_{j,T_t,s}(t), \qquad x_{i,t,s} = \bar{q}_{-i,T_t,s}(t) - q_{i,t}.$$

A positive $x_{i,t,s}$ means that agent i scored below **normal** neighbors on the same task type and ability s. This comparison removes differences in absolute review scores across task types by comparing an agent only with normal neighbors who work on the same task type and have the same ability.

Persistent degradation. A single positive shortfall may result from LLM randomness. We therefore maintain a separate detector for each (T, s) and average its most recent 8 shortfalls, yielding $\bar{x}_{i,T,s}$. For a healthy agent, it should stay near zero; if the agent remains degraded, it should stay positive.

However, a fixed threshold on $\bar{x}_{i,T,s}$ may not transfer across model backends because the dispersion of healthy agents' mean shortfalls can differ substantially across backends. We therefore also measure how much the mean shortfalls of **normal** neighbors vary. We pool the mean shortfalls $\bar{x}_{j,T,s'}$ of all **normal** neighbors j over all abilities s' under task type T, and let $m_{i,T}$ and $p^{25}_{i,T}$ be their median and 25th percentile. We measure the normal variation by the lower spread $m_{i,T} - p^{25}_{i,T}$, because degraded agents that have not yet been detected can inflate the upper tail. We normalize agent i's mean shortfall by this variation:

$$z_{i,T,s} = \frac{\bar{x}_{i,T,s} - m_{i,T}}{\max\Big(m_{i,T} - p^{25}_{i,T},\, 0.10\Big)}.$$

Thus, $z_{i,T,s}$ measures how far agent i's mean shortfall is above normal level, relative to how much healthy agents vary. This second comparison accounts for differences in how much healthy agents' mean shortfalls vary across model backends, so same detection thresholds can be used across models.

State transitions, rerouting, and recovery. Each update uses the current $z_{i,T,s}$; if fewer than three matched observations are available h_i remains unchanged. Otherwise, $z_{i,T,s} \ge \alpha$ moves **normal** to **watched**, $z_{i,T,s} < \alpha$ returns **watched** to **normal**, and two consecutive updates with $z_{i,T,s} \ge \beta$ move

a **normal** or **watched** agent to **isolated**; the isolation count resets when $z_{i,T,s} < \beta$. Here $\alpha < \beta$ are thresholds, and in our experiments we choose $\alpha = 1.2$ and $\beta = 2$. The thresholds were selected empirically to balance detection delay and false isolation of healthy agents.

We further design *recovery probes* that use the same review and detection mechanism. After every 10 tasks, an isolated agent becomes eligible for a recovery probe. When a neighbor next receives work matching its abilities, the work is routed to the isolated agent and its output receives mandatory committee review. The resulting score updates the same detector. The agent returns to **watched** after one update with $z_{i,T,s} < \beta$, even if $z_{i,T,s} < \alpha$, and to **normal** after a later update with $z_{i,T,s} < \alpha$.

5 Model-Backed MAS Evaluation

System prompt fails to constrain the agent's ability. Many LLM MAS evaluations assign abilities through system prompts while using the same underlying model for all agents (Li et al., 2023; Qian et al., 2024; Chen et al., 2024; Hong et al., 2024). To test whether such prompts constrain agents, we compare two prompting conditions across five models on BBH benchmark (Suzgun et al., 2023): one makes no statement about unassigned abilities, while the other explicitly describes them as unreliable. If these prompts truly limited agents to their assigned abilities, accuracy on unassigned abilities should fall while assigned-ability accuracy remains stable. The accuracy gap between assigned and unassigned abilities increases by only 0.015–0.058 (see Tab. 1). Accuracy on unassigned abilities remains substantial, while assigned-ability accuracy also falls for four models. Prompt instructions change agent behavior, but do not reliably prevent agents from solving tasks outside their assigned abilities. App. A gives the full protocol.

Table 1: BBH ability-assignment test. Arrows show no competence statement → unassigned abilities declared unreliable; Δ_{gap} is the resulting increase in the accuracy gap between assigned and unassigned abilities.

Model	Unassigned ability acc.	Assigned ability acc.	Δ_{gap}
DeepSeek-Chat	$.812 \rightarrow .775$	$.800 \rightarrow .792$	.029
Llama-3.1-70B	$.781 \rightarrow .750$	$.783 \rightarrow .767$	.015
GPT-4o-mini	$.744 \rightarrow .675$	$.708 \rightarrow .667$	.027
GPT-OSS-120B	$.869 \rightarrow .831$	$.825 \rightarrow .833$	.046
Qwen-2.5-7B	$.494 \rightarrow .219$	$.558 \rightarrow .342$	.058

Controlled ability realization and gray-failure injection. We use model backends to realize ability assignments and inject gray failures, shown in Fig. 1(c). A healthy agent uses a **strong model** for assigned abilities and a **weak model** otherwise. Degradation is simulated by replacing its strong backend with the weak one while leaving communication and execution interfaces unchanged. This preserves responsiveness while reducing task competence, providing known failure ground truth and making routing errors visible in end-to-end accuracy.

Tasks and degraded agents. We apply this protocol to seven task groups from each of BBH, MATH, and MMLU-Pro (Suzgun et al., 2023; Hendrycks et al., 2021; Wang et al., 2024). Each task group contains tasks of the same type and is associated with two complementary abilities required for solving those tasks. The two abilities are assigned to disjoint sets of agents, so no single agent holds both. Every task requires at least two agents. Full task-to-ability mappings appear in App. B.

6 Experiments

We evaluate on benchmarks of BBH, MATH, and MMLU-Pro. The main system contains six agents, A1–A6, with three agents per ability and disjoint groups for the two abilities required by each task. A3 and A5 are selected for degradation, in the degraded phase, every LLM call by them, including review calls, switches to the weak model. Their controllers, abilities, communication links, and availability remain unchanged. Unless stated otherwise, healthy agents use `openai/gpt-oss-120b` for declared abilities; undeclared abilities and degraded agents use `meta-llama/llama-3.2-1b-instruct`.

We compare MeshHeal with AgentNet's decentralized routing (Yang et al., 2025b), RAPS's reputation-based routing (Li et al., 2026), and a centralized AutoGen implementation with a mandatory manager (Wu et al., 2023). SAC uses three teams with two rounds of filtering and refinement followed by majority aggregation (Lee et al., 2026), while Symphony uses parallel plans and voting (Guan et al., 2026). All methods use the same tasks, agents, ability assignments, models, degradation targets, and schedule. In AutoGen, A3 is the mandatory manager and appears on every task path.

Table 2: (a) Accuracy and total model-token cost under persistent degradation (mean ± std. over three datasets, each with five seeds). H/D denote healthy/degraded phases; tokens are thousands per degraded-phase task. (b) Scaling in complete and sparse networks over five seeds. I/D denotes isolated/degraded agents; edge fraction is relative to the complete graph.

(a) Persistent degradation

Method	H acc.	D acc.	Tokens
MESHHEAL	**.831±.013**	**.839±.012**	51±5
Symphony	.827±.013	.807±.014	115±9
SAC	.822±.014	.746±.021	178±13
RAPS	.800±.019	.653±.041	48±3
AgentNet	.729±.023	.557±.039	43±2
AutoGen	.787±.014	.424±.023	15±0

(b) Network scaling

	Complete		Sparse		Edge
Agents	Acc.	I/D	Acc.	I/D	fraction
96	.827	32/32	.818	32/32	5.00%
288	.813	96/96	.803	96/96	1.71%
576	.808	192/192	.814	192/192	0.88%

Table 3: Self-healing under staggered degradation and recovery over five seeds. (a) Mean accuracy and total model-token cost across phases. (b) Detection (Det.) and reintegration (Rein.) delays are mean ± std. (min–max); closure reports successful seeds.

(a) Degradation and Recovery at Different Times

Phase	MESHHEAL	Sym.	SAC	A.Net	RAPS	AutoGen
Healthy	.833	.810	.825	.794	.722	.627
A5 deg.	.778	.786	.722	.579	.492	.659
A3+A5 deg.	.770	.722	.643	.452	.405	.183
A5 restored; A3 deg.	.841	.841	.706	.532	.524	.135
All restored	.857	.873	.860	.563	.524	.770
Pooled after deg.	**.812**	.806	.733	.532	.486	.437
Tokens/task after deg.	49k	105k	124k	39k	38k	15k

(b) Self-healing metrics of MESHHEAL

Agent	Det.↓	Closure	Rein.↓
A5	8.0 ± 5.2 (2–16)	5/5	46.0 ± 11.7 (30–63)
A3	12.4 ± 6.2 (6–22)	5/5	45.0 ± 3.5 (42–51)

Each setting uses five seeds, defining task samples and order shared across methods. We report means and sample standard deviations. Except in targeted ablations, review rubrics and detector parameters are fixed across datasets, model pairs, and network sizes (Tab. B11). We measure accuracy, token consumption (counting both input and output tokens) , detection delay δ_i^{delay}, closure, and reintegration delay δ_i^{rein}; Appendices B–E provide settings, prompts, traces, and full results.

6.1 PERSISTENT DEGRADATION: ACCURACY AND COST

Each stream contains 252 healthy-phase tasks followed by 252 disjoint degraded-phase tasks with the same task-type distribution. During degraded-phase, A3 and A5 switch to the weak model. We report healthy- and degraded-phase accuracy and total model-token cost. After degradation, MESHHEAL reaches 0.839 ± 0.012 accuracy at 51 ± 5k total model tokens per task, compared with Symphony's 0.807 ± 0.014 at 115 ± 9k. AgentNet and RAPS use fewer tokens but have substantially lower accuracy (Tab. 2(a)). Full dataset-level and cost results are in App. C.

6.2 DEGRADATION AND RECOVERY: DETECTION, CLOSURE, AND REINTEGRATION

We test the three self-healing criteria on five consecutive 126-task MATH phases: healthy operation, A5 degraded, A3 and A5 degraded, A3 degraded after A5's backend restoration, and both backends restored. All methods complete the 630-task stream. Across the 504 tasks after the first degradation, including the fully restored phase, MESHHEAL reaches 0.812 accuracy at 49k total model tokens per task, compared with Symphony's 0.806 at 105k (shown in Tab. 3).

Across five seeds, after agent i becomes degraded at b_i, it is first excluded from ordinary task execution at c_i. The resulting detection delays $\delta_i^{\text{delay}} = c_i - b_i$ are 8.0 ± 5.2 tasks for A5 and 12.4 ± 6.2 tasks for A3. Once isolated, both agents satisfy closure in all five seeds, remaining excluded throughout $c_i \le t < r_i$ except for committee-reviewed recovery probes. After backend restoration at r_i, the agents return to **normal** routing at u_i, giving reintegration delays $\delta_i^{\text{rein}} = u_i - r_i$ of 46.0 ± 11.7 tasks for A5 and 45.0 ± 3.5 tasks for A3 (Tab. 3). Fig. C1 in App. C.2 shows one representative trajectory, with its state changes listed in Tab. C5. None of the baseline methods fully isolates degraded agents; under our definitions, their detection delay is therefore infinite, closure is not established, and reintegration delay is undefined.

Table 4: (a) Accuracy over five seeds on BBH; H and D denote the healthy and degraded phases. (b) Missed degraded agents out of two and falsely **isolated** healthy agents out of four on BBH.

(a) Fast-timescale components

Setting	H	D	Δ
Full	$.899 \pm .027$	$\mathbf{.900 \pm .014}$	$\mathbf{+.001 \pm .016}$
Single reviewer	$\mathbf{.909 \pm .022}$	$.865 \pm .029$	$-.044 \pm .024$
No Takeover	$.879 \pm .021$	$.844 \pm .036$	$-.035 \pm .030$

(b) Detector comparison

	Pair A		Pair B	
Rule	Miss	False	Miss	False
Absolute threshold	0/2	4/4	0/2	0/4
CUSUM	0/2	2/4	1/2	0/4
Relative peer comparison	0/2	0/4	0/2	0/4

6.3 Scaling with Local Information

We replicate the six-agent profiles in complete and sparse graphs with 96, 288, and 576 agents, with one third degraded and all review, summary exchange, and routing updates remaining local. MeshHeal isolates every degraded agent in all settings, while sparse and complete accuracies differ by at most 0.010. At 576 agents, the sparse graph uses only 0.88% of complete-graph edges and reaches 0.814 accuracy (Tab. 2(b)), demonstrating MeshHeal's effectiveness in sparse networks.

6.4 Component and Detector Studies

Fast timescale. We replace committee review with a single reviewer and separately remove Takeover on BBH while retaining detection and rerouting (Tab. 4(a); App. C.4, Tab. C6). Single reviewer uses the initial peer for judgment and correction; it has a slightly higher healthy-phase mean but 0.035 lower degraded-phase accuracy and a mean phase change of -0.044. No Takeover leaves low-scoring outputs unchanged, reducing degraded-phase accuracy by 0.056 and healthy-phase accuracy by 0.020. This suggests that Takeover also corrects some errors from healthy agents. An audit (App. C.5, Tab. C8(a,b)) shows that healthy agents receive much higher review scores than degraded agents (0.89 vs. 0.08), and 98% of degraded outputs are sent to committee review. Committee reassessment reduces false alarms on healthy agents. A separate Takeover audit (Tab. C8(c)) shows that Takeover corrects 15 of 32 wrong answers while turning only 1 of 32 correct answers into an incorrect one.

Slow timescale. We compare detectors under Pair A (GPT-OSS-120B/Gemma-3-4B) and Pair B (Llama-3.3-70B/Llama-3.2-1B). The absolute threshold falsely isolates all four healthy agents under Pair A; CUSUM (Page, 1954) falsely isolates two under Pair A and misses one of two degraded agents under Pair B. Relative peer comparison detects both degraded agents without false isolations under either pair using fixed rubrics and parameters (Tab. 4(b); App. C.4, Tab. C7).

Model-pair sensitivity. To test robustness under a milder degradation and a weaker healthy model, we additionally evaluate Pair A and Pair B, respectively. MeshHeal detects both degraded agents without false isolation across both settings, while maintaining competitive degraded-phase accuracy at substantially lower cost: it outperforms Symphony by 0.040 on Pair A and is within 0.004 on Pair B, using only 52–56% as many total model tokens. Full results are in App. C.2.

7 Conclusion

We introduced MeshHeal, a fully decentralized framework for self-healing gray failures in LLM agent networks with a novel two-timescale review design. At the fast timescale, review protocol evaluates every produced output and can replace low-quality outputs before they enter the task state; and at the slow timescale, a task- and ability-conditioned peer-relative detector aggregates persistent shortfalls to drive staged state transitions and routing updates, while recovery probes support evidence-based reintegration. We further showed that prompt-only ability assignments do not reliably enforce constraints on agents' ability and proposed Model-Backed MAS Evaluation, which uses controlled backend selection to realize complementary abilities and inject gray failures with known ground truth while keeping affected agents responsive. With the default model pair, MeshHeal reaches 0.839 degraded-phase accuracy at 51k tokens per task across three benchmarks and five seeds, versus the strongest baseline of Symphony 0.807 at 115k tokens per task. The experiment with degradation and recovery at different times meets the three self-healing criteria under the evaluated schedule. The scaling experiments isolate every degraded agent in the evaluated networks of up to 576 agents.

AI USE STATEMENT

We used generative AI tools to assist with manuscript editing and the formatting of LaTeX.

REPRODUCIBILITY STATEMENT

The appendices provide the complete procedure and routing rules (App. B.1), ability assignments, model and degraded settings, baseline adaptations, detector parameters (App. B.5), detailed results (App. C), representative runtime prompt cards (App. D), and execution traces (App. E). Together with the descriptions of task streams and token counting in Sec. 6, these materials specify the reported evaluation and the conditions under which the self-healing cycle is measured.

REFERENCES

Anish Acharya, Kris W Pan, and Brian Verkhovsky. Ropoll: Robust panel of llm judges. *arXiv preprint arXiv:2606.30931*, 2026.

Anish Arora and Mohamed G. Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, 1993.

Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

Dimitri Bertsekas. Distributed dynamic programming. *IEEE transactions on Automatic Control*, 27 (3):610–616, 1982.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Keru Chen, Jun Luo, Sen Lin, Yingbin Liang, Alvaro Velasquez, Nathaniel Bastian, and Shaofeng Zou. Hipo: Instruction hierarchy via constrained reinforcement learning. *arXiv preprint arXiv:2603.16152*, 2026.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024.

Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974. doi: 10.1145/361179.361202.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763, 2024.

Keheliya Gallaba, Ali Arabat, Dayi Lin, Mohammed Sayagh, and Ahmed E Hassan. Towards conversational development environments: Using theory-of-mind and multi-agent architectures for requirements refinement. *arXiv preprint arXiv:2505.20973*, 2025.

Sukumar Ghosh, Arobinda Gupta, Ted Herman, and Sriram V. Pemmaraju. Fault-containing self-stabilizing distributed protocols. *Distributed Computing*, 20(1):53–73, 2007. doi: 10.1007/s00446-007-0032-2.

Zhaoyang Guan, Huixi Cao, Ming Zhong, Yin Wang, Guanyu Liu, Eric Yang, Lynn Ai, Yongxin Ni, and Bill Shi. Symphony-Coord: Adaptive routing for multi-agent LLM systems. arXiv preprint arXiv:2602.00966, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024.

Jen-Tse Huang, Jiaxu Zhou, Tailin Jin, Xuhui Zhou, Zixi Chen, Wenxuan Wang, Youliang Yuan, Michael Lyu, and Maarten Sap. On the resilience of LLM-based multi-agent collaboration with faulty agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 26202–26226, 2025.

Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pp. 150–155, 2017. doi: 10.1145/3102980.3103005.

Peng Huang, Chuanxiong Guo, Jacob R. Lorch, Lidong Zhou, and Yingnong Dang. Capturing and enhancing in situ system observability for failure detection. In *13th USENIX Symposium on Operating Systems Design and Implementation*, pp. 1–16. USENIX Association, 2018.

Yongrae Jo and Chanik Park. Byzantine-robust decentralized coordination of LLM agents. arXiv preprint arXiv:2507.14928, 2025.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. BIG-Bench Extra Hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26473–26501, 2025. doi: 10.18653/v1/2025.acl-long.1285.

Haejoon Lee, Vincent-Daniel Yun, Dimitra Panagou, and Sai Praneeth Karimireddy. Robust multi-agent LLMs under byzantine faults. arXiv preprint arXiv:2605.09076, 2026.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008, 2023.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *Transactions on Machine Learning Research*, 2024.

Rui Li, Zeyu Zhang, Xiaohe Bo, Quanyu Dai, Chaozhuo Li, Feng Wen, and Xu Chen. Towards adaptive, scalable, and robust coordination of LLM agents: A dynamic ad-hoc networking perspective. *arXiv preprint arXiv:2602.08009*, 2026.

E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1–2):100–115, 1954. doi: 10.1093/biomet/41.1-2.100.

Harald Psaier and Schahram Dustdar. A survey on self-healing systems: Approaches and systems. *Computing*, 91(1):43–73, 2011. doi: 10.1007/s00607-010-0107-y.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, 2024. doi: 10.18653/v1/2024.acl-long.810.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Hanqing Yang, Jingdi Chen, Marie Siew, Tania Lorido-Botran, and Carlee Joe-Wong. LLM-powered decentralized generative agents with adaptive hierarchical knowledge graph for cooperative planning. *arXiv preprint arXiv:2502.05453*, 2025a. URL https://arxiv.org/abs/2502.05453.

Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. AgentNet: Decentralized evolutionary coordination for LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems*, 2025b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.02485.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025.

Lifan Zheng, Jiawei Chen, Qinghong Yin, Jingyuan Zhang, Xinyi Zeng, and Yu Tian. Rethinking the reliability of multi-agent system: A perspective from byzantine fault tolerance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35012–35020, 2026.

A Ability Assignment Through System Prompts

We test whether an ability assignment in the system prompt can create a reliable performance gap between assigned and unassigned abilities. We compare two conditions on BBH: no statement about competence and a statement that the model is unreliable on unassigned abilities. We use four ability assignments and five questions per task type. Each question is run twice at temperature zero, and a given model receives the same questions in both conditions. Llama-3.2-1B is a weak-model reference and is tested only without the statement; the other five models are evaluated under both conditions.

The accuracy gap is the accuracy on assigned abilities minus the accuracy on unassigned abilities. The additional gap is the change in this quantity relative to the condition with no competence statement. If the prompt reliably enforced specialization, it would enlarge this gap by lowering accuracy on unassigned abilities while preserving accuracy on assigned abilities.

Table A1: Results of ability assignment by system prompt on BBH.

Model	Condition	Unassigned ability acc.	Assigned ability acc.	Accuracy gap	Additional gap
DeepSeek-Chat	No statement	0.812	0.800	-0.012	–
	Told unreliable	0.775	0.792	0.017	0.029
Llama-3.1-70B	No statement	0.781	0.783	0.002	–
	Told unreliable	0.750	0.767	0.017	0.015
GPT-4o-mini	No statement	0.744	0.708	-0.035	–
	Told unreliable	0.675	0.667	-0.008	0.027
GPT-OSS-120B	No statement	0.869	0.825	-0.044	–
	Told unreliable	0.831	0.833	0.002	0.046
Qwen-2.5-7B	No statement	0.494	0.558	0.065	–
	Told unreliable	0.219	0.342	0.123	0.058
Llama-3.2-1B	No statement	0.212	0.192	-0.021	–

Declaring unassigned abilities unreliable increases the accuracy gap by 0.015–0.058 across the five models tested under both conditions. Accuracy on assigned abilities also declines for four of the five models. These results suggest that the tested instructions affect overall response behavior without reliably restricting performance to the assigned abilities. The models can still solve tasks outside their assigned abilities.

In the main evaluation, each agent's declared abilities determine the model used for execution together with its healthy or degraded condition. A healthy agent uses the strong model for a declared ability and the weak model for a missing ability; a degraded agent uses the weak model. With complementary ability pairs, an incorrect route can therefore change the model applied to part of the task and affect task accuracy.

B Implementation Details

B.1 Full Procedure

Alg. 1 places routing, review, Takeover, detection, and reintegration in one procedure. The task state $X_t = (Q_t, T_t, R_t, C_t, Y_t, P_t)$ contains the query, task type, required and covered abilities, ordered accepted outputs, and routing history. Controller i stores its inferred routing state $h_i \in \{\textbf{normal}, \textbf{watched}, \textbf{isolated}\}$, recent evidence W_i indexed by task type and ability, a cache L_i of score and routing-state summaries received from peers, and Bellman–Ford distance $D_i(s)$ to each ability s.

In Assess, H and X denote the ability set and task state supplied by the caller, and b_{probe} is true for a recovery probe. The procedure returns the final score q and the output $\widehat{y}$, which is the original output under **Accept** and the Takeover synthesis under **Takeover**. We use $\tau_{\text{to}} = 0.40$ for the low-score threshold and $\sigma_{\text{com}} = 0.55$ for the review-disagreement threshold.

Algorithm 1 MESHHEAL task execution and routing adaptation

Require: Graph $G = (\mathcal{V}, \mathcal{E})$, abilities $\{\mathcal{K}_i\}$, task stream, parameters in Tab. B11

Initialize $h_i \leftarrow$ **normal**, $\mathcal{K}_i^{\text{original}} \leftarrow \mathcal{K}_i$, $W_i, L_i \leftarrow \emptyset$, $D_i(s)$, $p_i^{\text{probe}} \leftarrow -\infty$
for each task t with query Q_t and type T_t **do**
  sample $i \sim \text{Unif}(\mathcal{V})$; $C_t, Y_t, P_t \leftarrow \emptyset$; $b_{\text{probe}} \leftarrow$ false; $\widehat{y}_t \leftarrow \perp$
  if $h_i =$ **isolated then**
    $i \leftarrow \text{RAWRELAY}(i, L_i, D_i)$
  end if
  $R_t \leftarrow \text{INFERABILITIES}(i, Q_t, T_t)$
  while $\widehat{y}_t = \perp$ and budget remains **do**
    $M_t \leftarrow R_t \setminus C_t$
    if $h_i =$ **isolated** and not b_{probe} **then**
      $i \leftarrow \text{RAWRELAY}(i, L_i, D_i)$; **continue**
    end if
    if not b_{probe} and there exists probe-eligible $j \in \mathcal{N}_i$ with $\mathcal{K}_j^{\text{original}} \cap M_t \neq \emptyset$ **then**
      select one such j; $p_j^{\text{probe}} \leftarrow t$; $i \leftarrow j$; $b_{\text{probe}} \leftarrow$ true; **continue**
    end if
    $H_{i,t} \leftarrow \mathcal{K}_i^{\text{original}} \cap M_t$
    if $H_{i,t} = \emptyset$ **then**
      $i \leftarrow \text{NEXTHOP}(i, M_t, L_i, D_i)$; **continue**
    else if $H_{i,t} \subsetneq M_t$ **then**
      $a \leftarrow$ **Contribute**
    else
      $a \leftarrow$ **Finalize**
    end if
    $X_t \leftarrow (Q_t, T_t, R_t, C_t, Y_t, P_t)$
    $y \leftarrow \text{GENERATE}(i, a, X_t, H_{i,t})$
    $(q_{i,t}, \widehat{y}) \leftarrow \text{ASSESS}(i, y, H_{i,t}, h_i, b_{\text{probe}}, X_t)$
    $h_i' \leftarrow \text{UPDATEEVIDENCE}(i, T_t, H_{i,t}, q_{i,t}, W_i, L_i)$
    if $h_i' \neq h_i$ **then**
      $h_i \leftarrow h_i'$
      if $h_i =$ **isolated then**
        $p_i^{\text{probe}} \leftarrow t$
      end if
      advertise eligibility and distance changes
    end if
    $b_{\text{probe}} \leftarrow$ false
    if $a =$ **Contribute then**
      append $\widehat{y}$ to Y_t; $C_t \leftarrow C_t \cup H_{i,t}$; append i to P_t
      $i \leftarrow \text{NEXTHOP}(i, R_t \setminus C_t, L_i, D_i)$
    else
      $\widehat{y}_t \leftarrow \widehat{y}$
    end if
  end while
  if $\widehat{y}_t = \perp$ **then**
    mark task t unanswered
  else
    output $\widehat{y}_t$
  end if
end for

The three collaboration actions specify how the task state changes. Handoff forwards the state unchanged; Contribute adds a reviewed partial result and expands C_t; and Finalize completes the remaining work and proposes a full answer. Review then chooses Accept or Takeover. For Finalize, the controller returns the accepted answer or its replacement. Review determines whether the current output is kept or replaced, while the inferred routing state affects later routes. The displayed prompts and traces use these same action names.

UPDATEEVIDENCE records $q_{i,t}$ for every $s \in H_{i,t}$, updates the matched peer-relative shortfall $x_{i,t,s}$ when the required summaries are available, and appends the resulting evidence to W_i. It then applies RELATIVEUPDATE to obtain the agent-level routing state h_i', following the detector rules in Sec. 4.2.2.

Algorithm 2 MESHHEAL review and correction

procedure ASSESS($i, y, H, h_i, b_{\text{probe}}, X$)
  obtain independent scores u_1, u_2, u_3 from one eligible **normal** reviewer with the required ability
  $q_0 \leftarrow \text{median}(u_1, u_2, u_3)$; $s_u \leftarrow \text{sd}_{\text{sample}}(u_1, u_2, u_3)$
  if $q_0 \le \tau_{\text{to}}$ or $h_i =$ **watched** or b_{probe} or $s_u \ge \sigma_{\text{com}}$ **then**
    select two distinct eligible reviewers A and B when possible; otherwise make two independent calls to the same eligible reviewer
    reviewers judge independently, exchange explanations, and judge again; let the final scores be v_A, v_B
    $q \leftarrow (v_A + v_B)/2$
    if $q \le \tau_{\text{to}}$ **then**
      $\widehat{y} \leftarrow$ TAKEOVERSYNTHESIS(A, B, y, X)
      return $(q, \widehat{y})$
    end if
  else
    $q \leftarrow q_0$
  end if
  return (q, y)
end procedure

RAWRELAY forwards the unchanged task state from an isolated agent, either the entry agent or an intermediate relay, toward a normal agent using D_i, without any LLM call.

NEXTHOP reads only the cached eligibility states of the current controller's neighbors in L_i and its distance table D_i. It samples a missing ability uniformly and routes toward its nearest holder using the cached Bellman–Ford distances. Normal and watched agents have identical execution eligibility; isolated agents are excluded as execution providers but may relay the task state without LLM calls.

ASSESS attributes $q_{i,t}$ to the agent that generated the original output under both Accept and Takeover. RELATIVEUPDATE computes the shortfall $x_{i,t,s}$ only from observations with the same task type and ability as the current output; the normal range $m_{i,T}$ and $p^{25}_{i,T}$ pools the mean shortfalls of normal neighbors over all abilities within the same task type T. The resulting threshold decision updates one inferred routing state for the agent. The procedure also requires a minimum number of observations, checks for two consecutive crossings of the isolation threshold, and applies reversible routing-state transitions. Score summaries update neighbor caches, while eligibility and distance advertisements propagate from neighbor to neighbor. Probe timers are independent local events, so no coordinator scans the network. Later tasks use the latest local eligibility and distance tables.

B.2 AGENTS AND ABILITIES

Table B1: BBH ability assignment in the six-agent setting.

Agent	Declared abilities
A1	reasoning, mathematical, knowledge
A2	reasoning, mathematical, knowledge
A3 (degraded)	mathematical, knowledge, sequence, inference
A4	language, sequence, spatial, inference
A5 (degraded)	reasoning, language, spatial
A6	language, sequence, spatial, inference

Table B2: BBH task types and required abilities.

Task type	Required abilities
`causal_judgement`	reasoning + inference
`formal_fallacies`	reasoning + inference
`tracking_shuffled_objects_five_objects`	reasoning + sequence
`geometric_shapes`	mathematical + spatial
`object_counting`	mathematical + spatial
`date_understanding`	mathematical + language
`ruin_names`	language + knowledge

The MATH mapping separates operations on mathematical objects from the reasoning needed to formulate or complete a solution. Algebraic subjects pair symbolic transformation with multi-step deduction. Geometry and precalculus pair calculation or theorem knowledge with reasoning about figures, while subjects solved by considering separate cases pair symbolic constraints with

Table B3: MATH ability assignment.

Agent	Declared abilities
A1	symbolic_manipulation, numeric_computation, theorem_knowledge
A2	symbolic_manipulation, numeric_computation, theorem_knowledge
A3 (degraded)	numeric_computation, theorem_knowledge, enumeration, multi_step_deduction
A4	enumeration, multi_step_deduction, problem_translation, figure_reasoning
A5 (degraded)	symbolic_manipulation, problem_translation, figure_reasoning
A6	enumeration, multi_step_deduction, problem_translation, figure_reasoning

Table B4: MATH subjects, required abilities, and assignment rationale.

Subject	Required abilities	Rationale
prealgebra	numeric_computation + problem_translation	word-problem modeling + arithmetic
algebra	symbolic_manipulation + multi_step_deduction	expression transformation + multi-step solving
geometry	numeric_computation + figure_reasoning	geometric relations + numerical calculation
number_theory	symbolic_manipulation + enumeration	algebraic constraints + systematic cases
counting_and_probability	symbolic_manipulation + enumeration	counting expressions + complete case coverage
intermediate_algebra	symbolic_manipulation + multi_step_deduction	deeper transformations + longer deduction
precalculus	theorem_knowledge + figure_reasoning	function properties + graphical intuition

enumeration. The assignments follow common stages of mathematical problem solving and associate each ability with a specific step in the solution process.

Table B5: MMLU-Pro ability assignment.

Agent	Declared abilities
A1	humanities_knowledge, science_knowledge, quantitative_methods
A2	humanities_knowledge, science_knowledge, quantitative_methods
A3 (degraded)	science_knowledge, quantitative_methods, scenario_inference, stepwise_analysis
A4	scenario_inference, stepwise_analysis, text_comprehension, structural_reasoning
A5 (degraded)	humanities_knowledge, text_comprehension, structural_reasoning
A6	scenario_inference, stepwise_analysis, text_comprehension, structural_reasoning

The MMLU-Pro mapping pairs the main source of domain knowledge with the operation needed to apply it. Here, structural_reasoning denotes reasoning about structures, processes, and physical representations. Knowledge-heavy domains require text comprehension, contextual inference, or reasoning about structure. Economics and physics combine quantitative methods with textual or structural interpretation. The two parts remain complementary: one agent supplies relevant knowledge or methods, and another applies them to the question.

The BBH assignments are inspired by the reasoning capabilities represented across BBH / BIG-Bench Extra Hard (Kazemi et al., 2025); MATH and MMLU-Pro use names chosen for each dataset while preserving the same controlled structure. Within each dataset, exactly three agents hold each ability, and every task combines two abilities with disjoint sets of agents. Under this assignment, at least two agents are needed to cover both required abilities. Every ability held by degraded agents A3 and A5 is also held by two healthy agents after both targets degrade. The mapping defines complementary ability sets and preserves alternatives to each degraded agent.

At task entry, INFERABILITIES receives the complete seven-ability vocabulary, brief ability descriptions, the query and task type, and examples covering all evaluated task types and their required ability pairs. The entry LLM selects a pair and stores it in R_t; controllers then use this inferred pair for action selection and routing. Ability inference is therefore a constrained choice over the documented mappings in Tables B2, B4, and B6.

B.3 MODELS, DATA, AND DEGRADATION

The main stream contains 504 tasks in 12 balanced blocks. Tasks 1–252 form the healthy phase. Tasks 253–504 use different questions from the same task-type distribution and form the degraded phase. At task 253, A3 and A5 switch from the strong model to the weak model while their declared ability sets remain unchanged.

A missing or unparsable answer is counted as incorrect. Table B7 shows that the LLM can accurately infer the abilities required for each task. Results for persistent degradation on each of BBH, MATH, and MMLU-Pro, together with the studies of alternative models and individual components, average

Table B6: MMLU-Pro domains, required abilities, and assignment rationale.

Domain	Required abilities	Rationale
health	science_knowledge + text_comprehension	medical knowledge + comprehension of long questions
history	humanities_knowledge + scenario_inference	historical facts + contextual inference
biology	science_knowledge + structural_reasoning	biology knowledge + structural/process reasoning
psychology	humanities_knowledge + scenario_inference	theory + scenario judgment
philosophy	humanities_knowledge + stepwise_analysis	doctrine + step-by-step argument analysis
economics	quantitative_methods + text_comprehension	interpretation + model/calculation
physics	quantitative_methods + structural_reasoning	quantitative analysis + physical representation

Table B7: Exact match between inferred ability pairs and the documented task-type mappings.

Dataset	BBH	MATH	MMLU-Pro	Overall
Exact pair match	100%	100%	100%	100%

five seeded task streams per setting. For the studies of degradation and recovery at different times, network scaling, and detection, the corresponding tables distinguish aggregated accuracies from agent counts; Tab. C5 reports a representative state trajectory.

The healthy and degraded phases preserve the task-type distribution and use disjoint questions, so the task mix remains comparable across distinct questions. Degradation is introduced by changing the model while preserving the agent identifier, declared ability set, and communication links. Task outputs and peer reviews provide the resulting evidence for routing-state updates.

B.4 Baselines

Each baseline uses the authors' released implementation. We retain its native routing, selection, filtering, and aggregation logic and connect it to a shared execution layer that fixes the task order, agent identities, ability assignments, models, degradation schedule, worker prompts, and Handoff–Contribute–Finalize task interface. The resulting comparison covers decentralized routing or reputation (AgentNet and RAPS), mandatory central delegation (AutoGen), and repeated candidate generation and aggregation (SAC and Symphony). We count every model call and all model tokens across routing, execution, review, and aggregation, including repeated review samples and committee rounds. Token cost includes both input prompt tokens and generated output tokens.

B.5 Review, Accept, and Takeover

For each Contribute or Finalize output, the controller samples one reviewer in the **normal** state that holds the abilities used to produce it and calls that reviewer three times in independent contexts. **Watched** and **isolated** peers are ineligible throughout initial review and committee selection. Every call uses the same rubric for that type of output and the same score definitions shown in App. D. The median of the initial scores is q_0. Let $\bar{u} = (u_1 + u_2 + u_3)/3$. Disagreement among the three scores is measured by the sample standard deviation

$$s_u = \sqrt{\frac{1}{3-1}\sum_{r=1}^{3}(u_r - \bar{u})^2}.$$

A two-reviewer committee is called when $q_0 \leq 0.40$, the agent is **watched**, the output is a recovery probe, or $s_u \geq 0.55$. All other outputs are accepted using the initial median score.

The controller selects distinct peers as Reviewer A and Reviewer B whenever two are available. If only one eligible reviewer is available, the controller uses that peer in two independent review contexts. The reviewers first score the output independently and explain their judgments. Each reviewer then sees both first-round scores and explanations and independently returns a second judgment. A score above 0.40 leads to Accept. A score at most 0.40 leads to Takeover: the two reviewers produce correction candidates, and a normal committee reviewer synthesizes them with the accepted task context. The controller replaces the original output with this synthesis. This replacement is not reviewed recursively; its cost is included, while the score attributed to the agent that generated the original output remains the detector evidence.

Table B8: Models used for task execution and induced degradation.

Execution condition	Model
Declared ability, healthy agent	`openai/gpt-oss-120b`
Missing ability or degraded agent	`meta-llama/llama-3.2-1b-instruct`

Table B9: Datasets and scoring rules.

Dataset	Scoring rule
BIG-Bench Hard	Exact answer extraction and matching
MATH	Mathematical equivalence with `math_equal`
MMLU-Pro	Extracted option letter

Mixture-of-Agents allows the same LLM to be reused within or across layers and to generate independently sampled candidates before aggregation (Wang et al., 2025). We use this form of model reuse when only one eligible peer in the **normal** state has the relevant ability. The controller calls that peer in two independent contexts labeled Reviewer A and Reviewer B. This preserves review and Takeover availability and provides two separately sampled judgments. Every reviewed output produces one detector score, attributed to the agent that generated the original output under both Accept and Takeover.

The two-timescale procedure connects correction at the fast timescale to detection at the slow timescale. One reviewer provides the initial check, and committee discussion is reserved for low-scoring or uncertain outputs, outputs from **watched** agents, and recovery probes. Under Accept, the reviewed output remains in the task state. Under Takeover, the synthesized replacement takes its place. In both cases, the final review score remains attributed to the agent that generated the original output and supports later routing updates.

Degradation detector. Each output produced by agent i on task t receives a final review score $q_{i,t}$. For every ability $s \in H_{i,t}$ used in that output, the controller records $(T_t, s, q_{i,t})$ and maintains a separate detector for (T_t, s). Let $\mathcal{P}_{i,T,s}(t)$ contain the direct neighbors of agent i that are currently in the **normal** state and have recent reviewed outputs for the same task type T and ability s. Let $\mu_{j,T,s}(t)$ be neighbor j's mean review score over these recent outputs. The reference score is

$$\bar{q}_{-i,T,s}(t) = \frac{1}{|\mathcal{P}_{i,T,s}(t)|} \sum_{j \in \mathcal{P}_{i,T,s}(t)} \mu_{j,T,s}(t),$$

and the shortfall for ability s is

$$x_{i,t,s} = \bar{q}_{-i,T_t,s}(t) - q_{i,t}.$$

For each (T, s), controller i averages its latest eight matched shortfalls to obtain $\bar{x}_{i,T,s}$. For the second comparison, the controller uses the mean shortfalls maintained by its **normal** neighbors for all abilities under the same task type T. Let

$$\mathcal{C}_{i,T}(t) = \{(j, s') : j \in \mathcal{N}_i,\ h_j = \textbf{normal},\ \text{and } \bar{x}_{j,T,s'} \text{ is available}\}.$$

Thus, a neighbor may contribute multiple values, one for each ability s' for which it has enough matched evidence. Let

$$m_{i,T} = \text{median}\left\{\bar{x}_{j,T,s'} : (j, s') \in \mathcal{C}_{i,T}(t)\right\},$$

and

$$p^{25}_{i,T} = Q_{0.25}\left(\{\bar{x}_{j,T,s'} : (j, s') \in \mathcal{C}_{i,T}(t)\}\right).$$

The detector statistic is

$$z_{i,T,s} = \frac{\bar{x}_{i,T,s} - m_{i,T}}{\max\Big(m_{i,T} - p^{25}_{i,T},\ 0.10\Big)}.$$

The first comparison is ability-specific, while the second comparison pools the mean shortfalls $\bar{x}_{j,T,s'}$ across abilities to estimate the normal range for task type T. The current output updates the statistic for its own task type and ability, while the inferred routing state applies to the agent. After three matched observations, a **normal** agent becomes **watched** when $z_{i,T,s} \geq 1.2$. A **normal** or **watched** agent becomes **isolated** after $z_{i,T,s} \geq 2.0$ on two consecutive updates with matched evidence; the

Table B10: Baseline implementations.

Label	Source	Implementation used in our framework
AgentNet	Yang et al. (2025b)	Native decentralized routing with adaptive edge weights and pruning.
RAPS	Li et al. (2026)	Peer reputation updates local links and routing avoids low-reputation peers.
AutoGen	Wu et al. (2023)	A mandatory GroupChatManager-style hub delegates work, receives worker outputs, and produces the final answer.
SAC	Lee et al. (2026)	Three teams produce answers, run two filtering and refinement rounds, and aggregate by majority.
Symphony	Guan et al. (2026)	GlobalLinUCB selects agents with the required abilities; three plans run in parallel; the final result is selected by voting and used to update routing statistics.

counter resets when $z_{i,T,s} < 2.0$. Recovery applies the thresholds in reverse: an **isolated** agent moves to **watched** when $z_{i,T,s} < 2.0$, and a later **watched** update with $z_{i,T,s} < 1.2$ restores **normal** status. The minimum of three matched observations applies separately to each agent, task type, and ability. If this minimum is not met, or either $\mathcal{P}_{i,T,s}(t)$ or $\mathcal{C}_{i,T}(t)$ is empty, the current routing state is retained.

Peer summaries use recent answers to different questions with the same task type and ability. The rolling window averages over question-level variation. After every matched review, the controller sends updated score and shortfall summaries to direct neighbors. Routing-state and distance summaries are sent when those values change so that neighboring controllers can update their routing tables.

Under the evaluated ability mappings, each produced output is associated with exactly one remaining required ability, i.e., $|H_{i,t}| = 1$. Thus, each reviewed output updates one (T, s) detector and one agent-level routing state.

B.6 PARAMETERS

Table B11: Main MESHHEAL parameters.

Parameter	Value
Committee reviewer slots	2
Committee deliberation rounds	2
Takeover correction candidates	2
Routing hop limit	4 hops
Maximum agents per task	6
Maximum reviewed outputs per task	3
Evidence window per agent, task type, and ability	8 observations
Isolation threshold	$z \geq 2.0$ (2 consecutive updates)
Watch threshold	$z \geq 1.2$
Scale floor	0.10
Minimum observations	3
Minimum recovery-probe interval	10 tasks
Takeover score threshold	$q \leq 0.40$
Initial review samples	3
Committee sample-s.d. trigger	$s_u \geq 0.55$

The observation window and the requirement of two consecutive threshold crossings provide repeated evidence before isolation. The **watched** state adds scrutiny before isolation, and the local probe timer controls how quickly fresh evidence can replace degraded observations after recovery. Each isolated agent's controller counts tasks since isolation or its last probe. After 10 tasks, the agent becomes probe-eligible; a neighbor schedules the probe when matching work next arrives, and the counter resets when the probe runs. Routing and review limits bound the amount of work per task. Low initial scores and high uncertainty trigger committee review and its additional cost; Takeover follows only when the committee's final score remains at or below 0.40. Unless explicitly varied in an ablation, all rubrics and parameters for review and detection are fixed before evaluation and used unchanged across datasets, model pairs, and network sizes.

B.7 SPARSE NETWORKS

All reviewers, including committee members, are direct neighbors of the agent whose output they review. In the evaluated sparse graphs, every agent producing an ordinary task output has at least one neighbor in the **normal** state with the same ability. Each agent selected for degradation has two such neighbors after both targets are excluded from reviewer selection, so its **watched** outputs and recovery probes use distinct peers for the two reviewer contexts. Another agent can have only one eligible neighbor with the same ability after an agent with that ability degrades; in that case, the controller reuses that peer in the two independent reviewer contexts described above. Every agent is also within three hops of an agent in the **normal** state for each required ability, leaving one hop of slack under the routing limit of four.

Table B12: Sparse graph size. Edge counts treat each bidirectional connection as one undirected link.

Agents	Sparse edges	Fraction of complete graph	Maximum ability distance
96	228	5.00%	3
288	705	1.71%	3
576	1454	0.88%	3

The sparse graphs preserve the local conditions for review and routing: every agent has a reviewer with the same ability, and every required ability is available from a reachable healthy agent within the routing limit. The detector compares eligible neighbors in the **normal** state. Edge density falls from 5.00% at 96 agents to 0.88% at 576 agents. The comparison between complete and sparse graphs changes connectivity while preserving these conditions.

C DETAILED RESULTS

This appendix provides the detailed results summarized in Sec. 6. It first separates accuracy by dataset and reports inference cost. It then varies the model pair, degradation timing, and network topology before examining the committee and detector. Unless stated otherwise, all experiments use the task construction and cost accounting described in App. B.

C.1 MAIN RESULTS

Table C1: Full persistent-degradation results, reported as mean ± standard deviation over five seeded task streams per dataset. Each stacked entry gives H above D for the healthy and degraded phases. The Mean column averages the three datasets within each seed, and Change is computed within seed before aggregation. Bold marks the best mean.

Method	BBH	MATH	MMLU-Pro	Mean	Change
	$.899 \pm .027$	$.821 \pm .019$	$.774 \pm .016$	$\mathbf{.831 \pm .013}$	
MESHHEAL	$\mathbf{.900 \pm .014}$	$\mathbf{.802 \pm .021}$	$\mathbf{.814 \pm .018}$	$\mathbf{.839 \pm .012}$	$\mathbf{+.007 \pm .012}$
	$.896 \pm .022$	$.818 \pm .020$	$.766 \pm .018$	$.827 \pm .013$	
Symphony	$.889 \pm .007$	$.790 \pm .022$	$.742 \pm .026$	$.807 \pm .014$	$-.020 \pm .018$
	$.896 \pm .016$	$.794 \pm .026$	$.778 \pm .020$	$.822 \pm .014$	
SAC	$.894 \pm .007$	$.639 \pm .045$	$.706 \pm .034$	$.746 \pm .021$	$-.076 \pm .026$
	$.884 \pm .029$	$.726 \pm .034$	$.790 \pm .024$	$.800 \pm .019$	
RAPS	$.740 \pm .070$	$.627 \pm .061$	$.591 \pm .067$	$.653 \pm .041$	$-.147 \pm .045$
	$.862 \pm .035$	$.631 \pm .041$	$.694 \pm .034$	$.729 \pm .023$	
AgentNet	$.668 \pm .066$	$.476 \pm .056$	$.528 \pm .061$	$.557 \pm .039$	$-.172 \pm .047$
	$.898 \pm .013$	$.665 \pm .030$	$.796 \pm .019$	$.787 \pm .014$	
AutoGen	$.525 \pm .034$	$.214 \pm .029$	$.532 \pm .043$	$.424 \pm .023$	$-.363 \pm .028$

MESHHEAL has the highest mean degraded-phase accuracy on all three datasets. Its aggregate accuracy is 0.831 ± 0.013 in the healthy phase and 0.839 ± 0.012 in the degraded phase, with a paired phase change of $+0.007 \pm 0.012$. On MATH, its mean changes from 0.821 ± 0.019 to 0.802 ± 0.021, while Symphony and SAC show mean changes of -0.028 and -0.155. BBH and MMLU-Pro have nonnegative mean changes. RAPS, AgentNet, and AutoGen show mean phase changes of -0.147 ± 0.045, -0.172 ± 0.047, and -0.363 ± 0.028.

After degradation onset, MESHHEAL's mean model calls increase from 12.4 ± 0.2 to 13.6 ± 0.3 per task because the system performs more review and rerouting, while mean total model tokens are (51 ± 3)k and (51 ± 5)k in the two phases. Symphony's mean token cost increases from (86 ± 5)k

Table C2: Healthy- and degraded-phase cost per task, reported as mean ± standard deviation over five seeds after averaging datasets within seed. Tokens include both input and output tokens; their means and standard deviations are rounded to the nearest thousand.

Method	H calls	D calls	H tokens (k)	D tokens (k)
MESHHEAL	12.4 ± 0.2	13.6 ± 0.3	51 ± 3	51 ± 5
Symphony	18.1 ± 0.1	23.4 ± 0.1	86 ± 5	115 ± 9
SAC	30.3 ± 0.1	39.9 ± 0.6	116 ± 4	178 ± 13
RAPS	8.2 ± 0.5	10.9 ± 0.3	39 ± 5	48 ± 3
AgentNet	7.8 ± 0.7	10.3 ± 0.2	34 ± 4	43 ± 2
AutoGen	6.1 ± 0.0	6.0 ± 0.0	15 ± 0	15 ± 0

to (115 ± 9)k, and SAC's from (116 ± 4)k to (178 ± 13)k as they repeatedly generate and refine parallel candidates. RAPS and AgentNet have lower mean token costs together with mean accuracy changes of -0.147 ± 0.045 and -0.172 ± 0.047. AutoGen has both the lowest mean cost and the lowest mean degraded-phase accuracy when its mandatory manager degrades.

C.2 MODELS AND RECOVERY

Table C3: Model pair A: GPT-OSS-120B is the strong model and Gemma-3-4B is the weak model. Results are mean ± standard deviation over five seeded task streams; Change is paired within seed and tokens include both input and output tokens.

Method	Healthy acc.	Degraded acc.	Change	Tokens/task
MESHHEAL	$\mathbf{0.929 \pm 0.018}$	$\mathbf{0.905 \pm 0.016}$	$\mathbf{-0.024 \pm 0.014}$	$56{,}687 \pm 2{,}431$
Symphony	0.917 ± 0.020	0.865 ± 0.021	-0.052 ± 0.017	$100{,}685 \pm 4{,}782$
SAC	0.917 ± 0.019	0.829 ± 0.030	-0.088 ± 0.026	$130{,}938 \pm 6{,}214$
RAPS	0.877 ± 0.031	0.623 ± 0.064	-0.254 ± 0.055	$35{,}746 \pm 2{,}963$
AgentNet	0.857 ± 0.037	0.694 ± 0.056	-0.163 ± 0.048	$36{,}978 \pm 3{,}284$
AutoGen	0.905 ± 0.016	0.734 ± 0.041	-0.171 ± 0.038	$15{,}377 \pm 1{,}126$

Table C4: Model pair B: Llama-3.3-70B is the strong model and Llama-3.2-1B is the weak model. Results are mean ± standard deviation over five seeded task streams; Change is paired within seed and tokens include both input and output tokens.

Method	Healthy acc.	Degraded acc.	Change	Tokens/task
MESHHEAL	0.790 ± 0.025	0.782 ± 0.023	$\mathbf{-0.008 \pm 0.015}$	$50{,}802 \pm 2{,}317$
Symphony	$\mathbf{0.806 \pm 0.023}$	$\mathbf{0.786 \pm 0.021}$	-0.020 ± 0.017	$96{,}982 \pm 4{,}563$
SAC	$\mathbf{0.806 \pm 0.022}$	0.754 ± 0.032	-0.052 ± 0.027	$122{,}234 \pm 5{,}908$
RAPS	0.790 ± 0.030	0.571 ± 0.061	-0.219 ± 0.053	$30{,}761 \pm 2{,}684$
AgentNet	0.774 ± 0.038	0.476 ± 0.065	-0.298 ± 0.056	$29{,}487 \pm 2{,}792$
AutoGen	0.754 ± 0.028	0.349 ± 0.042	-0.405 ± 0.038	$12{,}986 \pm 956$

The accuracy ranking changes with the model pair, while MESHHEAL uses fewer tokens than Symphony under both pairs. Under pair A, MESHHEAL's mean degraded-phase accuracy is 0.905 ± 0.016, compared with 0.865 ± 0.021 for Symphony, and its mean token cost is 56,687 versus 100,685. Under pair B, the degraded-phase means are 0.782 ± 0.023 and 0.786 ± 0.021; MESHHEAL has a smaller mean paired phase change (-0.008 ± 0.015 versus -0.020 ± 0.017) and uses 52% of Symphony's mean tokens.

Tab. 3 reports mean results over five seeds for the experiment with degradation and recovery at different times. All methods complete the full 630-task stream, with 126 tasks in each phase. The fully recovered mean accuracy is 0.857, equivalent to approximately 108 correct answers per 126-task phase. Across the 504 tasks after the first agent degrades, the reported mean pooled accuracy is 0.812 at 49k tokens per task, compared with 0.806 and 105k for Symphony. Mean accuracy rises from 0.770 when both targets are degraded to 0.841 after A5 recovers and 0.857 after both recover. RAPS and AgentNet reach 0.524 and 0.563 after both models are restored.

Warm-up is an evidence-collection phase in which all agents remain in ordinary routing. In the representative trajectory in Tab. C5, A5 becomes watched at task 129 and isolated at task 132, five tasks after degradation begins. A3 becomes watched at task 267 and isolated at task 268, giving a detection delay of 15 tasks. Both remain isolated throughout their degraded intervals, during which they are observed only through recovery probes. After backend restoration, A5 and A3 move to watched at tasks 422 and 548 and return to normal routing at tasks 430 and 550, giving reintegration delays of 51 and 45 tasks, respectively.

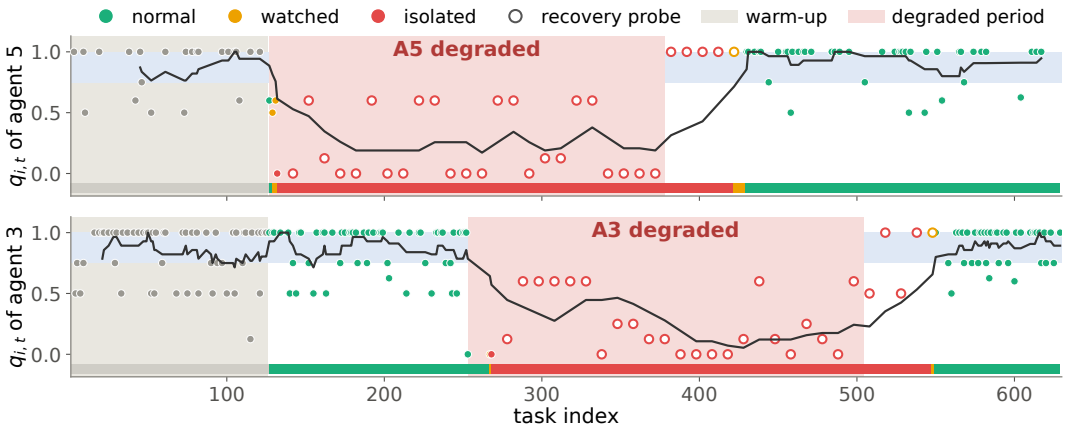


Figure C1: Detection and reintegration under degradation and recovery at different times. Pink regions mark injected degradation. Each marker is one reviewed output, colored by the inferred state after the update; hollow markers are recovery probes. The bottom band shows the inferred state at each task, the curve shows the mean review score, and the blue region is the reference range for normal agents.

Table C5: Inferred routing-state changes in a representative run with degradation and recovery at different times.

Agent	True degraded interval	State sequence
A5	127–378	first evidence at task 3; **normal** during warm-up → 129 **watched** → 132 **isolated** → 422 **watched** → 430 **normal**
A3	253–504	first evidence at task 4; **normal** during warm-up → 267 **watched** → 268 **isolated** → 548 **watched** → 550 **normal**

C.3 Larger Networks

Tab. 2(b) gives the accuracy and isolation results for complete and sparse graphs. Every degraded agent is **isolated** in every topology. Complete-graph accuracy ranges from 0.808 to 0.827, and sparse-graph accuracy ranges from 0.803 to 0.818. The largest difference is 0.010 at 288 agents; the sparse graph is 0.006 higher at 576 agents. Expanding the task stream with the number of replicated six-agent ability profiles keeps detector evidence per agent comparable across scales. Tab. B12 gives the graph sizes and confirms that the maximum distance to a healthy agent with a required ability remains within the routing hop limit.

C.4 Ablations

Table C6: Component study, reported as mean ± standard deviation over five seeded task streams. Phase change is paired within seed.

Metric	Full method	Single reviewer	No Takeover
Healthy accuracy	0.899 ± 0.027	0.909 ± 0.022	0.879 ± 0.021
Degraded-phase accuracy	$\mathbf{0.900 \pm 0.014}$	0.865 ± 0.029	0.844 ± 0.036
Paired phase change	$\mathbf{+0.001 \pm 0.016}$	-0.044 ± 0.024	-0.035 ± 0.030

The Single reviewer variant uses the initial reviewer with the same ability for the final judgment and any Takeover correction; it retains detection and rerouting while removing multi-reviewer discussion. The No Takeover variant retains review, committee escalation, detection, and rerouting but leaves a low-scoring output in the current task. The full configuration exceeds their mean degraded-phase accuracies by 0.035 and 0.056, respectively. No Takeover also lowers the healthy-phase mean by 0.020 because it leaves occasional low-quality outputs from healthy LLMs unchanged. The reported paired phase changes are $+0.001 \pm 0.016$, -0.044 ± 0.024, and -0.035 ± 0.030.

We next compare the detector based on peer comparisons with a fixed-threshold rule and a cumulative-sum (CUSUM) rule under the two model pairs (Page, 1954).

Table C7: Detector performance across two BBH model pairs. Missed counts are out of two degraded agents and false-isolation counts are out of four healthy agents; each setting uses the full 504-task stream.

		Pair A		Pair B	
Rule	Evidence	Missed	False	Missed	False
Absolute threshold	Own scores	0/2	4/4	0/2	0/4
CUSUM	Own scores	0/2	2/4	1/2	0/4
Relative peer comparison	Scores relative to peers	**0/2**	**0/4**	**0/2**	**0/4**

The absolute rule detects both degraded agents under both model pairs. Under pair A, it also isolates all four healthy agents. CUSUM reduces those false isolations to two and misses one degraded agent under pair B. Using the same review rubrics and detector parameters, relative peer comparison produces neither error under either model pair. Its reference adapts to peer scores for the same task type and ability while the decision thresholds remain fixed. The peer-relative reference relies on matched normal-neighbor summaries remaining representative of healthy performance.

The component study evaluates committee review and Takeover at the fast timescale. The detector comparison evaluates how peer-based evidence changes routing at the slow timescale under the two tested model pairs. Together, these studies support using review scores for both current-task correction and future routing.

C.5 Review and Takeover Audit

We examine the initial review, committee reassessment, and Takeover across five seeded task streams. In the first two panels of Tab. C8, outputs from degraded agents are treated as positives and outputs from healthy agents as negatives. These labels describe the agent's injected condition, not the correctness of each output. A healthy agent can still produce an incorrect output, so the reported rates measure discrimination between injected conditions rather than detection of incorrect answers.

The initial review assigns mean scores of 0.89 ± 0.01 and 0.08 ± 0.01 to outputs from healthy and degraded agents, respectively. Each seed contains 31–33 outputs from degraded agents in this audit, of which 2–4 are missed by the initial low-score decision. The committee escalation rate is higher because **watched** status or disagreement among the initial scores can also trigger committee review.

After committee reassessment, recall changes from 0.90 ± 0.03 to 0.88 ± 0.01, while the false-alarm rate decreases from 0.053 ± 0.007 to 0.044 ± 0.008. Among initially flagged outputs, the committee clears 0.33 ± 0.07 of healthy-agent cases and retains 0.95 ± 0.02 of degraded-agent cases.

Finalize answers can be checked directly against the gold answer. Among 32 initially wrong answers entering Takeover, 15 are corrected. The synthesis changes 18 of these answers, and 15 of the 18 changes are correct; the other 14 wrong answers are left unchanged. Among 32 initially correct answers entering Takeover, only one is changed to an incorrect answer.

A Contribute output cannot be scored directly from the final gold answer. Among the 137 tasks where Takeover replaces a Contribute output, 86% end with a correct final answer. This number reports the final task outcome and is not a direct measure of intermediate output correctness.

D Runtime Prompts

This appendix presents representative runtime prompts for communicating ability matches to the LLM and reviewing outputs. Shaded cards distinguish prompt instructions from explanatory prose. Alg. 1 specifies the surrounding controller logic. For presentation, we standardize ability terminology and the action names Handoff, Contribute, and Finalize. Executable JSON field names, other wording, thresholds, and placeholders are retained; line breaks are adjusted to fit the page.

D.1 Review Prompts

Review depends on the task action. A Contribute reviewer evaluates the assigned intermediate output. A Finalize reviewer evaluates the finalizer's own output and whether it commits a concrete answer.

(a) Initial review

Metric	Value
Mean score, healthy-agent outputs	0.89 ± 0.01
Mean score, degraded-agent outputs	0.08 ± 0.01
Recall on degraded-agent outputs	0.90 ± 0.03
False-alarm rate on healthy-agent outputs	0.053 ± 0.007
Committee escalation rate for degraded-agent outputs	0.98 ± 0.02

(b) Initial review and committee reassessment

Metric	Initial review	After committee
Recall on degraded-agent outputs	0.90 ± 0.03	0.88 ± 0.01
False-alarm rate on healthy-agent outputs	0.053 ± 0.007	0.044 ± 0.008
Initial flags cleared, healthy-agent outputs	–	0.33 ± 0.07
Initial flags kept, degraded-agent outputs	–	0.95 ± 0.02

(c) Takeover outcomes

Outcome	Result
Finalize answers wrong before Takeover ($n = 32$)	
corrected	$15/32$ (47%)
changed but still wrong	$3/32$ (9%)
left unchanged	$14/32$ (44%)
corrected, among those changed	$15/18$ (83%)
Finalize answers correct before Takeover ($n = 32$)	
changed to incorrect	$1/32$ (3%)
Takeover of Contribute outputs ($n = 137$)	
task correct at the end	86%

Table C8: Agent review performance and Takeover outcomes.

The prompt identifiers use the same Contribute and Finalize names as the method. In both cases, the controller selects a reviewer that holds the relevant ability. Process issues such as exceeding the assigned scope or ignoring a draft remain advisory unless they create a concrete correctness error. Accept and Takeover are controller actions derived from the resulting score.

The cards retain the executed instructions with ability terminology and action labels standardized for presentation. The assigned part refers to work requiring one of the task's two abilities; `draft-half` refers to the earlier output covering the other ability. Executable JSON field names are unchanged.

Contribute judge system prompt (COMMITTEE_CONTRIBUTE_JUDGE_SYSTEM)

```
You review a DRAFTER's INTERMEDIATE contribution in a multi-agent system. The
drafter was delegated just ONE part of a task requiring two abilities; it
should Contribute that half and leave the other half for Handoff to a teammate -
it should NOT Finalize (that needs the other ability too). You give one noisy
review signal; do not solve, repair, or decide degradation. Judge ONLY the
drafter's half, using only a ability you actually hold (see
REVIEWER_COMPETENCE). Return only valid JSON.
```

The scoring schema anchors the returned signal and requires every low score to identify a concrete correctness error. The controller assigns reviewers who hold the relevant ability. If a reviewer does not hold that ability, the prompt directs it to return a neutral score with low confidence.

Contribute judge scoring schema

```
Score each in [0,1]:
- half_correct: ONLY IF you are competent at the drafter's ability - re-
  derive that half; is it correct? LOW for a concrete error (miscalculation,
  invalid step, contradiction with the given facts/objects). If you are NOT
  competent at that ability, set 0.5 and lower confidence.
- in_scope: did the drafter STAY IN ITS LANE - do its half and leave the final
    decision OPEN?

SCORING RULE - normal_working_score MUST equal half_correct: score the
CORRECTNESS of the delegated half only. in_scope is ADVISORY: it must NOT lower
normal_working_score.
anchors: 1.00 half is sound / 0.50 cannot verify or weak / 0.25 concrete error /
0.00 unusable garbage for the delegated half.

Return ONLY valid JSON:
{ "half_correct": .., "in_scope": .., "normal_working_score": .., "confidence": ..,
  "specific_failure_found": bool, "failure_type": "none|wrong_half|over_reach|other",
  "failure_evidence": "quote from the drafter output; empty if none",
  "one_sentence_reason": ".." }
Constraints: normal_working_score<=0.40 => specific_failure_found=true + non-
empty evidence of a CORRECTNESS failure; over_reach alone NEVER sets
specific_failure_found.
```

Finalize judge system prompt (COMMITTEE_FINALIZE_JUDGE_SYSTEM)

```
You review a FINALIZER's answer in a multi-agent system. The finalizer did its
OWN part and must COMBINE it with the draft-half already in the trajectory to
commit ONE concrete final answer. The draft-half was produced by a teammate
using a ability the finalizer (and you) do NOT hold: treat it as GIVEN - this
review does not score whether the draft-half is right. Judge only the
finalizer's own half and how faithfully it combined. You give one noisy review
signal; do not solve/repair. Return only valid JSON.
```

Finalize returns one concrete, scoreable answer; if no concrete answer is given, the score is at most 0.25. It uses the same numerical anchors and the same requirement to identify a concrete error as Contribute, applied to the finalizer's work within its declared ability and the committed answer. Draft handling affects the score when it creates a concrete error in the finalizer's own work or in the combined answer.

D.2 ABILITY PROMPTS

Before execution, the entry LLM infers the required ability pair from the complete dataset ability vocabulary, brief descriptions, and examples mapping all evaluated task types to ability pairs. The controller stores this pair in the task state. It then compares the inferred pair with the agent's declared abilities and inserts one of three directives into the execution prompt. The evaluation selects the strong or weak model from the ability match and the agent's injected condition; the directive communicates the ability match to the LLM.

Complete competence match

```
[COMPETENCE VERDICT - DECIDED BY SYSTEM, BINDING]
This task requires: {req}. You ARE competent at all of them. Solve it correctly and
    carefully.
On the FIRST line output exactly 'COMPETENCE_MATCH: yes', then give your answer.
```

Partial competence match

```
[COMPETENCE NOTE - PARTIAL FIT]
You can reliably handle only the parts of this task that fall WITHIN YOUR OWN
    abilities...
- If you select Finalize: you MUST commit a single concrete final answer,
  combining the part you can do with whatever the progress already provides for the
    rest.
- If you are only contributing an INTERMEDIATE part (a Contribute draft): do ONLY the
    part your
  abilities cover, state that partial result, and request Handoff for the remaining
    work.
On the FIRST line output exactly 'COMPETENCE_MATCH: partial', then give your answer/
    contribution.
```

No competence match

```
[COMPETENCE NOTE]
This task requires: {req}, none of which are your strengths. Answer as best you can,
but treat your own answer here as unreliable.
On the FIRST line output exactly 'COMPETENCE_MATCH: no', then give your answer.
```

The no-match directive applies to generation assigned outside an agent's declared abilities. In MESHHEAL, an agent with none of the remaining required abilities instead performs controller-only Handoff. App. A reports the limited increase in the accuracy gap between assigned and unassigned abilities under the tested prompts.

E CASE STUDIES

We present four representative execution traces showing collaboration across complementary abilities, multi-hop routing around an **isolated** agent, committee review and Takeover after an incorrect output, and correction of a localized reasoning error. For presentation, we standardize ability terminology and action labels to match Sections 4 and B. Recorded answers, numerical values, and executable field names are retained; logging metadata not needed to follow the execution is omitted.

E.1 COLLABORATION AND ROUTING

Case 1: Complementary abilities across two agents

Task: `object_counting`.
Required abilities: mathematical + spatial.
Route: A4 → A1.
Outcome: success; gold answer 15.
Question: *"I have a drum, four pianos, two accordions, a carrot, a stalk of celery, three flutes, a cauliflower, a violin, a clarinet, a trombone, and two trumpets. How many musical instruments do I have?"*

A4 is assigned the spatial ability but not the mathematical ability. It contributes the part corresponding to its assigned ability and leaves the remaining work to another agent.

A4 Contribute: Filter and enumerate

```
RESULT: Based on the provided list, the musical instruments and their quantities are:
- Drum: 1
- Piano: 4
- Accordion: 2
- Flute: 3
- Violin: 1
- Clarinet: 1
- Trombone: 1
- Trumpet: 2

Non-instrument items (carrot, celery, cauliflower) are excluded.
The arithmetic summation can be completed by a teammate.
```

The spatial component is now covered, while the mathematical component remains. The controller therefore routes the updated task state to A1, which holds the missing ability and is in the normal routing state.

Controller routing: A4 to A1

```
Covered ability: spatial
Remaining ability: mathematical
NEXT_AGENT_ID: 1
```

A1 receives the accepted output and completes the remaining operation.

A1 Finalize: Complete the mathematical component

```
DESCRIPTION: Using the counts from Agent 4:
1 + 4 + 2 + 3 + 1 + 1 + 1 + 2 = 15.

RESULT: 15
```

What this case shows

The task is solved through complementary outputs from two agents. A4 performs the spatial filtering and enumeration, routing carries that result, and A1 uses the remaining mathematical

What this case shows (continued)

ability in the Finalize step. The trace illustrates how declared abilities determine both the division of work and the next routing decision.

Case 2: Multi-hop routing around an isolated agent

Task: `number_theory`.
Required abilities: symbolic manipulation + enumeration.
Network: 288 agents, 705 bidirectional links, average degree 4.9, with 1.7% as many edges as the corresponding complete graph.
Outcome: success; gold answer 1575.
Question excerpt: *"A school has between 150 and 200 students... What is the sum of all possible numbers of students enrolled?"*

Agent 61 first derives the candidate values needed for the remainder of the solution.

Agent 61 Contribute: Symbolic manipulation

```
Assigned ability: symbolic manipulation

RESULT: 151, 157, 163, 169, 175, 181, 187, 193, 199
```

The symbolic-manipulation component is now covered, while enumeration remains. Agent 12, the nearest holder of enumeration, is isolated, so its ability set is empty for routing. The updated Bellman–Ford distances lead to Agent 82, the nearest **normal** holder of enumeration, three hops away.

Controller-only relays in a sparse network

```
Remaining ability: enumeration
Agent 61 -> Agent 49    (routing after Contribute)
Agent 49 -> Agent 151   (Handoff)
Agent 151 -> Agent 82   (Handoff)
```

The intermediate agents relay the task state without asking an LLM to generate task content.

Agent 82 Finalize: Complete the enumeration component

```
Assigned ability: enumeration

DESCRIPTION: Add the nine values provided by Agent 61:
151 + 157 + 163 + 169 + 175 + 181 + 187 + 193 + 199 = 1575.

RESULT: 1575
```

What this case shows

Agent 61 contributes the symbolic-manipulation component. Local routing-state information then prevents the task from being assigned to an **isolated** agent. Bellman–Ford distance updates guide two controller-only relays to Agent 82, which holds the remaining enumeration ability and uses Finalize to complete the task. The route uses only local eligibility and distance information.

E.2 Review and Takeover

Case 3: Fluent but incorrect output, committee reassessment, and Takeover

Task: `geometric_shapes`.
Required abilities: mathematical + spatial.
Degraded agent: A5.

Case 3: Fluent but incorrect output, committee reassessment, and Takeover (continued)

Outcome: success; gold answer (J), triangle.

Question: SVG path `'M 30.17,45.97 L 58.79,40.36 L 18.10,15.70 M 18.10,15.70 L 30.17,45.97'`.

A5 remains responsive after degradation and produces a well-formed output. The failure is therefore not visible from availability or surface form alone.

Step 1: A5 produces a fluent but incorrect contribution

```
RESULT: 6 vertices. The segments connect in the order A-B-C-D-E-F-A,
forming a closed contour. This describes a hexagon.
```

The path contains only three distinct vertices, so the output contains a concrete correctness error. The output is escalated to committee review. Committee members first assess the same output independently.

Step 2: Independent committee judgments

```
Reviewer A: score 0.0

Reason: "The path contains only three distinct points and three line segments,
not six vertices forming a hexagon."

Reviewer B: score 0.5

Reason: "The shape classification is incorrect, but the vertex interpretation
requires reassessment."
```

The first-round judgments disagree. Each reviewer then receives the other reviewer's assessment and explanation before independently reassessing the original output.

Step 3: Committee reassessment

```
First-round scores:  [0.0, 0.5]
Second-round scores: [0.0, 0.0]
Final review score:  0.0
Decision: Takeover
```

After reassessment, both reviewers identify the output as incorrect. The final low score remains attributed to A5 as evidence for routing-state updates, while Takeover repairs the current task immediately.

Step 4: Takeover correction

```
The path contains three distinct vertices:
(30.17, 45.97), (58.79, 40.36), and (18.10, 15.70).

They are connected by three straight-line segments forming a closed
three-sided polygon: a triangle. No additional vertices or segments are present.
```

The corrected output is placed in the task state, and the downstream Finalize step returns (J), matching the gold answer.

What this case shows

Degradation need not produce malformed or obviously unusable text: A5 returns a fluent and confident output containing a substantive error. Independent reviewers initially disagree, but committee reassessment resolves the disagreement. Takeover then replaces the incorrect output before it propagates to the final answer, while the review score remains available as evidence for subsequent routing-state updates.

Case 4: A localized error inside an otherwise coherent solution

Task: `tracking_shuffled_objects_five_objects`.
Required abilities: reasoning + sequence.
Degraded agent: A5.
Outcome: success; gold answer (C), brown.

In this example, the overall response is coherent, but one state transition is incorrect. A5 state is watched. Its output therefore receives mandatory committee review.

A5 Contribute: Reason through the state updates

```
1. Claire and Alice swap:
   Claire (red), Alice (brown), Bob (green), Dave (black), Eve (orange)

2. Dave and Eve swap:
   Claire (red), Alice (brown), Bob (green), Dave (orange), Eve (black)

3. Claire and Eve swap:
   Claire (black), Alice (brown), Bob (green), Dave (orange), Eve (red)

4. Bob and Claire swap:
   Claire (green), Alice (brown), Bob (black), Dave (orange), Eve (red)

5. Alice and Claire swap:
   Claire (green), Alice (black), Bob (brown), Dave (orange), Eve (red)

Therefore, Claire has the green ball.
```

The first four state transitions are correct. The fifth is not: immediately before the final swap, Claire holds green and Alice holds brown, so exchanging their balls should leave Claire with brown and Alice with green.

Both committee reviewers then identify the incorrect transition.

Committee review identifies the failing step

```
Reviewer A: The final swap states that Claire remains green, but replaying the
exchange gives Claire brown.

Reviewer B: Step 5 is the first incorrect state update; the preceding four
transitions are consistent.
```

The committee's final score is at most τ_{to}, so Takeover replaces the output.

Takeover correction

```
After the final Alice--Claire swap:
Claire holds brown and Alice holds green.
```

The corrected output is placed in the task state, and the downstream Finalize step by an agent holding the sequence ability returns (C), matching the gold answer.

What this case shows

The failure is subtle: four consecutive state updates are correct and only the final transition is wrong. The response otherwise has the structure and fluency of a valid solution. The review identifies the specific correctness error, and Takeover repairs the current task without restarting the full task trajectory. This complements Case 3, where committee reassessment resolves disagreement over a more visible semantic error.