DEALS: DECENTRALIZED EXPERTISE-AWARE LOAD SERVING FOR MULTI-AGENT LLM SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) have recently emerged as an effective approach for coordinating large language model (LLM)-based agents to solve complex tasks through structured interactions. In practice, MASs often handle a stream of heterogeneous and complex tasks, requiring agents to decompose each task and then self-organize and self-evolve to adapt to incoming tasks while sharing execution resources. However, most early approaches to MASs rely on centralized controllers or fixed coordination patterns, which can limit scalability or adaptability. In contrast, existing decentralized and dynamic MASs often require training dedicated routers or invoking LLMs for agent selection, resulting in substantial computational costs and coordination overhead. To address these challenges and enable efficient task-level self-organization and self-evolution for task- and workload-level collaboration, we propose Decentralized Expertise-Aware Load Serving (DEALS), a decentralized and low-complexity framework that enables agents to self-organize and dynamically route concurrent tasks for processing. Specifically, each agent maintains local queues of incoming tasks, and its router decides whether to process a task locally or forward it to a neighbor based on differences in backlog and success rate. Meanwhile, executors process independent tasks concurrently within and across agents, and partially solved tasks can be resumed by other agents. Experiments show that DEALS not only improves performance along multiple dimensions (e.g., answer accuracy and task throughput) in both homogeneous and heterogeneous agent pools, but also balances agent expertise and workload in a self-organized manner, enabling effective decentralized coordination.

1 INTRODUCTION

In recent years, large language model (LLM)-based agents have demonstrated astonishing capabilities in reasoning and tool use (Brown et al., 2020; Yao et al., 2022). However, as agentic tasks grow increasingly complex, a single LLM-based agent may struggle to reliably solve complex problems and sustain long-horizon executions (Liu et al., 2024a;b). To address this challenge, multi-agent systems (MAS) have recently emerged as an effective approach for coordinating multiple LLM-based agents to solve complex agentic tasks through structured interactions, such as decomposing a complex tasks, assign subtasks to agents with specialized roles, and collaboratively producing solutions (Li et al., 2023; Wu et al., 2023; Hong et al., 2024).

In the literature, early approaches to MASs relied on centralized controllers or hand-designed static role specializations with fixed interaction patterns (Li et al., 2023; Hong et al., 2024). Later methods focused on optimizing collaboration in terms of graph connectivity (Zhuge et al., 2024; Zhang et al., 2025b), workflow structures (Zhang et al., 2025c), and team composition (Liu et al., 2023). However, these approaches optimize collaboration statically and therefore still lack the capacity for self-organization and self-evolution. More recently, researchers have started to consider self-organization and self-evolution for MASs through decentralized routing decisions (Yang et al., 2026) and private memories (Hao et al., 2026). Across these approaches, the primary focus is on how agents collaborate to solve an individual task. Moreover, emergent self-organization and self-evolution behaviors are typically induced through reinforcement learning techniques (Zhang et al., 2025a), which can incur substantial training costs and suffer from stability issues.

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.

In practice, however, MASs often need to serve a continuous stream of heterogeneous tasks arriving concurrently and/or sequentially (Lin et al., 2024; Luo et al., 2025). Supporting multiple heterogeneous tasks simultaneously not only introduces more complex challenges in self-organization and self-evolution, but can also lead to workload imbalance across agents if not carefully managed, thereby degrading system throughput and increasing latency. To date, most existing work on multi-task routing in MASs relies on centralized coordination (Ni et al., 2026; Da & Kalyvianaki, 2026). However, centralized coordination can become a scalability bottleneck and a single point of failure as the system grows. Moreover, some methods require separately trained routers (Yue et al., 2025), while others rely on LLM inference for agent selection (Wu et al., 2023). These approaches can incur substantial training costs, consume significant inference resources, and introduce additional coordination latency. Motivated by these limitations, we ask the following research question:

(Q): Can we design a self-organizing and self-evolving MAS with low-complexity decentralized routing to collaboratively solve multiple concurrent tasks with high accuracy and low latency?

In this paper, we answer this question affirmatively by proposing the Decentralized Expertise-Aware Load Serving method (DEALS), a low-complexity decentralized framework that facilitates self-organization and self-evolution in MASs for concurrent task processing. In DEALS, each agent maintains local queues for unresolved tasks that arrive randomly. Based on its queue backlog and the differential in task completion success rates relative to its neighbors, each agent computes a score to determine whether to forward a task or process it locally. A key feature of DEALS is that task decomposition, allocation, and routing jointly account for both agent expertise and workload, without requiring additional LLM calls or a separately trained routing model. Moreover, each agent is equipped with multiple execution slots, enabling it to process independent tasks concurrently. Collectively, these decentralized interactions organize collaboration within individual tasks and coordinate execution across the workload without a central controller, while balancing answer quality and workload. Our contributions are summarized as follows:

- **Self-Organizing Decentralized MAS Framework.** We develop a decentralized MAS framework that enables self-organized collaboration and role specialization across concurrent tasks to achieve improvements in multiple dimensions, including accuracy, throughput, and latency. Specifically, each agent maintains a local queue for each task type and makes local routing decisions based on the queue state to direct task flows. Meanwhile, independent tasks can be executed concurrently both across agents and within individual agents. By jointly coordinating local routing, task decomposition, and concurrent execution, DEALS enables self-organized and self-evolving collaboration within tasks while efficiently allocating resources across the workload.
- **Lightweight Expertise-Aware Load-Balancing Routing.** We design a decentralized expertise-aware backpressure-type routeing mechanism, which coordinates concurrent task execution while balancing answer quality and execution time. Specifically, each router computes a routing score from backlog and task-specific success-rate differentials between the agent and its neighbors. Each decision uses only local state and direct-neighbor information and requires neither additional LLM calls nor a separately trained routing model.
- **Empirical Validation.** We evaluate DEALS across multiple benchmarks with homogeneous and heterogeneous agent pools. The results show that the complete system achieves the highest average accuracy among the compared methods and $4.7\times$ to $22.7\times$ of the throughput of the baseline methods. Further analysis shows that incorporating expertise information improves accuracy over backlog-only allocation and facilitates self-organization and self-evolution in heterogeneous MASs.

2 Related Work

1) LLM-based Multi-Agent Systems. MAS orchestrate LLM-based agents into structured workflows to solve tasks collaboratively (Guo et al., 2024; Qi et al., 2026). Early systems structure collaboration through role-based conversations and predefined workflows (Li et al., 2023; Wu et al., 2023; Hong et al., 2024). To improve answer quality, some works focus on debate or majority voting (Du et al., 2023; Li et al., 2024), which raise cost for each task. Another line optimizes the collaboration structure itself. For example, GPTSwarm optimizes agent computation graphs (Zhuge et al., 2024), while AFlow searches for effective agentic workflows (Zhang et al., 2025c). Furthermore, AgentPrune removes redundant communication to reduce token cost (Zhang et al., 2025b). Beyond communication structure, DyLAN selects task-oriented teams and AgentVerse adjusts team composition during

collaboration (Liu et al., 2023; Chen et al., 2024). These works considered solving one task at a time, whereas our method processes many independent tasks concurrently across agents.

2) Centralized Routing for Task Allocation. Routing was first studied in the centralized setting, where a trained router assigns each query to one LLM from a candidate pool by balancing predicted answer quality against cost (Ding et al., 2024; Ong et al., 2025; Feng et al., 2025). MasRouter extends routing to the multi-agent setting for agent collaboration and role assignment (Yue et al., 2025). When independent tasks run concurrently, task allocation must consider both model ability and load balancing. Toward this end, Chimera and RouteBalance combine model-quality estimates with runtime load (Ni et al., 2026; Da & Kalyvianaki, 2026). However, both systems rely on separately trained predictors, which introduce additional training overhead. All of these methods allocate work from a single decision point that needs global information of all agents, which limits scalability and requires each agent to expose its internal state. Instead, our method enables agent to route tasks to neighbors by using *local state information and direct-neighbor communication*.

3) Decentralized Coordination and Workload Allocation. In decentralized MAS, agents interact directly without a single orchestrator (Chen et al., 2025; Lu et al., 2024). Recent frameworks decide which agent solves a task, either from memory of trajectories (Yang et al., 2026; Hao et al., 2026) or from a price that agents bid for the work (Bhatt et al., 2025; Liu et al., 2026). Recent work draws on queueing and backpressure principles to schedule LLM inference and agent workloads (Dai et al., 2025; Zhang et al., 2026). The resulting guarantees hold under specific assumptions that simplify real-world system dynamics. Some methods select agents by capability or cost, and some consider queue backlogs and memory reuse when scheduling work. Instead, our method combines success-rate differences with backlog differences for local routing. Unlike AgentNet and AgentLance (Yang et al., 2026; Liu et al., 2026), our method requires no additional LLM calls for task allocation.

3 The Proposed DEALS Framework

3.1 Problem Statement and System Model

We consider a multi-agent system (MAS) whose topology is defined by a fixed directed graph $\mathcal{G} = (\mathcal{A}, \mathcal{E})$, where $\mathcal{A} = \{a_1, \ldots, a_n\}$ denotes the set of LLM agents, and $\mathcal{E} \subseteq \mathcal{A} \times \mathcal{A}$ denotes the connections between agents. Tasks randomly arrive at one of the agents in the MAS system over time, and each task has a type $d \in \mathcal{D}$, where $\mathcal{D}$ denotes a finite task type set (e.g., coding, math, etc.). Each agent $a_i \in \mathcal{A}$ contains three components: (i) a set of first-in, first-out (FIFO) queueing buffers $\{Q_i^d\}_{d\in\mathcal{D}}$, which store tasks entering the system at agent a_i and tasks forwarded by neighboring agents; (ii) a router, which decides whether a task is executed locally or transferred to a neighbor; and (iii) an executor, which processes at most C_i tasks concurrently and returns an answer. The goal of the MAS is that each agent executes or decompose and routes tasks in a decentralized fashion, so that the system is self-organized, self-improve, and self-evolve to optimize performance in multiple dimensions (e.g., answer accuracy, throughput, and latency).

Figure 1 illustrates the task processing in DEALS, which is organized in three stages. In Stage (i), a task of type-d randomly arrives at an ingress agent a in $\mathcal{A}$ following some stochastic arrival process and joins the type-d queue of that agent. Next, in Stage (ii), agent a_i dequeues a task when service capacity is available and marks the task as "in flight." Then, the router either forwards the task to the queue of a neighbor or assigns the task to local execution, and a forwarded task repeats this stage at the receiving agent. Lastly, in Stage (iii), a locally assigned task occupies one of the C_i execution slots of agent a_i. Each execution slot retrieves similar successful trajectories from the private memory of agent a_i as demonstrations and then uses an execution policy (e.g., the ReAct execution policy (Wei et al., 2022; Yao et al., 2022)). Local execution subsequently returns a final answer or produces a continuation that re-enters the corresponding type-specific queue. Once all queues are empty and no tasks remain in flight, the processing terminates.

3.2 Decentralized Routing Design in DEALS

We design a decentralized router that i) enables self-organization and self-evolution and ii) balances execution efficiency and answer quality. Inspired by classical backpressure routing, our router uses backlog differentials to move tasks from more congested agents toward less congested neighbors (Tassiulas & Ephremides, 1990). However, backlog-only routing may favor an idle agent with low

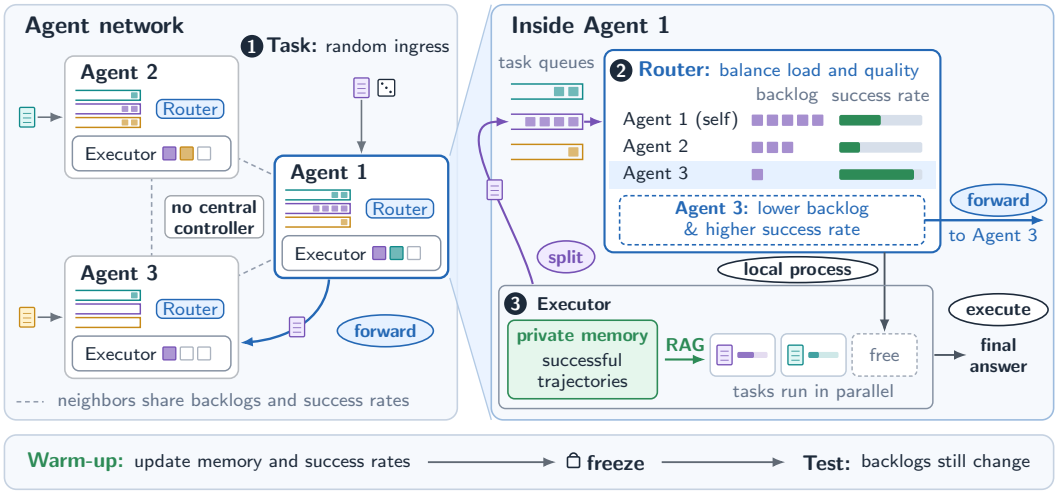


Figure 1: Overview of the DEALS framework. **Left** shows collaboration among three agents without a central controller. **Right** shows Agent 1's internal structure and explains its decision to forward the task to Agent 3. **Bottom** shows how memories and success-rate estimates are updated during warm-up and frozen during testing, while backlogs continue to update.

task-specific capability (e.g., in heterogeneous MAS). Therefore, our router combines the backlog differential with the success-rate differential for each task type. Different from existing works that process tasks sequentially based on expensive LLM nominations for routing (e.g., (Yang et al., 2026)), our method supports concurrent task processing both across agents and within each agent. Each decision uses only the current state of the agent and the state reported by its direct neighbors. In what follows, we present the key components of in the decentralized routing design in DEALS.

1) Backlog. With a slight abuse of notation, we also use $Q_i^d(t)$ to denote the number of type-d tasks stored in the queue Q_i^d at time t. Let $I_i^d(t)$ denote the number of dequeued type-d tasks that are still being routed or executed by agent a_i. The backlog $B_i^d(t)$ captures both waiting and active tasks and is defined as follows:

$$B_i^d(t) = Q_i^d(t) + I_i^d(t). \tag{1}$$

The backlog $B_i^d(t)$ remains unchanged when a task moves from the queue to local execution, and decreases only when the task is forwarded or completed. Note that the classical backpressure formulation for routing in traditional data networks models does not include task execution at nodes. However, local task execution in MAS requires non-negligible time. Therefore, $B_i^d(t)$ augments the classical queueing backlog by explicitly considering tasks under execution at agent a_i.

2) Task-Specific Quality. Backlog reflects workload but ignores each agent's problem-solving capability across different task types. Therefore, we let each agent maintain a Beta-smoothed success rate for every task type. Let $N_{i,d}^+$ and $N_{i,d}^-$ denote the numbers of successful and failed type-d tasks executed by agent a_i. The success rate of agent a_i for task type d can be estimated as:

$$q_i^d = \frac{N_{i,d}^+ + \alpha_q}{N_{i,d}^+ + N_{i,d}^- + \alpha_q + \beta_q}. \tag{2}$$

The Beta prior can stabilize the estimate when few outcomes are available. During the warm-up (training) split, every agent that executes or splits a task updates the corresponding success rate based on the verified outcome, while agents that only forward the task receive no update. The resulting success rates are frozen before the test split.

3) Routing Score. To balance execution efficiency and answer quality, we combine the backlog differential $B_i^d - B_j^d$ and the *success-rate differential* $q_j^d - q_i^d$ into a routing score. Given a nonnegative weight V, we define the routing score for forwarding a type-d task from agent a_i to neighbor a_j as:

$$R_{i\to j}(d) = B_i^d - B_j^d + V\left(q_j^d - q_i^d\right). \tag{3}$$

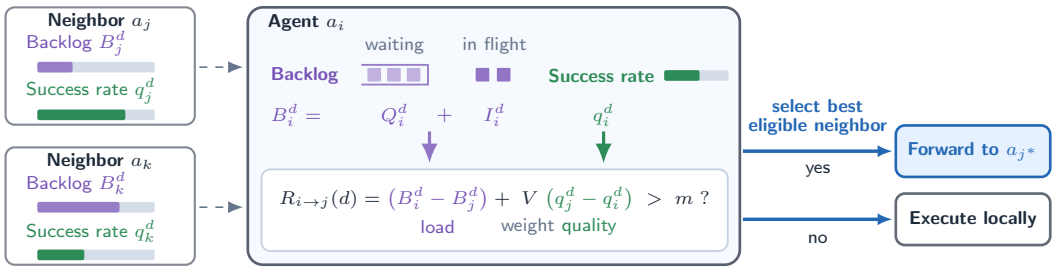


Figure 2: Local routing in DEALS. Agent a_i combines the local state with neighbor advertisements to score eligible neighbors with backlog and success-rate differentials. Agent a_i forwards the task when the best score exceeds the margin m and executes the task locally otherwise.

Intuitively, the routing score favors neighbors with lower backlog or a higher success rate for the type-d task. Specifically, $V = 0$ removes the success-rate differential and considers only the backlog. When $V \neq 0$, a more capable neighbor can receive tasks despite a larger backlog until the backlog differential cancels the quality advantage. Therefore, the router weighs congestion against capability rather than giving every agent the same number of tasks.

Figure 2 shows how agent a_i turns the local state and the neighbor advertisements into one routing decision. For each type-d task, agent a_i evaluates all eligible direct neighbors and selects the neighbor with the highest routing score. Agent a_i forwards the task only when the maximum score exceeds margin m and otherwise executes or splits the task locally. The router excludes the previous agent and enforces hop limit H to prevent immediate returns and unbounded forwarding.

3.3 The Executor Design in DEALS

Although our proposed DEALS is a general MAS framework, for fair performance comparisons, we adopt the ReAct-based retrieval-augmented executor that are widely used in the literature (e.g., (Yao et al., 2022; Yang et al., 2026)), while extending ReAct to support concurrent executions. Specifically, DEALS equips each agent a_i with C_i execution slots for concurrently executing independent tasks both within and across agents. At each step, the executor either completes the task or solves a subproblem and identifies the remaining work. Figure 1 illustrates the executor in an agent, which contains the following two key components:

1) Memory-Guided Reasoning. Each agent a_i maintains a private memory M_i of successful trajectories. These trajectories record the agent's local execution steps on warm-up tasks whose final answers are verified as correct. The memory is frozen before testing and remains available for retrieval. For each local step, the executor retrieves up to k relevant successful trajectories from M_i and uses them as demonstrations to guide its response. The detailed calculation is deferred to Appendix A.

2) Concurrent Execution. The executor supports two actions: `execute` produces a final answer, while `split` solves a subproblem and returns any unfinished work to the local queue. Independent tasks run asynchronously, both in parallel across agents and concurrently within the C_i execution slots of each agent a_i. When a local step releases its slot, another waiting task can proceed without waiting for the other slots.

3.4 Overall Operations of DEALS

Algorithm 1 summarizes DEALS at agent a_i, and every agent runs Dispatch, ProcessTask and Finish concurrently on local state as follows.

1) Initialization. First, agent a_i places each incoming task into the corresponding local task-type queue (Lines 3–8). A type-d task enters the system at agent a_i with probability p_i^d, where $p^d = (p_1^d, \ldots, p_n^d)$ is an arbitrary probability distribution over the agents.

2) Dispatch. When agent a_i has at least one waiting task and an available execution slot, Dispatch admits a task from its longest typed queue (Lines 9–13). Each admitted task is handled by a separate ProcessTask call, allowing up to C_i tasks to progress concurrently within the agent.

Algorithm 1 DEALS: Decentralized Expertise-Aware Load Serving at Agent a_i.

1: **Input:** $phase \in \{\text{warm-up}, \text{test}\}$, tasks $\mathcal{T}_i$ that enter the system at agent a_i, neighbors $\mathcal{N}(i)$
2: **Parameters:** capacity C_i, quality weight V, margin m, hop limit H, split limit K
3: **Initialization:** queue $Q_i^d \leftarrow \varnothing$, in-flight count $I_i^d \leftarrow 0$, $\forall d \in \mathcal{D}$ ▷ Step ❶: random ingress
4: **if** $phase =$ warm-up **then**
5: memory $M_i \leftarrow \varnothing$, success counts $N_{i,d}^+ \leftarrow 0$, $N_{i,d}^- \leftarrow 0$, $\forall d \in \mathcal{D}$
6: **for** each type-d task $x \in \mathcal{T}_i$ **do**
7: ENQUEUE(Q_i^d, x), hop count $h(x) \leftarrow 0$, split count $k(x) \leftarrow 0$
8: previous agent $\text{prev}(x) \leftarrow \varnothing$, executor set $S(x) \leftarrow \varnothing$
9: **procedure** DISPATCH
10: **while** $\sum_d I_i^d < C_i \wedge \exists d\colon Q_i^d \neq \varnothing$ **do**
11: $d \leftarrow \arg\max_{d'} Q_i^{d'}$
12: $x \leftarrow$ DEQUEUE(Q_i^d), $I_i^d \leftarrow I_i^d + 1$
13: **async** PROCESSTASK(x, d)
14: **procedure** PROCESSTASK(x, d)
15: candidate set $\mathcal{J} \leftarrow \{a_j \in \mathcal{N}(i) \mid a_j \neq \text{prev}(x) \wedge h(x) < H\}$ ▷ Step ❷: router
16: routing score $R_{i\to j}(d) \leftarrow B_i^d - B_j^d + V(q_j^d - q_i^d), \forall a_j \in \mathcal{J}$
17: **if** $\mathcal{J} \neq \varnothing \wedge \max_{a_j \in \mathcal{J}} R_{i\to j}(d) > m$ **then**
18: best neighbor $a_{j^*} \leftarrow \arg\max_{a_j \in \mathcal{J}} R_{i\to j}(d)$
19: $\text{prev}(x) \leftarrow a_i$, $h(x) \leftarrow h(x) + 1$, ENQUEUE$(Q_{j^*}^d, x)$ ▷ forward
20: **else** ▷ Step ❸: executor
21: $S(x) \leftarrow S(x) \cup \{a_i\}$
22: allowed actions $\Omega(x) \leftarrow \{\mathcal{O}_{exec}\} \cup \{\mathcal{O}_{split} \mid k(x) < K\}$
23: $(action, output) \leftarrow$ REACT$\big(x, \text{RETRIEVE}(M_i, x), \Omega(x)\big)$
24: **if** $action = \mathcal{O}_{split}$ **then**
25: $x \leftarrow output$, $k(x) \leftarrow k(x) + 1$, ENQUEUE(Q_i^d, x)
26: **else if** $action = \mathcal{O}_{exec}$ **then**
27: FINISH$(x, d, output)$
28: $I_i^d \leftarrow I_i^d - 1$, DISPATCH()
29: **procedure** FINISH$(x, d, answer)$
30: **if** $phase =$ warm-up **then**
31: $correct \leftarrow$ VERIFY$(x, answer) \in \{0, 1\}$
32: **for** each $a_j \in S(x)$ **do**
33: $N_{j,d}^+ \leftarrow N_{j,d}^+ + correct$, $N_{j,d}^- \leftarrow N_{j,d}^- + 1 - correct$
34: **if** $correct = 1$ **then**
35: $M_j \leftarrow M_j \cup \{\text{TRAJECTORY}(a_j, x)\}$
36: **return** $answer$
37: **else if** $phase =$ test **then**
38: **return** $answer$

3) Task Processing. Next, PROCESSTASK scores the eligible neighbors with Eq. (3) and forwards the task only when the best routing score exceeds the margin m (Lines 15–19). If the task is retained locally, the executor uses ReAct with successful trajectories retrieved from M_i to either return a final answer ($\mathcal{O}_{exec}$) or solve a subproblem and return the remaining work to the same typed queue ($\mathcal{O}_{split}$) (Lines 21–28). Different PROCESSTASK calls can run concurrently. The hop limit H and split limit K bound the numbers of forwarding and split steps per task.

4) Warm-up and Test. Finally, FINISH verifies each warm-up answer and updates the success and failure counts of each agent in $S(x)$, the set of agents that executed part of task x (Lines 29–38). These agents add only their own successful trajectories to their private memories. In contrast, the test phase only returns answers, and the memory and the success counts stay as learned in warm-up. Backlogs continue to change during testing as tasks are queued and processed. Thus, the router combines fixed learned quality estimates with changing workload information.

4 EXPERIMENTAL RESULTS

Our experiments examine both the performance of the complete system and the role of quality information in workload allocation. First, we compare final-answer accuracy and report test through-

put relative to baselines. Second, we vary the quality weight to examine changes in accuracy and execution allocation. Finally, we evaluate larger agent pools with different model compositions.

4.1 Experimental setup

1) Tasks. We evaluate on three benchmarks with task-type labels. For MATH and BBH, we reuse the released splits of Yang et al. (2026). We construct the MMLU-Pro split separately.

(i) MATH. The split contains 700 training problems and 140 test problems across seven mathematical types (Hendrycks et al., 2021).
(ii) BBH. The split contains 627 training questions and 100 test questions drawn from 20 subtasks (Suzgun et al., 2023).
(iii) MMLU-Pro. We randomly sample 40 training questions and 20 test questions from each of five subjects: biology, computer science, history, law, and mathematics. This gives 200 training questions and 100 test questions in total (Wang et al., 2024).

For DEALS, we use Dirichlet-distributed ingress probabilities to model uneven task arrivals across agents. Appendix A gives the distribution parameters and sampling procedure.

2) Models. We use Qwen2.5-7B-Instruct, Mistral-7B-Instruct-v0.3, and Llama-3.1-8B-Instruct as agent backbones (Yang et al., 2024; Thakkar & Manimaran, 2023; Grattafiori et al., 2024). For each backbone, we first evaluate a homogeneous group of three agents. We also evaluate a heterogeneous group with one agent from each model family. Next, to examine larger agent pools, we extend both settings to five and seven agents. The homogeneous pools H5 and H7 contain five and seven Qwen agents, respectively. In the heterogeneous setting, M5 combines three Qwen agents with one Mistral agent and one Llama agent. Building on M5, M7 adds two more Llama-based agents.

3) Baselines. We compare DEALS with single-agent and multi-agent frameworks as follows:

(i) Single Agent. A single agent that uses ReAct and reasons step by step before acting (Wei et al., 2022; Yao et al., 2022).
(ii) AgentNet. A decentralized multi-agent framework in which each agent uses the LLM to execute, split or forward a task based on agent abilities and retrieval-based memory (Yang et al., 2026).
(iii) GPTSwarm. A multi-agent framework that models agents as optimizable computational graphs, learns the edges between agents and aggregates the answers by majority voting with benchmark-specific prompts (Zhuge et al., 2024).
(iv) AgentPrune. A multi-agent framework that prunes redundant messages from the communication graph between agents to reduce token cost (Zhang et al., 2025b).

4.2 Accuracy and Throughput Analysis

As shown in Table 1 and Figure 3, our method improves both accuracy and execution efficiency. Since many existing methods lack public implementations or are costly to reproduce, we compare DEALS with the baselines listed in Table 1.

1) Accuracy. Table 1 reports accuracy on MATH, BBH, and MMLU-Pro across four agent configurations. DEALS achieves the highest average accuracy in all four configurations and ranks first in nine of the twelve individual evaluations. It also outperforms AgentNet on all three benchmarks in every configuration. The largest average improvement over the strongest baseline occurs in the heterogeneous setting, where DEALS reaches 66.52% compared with 57.76% for AgentNet.

2) Throughput. Figure 3 reports throughput measured as completed tasks per minute. Across all settings, the throughput of our method is $4.7\times$ to $22.7\times$ that of AgentNet. This gain comes from *concurrent* task execution and decentralized workload allocation. Therefore, multiple agents work simultaneously and each agent can execute up to C_i tasks concurrently. In contrast, AgentNet processes one task at a time which limits throughput.

4.3 Guiding Self-Organization through Local Routing

We vary the quality weight V to examine how local routing decisions shape task allocation across the MAS. This experiment also examines whether changing the routing preference can guide self-organization without prescribing individual task assignments and agent role specialization.

Table 1: Accuracy (%) across backbones and datasets. Bold indicates the best result, and underlining indicates the second-best result. The quality weight is set to $V = 20$ for homogeneous configurations and $V = 80$ for the mixed-family configuration.

Backbone	System	MATH	BBH	MMLU-Pro	Average
Qwen2.5-7B	Single Agent	66.43	62.00	52.00	60.14
	AgentNet	68.57	71.00	50.00	63.19
	GPTSwarm	65.00	**75.00**	21.00	53.67
	AgentPrune	65.00	68.00	34.00	55.67
	DEALS (Ours)	**71.43**	74.00	**53.00**	**66.14**
Mistral-7B	Single Agent	6.43	43.00	34.00	27.81
	AgentNet	7.86	50.00	30.00	29.29
	GPTSwarm	**13.57**	30.00	35.00	26.19
	AgentPrune	6.43	14.00	19.00	13.14
	DEALS (Ours)	12.86	**51.00**	**38.00**	**33.95**
Llama-3.1-8B	Single Agent	52.86	53.00	40.00	48.62
	AgentNet	31.43	53.00	27.00	37.14
	GPTSwarm	**57.14**	50.00	43.00	50.05
	AgentPrune	33.57	49.00	35.00	39.19
	DEALS (Ours)	51.43	**60.00**	**48.00**	**53.14**
Mixed Family	AgentNet	64.29	57.00	52.00	57.76
	GPTSwarm	63.57	61.00	38.00	54.19
	AgentPrune	50.71	65.00	37.00	50.90
	DEALS (Ours)	**73.57**	**73.00**	**53.00**	**66.52**

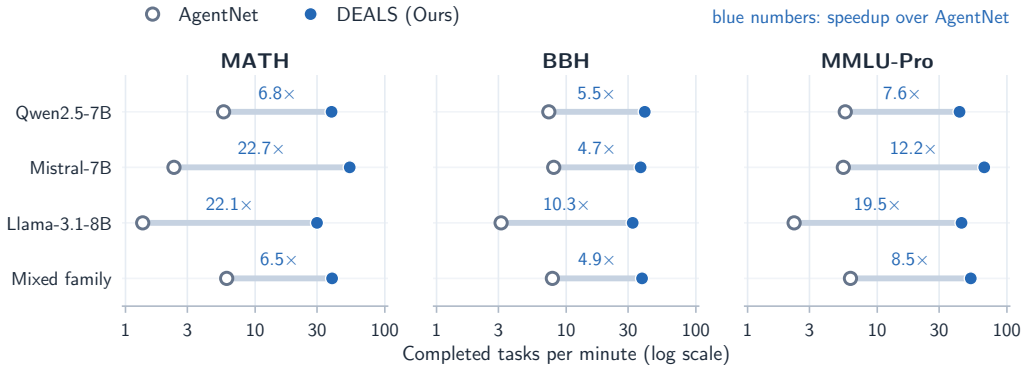
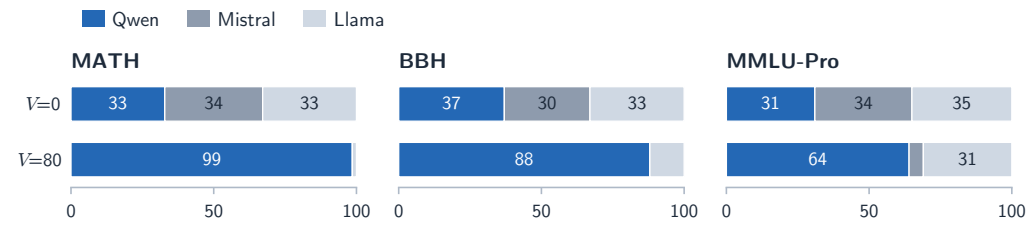


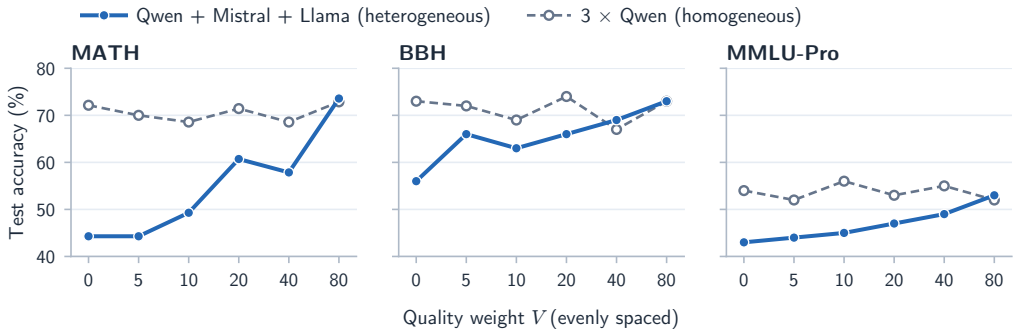
Figure 3: Test throughput of sequential AgentNet and concurrent DEALS across three benchmarks and four backbone configurations. Each row links AgentNet throughput (open circle) to DEALS throughput (filled circle), measured in completed tasks per test wall-clock minute on a log scale. Blue numbers give the ratio of DEALS throughput to AgentNet throughput.

1) Self-Organized Task Allocation. Figure 4(a) shows the percentage of final answers produced by each agent. At $V = 0$, the router uses backlog alone without considering task-solving capability, and final answers are distributed approximately evenly among the three agents. At $V = 80$, the router shifts final task executions toward the stronger Qwen agent. For example, Qwen produces approximately 99% of the final answers on MATH and 64% on MMLU-Pro. These allocation patterns arise from local routing decisions rather than predefined assignment policy. It also illustrate how agents organize task completion through local interactions.

2) Controlling the Expertise-Load Balance. The weight V allows users to adjust the relative importance of agent expertise and workload in local routing. Figure 4(b) examines how this preference affects answer accuracy. In the homogeneous setting, accuracy varies less with V because agents share the same backbone and have more similar task-specific capabilities. However, in the heterogeneous pool, $V = 0$ gives the lowest accuracy on all three datasets. Increasing V from 0 to 80 then improves accuracy on every dataset, for example from 44.29% to 73.57% on MATH. These results show that task allocation benefits from quality information instead of relying on backlog alone.



(a) Final-answer percentage by agent in the heterogeneous pool.



(b) Test accuracy versus the quality weight V.

Figure 4: Effect of the quality weight V in three-agent systems. (a) Final-answer percentage by agent in the heterogeneous pool for $V = 0$ and $V = 80$. (b) Test accuracy of the heterogeneous and the homogeneous pools at different quality weights V.

4.4 Expertise-aware routing at larger pool sizes

Following the experimental setup in Section 4.1, we evaluate DEALS on MATH with five and seven agents, using Qwen-only homogeneous pools (H5 and H7) and heterogeneous pools that combine Qwen with Mistral and Llama (M5 and M7). Table 2 shows that expertise-aware routing improves accuracy in both heterogeneous pools. Increasing V from 0 to 20 and then to 80 yields substantial accuracy gains. In contrast, homogeneous accuracy remains stable across different values of V.

Both M5 and M7 contain Mistral and Llama agents that are weaker than Qwen on MATH. With backlog-only routing, M5 and M7 reach 47.86% and 40.00%, which falls below the 66.43% of the strongest single agent in Table 1. With $V = 80$, the same two pools reach 72.86% and 67.86% and pass that single agent. Accuracy drops from M5 to M7 because M7 adds two weaker Llama agents, yet M7 still passes that single agent. The homogeneous pools change little, because every agent shares the same backbone and leaves little quality differential for the router to exploit.

Table 2: MATH test accuracy (%) with five and seven agents. Bold marks the highest accuracy within each group, including ties.

Setting	Group	$V = 0$	$V = 20$	$V = 80$
Homogeneous	H5	**68.57**	**68.57**	**68.57**
	H7	70.71	70.71	**72.14**
Heterogeneous	M5	47.86	64.29	**72.86**
	M7	40.00	60.71	**67.86**

5 Conclusion

In this paper, we developed DEALS for decentralized coordination for serving concurrent task workloads in multi-agent systems. The key principle is that available execution capacity should be considered together with task-solving capability. DEALS implements this principle through local backlog and success-rate differentials while independent tasks proceed asynchronously. Our experiments showed that incorporating quality information is particularly useful in heterogeneous pools. Furthermore, our framework achieved competitive accuracy and higher workload throughput than state-of-the-art baselines.

AI USE STATEMENT

Generative AI tools were used to polish the writing and to improve the presentation of the figures. They were also used to debug the code and to improve the readability of the code. Generative AI tools were not used to develop the method or to generate the data. The authors independently verified all AI-assisted text and all AI-assisted code. The authors also verified the mathematical claims, the citations and the reported data. The authors take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

The supplementary material contains the complete implementation of DEALS. The package also contains the benchmark splits used in our experiments, the configuration of every reported run and the launch scripts. Furthermore, the README states the requirements, the installation steps and the serving setup. Section 3 defines the router and the executor, Algorithm 1 states the full procedure, and Appendix A lists every parameter setting.

REFERENCES

Manish Bhatt, Ronald F Del Rosario, Vineeth Sai Narajala, and Idan Habler. Coalesce: Economic and security dynamics of skill-based task outsourcing among team of autonomous llm agents. In *2025 Cyber Awareness and Research Symposium (CARS)*, pp. 1–9. IEEE, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024.

Weize Chen, Ziming You, Ran Li, Chen Qian, Chenyang Zhao, Cheng Yang, Ruobing Xie, Zhiyuan Liu, Maosong Sun, et al. Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. In *International Conference on Learning Representations*, volume 2025, pp. 36374–36411, 2025.

Wei Da and Evangelia Kalyvianaki. Routebalance: Fused model routing and load balancing for heterogeneous llm serving. *arXiv preprint arXiv:2606.17949*, 2026.

JG Dai, Tianze Deng, Yueying Li, and Tianyi Peng. Throughput-optimal scheduling algorithms for llm inference and ai agents. *arXiv preprint arXiv:2504.07347*, 2025.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pp. 41348–41366, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *International Conference on Learning Representations*, volume 2025, pp. 26186–26203, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

Guangya Hao, Yunbo Long, and Zhuokai Zhao. Self-evolving multi-agent systems via decentralized memory. *arXiv preprint arXiv:2605.22721*, 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. *arXiv preprint arXiv:2103.03874*, 2021.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *arXiv preprint arXiv:2402.05120*, 2024.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of {LLM-based} applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945, 2024.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024b.

Xiao Liu, Haoyang Li, Songwei Li, Hongbo Fang, Fengli Xu, Feng Shi, and James Evans. Markets, not planners: Decentralized orchestration of llm agents with private information. *arXiv preprint arXiv:2608.23867*, 2026.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023.

Siyuan Lu, Jiaqi Shao, Bing Luo, and Tao Lin. Morphagent: Empowering agents through self-evolving profiles and decentralized collaboration. *arXiv preprint arXiv:2410.15048*, 2024.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E Gonzalez, et al. Autellix: An efficient serving engine for llm agents as general programs. *arXiv preprint arXiv:2502.13965*, 2025.

Kangqi Ni, Wenyue Hua, Xiaoxiang Shi, Jiang Guo, Shiyu Chang, and Tianlong Chen. Chimera: Latency-and performance-aware multi-agent serving for heterogeneous llms. *arXiv preprint arXiv:2603.22206*, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Shihao Qi, Jie Ma, Rui Xing, Wei Guo, Xiao Huang, Zhitao Gao, Jianhao Deng, Jun Liu, Lingling Zhang, Bifan Wei, et al. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in llm-based multi-agent systems. *arXiv preprint arXiv:2605.14892*, 2026.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. In *29th IEEE Conference on Decision and Control*, pp. 2130–2132. IEEE, 1990.

Hiren Thakkar and A Manimaran. Comprehensive examination of instruction-based language models: A comparative analysis of mistral-7b and llama-2-7b. In *2023 International Conference on Emerging Research in Computational Science (ICERCS)*, pp. 1–6. IEEE, 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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pp. 641–649, 2024.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

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. *Advances in Neural Information Processing Systems*, 38:107309–107336, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025a.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 75389–75428, 2025b.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025c.

Zuyuan Zhang, Sizhe Tang, and Tian Lan. Augmented backpressure for decentralized management of agentic networks. *arXiv preprint arXiv:2608.00914*, 2026.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

A IMPLEMENTATION DETAILS

This appendix describes how DEALS runs in our experiments, and Table 3 lists the parameter settings.

Table 3: Implementation parameter settings for all experiments.

Parameter	Setting
Number of agents	3, 5, 7
Graph topology	Complete graph
Dirichlet concentration α	0.7
Execution capacity per agent C_i	9
Hop limit H	3
Split limit K	3
Forwarding margin m	1
Beta prior (α_q, β_q)	$(1, 1)$
Quality weight V	20 (homogeneous), 80 (mixed family)
Temperature	0
Request seed	1
Hardware	8× A100 80GB (five and seven agents), 4× H100 NVL or 4× H200 NVL (three agents)

Serving. Each agent serves the backbone model through a private vLLM replica on a dedicated GPU (Kwon et al., 2023). A separate GPU hosts the BGE-large retrieval encoder (Xiao et al., 2024). All model calls use temperature 0, request seed 1, and at most 2048 generated tokens.

Concurrency. Each agent processes tasks with $C_i = 9$ worker slots, and a task holds one slot while agent a_i routes or executes the task. Whenever a slot becomes free, agent a_i admits the next task from the longest typed queue and breaks ties by the order of task types.

Routing. The router evaluates Eq. (3) for every direct neighbor except the previous agent of the task. The task moves to the best neighbor only when the best score is strictly larger than $m = 1$, and ties go to the neighbor with the smallest index. The hop limit $H = 3$ counts every forward of a task, including forwards after a split. Once a task reaches the hop limit, the task runs at the agent that currently holds the task.

Trajectory retrieval. Let $e(\cdot)$ denote the BGE-large text embedding (Xiao et al., 2024), and let $t(\tau)$ denote the recorded execution time of a successful trajectory τ in minutes. For a query text z, we define the relevance of trajectory τ as

$$s(z, \tau) = \cos\big(e(z), e(\tau)\big) \left(1 + \frac{0.2}{1 + t(\tau)}\right), \tag{4}$$

where $e(\tau)$ embeds the stored task context or thought. The second factor favors trajectories with shorter execution times. The executor retrieves relevant successful trajectories as follows:

$$\textsc{Retrieve}(M_i, z) = \mathrm{top}_k\big\{\tau \in M_i \;\big|\; s(z, \tau) \geq \theta\big\}, \tag{5}$$

where top_k selects up to k trajectories with the highest relevance scores of at least θ.

Equation (4) combines cosine similarity with a time-based factor whose coefficient is fixed at 0.2. Here, z is the task context or a preliminary thought, and $e(\tau)$ embeds the corresponding stored context or thought. The recorded time $t(\tau)$ covers local trajectory execution and excludes action selection, preliminary-thought generation and completion checks. In Eq. (5), M_i is the agent's private memory of successful trajectories. We set $\theta = 0.7$ and $k = 3$, so retrieval returns at most three trajectories that meet the relevance threshold. If fewer than three trajectories qualify, all qualifying trajectories are returned. The retrievable execution pool of each agent holds at most 40 successful trajectories.

Warm-up and test. In our experiments, ingress probabilities follow $p^d \sim \mathrm{Dirichlet}(\alpha \mathbf{1}_n)$ with $\alpha = 0.7$. We draw one vector per task type in each phase and keep it fixed within that phase. Warm-up and testing use different random seeds for ingress. During warm-up, each finished task is verified immediately, so later routing decisions already use the updated success rates. During test, the memories and the success rates stay fixed, and answers are scored only after all test tasks finish.

B LIMITATIONS

Our evaluation covers complete graphs with three, five, and seven agents across three LLM families. Future work can study other communication topologies such as chain, ring, and sparse graphs. Furthermore, future work can evaluate more challenging benchmarks and include a broader range of LLM families.

C EXTENDED RESULTS

C.1 QUALITY-WEIGHT SENSITIVITY AND FINAL-TASK ALLOCATION

Tables 4 and 5 preserve the exact numerical values used in Figure 4.

Table 4: Accuracy (%) sensitivity to the quality weight V. Spread is the difference between the highest and lowest accuracy across the reported values of V. Bold indicates the best result within each row.

Agent pool	Dataset	V=0	V=5	V=10	V=20	V=40	V=80	Spread
Homogeneous Qwen2.5-7B $\times 3$	MATH	72.14	70.00	68.57	71.43	68.57	**72.86**	4.29
	BBH	73.00	72.00	69.00	**74.00**	67.00	73.00	7.00
	MMLU-Pro	54.00	52.00	**56.00**	53.00	55.00	52.00	4.00
Heterogeneous Qwen/Mistral/Llama	MATH	44.29	44.29	49.29	60.71	57.86	**73.57**	29.28
	BBH	56.00	66.00	63.00	66.00	69.00	**73.00**	17.00
	MMLU-Pro	43.00	44.00	45.00	47.00	49.00	**53.00**	10.00

Table 5: Test tasks completed by each model-backed agent in the heterogeneous configuration. The rows correspond to the selected $V = 0$ and $V = 80$ runs in Table 4. Each count identifies the agent responsible for the final task completion.

Dataset	V	Qwen	Mistral	Llama
MATH	0	46	48	46
	80	138	0	2
BBH	0	37	30	33
	80	88	0	12
MMLU-Pro	0	31	34	35
	80	64	5	31

D CASE STUDY

We follow one BBH movie-recommendation task through DEALS with three Qwen2.5-7B-Instruct agents and $V = 20$. The trace shows one split followed by one forwarding transition and execution at a second agent.

1. Task entry. The task initially enters agent a_2 with the following question:

> Find a movie similar to Pulp Fiction, The Shawshank Redemption, Dances with Wolves, Stargate:
> Options: (A) The Fugitive (B) The Boss of It All
> (C) Barb Wire (D) Basic Instinct 2

2. Split and partial execution at a_2. The executor returns `DECISION: split` and first summarizes the genres and themes of the four reference films. For example, its partial output includes:

> "The Shawshank Redemption is a Drama, Crime film about hope, friendship, and redemption, with deep character studies and powerful performances."

This stage generates film descriptions rather than a final choice. The completion check returns `DECISION: Incompleted` because the four candidate movies have not yet been compared with the reference films.

3. Requeue and forwarding. Next, the remaining work returns to the local queue and is forwarded from a_2 to a_3 by the DEALS router. The continuation carries the partial result rather than restarting the task from the original question alone. Specifically, the prompt at a_3 includes the film descriptions under `Completed Subtasks and Results`.

4. Continuation and final answer at a_3. Agent a_3 selects `execute` and compares the candidate movies using the preceding descriptions and its retrieved demonstrations. Its reasoning identifies *The Fugitive* as the closest match through shared crime and drama elements. The recorded final response is `RESULT: (A)`, which matches the reference answer.

Overall, the example illustrates the workflow. The task contains one split and one forward with two distinct executing agents. Agent a_2 characterizes the reference films, while agent a_3 completes the candidate comparison.