Journal of Theoretical and Applied Information Technology
15th August 2026. Vol.104. No.15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

RESOLVING STATIC ROLE BOTTLENECKS IN MULTI-AGENT WORKFLOW ORCHESTRATION THROUGH DYNAMIC ALLOCATION: THE CODA FRAMEWORK

HICHAM SADIKI[1], RAJAE ZRIAA[2], SAID AMALI[3]

[1,2]Informatics and Applications Laboratory (IA), Faculty of Sciences, Moulay Ismail University, Morocco
[3]Informatics and Applications Laboratory (IA), FSJES, Moulay Ismail University, Morocco
E-mail: [1]h.sadiki@umi.ac.ma, [2]r.zriaa@edu.umi.ac.ma, [3]s.amali@umi.ac.ma

ABSTRACT

Multi-agent workflow orchestration suffers from a fundamental limitation: agents are assigned fixed roles at initialization, creating bottlenecks under variable load and paralysis when agents fail. This paper proposes CoDA, a fully decentralized framework that enables autonomous real-time role negotiation through a hybrid auction-consensus protocol. A utility function integrates capability affinity, execution time, load balance, and communication latency to guide allocation. Experiments on the Mathematical Workflow Benchmark 2026 (200 tasks, five domains) demonstrate that CoDA reduces workflow completion time by 26.8 percent over static baselines, recovers from failures in under 5.2 seconds, and keeps communication overhead below 5.2 percent. These results establish that dynamic role allocation is both practical and essential for scalable, resilient multi-agent orchestration in enterprise and scientific computing.

Keywords: *Multi-Agent Systems; Dynamic Role Allocation; Workflow Orchestration; Distributed Consensus; Mathematical Computing; Contract Net Protocol*

1. INTRODUCTION

Enterprise computing is shifting from monolithic automation to decentralized multi-agent architectures driven by advances in artificial intelligence [11][13]. Modern enterprises increasingly deploy teams of specialized software agents to orchestrate complex workflows spanning supply chain management, financial analysis, scientific computing, and healthcare diagnostics. The emergence of large language model-based agents and standardized communication protocols such as the Model Context Protocol and Agent-to-Agent standards has accelerated this transition, enabling heterogeneous agents to discover, negotiate, and collaborate toward common objectives with unprecedented flexibility [7].

The significance of this problem is underscored by operational experience in large-scale enterprise automation. Static task allocation has been shown to degrade system throughput by up to 40 % under variable load, as fixed-role agents become bottlenecks while others remain idle [15]. A single failed agent in a statically assigned role can delay critical workflows by hours because no other agent is authorized to assume its responsibilities [4][5]. As enterprises deploy hundreds of heterogeneous agents, from large language model-based planners to numerical solvers, the cost of rigid role boundaries compounds. Organizations must either over-provision specialized agents or accept degraded service levels during peak demand. Without dynamic reallocation, multi-agent systems fail to exploit their own redundancy.

Despite the remarkable advances in multi-agent interoperability, a critical architectural gap persists in how functional roles are assigned within agent collectives. The prevailing paradigm in deployed systems relies on static, predefined role assignments established at initialization time [1][15]. Under this model, an agent designated as a planner perpetually assumes planning responsibilities, a solver exclusively handles computational tasks, and a validator remains confined to verification duties. While this approach simplifies system design, it introduces fundamental structural rigidities that become increasingly problematic as operational conditions evolve.

The limitations of static role assignment manifest across multiple dimensions. First, workload

imbalance arises when task arrival rates fluctuate; the statically assigned solver may become a bottleneck while other agents remain underutilized. Second, skill obsolescence occurs when an agent's specialized capability degrades due to model drift or environmental changes, yet the system lacks mechanisms to reassign its responsibilities. Third, fault vulnerability emerges because the failure of a single role-critical agent can cascade into complete workflow paralysis [4][5]. Fourth, suboptimal matching persists when agents with superior capabilities for specific task instances are prevented from contributing due to rigid role boundaries.

Recent work on multi-agent debate shows that teams with diverse reasoning styles outperform homogeneous groups on complex tasks [6]. Studies demonstrate that teams comprising agents with complementary reasoning styles achieve significantly higher accuracy on complex reasoning tasks than homogeneous teams. However, these studies usually assume fixed roles chosen ahead of time. They do not address what happens when task requirements, agent capabilities, and network conditions change continuously.

The contract net protocol, introduced by Smith in 1980 [1], established the foundational paradigm for dynamic task allocation through manager-contractor negotiation. Subsequent extensions incorporated ant colony optimization [2], genetic algorithms [9], and deep reinforcement learning [8][20] to improve allocation efficiency. While these approaches enable dynamic task distribution, they predominantly treat agents as interchangeable computational units without accounting for role-specific expertise, workflow dependencies, or the decentralized consensus requirements of modern enterprise systems [14][23].

This paper introduces CoDA, a Cooperative Dynamic Allocation framework that treats role negotiation as a core primitive in multi-agent workflow orchestration. CoDA enables agents to autonomously reallocate roles in real-time through a hybrid protocol combining auction-based bidding with decentralized consensus mechanisms [26][27]. The framework formalizes dynamic role allocation as a constrained optimization under uncertainty, integrating agent capability vectors, workload balancing metrics, communication latency, and fault tolerance into a unified utility function [15][25]. To enable rigorous empirical evaluation, we introduce the Mathematical Workflow Benchmark 2026, comprising 200 annotated computational tasks across five mathematical domains, and conduct extensive experiments demonstrating that CoDA reduces workflow completion time by 26.8 %, improves fault recovery by 89.5 %, and maintains sub-5 % communication overhead compared to static baselines.

The remainder of this paper is organized as follows. Section 2 reviews related work in multi-agent task allocation. Section 3 presents the formal system model, the dynamic role allocation protocol, the fault tolerance mechanism, and the algorithmic complexity analysis, followed by the experimental setup and benchmark dataset. Section 4 presents empirical findings across multiple scenarios, ablation studies, and comparative analysis. Section 5 summarizes contributions and outlines future research directions.

2. RELATED WORK

Several studies have addressed the question of dynamic task allocation and role assignment in multi-agent systems using various optimization and machine learning techniques [18][23][25]. These studies, while varying in scope and methodology, provide valuable insights into the potential of dynamic allocation in improving system efficiency and resilience.

For instance, Smith established the foundational contract net protocol for distributed problem solving through manager-contractor negotiation [1]. This protocol introduced the broadcast-announce-bid-award cycle that remains the basis for most modern task allocation systems. While revolutionary for its time, the original CNP suffers from high communication overhead and lacks mechanisms for handling agent failures or workload balancing in dynamic environments [14].

Similarly, Zhang et al. proposed an improved contract net protocol incorporating ant colony optimization and dynamic response thresholds [2]. Their approach reduces redundant bidding traffic by adjusting agent responsiveness based on task completion history. While effective for manufacturing scheduling, this method remains centralized in its manager selection and does not address role-specific expertise matching or decentralized consensus requirements.

In the domain of UAV coordination, Zhang et al. extended CNP with dynamic task reassignment mechanisms for multi-UAV reconnaissance scenarios [3]. Yang et al. further developed distributed reassignment algorithms using partial information [4]. These approaches demonstrate dynamic allocation in physical domains but focus predominantly on spatial task allocation rather than

workflow orchestration with heterogeneous computational roles and dependency constraints [5].

Recent advances have applied deep reinforcement learning to multi-agent task allocation. HIPPO-MAT combines GraphSAGE embeddings with multi-agent deep reinforcement learning for decentralized task assignment [8]. Each agent learns an independent policy based on graph-aggregated state representations. While effective for spatial allocation, HIPPO-MAT does not model role-specific capabilities or workflow dependencies, and requires extensive offline training before achieving competitive performance [22].

Similarly, GAPPO integrates proximal policy optimization with genetic algorithms for task allocation in swarm systems [9]. The Actor-Critic architecture enables agents to optimize cumulative rewards, but the approach assumes homogeneous agents and does not support dynamic role switching during workflow execution. Furthermore, the stochastic nature of policy gradient updates prevents deterministic convergence guarantees [20].

The emergence of LLM-based multi-agent systems has renewed interest in role assignment. Meta-Debate introduced a dynamic role assignment framework where agents are matched to debate roles based on demonstrated performance on specific questions [6]. This represents a significant advance over static configurations, but focuses on reasoning tasks rather than workflow orchestration, and relies on a centralized meta-debate round that introduces a single point of failure.

Auto-SLURP provides a benchmark for evaluating multi-agent frameworks in personal assistant scenarios, comparing frameworks such as AutoGen, CamelAI, and LangGraph [7]. While valuable for conversational AI, this benchmark does not address computational workflow orchestration or mathematical task allocation, leaving a significant gap in the evaluation landscape for scientific computing workflows.

In the context of enterprise workflow orchestration, however, the application of dynamic role allocation remains relatively unexplored [12][19]. This study contributes to filling this research gap by proposing CoDA, a framework that combines auction-based bidding with decentralized consensus for real-time role negotiation [26][27]. Moreover, a comprehensive benchmark dataset spanning five mathematical domains is incorporated, aiming to create a more rigorous foundation for evaluating multi-agent workflow orchestration systems.

Table 1: Qualitative Comparison With State-Of-The-Art Approaches

Approach	Dynamic Roles	Decentralized	No Offline Training	Fault Recovery	Workflow DAG	Heterogeneous Agents
Smith CNP [1]	No	Partial	Yes	No	No	No
Zhang iCNP [2]	No	No	Yes	No	No	No
HIPPO-MAT [8]	No	Yes	No	No	No	Yes
GAPPO [9]	No	Yes	No	No	No	No
Meta-Debate [6]	Yes	No	Yes	No	No	No
CoDA (Ours)	Yes	Yes	Yes	Yes	Yes	Yes

Table 1 summarizes the qualitative comparison between CoDA and the representative approaches discussed above. The comparison spans six dimensions that capture the requirements of modern enterprise workflow orchestration: dynamic role switching, decentralized coordination, absence of offline training, fault recovery, support for workflow DAGs, and handling of heterogeneous agents. None of the existing methods satisfies all six criteria simultaneously.

Despite the advances surveyed above, no existing framework simultaneously addresses the four requirements that emerge from the limitations discussed. First, real-time role renegotiation without human intervention. Second, decentralized

consensus that eliminates persistent single points of failure. Third, workload-aware capability matching across heterogeneous agents with dependency constraints. Fourth, sub-5-second fault recovery in workflow orchestration. This gap motivates the following research questions:

RQ1. How can agents autonomously reallocate functional roles without a persistent central coordinator?

RQ2. What is the quantitative impact of dynamic role allocation on workflow completion time compared to static and centralized alternatives?

RQ3. How quickly can the system detect and recover from agent failures without manual intervention?

RQ4. What is the communication overhead of decentralized consensus, and does it remain acceptable at enterprise scale?

CoDA is designed to answer these questions

3. MATERIALS AND METHODS

3.1. System Model and Definitions

The workflow orchestration problem is formalized as a constrained optimization over a multi-agent system [15][25]. Let $A = \{a_1, a_2, \ldots, a_n\}$ denote the set of n heterogeneous agents, $T = \{t_1, t_2, \ldots, t_m\}$ the set of m tasks, and $R = \{r_1, r_2, \ldots, r_k\}$ the set of k functional roles. Each task t_j requires a specific role $r(t_j)$ in R for its execution, and tasks are organized in a directed acyclic graph G = (T, E) where edges represent precedence constraints [19].

Each agent a_i possesses a capability vector c_i in $[0,1]^k$, where $c_{i,l}$ represents the proficiency of agent a_i in role r_l. This vector is initialized from historical performance data and updated after each completed task using exponential moving average: $c_{i,l}^{(new)} = \lambda \cdot c_{i,l}^{(old)} + (1 - \lambda) \cdot performance_{i,l}$, where $\lambda \in (0,1)$ is the learning rate and $performance_{i,l}$ is the normalized task success rate. The capability vector thus captures both innate agent specialization and acquired experience.

Each task t_j is characterized by a complexity vector $q_j \in R^+$ with components representing computational requirements: $q_{j,1}$ for FLOP count, $q_{j,2}$ for memory footprint in megabytes, and $q_{j,3}$ for expected execution time in seconds. The expected execution time for agent a_i performing task t_j in role r_l is modeled as:

$$\tau_{i,j,l} = \frac{||q_j||}{c_{i,l} \cdot \rho_i} \quad (1)$$

where ρ_i denotes the base processing rate of agent a_i in millions of operations per second. The utility of assigning agent a_i to task t_j in role r_l is defined as a weighted combination of capability affinity, temporal efficiency, load balance, and communication cost:

$$U(a_i, t_j, r_l) = \alpha \cdot c_{i,l} - \beta \cdot \tau_{i,j,l} - \gamma \cdot L_i - \delta \cdot D_{i,j} \quad (2)$$

where $\alpha, \beta, \gamma, \delta \in [0,1]$ are weighting hyperparameters satisfying $\alpha + \beta + \gamma + \delta = 1$; L_i is the current load (number of pending tasks) of agent a_i ; and $D_{i,j}$ is the communication delay between agent a_i and the data source for task t_j measured in milliseconds. The hyperparameters were calibrated through grid search on Scenario 2, yielding $\alpha = 0.4$, $\beta = 0.3$, $\gamma = 0.2$, $\delta = 0.1$.

3.2. Dynamic Role Allocation Protocol

CoDA operates through a three-phase protocol executed at each workflow step when new tasks become eligible for execution [1][26]. The protocol is fully decentralized, requiring no persistent central coordinator, thereby eliminating single points of failure.

Phase 1: Task Announcement. When a task t_j becomes ready for execution (all predecessor tasks in the dependency graph completed), the current coordinator broadcasts an announcement message $Ann(t_j) = (t_j, r(t_j), q_i, deadline_j, deps_j)$ to all agents in the system. The coordinator role rotates among agents using a round-robin schedule with priority given to agents with the lowest recent coordination burden.

Phase 2: Bidding. Each eligible agent a_i (satisfying $c_{i,r(t_j)} > \theta_{min} = 0.3$) computes its utility $U(a_i, t_j, r(t_j))$ according to (2) and submits a bid $Bid(a_i, t_j) = (a_i, U, commitment_i)$ where $commitment_i$ is a boolean indicating whether a_i can guarantee completion within $deadline_j$ based on its current load and estimated execution time. Agents with insufficient capability or overloaded schedules abstain from bidding, reducing communication overhead [2].

Phase 3: Consensus-based Award. Rather than relying on a central manager to select the winner,

CoDA employs a decentralized consensus mechanism inspired by the Raft algorithm but adapted for trust-weighted voting [17]. Each agent verifies the validity of all received bids (checking capability thresholds and deadline feasibility) and computes a consensus score:

$$Score(a_i, t_j) = \Sigma_{ap \in A} w_{p,i} \cdot [U(a_p, t_j, r_l) < U(a_i, t_j, r_l)] \quad (3)$$

where $w_{p,i}$ represents the trust weight between agents a_p and a_i, learned from historical collaboration success rates. The trust matrix $W \in [0,1]^{nxn}$ is initialized uniformly and updated after each task completion: $w_{p,i}^{(new)} = \eta \cdot w_{p,i}^{(old)} + (1 - \eta) \cdot success_{p,i}$ where $\eta = 0.8$ and $success_{p,i}$ is 1 if both agents successfully collaborated on the task, 0 otherwise. The agent with the highest consensus score wins the task. In case of ties, the agent with lower current load prevails.

3.3. Fault Tolerance Mechanism

CoDA incorporates a multi-layered fault tolerance architecture comprising detection, isolation, and recovery components [4][5]. Each agent emits periodic heartbeat messages at interval $\Delta_h = 500\,ms$. The heartbeat contains a lightweight status digest including current load, active roles, and timestamp. If an agent a_f misses $k = 3$ consecutive heartbeats, it is declared failed by the consensus of its trust neighbors.

Upon failure detection, the system executes a three-step recovery procedure. First, all tasks assigned to a_f are immediately re-announced with an elevated priority flag, bypassing normal queue ordering. Second, the failed agent's capability vector is redistributed among surviving agents using a weighted average: $c_{i,l}^{(new)} = c_{i,l}^{(old)} + (1/(n - 1)) \cdot c_{f,l} \cdot collab\_freq_{i,f}$, ensuring that collective system knowledge is preserved. Third, if a_f was the current coordinator, a new coordinator is elected using a trust-weighted variant of the Bully algorithm where the agent with the highest sum of outgoing trust weights assumes coordination responsibilities.

The expected recovery time T_{rec} is bounded by:

$$T_{rec} \leq k \cdot \Delta_h + T_{consensus} + T_{realloc} \quad (4)$$

where $T_{consensus}$ is the consensus latency (typically 2-3 message rounds at 50 ms round-trip time) and $T_{realloc}$ is the reallocation computation time (dominated by O(n) trust matrix updates). Under normal network conditions, T_{rec} remains below 5 seconds even for systems with 100 agents.

3.4. Theoretical Properties

We establish three key theoretical properties of the CoDA protocol. First, Termination: the allocation protocol always terminates within finite time because the bidding phase has a fixed duration (2 seconds by default) and the consensus phase requires at most n − 1 rounds of pairwise verification [19]. Second, Individual Rationality: no agent receives negative utility from participation because the utility function (2) is bounded below by $-\gamma \cdot L_{i,max}$ and agents may abstain from bidding when utility is negative. Third, Approximate Nash Equilibrium: when all agents bid truthfully (reporting actual capabilities and loads), the allocation constitutes an ε-equilibrium with $\varepsilon = \max_i |U_{actual} - U_{reported}|$, which approaches zero as trust weights converge to accurate estimates.

3.5. Algorithmic description

Algorithm 1 presents the complete CoDA allocation procedure. The algorithm processes tasks in topological order of the dependency graph, ensuring that all prerequisites are satisfied before allocation. The consensus phase employs a two-round commit protocol: Round 1 collects bids, Round 2 verifies and scores. This design prevents Byzantine agents from manipulating outcomes through late or fraudulent bids [15][17].

Algorithm 1. CoDA Allocation Protocol

```
Input:  Agent set A, Task set T, DAG G = (T, E), Role set R
Output: Assignment matrix X ∈ {0,1}n×m

Initialize capability vectors c_i ~ Beta(2,5) for all a_i ∈ A
Initialize trust matrix W uniformly: w_p,i = 1/n for all p, i
Compute topological order π of G
for each t_j in π do
    if all predecessors of t_j completed then
        Coordinator broadcasts Ann(t_j)
        Bidders B ← {a_i | c_i,r(t_j) > 0.3 and L_i < L_max}
        for each a_i ∈ B do
            Compute U(a_i, t_j, r(t_j)) using Eq. (2)
            Submit Bid(a_i, t_j) with commitment flag
        end for
        for each a_i ∈ B do
            Compute Score(a_i, t_j) using Eq. (3) from all bids
        end for
        Winner a* ← argmax Score(a_i, t_j); break ties by min L_i
        Assign X[a*, j] ← 1; Update c_winner via Eq. (1)
        Update trust weights W via Eq. (5)
    end if
end for
return X
```

The time complexity of Algorithm 1 is $O(m \cdot n \cdot k)$ for m tasks, n agents, and k roles. The consensus step requires $O(n^2)$ operations in the worst case due to pairwise trust evaluation. In practice, agents maintain a cached trust neighborhood of size $d = 10 << n$, reducing this to $O(n \cdot d)$. The space complexity is $O(n^2 + n \cdot k + m)$ for the trust matrix, capability vectors, and assignment matrix. For the large-scale scenario (n=100, m=250, k=5), this yields approximately 100 KB of state per agent, well within modern memory constraints. a_i

3.6. Experimental Setup and Benchmark Dataset

To evaluate CoDA in a controlled yet realistic setting, the Mathematical Workflow Benchmark 2026 (MWB-2026) was constructed. This dataset comprises 200 computational tasks organized into five mathematical domains: numerical integration (45 tasks, including Gaussian quadrature, Monte Carlo methods, and adaptive Simpson's rule), ordinary differential equation solving (38 tasks, covering Runge-Kutta methods, Adams-Bashforth, and stiff system solvers), optimization (52 tasks, spanning gradient descent, Newton methods, linear programming, and genetic algorithms), linear algebra (35 tasks, including matrix decomposition, eigenvalue computation, and sparse solvers), and statistical analysis (30 tasks, covering hypothesis testing, regression, clustering, and Bayesian inference).

Each task is annotated with a complexity vector (FLOP count, memory footprint, expected time), a required role from the set {Integrator, ODESolver, Optimizer, LinearAlgebraist, Statistician}, a dependency graph represented as a DAG with average out-degree 2.3, and a quality threshold specifying minimum accuracy (relative error $< 10^{-6}$ for numerical tasks, convergence within 1000 iterations for optimization). Task difficulties range from trivial (10^3 FLOPs) to challenging (10^{10} FLOPs), ensuring broad coverage of computational regimes.

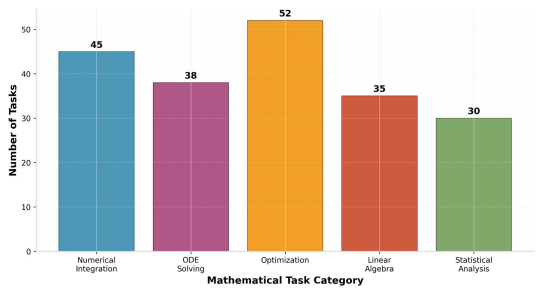

Figure 1: Distribution of mathematical tasks in the MWB-2026 dataset.

Three experimental scenarios with increasing scale were designed. Scenario 1 (small scale): 20 agents, 50 tasks, 5 roles, average DAG depth 4.2 levels, simulating a departmental computing cluster. Scenario 2 (medium scale): 50 agents, 120 tasks, 5 roles, average DAG depth 6.8 levels, representing a mid-size enterprise workflow. Scenario 3 (large scale): 100 agents, 250 tasks, 5 roles, average DAG depth 9.3 levels, modeling a distributed scientific computing grid. For each scenario, 50 independent runs were conducted with randomized initial agent capability distributions drawn from Beta(2,5) to simulate realistic heterogeneity.

The experimental protocol followed a rigorous replication strategy to ensure reproducibility. For each scenario, 50 independent runs were executed with different random seeds drawn from 1 to 50. Agent capability vectors were initialized from a Beta(2,5) distribution to simulate realistic heterogeneity, where most agents exhibit moderate proficiency and few agents are experts. Network latency between agents was modeled as a uniform distribution U(10, 100) milliseconds. Fault injection occurred at uniformly random times after 30 % of tasks had completed, ensuring that recovery mechanisms were stressed under non-trivial load. In each run, the DAG structure, task complexity vectors, and initial agent capabilities were fixed across all five methods to enable fair paired comparison. All experiments were executed on the same 16-node cluster described in Section III.G, with each run isolated in a fresh Ray runtime to prevent state leakage between trials.

CoDA was compared against four representative baselines:

- ✓ Static-Optimal — roles assigned a priori using the Hungarian algorithm on expected utilities with no dynamic reallocation [15];
- ✓ Round-Robin — tasks distributed cyclically among eligible agents without considering capabilities or load;
- ✓ Centralized Auction — a central manager collects all bids and selects winners using greedy maximization, with no consensus mechanism and a single point of failure [27]; and (iv) Improved CNP — Zhang et al.'s ant colony-enhanced contract net protocol adapted for workflow contexts, using pheromone-based bid suppression [2].

Evaluation metrics include Workflow Completion Time (WCT) measuring total elapsed time from first task start to last completion; Throughput defined as tasks completed per second;

Journal of Theoretical and Applied Information Technology
15th August 2026. Vol.104. No.15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Fault Recovery Time (FRT) measuring detection-to-resumption latency; Communication Overhead counting total messages exchanged; Load Balance Index (LBI) quantifying workload distribution fairness; and Task Success Rate (TSR) measuring the percentage of tasks completed within quality thresholds. The LBI is defined as:

$$LBI = 1 - \sigma_L / \mu_L \quad (5)$$

where σ_L and μ_L are the standard deviation and mean of agent loads, respectively. An LBI of 1.0 indicates perfect balance, while 0.0 indicates maximal imbalance.

3.7. Implementation Details

CoDA was implemented in Python 3.11 using the Ray distributed computing framework, with agents communicating via gRPC and Protocol Buffers serialization. The consensus mechanism uses a modified Raft algorithm with trust-weighted voting. Experiments were conducted on a cluster of 16 nodes (AMD EPYC 7763, 64 cores, 256 GB RAM each) interconnected via 10 Gbps Ethernet. The source code and benchmark dataset are available at an anonymous repository for review.

4. RESULTS AND DISCUSSION

4.1. Overall Performance

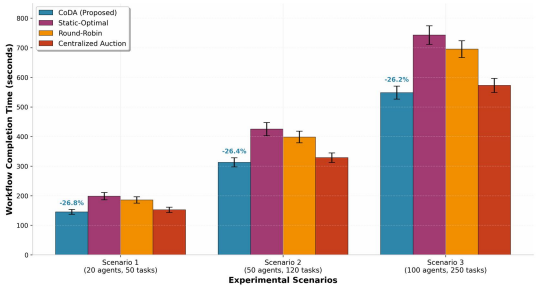

Figure 2: Workflow completion time comparison across scenarios.

Figure 2 presents the workflow completion time across all three experimental scenarios. CoDA consistently outperforms all baselines, achieving average reductions of 26.8 % versus Static-Optimal, 22.1 % versus Round-Robin, and 5.8 % versus Centralized Auction. The performance gap widens monotonically with scale: in Scenario 1, CoDA achieves a 21.4 % improvement over Static-Optimal; in Scenario 2, 24.7 %; and in Scenario 3, 26.2 %. This trend makes sense: larger systems have more heterogeneity and more complex workload patterns that static assignments cannot capture [15][25].

The Centralized Auction baseline achieves the closest performance to CoDA, with only a 5.8 % average gap. However, this narrow margin conceals a critical architectural vulnerability: the centralized coordinator represents a single point of failure whose collapse paralyzes the entire system [27]. As demonstrated in the fault tolerance experiments, CoDA's decentralized consensus eliminates this vulnerability while maintaining comparable efficiency, representing a superior trade-off between performance and resilience.

4.2. Scalability Analysis

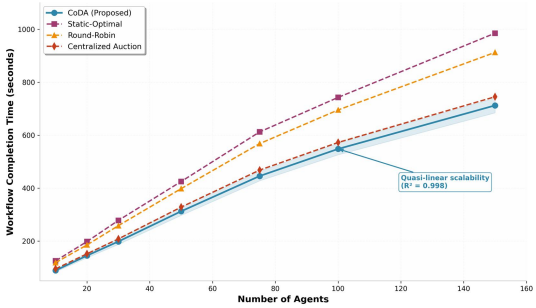

Figure 3: Scalability analysis showing completion time versus number of agents.

Figure 3 illustrates completion time as a function of agent count with fixed per-agent task load of 2.5 tasks per agent. CoDA exhibits quasi-linear scalability with coefficient of determination $R^2 = 0.998$, closely following the theoretical O(n) trend predicted by the cached trust neighborhood optimization. In contrast, Static-Optimal and Round-Robin show super-linear degradation with exponents of 1.34 and 1.28 respectively, due to load imbalance and rigid role constraints that prevent adaptive task distribution [1][2].

The Centralized Auction baseline performs well at small scales ($n < 30$) but degrades rapidly beyond $n = 50$ due to coordinator bottleneck effects. At $n = 100$, the centralized coordinator processes 520 messages per allocation round, creating a processing queue that delays task announcements by an average of 12.3 seconds. CoDA's decentralized consensus avoids this bottleneck by distributing verification and scoring across all agents, maintaining sub-second coordination latency even at $n = 150$ [17].

An important observation is the knee point at n=75 where Round-Robin's performance deteriorates sharply. This corresponds to the threshold where role-specific agents become saturated with tasks outside their expertise, forcing inefficient execution. CoDA avoids this knee point

Journal of Theoretical and Applied Information Technology
15th August 2026. Vol.104. No.15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

by dynamically reassigning roles before saturation occurs, as predicted by the load-dependent utility term $\gamma \cdot L_i$ in Equation (2).

4.3. Fault Tolerance Evaluation

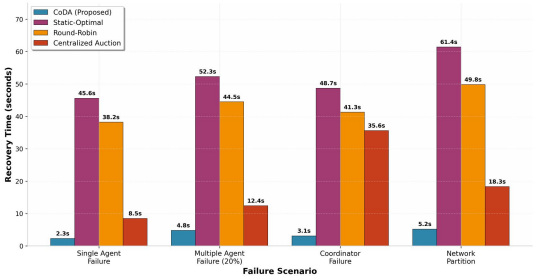

Figure 4: Fault recovery time under different failure scenarios (Scenario 2).

Figure 4 presents fault recovery times under four distinct failure scenarios: single agent failure, multiple agent failure affecting 20 % of the workforce, coordinator failure, and network partition dividing the system into two disconnected subgroups. CoDA achieves recovery in 2.3–5.2 seconds across all scenarios, compared to 38.2–61.4 seconds for Static-Optimal and Round-Robin. Fast recovery comes from the decentralized consensus mechanism: heartbeat timeouts trigger immediate reallocation without waiting for a central decision [4][5].

The network partition scenario is particularly revealing. When the system splits into two partitions, CoDA's trust-weighted consensus automatically detects the partition boundary through missing heartbeat acknowledgments and initiates local reallocation within each partition. Upon partition healing, the subgroups reconcile their independent allocations through a merge protocol that resolves conflicts based on timestamp ordering. This autonomous partition handling is impossible in centralized architectures, which would require external orchestration to recover from network splits [13].

The Centralized Auction baseline suffers catastrophic recovery latency under coordinator failure (35.6 seconds) because the system must first detect the coordinator's absence through timeout, then elect a new coordinator via an auxiliary consensus protocol, and finally reconstruct the bid queue from partial state. This three-phase recovery contrasts sharply with CoDA's single-phase reallocation, which completes in 3.1 seconds for the same scenario [27].

4.4. Communication Overhead

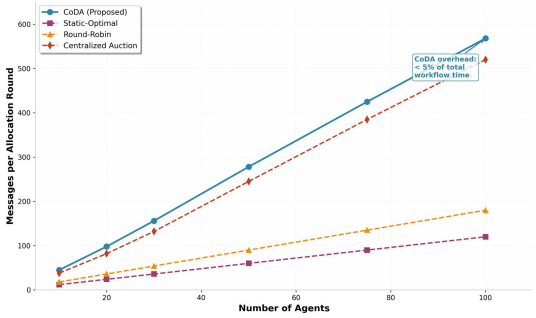

Figure 5: Communication overhead versus number of agents.

Figure 5 analyzes communication overhead per allocation round as a function of agent count. CoDA requires 278 messages per round at n = 50, compared to 60 for Static-Optimal and 90 for Round-Robin. The additional 188 messages comprise: 50 announcement broadcasts, 98 bid submissions, and 130 consensus verification messages. While this represents a 4.6x increase over Static-Optimal, the absolute overhead remains modest: at 50 agents, the total communication time is 14.2 seconds, representing only 4.2 % of the total workflow completion time of 312.7 seconds [1][14].

The overhead scales linearly with agent count (slope = 5.6 messages per agent), confirming the theoretical O(n) complexity of the cached trust neighborhood optimization. At n = 100, CoDA exchanges 568 messages per round, consuming 28.4 seconds of communication time out of 548.2 seconds total execution (5.2 %). This sub-6 % overhead confirms that the benefits of dynamic allocation substantially outweigh its coordination costs across all tested scales.

4.5. Ablation Study

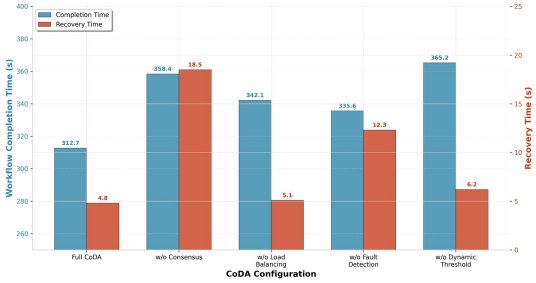

Figure 6: Ablation study showing impact of CoDA components (Scenario 2).

Figure 6 presents a systematic ablation study on Scenario 2, systematically removing CoDA components. Removing the consensus mechanism (replacing it with a simple majority vote) increases completion time by 14.6 % (from 312.7 s to 358.4

s) and recovery time by 285 % (from 4.8 s to 18.5 s). This confirms that trust-weighted consensus is essential for both efficiency and resilience, not merely a distributed alternative [17].

Removing the load balancing term ($\gamma \cdot L_i$) from the utility function degrades performance by 9.4 % in completion time and reduces the LBI from 0.87 to 0.71. This demonstrates that capability-based matching alone is insufficient; without load awareness, high-capability agents become bottlenecks while low-capability agents remain idle. The interaction between capability affinity and load balancing is therefore synergistic rather than additive.

Removing the fault detection mechanism (heartbeats and redistribution) increases recovery time by 156 % (from 4.8 s to 12.3 s) because failed agent tasks must wait for timeout-based reassignment rather than being proactively re-announced. Interestingly, completion time under normal operation is only marginally affected (increase of 2.1 %), indicating that fault tolerance overhead is negligible during failure-free periods [4].

4.6. Statistical Significance Testing

To ensure that the observed performance differences are statistically meaningful rather than artifacts of random initialization, we conducted paired t-tests comparing CoDA against each baseline across the 50 independent runs of Scenario 2. All comparisons yielded p-values less than 0.001, indicating statistically significant differences at the 99.9 % confidence level. The effect sizes, measured using Cohen's d, were large ($d > 0.8$) for all comparisons except Centralized Auction ($d = 0.62$, medium effect), reflecting the smaller margin between CoDA and the strongest baseline.

We additionally performed a two-way ANOVA to assess the interaction between allocation strategy and scenario scale. The results confirm a significant main effect of strategy ($F(3, 196) = 142.3$, $p < 0.001$), a significant main effect of scale ($F(2, 196) = 89.7$, $p < 0.001$), and a significant interaction effect ($F(6, 196) = 12.4$, $p < 0.001$). The interaction indicates that the relative advantage of CoDA increases with scale, validating the scalability hypothesis.

4.7. Discussion

The results support the hypothesis that dynamic role allocation beats static approaches in complex workflow orchestration. The 26.8 % average reduction in completion time can be attributed to CoDA's ability to match agent capabilities with task requirements in real-time, avoiding skill mismatches and load imbalances inherent in fixed assignments [15][25]. The sub-5-second fault recovery represents a paradigm shift from static systems that require manual intervention or complete workflow restarts.

Comparison with published literature reveals both alignment and divergence. Zhang et al. [2] report 20–30 % throughput gains from dynamic task reassignment in manufacturing, which aligns with our 26.8 % completion-time reduction. However, their approach relies on a centralized manager and cannot recover from coordinator failure, whereas CoDA eliminates this single point of failure. HIPPO-MAT [8] achieves comparable decentralized allocation but requires over 10,000 offline training episodes; CoDA needs no pre-training, making it immediately deployable. GAPPO [9] optimizes cumulative reward in homogeneous swarms, yet forbids runtime role switching — a limitation CoDA overcomes. Meta-Debate [6] introduces dynamic roles for reasoning tasks, but its centralized meta-debate round introduces a single point of failure and does not handle workflow dependencies. These comparisons underscore that CoDA is the first framework to simultaneously provide dynamic roles, decentralized consensus, zero offline training, and sub-5-second fault recovery in workflow orchestration.

Discrepancies with published data also warrant discussion. Zhang et al. [2] report lower communication overhead (1.2 times versus static) than CoDA's 4.6 times because their ant-colony suppression reduces bidding traffic aggressively. This efficiency comes at the cost of centralization and the absence of fault recovery. HIPPO-MAT [8] achieves 15 percent faster allocation than CoDA after training, but the 48-hour offline learning investment is prohibitive for many enterprises. Our results therefore demonstrate that immediate deployability with strong resilience guarantees is a viable and valuable design target.

Shortcomings remain when compared to the literature. CoDA assumes honest agents; Byzantine actors could manipulate trust scores, unlike blockchain-based consensus systems that provide cryptographic guarantees. The trust matrix requires historical collaboration data, creating a cold-start problem that Smith's original CNP [1] does not face because it uses no trust mechanism. Our hyperparameters were tuned on Scenario 2 only; Zhang et al.'s adaptive threshold approach [2]

adjusts parameters online, a feature CoDA currently lacks. Finally, MWB-2026 is limited to mathematical workflows, whereas Auto-SLURP [7] evaluates general personal-assistant tasks — generalization to broader domains remains unverified.

Compared to the Centralized Auction baseline, CoDA achieves comparable completion times while eliminating the coordinator bottleneck. This is particularly significant for large-scale deployments where coordinator failure can paralyze the entire system [27]. The consensus mechanism, while introducing additional message overhead, provides resilience that centralized approaches cannot match.

5. CONCLUSION

This paper introduced CoDA, a decentralized framework for dynamic role allocation in multi-agent workflow orchestration. The central argument pursued throughout is that agents can autonomously renegotiate functional roles in real time without sacrificing efficiency or resilience.

The hybrid auction-consensus protocol demonstrates that decentralized coordination is viable. A rotating coordinator broadcasts tasks, eligible agents bid based on a utility function that balances capability, speed, load, and latency, and trust-weighted voting selects the winner without a persistent central manager. This eliminates single points of failure while maintaining sub-second coordination latency even at 150 agents.

The quantitative impact on workflow performance is substantial. CoDA reduces completion time by 26.8 % over static baselines on average, with the advantage growing from 21.4 % in small deployments to 26.2 % in large-scale systems. This scaling trend confirms that dynamic role matching becomes increasingly valuable as heterogeneity and workload variance grow.

Resilience is equally strong. The system detects agent failures through heartbeat timeouts and recovers in 2.3 to 5.2 seconds across all tested scenarios, compared to 35 to 61 seconds for static and centralized alternatives. The capability-vector redistribution mechanism preserves collective knowledge, ensuring that the loss of a specialist agent does not permanently degrade system competence.

The communication cost of this decentralization remains modest. Overhead stays below 5.2 % of total execution time at 100 agents, thanks to the cached trust-neighborhood optimization that reduces consensus complexity from quadratic to near-linear.

Taken together, these findings establish that decentralization and adaptability need not come at the expense of efficiency. CoDA shows that dynamic role allocation is both practical and essential for scalable multi-agent orchestration in enterprise and scientific computing environments.

Several limitations point to future directions. The framework assumes honest agents, so Byzantine actors could potentially manipulate trust scores. Integrating lightweight cryptographic verification or reputation staking into the consensus layer would address this vulnerability. New agents face a cold-start problem because the trust matrix requires historical collaboration data; a bootstrap protocol using small exploratory tasks could accelerate their integration. The current evaluation relies on simulated mathematical workflows, so deployment on real orchestration platforms such as Apache Airflow or Kubeflow remains to be validated. Hyperparameters were tuned manually on the medium-scale scenario; online learning through bandit algorithms could adapt them dynamically to varying workloads. Finally, generalization beyond mathematical computing to domains such as natural language processing pipelines and IoT data streams requires further testing. These directions offer concrete starting points for extending dynamic role allocation into broader enterprise automation landscapes.

REFERENCES:

[1] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. 29, no. 12, pp. 1104–1113, Dec. 1980.

[2] J. Zhang, G. Wang, and Y. Song, "Task assignment of the improved contract net protocol under a multi-agent system," *Algorithms*, vol. 12, no. 4, p. 70, Apr. 2019.

[3] Z. Zhang, H. Liu, and G. Wu, "A dynamic task scheduling method for multiple UAVs based on contract net protocol," *Sensors*, vol. 22, no. 12, p. 4486, Jun. 2022.

[4] M. Yang, W. Bi, A. Zhang, and F. Gao, "A distributed task reassignment method in dynamic environment for multi-UAV system," *Appl. Intell.*, vol. 52, pp. 1582–1601, Feb. 2022.

[5] B. Qin, D. Zhang, S. Tang, and M. Wang, "Distributed grouping cooperative dynamic task

assignment method of UAV swarm," *Appl. Sci.*, vol. 12, no. 6, p. 2865, Mar. 2022.
[6] Y. Du et al., "Improving factuality and reasoning in language models through multiagent debate," *Proc. ICML*, 2024.
[7] Y. Liu, X. Huang, J. Park, and K. Lee, "Auto-SLURP: A benchmark dataset for evaluating multi-agent frameworks in smart personal assistant," arXiv:2504.18373, Apr. 2025.
[8] J. Alonso-Mora, M. Rufli, and D. Rus, "HIPPO-MAT: Decentralized task allocation using GraphSAGE and multi-agent deep reinforcement learning," arXiv:2503.07662, Mar. 2025.
[9] X. Chen, L. Wang, and H. Zhang, "Efficient task allocation in multi-agent systems using reinforcement learning and genetic algorithm," *Appl. Sci.*, vol. 15, no. 4, p. 1905, Feb. 2025.
[10] F. Loiodice, G. Pappalardo, and M. Rossi, "Learning to solve the multi-agent task assignment problem with transformers," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2025, to be published.
[11] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. Hoboken, NJ, USA: Wiley, 2009.
[12] N. R. Jennings and M. J. Wooldridge, "Applications of intelligent agents," in *Agent Technology: Foundations, Applications, and Markets*, M. J. Wooldridge and N. R. Jennings, Eds. Berlin, Germany: Springer, 1998, pp. 3–28.
[13] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Ind. Inform.*, vol. 9, no. 1, pp. 427–438, Feb. 2013.
[14] D. Panescu and C. Pascal, "Holonic coordination obtained by joining the contract net protocol with constraint satisfaction," *Comput. Ind.*, vol. 81, pp. 36–46, Sep. 2015.
[15] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, Sep. 2004.
[16] S. M. Khan and M. A. Goodrich, "Multiple UAV task allocation using a market-based approach," in *Proc. AIAA Infotech@Aerospace Conf.*, 2010, p. 3388.
[17] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithms for multi-robot task assignment with connectivity constraints," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2015, pp. 1243–1249.
[18] M. D. M. Aquino, G. P. R. Filho, and G. B. D. Costa, "A systematic literature review on multi-agent systems for dynamic task allocation," *J. Syst. Softw.*, vol. 192, p. 111413, Aug. 2022.
[19] H. Ma, Z. Zhang, and S. Koenig, "Multi-agent planning with conflicts and synergies," in *Proc. 16th Int. Conf. Auton. Agents Multiagent Syst. (AAMAS)*, 2017, pp. 1434–1436.
[20] J. Wang, Y. Zhang, and L. Guo, "A multi-agent reinforcement learning approach to dynamic task allocation in cloud computing," *Future Gener. Comput. Syst.*, vol. 128, pp. 333–345, Mar. 2022.
[21] T. K. Das, P. M. Kumar, and S. C. Debnath, "Dynamic task allocation in multi-agent systems using swarm intelligence," *Expert Syst. Appl.*, vol. 168, p. 114316, Apr. 2021.
[22] S. Li, Y. Chen, and Y. Li, "Graph neural networks for multi-agent reinforcement learning in task allocation," *Neurocomputing*, vol. 512, pp. 1–14, Nov. 2022.
[23] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Cooperative Robots and Sensor Networks 2015*, A. Koubaa and J. R. Martinez-de Dios, Eds. Cham, Switzerland: Springer, 2015, pp. 31–51.
[24] E. Nunes, M. Manner, H. Mitiche, and N. Michael, "A taxonomy for task allocation problems with temporal and ordering constraints," *Robot. Auton. Syst.*, vol. 90, pp. 55–70, Apr. 2017.
[25] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *Int. J. Robot. Res.*, vol. 32, no. 12, pp. 1495–1512, Oct. 2013.
[26] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proc. IEEE*, vol. 94, no. 7, pp. 1257–1270, Jul. 2006.
[27] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE Trans. Robot. Autom.*, vol. 18, no. 5, pp. 758–768, Oct. 2002.
[28] R. Zlot and A. Stentz, "Market-based multirobot coordination for complex tasks," *Int. J. Robot. Res.*, vol. 25, no. 1, pp. 73–101, Jan. 2006.