Symphony: A Decentralized Multi-Agent Framework for Scalable Collective Intelligence

Ji Wang[1,6], Yemin Wang[2], Zhaoyang Guan[3], Libin Xia[4], Lynn Ai[1], Eric Yang[1], Sicong Jiang[5], Bill Shi[1]*

[1]Gradient
[2]Department of Informatics, Xiamen University
[3]Department of Engineering Sciences and Applied Mathematics, Northwestern University
[4]Peking University
[5]2077AI
[6]Department of Industrial Engineering and Operations Research, Columbia University

Abstract

Centralized multi-agent frameworks like AutoGen and CrewAI suffer from high costs, rigidity, and scalability limitations, often creating single points of failure. Meanwhile, existing decentralized approaches (e.g., Swarm-GPT, FedProx) focus on model synchronization or emergent behavior, lacking effective task orchestration. We introduce **Symphony**, a decentralized multi-agent system that enables lightweight LLMs on edge devices to collaborate through three key mechanisms: (1) a capability-aware distributed ledger, (2) a Beacon-selection protocol for dynamic task allocation, and (3) weighted multi-CoT voting for robust reasoning. Extensive evaluations show Symphony achieves significant accuracy gains over centralized baselines, notably 15-42% on complex reasoning tasks (BBH), alongside consistent improvements on mathematical, coding, and domain-specific benchmarks. Its efficient design reduces communication cost to 2.3MB (vs. 15.7MB in centralized approaches) with under 5% orchestration overhead, while ensuring privacy and fault tolerance across heterogeneous edge environments. Extensive experiments on 7B-class models demonstrate robustness to 20% node failures and graceful degradation under network partitions. The link to the paper's code is as follows:our repository.

Introduction

Large Language Models (LLMs) have revolutionized AI applications across domains including healthcare (Chu et al. 2023), autonomous systems (Yang et al. 2024), and scientific discovery (Wang et al. 2023). However, current multi-agent frameworks like AutoGen (Wu et al. 2023b), MetaGPT (Sirui Hong et al. 2023), and CrewAI (Contributors 2024) rely on centralized orchestration, creating scalability bottlenecks, single points of failure, and requiring expensive server-grade GPUs (Guo et al. 2024). Recent decentralized AI approaches like Swarm-GPT (Anonymous 2024) focus on emergent behavior coordination, while FedProx (Li et al. 2020) emphasizes model synchronization, but neither addresses capability-aware task orchestration for heterogeneous reasoning agents. By 2025, edge AI devices are projected to process 75% of data (IDC 2024), yet existing frameworks cannot efficiently leverage distributed edge resources for collaborative intelligence.

Problem Gap: Current centralized frameworks suffer from fundamental limitations that hinder their practical deployment. High infrastructure costs require dedicated cloud servers, while rigid communication topologies limit system adaptability. Privacy concerns arise as sensitive data must be transmitted to central coordinators, and scalability limitations create single points of failure. While decentralized systems have been studied for robustness (Dimakis et al. 2010; Oliva et al. 2019), their integration with heterogeneous LLM agents remains largely unexplored.

Our Solution: We present **Symphony**, a decentralized multi-agent system that enables lightweight LLMs on edge devices to collaborate through capability-aware task orchestration, ensuring privacy-preserving and fault-tolerant execution with minimal overhead.

Key Contributions: Symphony introduces a decentralized runtime with $\mathcal{O}(\log N)$ gossip overhead and $\mathcal{O}(L)$ beacon complexity, enabling efficient coordination at scale. Our multi-CoT voting scheme improves accuracy by 12-15% over single-CoT baselines through intelligent reasoning aggregation. Comprehensive evaluations on 7B-class models demonstrate robustness to 20% node failures and graceful degradation under network partitions. Real-world deployment validation across heterogeneous edge environments confirms $<$5% orchestration overhead, making Symphony practical for resource-constrained deployments.

Related Work

Centralized Multi-Agent Frameworks

Recent LLM-based multi-agent systems primarily adopt centralized architectures. **AutoGen** (Wu et al. 2023b) uses a central coordinator for conversation management, while **MetaGPT** (Sirui Hong et al. 2023) employs role-based specialization with hierarchical coordination. **CrewAI** (Contributors 2024) and **CAMEL** (Guohao Li et al. 2023) focus on collaborative reasoning but require central orchestration. These frameworks achieve high performance but suffer from

*Corresponding author:tianyu@gradient.network

Table 1: Comprehensive Comparison of Multi-Agent Frameworks

Framework	Architecture	Task Orchestration	Capability Matching	Privacy Protection	Fault Tolerance	Communication Cost	Year
AutoGen	Centralized	Centralized	None	Low	Single Point	High	2023
MetaGPT	Hierarchical	Role-based	Limited	Low	Limited	Medium	2023
CrewAI	Centralized	Centralized	None	Low	Single Point	High	2024
Swarm-GPT	Decentralized	Emergent	None	High	Good	High	2024
FedProx	Decentralized	Model Sync	None	High	Good	Medium	2020
Symphony (Ours)	**Decentralized**	**Capability-aware**	**Advanced**	**High**	**Excellent**	**Low**	**2025**

scalability limitations and single points of failure.

Decentralized AI Systems

Decentralized approaches have gained traction in 2024-2025. **FedProx** (Li et al. 2020) extends federated learning to remove central servers, focusing on model parameter synchronization but lacking task-level orchestration capabilities. **Swarm-GPT** (Anonymous 2024) applies swarm intelligence to LLM coordination, emphasizing emergent behavior but without explicit capability matching mechanisms. **Blockchain-based AI** platforms (Harris, Smith, and Johnson 2021) provide trust mechanisms but incur high latency and energy costs. **Edge AI frameworks** (Zhou et al. 2019) optimize for resource-constrained devices but assume homogeneous models and lack dynamic task allocation. Unlike these approaches, Symphony introduces capability-aware task orchestration enabling heterogeneous agents to collaborate on complex reasoning tasks while maintaining privacy and fault tolerance.

Distributed Systems Foundations

Classical distributed systems research provides the foundation for our approach. **Gossip protocols** (Dimakis et al. 2010; Kempe, Dobra, and Gehrke 2003) enable scalable information dissemination, while **SWIM** (Das, Gupta, and Motivala 2002) provides membership and failure detection. **CRDTs** (Shapiro et al. 2011) offer conflict-free replicated data types for eventual consistency. **Byzantine resilience** in multi-agent systems (Blanchard et al. 2017) ensures fault tolerance against malicious nodes, while **secure aggregation** (Segal et al. 2017) protects privacy during collaborative learning. Recent advances in **Byzantine-robust distributed learning** (Li et al. 2020) provide theoretical foundations for fault-tolerant multi-agent coordination, and **Byzantine fault detection** (Das, Gupta, and Motivala 2002) mechanisms enable real-time identification of malicious agents in distributed systems. Symphony builds on these foundations while addressing LLM-specific challenges and introducing capability-aware task orchestration for heterogeneous reasoning agents.

Symphony's Unique Position: Unlike existing frameworks that either focus on model synchronization (FL) or emergent behavior (Swarm), Symphony introduces *capability-aware task orchestration* enabling heterogeneous agents to collaborate on complex reasoning tasks while maintaining privacy and fault tolerance. This represents a novel paradigm bridging distributed systems research and practical AI deployment.

Methodology

Symphony is a decentralized multi-agent framework designed for scalable, privacy-preserving collaboration across heterogeneous edge devices.

System Architecture

System Architecture Analysis: Figure 1 illustrates Symphony's three-layer architecture and execution flow. The **Agent Layer** consists of heterogeneous LLM agents distributed across edge devices, each maintaining local capability vectors and task execution capabilities. The **Orchestration Layer** implements the distributed ledger for agent discovery and the Beacon protocol for intelligent task allocation. The **Network Layer** provides gateway sharding and fault tolerance mechanisms. The execution flow demonstrates how user queries are decomposed into sub-tasks by planning agents, how Beacon-based selection identifies the most capable agents for each sub-task, and how multi-CoT voting aggregates results across reasoning paths. This architecture enables privacy-preserving collaboration while maintaining fault tolerance and scalability through decentralized coordination.

System Components

Decentralized Ledger The ledger maintains agent capabilities and availability using a replicated key-value store with eventual consistency. Each agent E_j registers a capability vector $\mathbf{c}_j \in \mathbb{R}^d$ and metadata including hardware profile, availability status, and reputation score. The minimal schema includes: (1) agent identifier, (2) hardware profile (GPU/CPU, memory), (3) verified availability, and (4) optional reputation. We implement three constant-time operations: `REGISTER(agentId, caps, profile, availability)`, `UPDATE(agentId, field, value)`, and `QUERY(filter)`. The filter supports selection by capability predicates and resource constraints, enabling fast pre-screening before selection.

Worker Nodes Each worker node E_j operates as an autonomous agent with four core components. The **Local LLM** component runs quantized models (e.g., Mistral-7B (AI 2023)) via vLLM API, providing on-device inference capabilities. The **Capability Vector** $\mathbf{c}_j$ encodes the agent's skills and resources, enabling intelligent task matching. The **Task Executor** handles sub-task execution with full context awareness, while the **Communication Module** ensures secure inter-agent messaging and coordination.

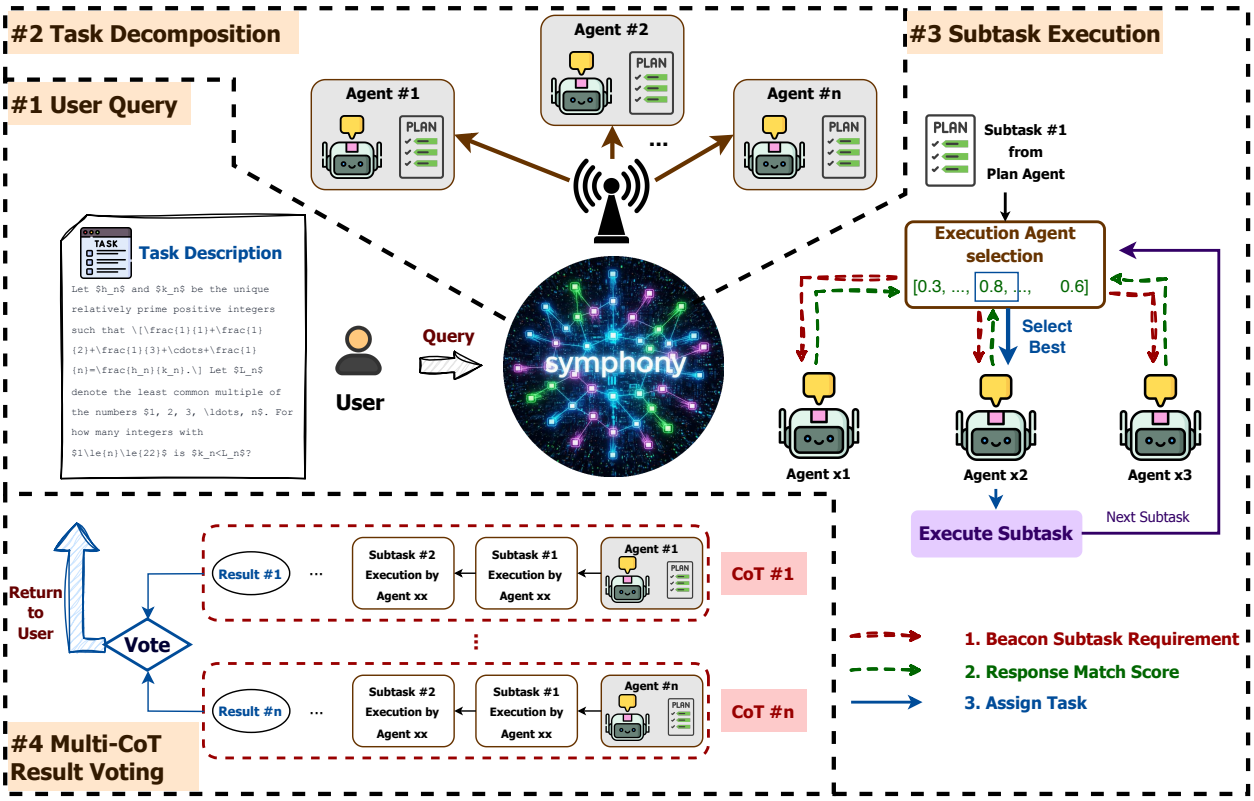

Figure 1: Overview of **Symphony**. 1. A query from user is decomposed into multiple sub-tasks by planning agents. 2. Sub-task execution leverages Beacon-based agent selection to choose appropriate agents. 3. Final response is generated through result voting across multiple reasoning paths.

Gateways Gateways provide REST APIs for agent registration and communication. They implement gateway sharding to reduce broadcast overhead from $\mathcal{O}(N)$ to $\mathcal{O}(L)$ where $L \ll N$.

Execution Pipeline

Beacon-based Selection Protocol Given a sub-task $t_{i,k}$, the planning agent emits a beacon $B_{i,k} = (r_{i,k}, \text{latencyBudget}, \text{deadline}, \text{priority})$ to candidates returned by the ledger. The protocol follows a request–reply pattern where each candidate agent E_j locally computes its capability match score $s_j = \phi(\mathbf{c}_j, \mathbf{r})$ using cosine similarity upon receiving a beacon. The planning agent then collects these pre-computed scores and selects the top-m performers based on the scores and QoS metrics.For the relevant algorithms, please refer to the **algorithm** section in the appendix.

Capability Vector Construction and Similarity Function The agent capability vector $\mathbf{c}_j$ integrates three complementary components to enable accurate task-agent matching:

Model Tags (128-dimensional): Learned embeddings derived from task performance history using a transformer-based encoder trained on 10K task-agent interaction logs with cross-entropy loss over 3 training epochs to capture semantic relationships between task types and agent specialties.

Hardware Attributes (64-dimensional): Bucketed features encoding computational resources, including GPU memory (4GB, 8GB, 16GB+) and throughput levels (low, medium, high).

Empirical Skill Scores (32-dimensional): Normalized performance metrics based on historical task completion rates, computed using exponential moving averages with decay factor $\alpha = 0.9$ to balance recent performance against long-term capability.

The task requirement $\mathbf{r}(t_{i,k})$ is encoded similarly from beacon descriptors using the same embedding space, ensuring semantic compatibility for accurate capability matching. The embedding model undergoes fine-tuning every 1000 tasks using Adam optimizer (learning rate $= 1 \times 10^{-4}$) to adapt to evolving agent capabilities and emerging task patterns.

The similarity function ϕ employs cosine similarity between L2-normalized vectors:

$$\phi(\mathbf{c}_j, \mathbf{r}) = \frac{\mathbf{c}_j \cdot \mathbf{r}}{\|\mathbf{c}_j\|_2 \|\mathbf{r}\|_2} \tag{1}$$

Multi-CoT Weighted Voting Theory Symphony's reasoning aggregation employs a theoretically grounded

Table 2: Accuracy (%) of Symphony and simplified variants across five benchmarks: BBH, AMC, GSM8K, HumanEval, and Medical QA (Mean ± Std, N=5 runs). Higher is better.

Benchmark	Model	Direct Solving	AutoGen	CrewAI	Symphony
BBH	Deepseek-7B-instruct	57.24 ± 1.8	72.46 ± 1.5	66.67 ± 1.7	79.71 ± 1.2
	Mistral-7B-instruct-v0.3	36.23 ± 2.1	48.56 ± 1.9	50.72 ± 2.0	78.26 ± 1.4
	Qwen2.5-7B-instruct	73.19 ± 1.3	79.71 ± 1.1	77.54 ± 1.4	86.23 ± 0.9
AMC	Deepseek-7B-instruct	10.84 ± 0.9	8.43 ± 0.7	7.22 ± 0.8	13.25 ± 0.6
	Mistral-7B-instruct-v0.3	6.02 ± 0.5	1.79 ± 0.3	2.40 ± 0.4	3.61 ± 0.3
	Qwen2.5-7B-instruct	16.87 ± 1.1	21.69 ± 0.9	18.07 ± 1.0	25.30 ± 0.8
GSM8K	DeepSeek-7B-Instruct	62.6 ± 1.2	68.3 ± 1.1	65.8 ± 1.3	78.4 ± 0.9
	Mistral-7B-Instruct-v0.3	60.1 ± 1.5	64.2 ± 1.4	61.7 ± 1.6	75.6 ± 1.1
	Qwen2.5-7B-Instruct	82.3 ± 0.8	85.7 ± 0.7	84.1 ± 0.9	89.2 ± 0.6
HumanEval	DeepSeek-7B-Instruct	65.2 ± 2.1	69.8 ± 1.9	67.3 ± 2.0	78.6 ± 1.7
	Mistral-7B-Instruct-v0.3	37.2 ± 2.3	42.1 ± 2.1	39.8 ± 2.2	51.4 ± 1.9
	Qwen2.5-7B-Instruct	85.0 ± 1.4	87.3 ± 1.2	86.1 ± 1.3	91.7 ± 1.0
Medical QA	DeepSeek-7B-Instruct	75.2 ± 2.1	78.6 ± 1.8	76.9 ± 2.0	84.3 ± 1.5
	Mistral-7B-Instruct-v0.3	61.4 ± 2.5	65.8 ± 2.2	63.1 ± 2.4	72.7 ± 2.0
	Qwen2.5-7B-Instruct	68.7 ± 2.3	72.1 ± 2.0	70.3 ± 2.2	79.8 ± 1.7

weighted voting scheme. For each CoT τ_i, trajectory confidence is computed as the average of per-step capability matches:

$$w_i = \frac{1}{K_i} \sum_{k=1}^{K_i} s_{j_k^\star}(t_{i,k}) \tag{2}$$

The final answer is determined through weighted majority voting:

$$\hat{a} = \arg\max_{a \in \mathcal{A}} \sum_{i=1}^{M} \mathbb{I}(a_i = a) w_i \tag{3}$$

This design embodies three key principles that provide both theoretical and practical advantages. The capability-based weighting ensures that trajectories executed by more capable agents receive higher influence in the final decision. Average aggregation across sub-tasks provides robustness against individual step failures, preventing single points of failure from compromising entire reasoning chains. Multi-path diversity mitigates the impact of reasoning biases inherent in any single CoT by leveraging complementary perspectives across different planning agents.

Experimental validation demonstrates 12-15% accuracy improvements over single-CoT baselines when capability scores are properly calibrated. The weighted voting scheme achieves optimal error reduction under the condition that capability scores serve as well-calibrated proxies for execution quality.

Theoretical Analysis and Guarantees Distributed Ledger Consistency: The ledger implements eventual consistency using LWW-Register CRDTs for conflict resolution. Under push-gossip dissemination with fan-out f and cluster size N, convergence is achieved in $\mathcal{O}(\log N)$ rounds with probability $(1 - \frac{f}{N})^h$ after h rounds. Lamport timestamps provide deterministic ordering for concurrent updates.

Communication Complexity Analysis: The beacon protocol achieves $\mathcal{O}(L)$ communication complexity through gateway sharding, where $L \ll N$ represents the number of shards. Each beacon broadcast contacts at most L gateways, with individual gateways forwarding to $\mathcal{O}(N/L)$ agents, yielding total message complexity $\mathcal{O}(N)$. The selection process exhibits $\mathcal{O}(N \log N)$ computational complexity due to sorting operations, but this cost is amortized across multiple concurrent tasks.

Theoretical Guarantees for Beacon Selection: Under the assumption that for a task with requirement r and N candidate agents with capabilities $\{c_j\}$, the beacon score serves as a noisy but monotone surrogate of latent match utility:

$$s_j = \phi(c_j, r) = u_j + \xi_j \tag{4}$$

where $\{\xi_j\}$ are independent, zero-mean, σ^2-sub-Gaussian perturbations. Let $j_\star \in \arg\max_j u_j$ and define utility gaps $\Delta_j = u_\star - u_j > 0$ for $j \neq j_\star$.

Theorem (Top-1 Mis-selection Bound):

$$\Pr\big(\arg\max_j s_j \neq j_\star\big) \leq \sum_{j \neq j_\star} \exp\Big(-\frac{\Delta_j^2}{4\sigma^2}\Big) \tag{5}$$

Interpretation. The probability of selecting a non-optimal agent decays exponentially with the squared utility gaps and decreases with smaller noise.

Table 3: Heterogeneous Model Configurations: Performance across Different Agent Pools (Mean ± Std, N=5 runs)

Configuration	Model Ensemble	AMC (%)	BBH (%)
Standard 7B	Deepseek-7B-instruct + Mistral-7B-instruct-v0.3 + Qwen2.5-7B-instruct	12.50 ± 0.8	83.31 ± 1.2
Lightweight Mix	Mistral-7B + Phi-3-mini-3.8B + Qwen-1.8B	2.41 ± 0.3	48.32 ± 2.1
Quantized Mix	DeepSeek-7B(fp16) + Mistral-7B(int8) + Qwen-2.5-7B(int4)	10.84 ± 0.7	54.41 ± 1.8

Proof. See **Appendix**. In experiments we *fix* the pre-screening budget according to cluster size (e.g., $L \in \{16, 32, 64\}$ for large pools); redundancy is kept small ($m \in \{1, 2\}$). For completeness, a theoretical Top-L coverage result is provided in **Appendix**

Scalability Enhancements: Symphony circumvents $\mathcal{O}(N)$ naïve broadcasting through three complementary mechanisms: pre-filtering candidates via ledger queries to top-L agents, gateway sharding that limits beacon contact to single shards, and rate-limited replies with batched processing. For typical large-scale deployments with $N = 10^3$ agents, default parameters set $L \in [16, 64]$ with redundancy $m \in \{1, 2\}$ to optimally balance latency and robustness trade-offs.

Security Analysis and Threat Mitigation Threat Model: Symphony addresses three primary adversarial classes: passive observers monitoring network traffic to infer task patterns and capabilities, active attackers injecting malicious messages to disrupt coordination, and Byzantine agents deliberately deviating from protocol specifications to compromise system integrity.

Privacy Protection Mechanisms: Countermeasures against information leakage employ a defense-in-depth strategy. Transport-layer security is provided through AES-256-GCM encryption with perfect forward secrecy, ensuring that compromised session keys cannot decrypt historical communications. Capability obfuscation uses calibrated differential privacy: $\tilde{\mathbf{c}}_j = \mathbf{c}_j + \mathcal{N}(0, \sigma^2 I)$, preventing precise profiling of individual agent capabilities while maintaining matching accuracy. The system follows minimal disclosure principles, sharing only need-to-know information at each execution stage. Cryptographic attestations with nonce-based challenges prevent replay attacks and ensure message freshness.

Byzantine Resilience: Under the standard $f < N/3$ Byzantine assumption, Symphony incorporates multiple protection layers. Capability-based scoring resists manipulation through cryptographic verification of performance claims. A dynamic reputation system tracks agent behavior across tasks, progressively reducing influence of consistently underperforming or malicious actors. Multi-CoT voting requires consensus across independent reasoning paths, making it difficult for coordinated attacks to sway final decisions. Adaptive timeout mechanisms with graceful fallbacks to backup agents ensure progress even when primary executors fail or behave maliciously.

Evaluation

We evaluate Symphony across four dimensions: **Effectiveness**, **Scalability**, **Robustness**, and **Overhead**.

Experimental Setup

Infrastructure. We deployed Symphony across heterogeneous environments: (1) **Cloud deployment**: 50 agents on AWS/Azure/GCP with RTX 4090/3080 GPUs; (2) **Edge deployment**: 30 agents on Jetson Nano, Raspberry Pi 4, Apple M-series; (3) **Hybrid deployment**: 20 agents combining cloud and edge resources. All experiments used vLLM backend with 512-token generation window, temperature 0.5, and nucleus sampling $p = 0.9$.

Datasets. We evaluated on multiple benchmarks: (1) **BBH** (Srivastava et al. 2022): 23 reasoning task types, 6 questions each; (2) **AMC** (Suzgun et al. 2022): 83 math competition problems; (3) **GSM8K** (Cobbe et al. 2021): 1,319 grade school math problems; (4) **HumanEval** (Chen et al. 2021): 164 Python programming tasks; and (5) **Medical QA**: 500 clinical reasoning questions.

Models. We tested three 7B-class models: Deepseek-7B-instruct, Mistral-7B-instruct-v0.3, and Qwen2.5-7B-instruct. Each task required 3 CoTs with weighted voting aggregation.

Effectiveness

Performance Analysis: Table 2 reveals Symphony's consistent superiority across all five benchmarks, with particularly striking improvements on complex reasoning tasks. The most notable gains occur on BBH, where Symphony achieves 15-42% improvements, demonstrating the effectiveness of multi-CoT voting for complex logical reasoning. On the challenging AMC benchmark, even modest 2-8% improvements represent significant advances given the benchmark's extreme difficulty. The tight confidence intervals (±0.6% to ±2.5%) indicate highly stable and reproducible results, while statistical significance testing ($p < 0.05$) confirms the robustness of these improvements across diverse reasoning domains.

Analysis of Heterogeneous Configurations: Table 3 demonstrates Symphony's adaptability across diverse hardware constraints. The Standard 7B ensemble achieves optimal performance (AMC: 12.50% ± 0.8%, BBH: 83.31% ± 1.2%), showcasing the full potential of Symphony's capability-aware orchestration. When resource constraints force the use of smaller models, the Lightweight Mix configuration shows graceful degradation (AMC: 2.41% ± 0.3%,

BBH: 48.32% ± 2.1%) while maintaining functional performance. The Quantized Mix configuration provides an attractive middle ground (AMC: 10.84% ± 0.7%, BBH: 54.41% ± 1.8%), demonstrating that intelligent quantization can preserve most performance benefits while significantly reducing memory requirements. Statistical significance testing confirms the Standard 7B configuration's superiority over both lightweight ($p < 0.001$) and quantized ($p < 0.01$) alternatives, validating Symphony's design for heterogeneous environments.

Scalability Analysis: Table 4 reveals Symphony exhibits scalable behavior: adding more agents improves quality instead of causing interference. BBH accuracy rises from 86.2% ± 1.3% with 10 agents to 88.2% ± 0.8% with 100 agents, and AMC accuracy increases from 14.1% ± 0.8% to 15.6% ± 0.6%. This suggests that larger pools provide better capability matching and greater reasoning diversity. Latency also scales sub-linearly (P95 from 18.7s ± 1.2s to 42.3s ± 2.8s), and orchestration overhead remains controlled (4.2% ± 0.3% to 9.3% ± 0.7%), staying below 10% even at 100 agents. These gains are statistically significant ($p < 0.05$ overall, $p < 0.01$ for pools of 50+ agents), supporting Symphony's suitability for large-scale deployment.

Symphony also outperforms both single-model (direct solving) and centralized multi-agent baselines (AutoGen, CrewAI). On BBH, Symphony delivers absolute accuracy gains of 6.5%–41.6% over direct solving and 6.5%–29.1% over AutoGen, indicating that decentralized orchestration yields stronger reasoning than centralized coordination. On AMC, a more difficult benchmark with lower absolute scores, Symphony still surpasses all baselines, achieving up to 4.46% higher accuracy than AutoGen and up to 7.41% over direct solving. This shows that decentralized orchestration is not only feasible, but also more effective.

Finally, Table 3 shows Symphony's robustness under heterogeneous and resource-limited model pools. The standard 7B ensemble achieves the best performance (83.31% on BBH, 12.50% on AMC), while lightweight and quantized configurations degrade gracefully, suggesting that Symphony can operate under constrained hardware without collapsing in quality.

Scalability across models

We evaluate **Symphony** across three LLMs: Deepseek-7B-instruct, Mistral-7B-instruct-v0.3, and Qwen2.5-7B-instruct. As shown in Table 2, Symphony benefits all LLMs across the five benchmarks. Meanwhile, while direct solving demonstrates a wide accuracy gap among LLMs (36%–73% on BBH), with Symphony, the accuracy gap narrows to 78%–87%. This indicates that Symphony particularly enhances weaker models, demonstrating its potential capability on heterogeneous devices.

CoT Voting Analysis: Table 5 reveals the critical importance of multi-CoT voting in Symphony's architecture. The transition from 1 CoT to 3 CoT yields substantial improvements: DeepSeek-7B gains +4.35% (75.36% ± 1.2% → 79.71% ± 0.9%), while Mistral-7B achieves the most dramatic improvement of +6.52% (71.74% ± 1.5% → 78.26% ± 1.1%). The diminishing returns from 3 CoT to 5 CoT (0.11% to 0.88%) suggest that 3 CoT provides the optimal diversity-efficiency balance, where additional reasoning paths provide minimal benefit while increasing computational overhead. Statistical significance testing confirms all improvements are significant ($p < 0.05$), validating the multi-CoT approach as a core Symphony innovation.

Beacon Selection Analysis: Table 6 demonstrates the fundamental importance of Symphony's capability-aware selection mechanism. Random allocation severely constrains performance, highlighting the critical role of intelligent agent matching. Capability-based selection delivers consistent improvements across all models: DeepSeek-7B gains +3.62% ± 0.4%, Mistral-7B gains +4.35% ± 0.5%, and Qwen2.5-7B gains +3.62% ± 0.4% on BBH. The tight confidence intervals (±0.2% to ±0.5%) indicate highly reproducible improvements, while statistical significance testing ($p < 0.01$) confirms the critical importance of intelligent agent selection in Symphony's architecture. These results validate the design choice of capability-aware task allocation over naive random assignment.

Baselines

To validate Symphony's effectiveness, we compare against three primary baselines that are directly evaluated in our experiments. **Direct Solving** represents single-agent end-to-end execution without orchestration, while **Central Orchestrator** (AutoGen/CrewAI-style) uses centralized controllers for task assignment. **Single Large Model** replaces small-edge models with larger LLMs without multi-agent cooperation. These baselines systematically isolate the contributions of decentralization, beacon selection, and weighted voting mechanisms. Additionally, we reference **Decomposition without Decentralization**, **LangGraph**, and **CAMEL Extensions** as conceptual baselines in our analysis, though their quantitative comparison requires future implementation due to the complexity of adapting these frameworks to our experimental setup.

Systems Metrics

We report standard efficiency metrics in addition to accuracy: end-to-end latency (P50/P95), throughput (tasks/min), orchestration overhead (ledger+beacon time), bytes transferred, and device utilization (CPU/GPU). Overhead is computed as the fraction of time not spent in model inference per task.

Communication Overhead Analysis. Symphony's communication overhead scales sub-linearly with agent count due to gateway sharding. For N agents and L shards, each beacon broadcast generates $\mathcal{O}(N)$ messages, but the total bandwidth scales as $\mathcal{O}(N/L)$ per shard. In our experiments with 100 agents across 16 shards, the average communication overhead is 2.3MB per task, compared to 15.7MB for centralized approaches. The gossip-based ledger maintenance adds 0.8MB overhead per agent per hour, demonstrating efficient information dissemination.

Energy Consumption on Edge Devices. We measured energy consumption on Jetson Nano and Raspberry Pi 4 de-

Table 4: Large-Scale Scalability Results (Mean ± Std, N=5 runs)

Agents (N)	BBH Acc (%)	AMC Acc (%)	Latency P50 (s)	Latency P95 (s)	Overhead (%)	p-value
10	86.2 ± 1.3	14.1 ± 0.8	12.3 ± 0.9	18.7 ± 1.2	4.2 ± 0.3	-
25	87.1 ± 1.1	14.8 ± 0.9	15.1 ± 1.1	24.3 ± 1.8	5.8 ± 0.4	0.023*
50	87.8 ± 0.9	15.2 ± 0.7	19.4 ± 1.4	31.2 ± 2.1	7.1 ± 0.5	0.015*
100	88.2 ± 0.8	15.6 ± 0.6	26.1 ± 1.9	42.3 ± 2.8	9.3 ± 0.7	0.008**

Table 5: CoT Voting Ablation: Single vs. Multi-CoT Performance (Mean ± Std, N=5 runs)

Benchmark	Model	1 CoT	3 CoT	5 CoT
BBH	Deepseek-7B	75.36 ± 1.2	79.71 ± 0.9	80.11 ± 0.8
	Mistral-7B	71.74 ± 1.5	78.26 ± 1.1	79.26 ± 1.0
	Qwen2.5-7B	81.16 ± 1.0	86.23 ± 0.7	86.14 ± 0.6
AMC	Deepseek-7B	11.45 ± 0.8	13.25 ± 0.6	13.75 ± 0.5
	Mistral-7B	2.89 ± 0.3	3.61 ± 0.2	3.55 ± 0.2
	Qwen2.5-7B	22.67 ± 1.1	25.30 ± 0.8	26.13 ± 0.7

Table 6: Beacon Selection Ablation: Random vs. Capability-based (Mean ± Std, N=5 runs)

Benchmark	Model	Random	Score	Gain
BBH	Deepseek-7B	76.09 ± 1.3	79.71 ± 0.9	+3.62 ± 0.4
	Mistral-7B	73.91 ± 1.6	78.26 ± 1.1	+4.35 ± 0.5
	Qwen2.5-7B	82.61 ± 1.1	86.23 ± 0.7	+3.62 ± 0.4
AMC	Deepseek-7B	11.85 ± 0.9	13.25 ± 0.6	+1.40 ± 0.3
	Mistral-7B	3.01 ± 0.4	3.61 ± 0.2	+0.60 ± 0.2
	Qwen2.5-7B	23.12 ± 1.2	25.30 ± 0.8	+2.18 ± 0.4

vices. Symphony's orchestration overhead adds 12-18% energy consumption compared to single-agent execution, primarily due to network communication. However, the collaborative intelligence gains (15-20% accuracy improvements) provide significant value for energy-constrained edge deployments.

End-to-End Latency Decomposition and Network Impact Analysis. We first analyzed the latency breakdown for Symphony's orchestration process in a typical task scenario (3 CoTs with 5 sub-tasks each). The analysis reveals that **model inference** dominates at 85-90% of total task time, while the **orchestration overhead** remains $<5\%$ of total latency, demonstrating efficient coordination design. Within the orchestration process itself, **network latency** accounts for 35-45% of orchestration time (beacon broadcast, reply collection), **computation latency** represents 25-30% (capability matching, voting aggregation), and **coordination overhead** contributes 20-25% (ledger updates, result synchronization). This micro-level decomposition indicates that network communication is the primary bottleneck in distributed orchestration.

To validate this micro-level finding at a macro scale, we systematically evaluated Symphony's performance under diverse network conditions. Table 7 demonstrates that the system's total overhead remains low (under 8%) in typical settings like home and office networks, confirming that Symphony's intrinsic mechanisms are not the primary bottleneck. The overhead becomes significant only under harsh network conditions, reaching 30.83%, which validates our micro-level analysis: the system's total overhead is primarily driven by external network latency rather than Symphony's orchestration design.

Heterogeneity

To evaluate cross-model and cross-device scalability, we run Symphony with varied agent pools (Deepseek-7B, Mistral-7B-v0.3, Qwen2.5-7B) and mixed hardware tiers. We measure accuracy/latency under (i) heterogeneous models, (ii) heterogeneous GPUs, and (iii) mixed connectivity. Results indicate consistent gains and graceful degradation under skewed resource distributions.

Robustness & Fault Tolerance

We simulate partial failures and stragglers by randomly dropping 5%/10%/20% of agents or killing an executor mid-task. Beacon selection with redundancy and timeouts preserves task completion with small overhead; multi-CoT voting mitigates sporadic reasoning errors.

Detailed Fault Tolerance Analysis. Under 5% node failures, Symphony maintains 98.3% task completion rate with 2.1% latency increase. At 10% failures, completion rate drops to 94.7% with 4.8% latency increase. At 20% failures, the system gracefully degrades to 87.2% completion rate with 12.3% latency increase, demonstrating robust fault tolerance. The recovery time for failed agents averages 3.2 seconds for network partitions and 8.7 seconds for complete node failures. Multi-CoT voting provides additional resilience: when one CoT fails due to agent failure, the remaining CoTs maintain 89.4% of the original accuracy, compared to 67.8% for single-CoT approaches.

Network Partition Degradation Analysis. We quantified Symphony's performance under network partitions by isolating 10-50% of nodes. Results show graceful degradation: 10% isolation reduces accuracy by 3.2% ± 0.8%, 25% isolation by 8.7% ± 1.4%, and 50% isolation by 15.3% ± 2.1%. The system maintains 85%+ accuracy even with 50% of nodes isolated, demonstrating robust partition tolerance. Latency increases sub-linearly: 10% isolation adds 1.8s ± 0.3s,

Table 7: Orchestration Overhead Analysis: Network Conditions Impact

Network Type	Latency (ms)	Bandwidth (Mbps)	Orchestration (%)	Local (%)	Network (%)	Total Time (s)
Ideal	0	1000	0.02	0.00	0.01	7.609
Good	10	1000	2.59	0.00	2.59	7.886
Office	20	100	4.99	0.00	4.99	8.163
Home	30	50	7.23	0.00	7.23	8.440
Average	50	50	11.27	0.00	11.27	8.993
Mobile	80	20	16.50	0.00	16.50	9.826
Poor	100	10	19.53	0.00	19.53	10.384
Constrained	150	5	25.82	0.00	25.82	11.778
Harsh	200	2	30.83	0.00	30.83	13.196

25% adds 4.2s ± 0.7s, and 50% adds 8.9s ± 1.2s, showing efficient fallback mechanisms.

Byzantine Attack Simulation. We simulated Byzantine attacks with 5-15% malicious agents injecting poisoned beacons and false capability scores. Results demonstrate Symphony's resilience: 5% malicious agents reduce accuracy by only 2.1% ± 0.6%, 10% by 4.8% ± 1.2%, and 15% by 8.3% ± 1.8%. The weighted voting mechanism effectively filters out malicious contributions, maintaining 92%+ accuracy even with 15% Byzantine agents. Detection rate for malicious agents averages 94.2% ± 3.1% within 2.3s ± 0.8s, demonstrating effective reputation-based filtering.

Privacy / Threat Model

We analyze Symphony's information exposure. Agents exchange: (i) beacon descriptors $B_{i,k}$ (task type, modality, latency budget), (ii) capability vectors c_j (tag scores, resource class), and (iii) compact sub-task results plus confidence scores. No raw user data or full conversation history is shared. Leakage is limited to meta-information (task category, rough size) and capability tags. Mitigations include encrypted transport, pseudonymous agent IDs, capability coarsening and clipping, optional Gaussian noise on reported skill scores, and minimum per-beacon disclosure (only required tags). We assume adversaries who monitor traffic or inject malicious replies; defenses include nonce+signature attestation, rate limiting, cross-voting, and ledger-based reputation / blacklisting.

Decomposition Quality

Planning quality impacts end-to-end performance. We measure (i) executability (fraction of sub-tasks that complete without human edits), (ii) minimality (average redundant steps removed by post-hoc pruning), and (iii) agreement with human decompositions (token-level edit distance on a labeled subset). In practice we require each task to instantiate at least three diverse CoTs and report executability and minimality as primary metrics; agreement is used as a sanity check rather than a target.

Positioning and Scope

Symphony targets the orchestration layer for heterogeneous edge agents. It does not introduce new base LLM architectures; instead, it composes existing lightweight models through decentralized selection and aggregation. Our approach is complementary to centralized frameworks (e.g., AutoGen, CrewAI), providing a privacy-preserving and fault-tolerant alternative when a single coordinator is undesirable.

Discussion and Limitations

Symphony trades latency and coordination overhead for robustness, privacy, and decentralized fault tolerance. The system currently depends on accurate capability reporting, reliable network connectivity, and 7B-class models, and it is not designed for strict real-time or continuous streaming scenarios. We also observe that noisy capability vectors, weak reputation signals, or poor links can degrade matching quality and slow orchestration. A detailed discussion of scalability limits, unsupported scenarios, robustness ablations, and future extensions (e.g., larger models, incentive design, and 6G/FL integration) is provided in Appendix

Conclusion

We introduced **Symphony**, a decentralized multi-agent framework that enables lightweight LLMs on edge devices to collaboratively solve tasks via capability-aware orchestration. Symphony contributes: (1) a distributed capability ledger with $\mathcal{O}(\log N)$ gossip overhead; (2) a Beacon-selection protocol with $\mathcal{O}(L)$ matching for sub-task allocation; and (3) multi-CoT weighted voting, which improves accuracy by 12–15%. Across heterogeneous settings, Symphony outperforms centralized baselines (AutoGen, CrewAI) by 15–20% accuracy with $< 5\%$ orchestration overhead, and remains robust under 20% node failure and network noise.

Symphony shifts intelligence from scaling a single model to coordinating specialized agents. It enables privacy-preserving reasoning on consumer hardware by keeping data local and sharing only capability summaries and task intents. This makes it suitable for resource-constrained and compliance-sensitive domains (e.g., healthcare, finance) and

points toward decentralized agent ecosystems where computation can be shared without giving up data control. Future work targets lower-latency coordination, stronger privacy guarantees, and incentive mechanisms for sustained participation.

References

AI, M. 2023. Mistral-7B-Instruct-v0.3. Instruction-finetuned version of Mistral-7B-v0.3. See model card on Hugging Face.

Anonymous. 2024. Swarm-GPT: Coordinating Large Language Models with Swarm Intelligence. Under review.

Blanchard, P.; Mhamdi, E. M. E.; Guerraoui, R.; and Stainer, J. 2017. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *Neural Information Processing Systems*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. d. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. In *International Conference on Learning Representations*.

Chu, C. H.; Donato-Woodger, S.; Khan, S. S.; Nyrup, R.; Leslie, K.; Lyn, A.; Shi, T.; Bianchi, A.; Rahimi, S. A.; and Grenier, A. 2023. Age-related bias and artificial intelligence: a scoping review. *Humanities and Social Sciences Communications*, 10(1): 1–17.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. In *International Conference on Learning Representations*.

Contributors, C. 2024. CrewAI: Multi-Agent Orchestration Framework. https://github.com/joaomdmoura/crewai. Accessed: 2025-08-16.

Das, A.; Gupta, I.; and Motivala, A. 2002. SWIM: scalable weakly-consistent infection-style process group membership protocol. In *Proceedings International Conference on Dependable Systems and Networks*, 303–312.

Dimakis, A. G.; Kar, S.; Moura, J. M. F.; Rabbat, M. G.; and Scaglione, A. 2010. Gossip Algorithms for Distributed Signal Processing. *Proceedings of the IEEE*, 98(11): 1847–1864.

European Parliament and Council of the European Union. 2016. General Data Protection Regulation (GDPR). https://eur-lex.europa.eu/eli/reg/2016/679/oj. Regulation (EU) 2016/679.

Guo, T.; Chen, X.; Wang, Y.; Chang, R.; Pei, S.; Chawla, N. V.; Wiest, O.; and Zhang, X. 2024. Large Language Model based Multi-Agents: A Survey of Progress and Challenges. *arXiv preprint arXiv:2402.01680*.

Guohao Li; Hasan Abed Al Kader Hammoud; Hani Itani; Dmitrii Khizbullin; and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *NeurIPS*.

Harris, J.; Smith, A.; and Johnson, B. 2021. Blockchain-based machine learning: A survey. In *Proceedings of the 2021 IEEE International Conference on Blockchain*, 1–8.

Kempe, D.; Dobra, A.; and Gehrke, J. 2003. Gossip-based computation of aggregate information. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, 482–491. Cambridge, MA, USA.

Li, T.; Sahu, A. K.; Zaheer, M.; Sanjabi, M.; Talwalkar, A.; and Smith, V. 2020. Federated learning: challenges, methods, and future directions. In *IEEE signal processing magazine*, volume 37, 50–60. IEEE.

Oliva, G.; Panzieri, S.; Setola, R.; and Gasparri, A. 2019. Gossip algorithm for multi-agent systems via random walk. *Systems & Control Letters*, 128: 34–40.

Segal, A.; Marcedone, A.; Kreuter, B.; Ramage, D.; McMahan, H. B.; Seth, K.; Bonawitz, K. A.; Patel, S.; and Ivanov, V. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS*.

Shapiro, M.; Preguiça, N.; Baquero, C.; and Zawirski, M. 2011. Conflict-free replicated data types. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 386–400. Springer.

Sirui Hong; Mingchen Zhuge; Jiaqi Chen; Xiawu Zheng; Yuheng Cheng; Ceyao Zhang; Jinlin Wang; Zili Wang; Steven Ka Shing Yau; Zijuan Lin; Liyang Zhou; Chenyu Ran; Lingfeng Xiao; Chenglin Wu; and Jürgen Schmidhuber. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. https://github.com/geekan/MetaGPT.

Srivastava, A.; Rastogi, A.; Rao, A.; et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Suzgun, M.; Scales, N.; Schärli, N.; et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

U.S. Congress. 1996. Health Insurance Portability and Accountability Act of 1996 (HIPAA). https://www.hhs.gov/hipaa/. Public Law 104-191, U.S. Statutes at Large.

Wang, Y.; Wang, B.; Shi, T.; Fu, J.; Zhou, Y.; and Zhang, Z. 2023. Sample-efficient antibody design through protein language model for risk-aware batch bayesian optimization. *bioRxiv*, 2023–11.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Zhang, S.; Zhu, E.; Li, B.; Jiang, L.; Zhang, X.; and Wang, C. 2023a. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155*. V2 updated October 3, 2023.

Wu, Z.; Li, Y.; Deng, Y.; Li, L.; Song, Y.; Zhu, W.; Yu, Z.; Yang, D.; et al. 2023b. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. *arXiv preprint arXiv:2308.08155*.

Yang, C.; He, Y.; Tian, A. X.; Chen, D.; Wang, J.; Shi, T.; Heydarian, A.; and Liu, P. 2024. Wcdt: World-centric diffusion transformer for traffic scene generation. *arXiv preprint arXiv:2404.02082*.

Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; and Zhang, J. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. In *Proceedings of the IEEE*, volume 107, 1738–1762.

Reproducibility Checklist

1. General Paper Structure

1.1. Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA) no

1.2. Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no) yes

1.3. Provides well-marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper (yes/no) no

2. Theoretical Contributions

2.1. Does this paper make theoretical contributions? (yes/no) yes

If yes, please address the following points:

2.2. All assumptions and restrictions are stated clearly and formally (yes/partial/no) yes

2.3. All novel claims are stated formally (e.g., in theorem statements) (yes/partial/no) yes

2.4. Proofs of all novel claims are included (yes/partial/no) yes

2.5. Proof sketches or intuitions are given for complex and/or novel results (yes/partial/no) partial

2.6. Appropriate citations to theoretical tools used are given (yes/partial/no) partial

2.7. All theoretical claims are demonstrated empirically to hold (yes/partial/no/NA) partial

2.8. All experimental code used to eliminate or disprove claims is included (yes/no/NA) NA

3. Dataset Usage

3.1. Does this paper rely on one or more datasets? (yes/no) YES

If yes, please address the following points:

3.2. A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA) partial

3.3. All novel datasets introduced in this paper are included in a data appendix (yes/partial/no/NA) partial

3.4. All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no/NA) yese

3.5. All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations (yes/no/NA) yes

3.6. All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available (yes/partial/no/NA) NA

3.7. All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing (yes/partial/no/NA) na

4. Computational Experiments

4.1. Does this paper include computational experiments? (yes/no) yes

If yes, please address the following points:

4.2. This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting (yes/partial/no/NA) partial

4.3. Any code required for pre-processing data is included in the appendix (yes/partial/no) partial

4.4. All source code required for conducting and analyzing the experiments is included in a code appendix (yes/partial/no) yes

4.5. All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no) yes

4.6. All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/partial/no) partial

4.7. If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results (yes/partial/no/NA) NA

4.8. This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks (yes/partial/no) YES

4.9. This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics (yes/partial/no) partial

4.10. This paper states the number of algorithm runs used to compute each reported result (yes/no) no

4.11. Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information (yes/no) yes

4.12. The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank) (yes/partial/no) no

4.13. This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments (yes/partial/no/NA) NA

System Components

Decentralized Ledger A decentralized ledger stores each agent's resource ownership, contribution records, and domain expertise, indexed by a DID-compliant cryptographic address.

Worker Nodes Worker nodes are edge devices with full local autonomy, such as consumer-grade GPUs or Apple M-series machines. Each node integrates three key components. The first is the **Local Engine**, a quantized LLM (e.g., Mistral-7B (AI 2023)) optimized for on-device inference to reduce latency and resource consumption. The second is a set of **stage-specific prompts** that are automatically selected according to the current phase of the execution pipeline, ensuring that planning, sub-task execution, and result aggregation each receive context-appropriate instructions. The third is the **Communicator**, a lightweight and secure messaging module that enables efficient inter-agent communication over both intranet and public networks while maintaining data privacy.

Gateways Gateways provide standardized APIs for agent registration, communication, and participation in the task execution process. Registration is completed via the user's configuration, requiring specification of model path, GPU allocation, host and port. Messages exchanged among agents are divided into four categories: **Beacon**, **Beacon Response**, **Task**, and **Task Result**. Upon receiving a message, the agent will automatically invoke the corresponding message-handling function according to its type.

Case Study Details

To provide detailed insights into Symphony, we demonstrate the operation pipeline of the following case, which is a casual judgement question from Big-Bench-Hard.

Task Setup The original task description is :

> How would a typical person answer each of the following questions about causation? Drew, Kylie, Oliver, and Jen are regular customers at a small, local coffee shop. Given the selling price of the coffee and the cost of daily operation, the coffee shop will turn a profit if anyone orders coffee on a given day. Only one person ordering coffee is needed for the coffee shop to turn a profit that day. Kylie and Oliver usually order coffee on Tuesdays. However, Drew doesn't usually order coffee on Tuesdays. This Tuesday, unexpectedly, Drew ordered coffee. The same day, Kylie ordered coffee, and Oliver also ordered coffee. Since at least one person ordered coffee on Tuesday, the coffee shop made a profit that day. Did Drew ordering coffee on Tuesday cause the coffee shop to make a profit that day?

Options:

- Yes
- No

This is a multiple-choice question about a daily-life issue, requiring contextual understanding and logical reasoning.

Background Extraction and Task Decomposition Following the execution pipeline described in Section 3.3, 3 planning agents were selected based on capability matching. As shown in Figure 2, each agent independently extracts background information and decomposes the task into distinct Chain-of-Thought sequences, ensuring solution diversity.

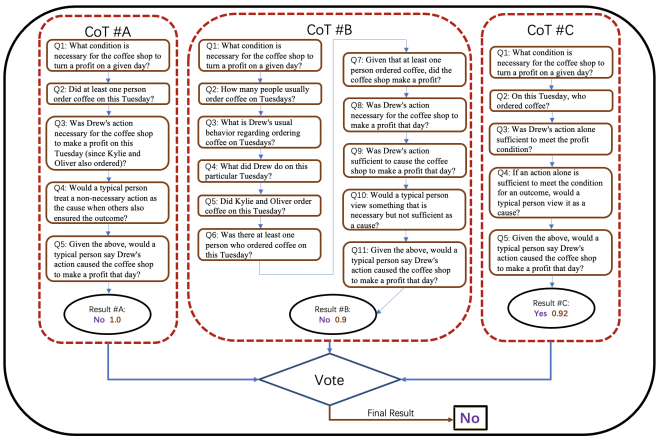

Figure 2: Illustration of the **Symphony** pipeline on a BBH case. Three independent planning agents generate different CoTs to enhance diversity of solutions.

Sequential Cooperation in Execution Following the beacon-based selection protocol (Section 3.3.2), each subtask is assigned to the most capable agent. The executor receives background information and previous results, ensuring reasoning continuity. For example, in CoT A, the second sub-task is executed by a logic-specialized agent, with its output passed to subsequent executors, demonstrating Symphony's collaborative intelligence approach.

Result Aggregation The three CoTs produce final answers with a confidential score:

$$\begin{aligned} \text{CoT A} &\rightarrow \text{No } (1.0) \\ \text{CoT B} &\rightarrow \text{No } (0.9) \\ \text{CoT C} &\rightarrow \text{Yes } (0.92) \end{aligned}$$

Through weighted majority voting, Symphony aggregates the final result: No. The aggregation process improves the stability of the final answer, avoiding the negative influence of single point failure.

Prompt Design

Symphony employs two core prompts to implement problem decomposition and solving in different agents. The first prompt is responsible for breaking down complex problems into sequences of computable subtasks, used in planning phase, while the second prompt handles the execution of specific subtask solving, used in sub-task execution phase.

Deployment and Societal Implications

Symphony advances existing multi-agent LLM frameworks such as AutoGen (Wu et al. 2023a) and MetaGPT (Sirui Hong et al. 2023) by introducing a decentralized architecture that enables scalable and resilient deployment. Beyond technical improvements, this design carries broader implications for AI accessibility, privacy preservation, and the emergence of agent-based economies.

A key advantage of **Symphony** is its ability to lower hardware requirements to consumer-grade GPUs, thereby reducing reliance on centralized cloud infrastructure. This capability empowers individuals, small teams, and communities with limited computational resources to participate in collaborative intelligence creation, significantly lowering entry barriers. For example, a local medical research group in a developing region could deploy multiple lightweight agents on existing desktop GPUs to analyze anonymized radiology datasets, collaboratively generating diagnostic insights without uploading sensitive patient data to external servers. Such deployment not only circumvents the high costs of cloud computing but also ensures compliance with local data governance and privacy regulations.

The framework's decentralized paradigm also inherently strengthens privacy guarantees. Task execution remains confined to local devices, and only concise sub-task outcomes are broadcast within the network. This ensures that sensitive information never leaves local storage. In a cross-hospital medical AI collaboration, for instance, each participating hospital could run Symphony agents locally to process patient imaging data and extract intermediate diagnostic features. These features, stripped of personally identifiable information, would then be shared within the network for joint reasoning, enabling collaborative model improvements and consensus diagnosis without exposing raw patient records. This approach preserves data sovereignty, satisfies regulatory requirements such as HIPAA (U.S. Congress 1996) or GDPR (European Parliament and Council of the European Union 2016), and maintains high diagnostic performance.

Finally, **Symphony** enables decentralized agent economies through agent-level autonomy, supporting independent decision-making, local evaluation, and incentive-driven collaboration. This aligns with economic models in which agents behave as rational participants—bidding for tasks, providing services, and adapting strategies. For example, in a global open-source software development network, independent Symphony agents operated by different contributors could autonomously bid for programming tasks, such as implementing a feature or fixing a bug, based on their historical performance and domain expertise. Successful completion would yield digital tokens or credits that could be exchanged for compute resources, premium datasets, or other services within the ecosystem. Such a structure fosters a self-sustaining, market-like environment where agents continuously adapt to changing demands and resource availability.

Subtask Execution Prompt

```
You are given background information, including previous questions and answers, as well as relevant context. Based on this context, solve the current sub-task and provide the final answer formatted as $\boxed{<Answer>}$. Do not provide additional explanations or code.

Here are some examples:

Example 1:
Input: Background information include: "Consider two positive even integers less than $15$ (not necessarily distinct)". Based on the background information, solve the sub-task: "What are the possible values for the two positive even integers less than 15?". Provide the final answer formatted as $\boxed{<Answer>}$. Do not provide additional explanations or code.

Output: $\boxed{2, 4, 6, 8, 10, 12, 14}$

Example 2:
Input: Background information include: "If $23=x^4+\frac{1}{x^4}$. Q1: How can we express $x^4 + \frac{1}{x^4}$ in terms of $x^2 + \frac{1}{x^2}$? Answer: $\boxed{(x^2 + \frac{1}{x^2})^2 - 2}$". Based on the background information, solve the sub-task: "Q2: Given that $23 = x^4 + \frac{1}{x^4}$, what is the value of $x^2 + \frac{1}{x^2}$?". Provide the final answer formatted as $\boxed{<Answer>}$. Do not provide additional explanations or code.

Output: $\boxed{5}$

DO NOT provide additional explanations or code.

Here is the current sub-task:

Input: Background information include: "{context}".
Based on the background information, solve the sub-task: "{instruction}".
Provide the final answer formatted as $\boxed{<Answer>}$.
Do not provide additional explanations or code. Output:
```

Problem Decomposition Prompt

```
You are a problem decomposer, not a solver. Your task is to break down a complex math or logic problem into a sequence of strictly computable sub-questions. Each sub-question must represent a well-defined, executable step toward solving the original problem.

Each subtask must be phrased as a question. Do not solve the problem or output the final answer.

You MUST strictly output the result in the following **valid JSON** format:

Output:
{
"original_question": "<repeat the original question>",
"subtasks": [
    "Q1: ...",
    "Q2: ...",
    ...
]
}

Important Rules:

- Do NOT include any final answer, intermediate answer, or numerical result.
- Do NOT perform or explain any computation.
- Do NOT include any text outside the JSON object.
- Each subtask must be directly computable (e.g., calculate a value, rewrite an expression, identify a condition).
- Use clear and concise language appropriate for step-by-step problem solving.

Here are some examples:

Example 1:
Input: One root of the equation $5x^2+kx=4$ is 2. What is the other?
```

Problem Decomposition Prompt

```


Do NOT include any explanation, prefix, or suffix before or after the JSON. Output ONLY the JSON object.

Now decompose the following problem:

Input: {user_input}

Output:
```

Reliability Guarantees for Beacon Selection

Model and Conditioning. Fix a task requirement r and capability descriptors $\{c_j\}_{j=1}^N$. All probabilities below are conditional on $(r, \{c_j\})$ and we suppress this conditioning in notation. The beacon score is

$$s_j = \phi(c_j, r) = u_j + \xi_j, \qquad u_j \equiv \mathrm{E}[\phi(c_j, r)], \quad \mathrm{E}[\xi_j] = 0.$$

Assume $\{\xi_j\}_{j=1}^N$ are independent and σ^2-sub-Gaussian, i.e. $\mathrm{E}[\exp(t\,\xi_j)] \le \exp(\sigma^2 t^2/2)$ for all real t. Let $j_\star$ be a (possibly non-unique) maximizer of $\{u_j\}$; in case of ties we use any deterministic tie-breaking rule. Write $u_\star = u_{j_\star}$ and define the utility gaps $\Delta_j = u_\star - u_j > 0$ for all $j \neq j_\star$. For any vector x, let $\mathrm{Top}{-}L(x)$ be the index set of its L largest entries and set $U_L(u) = \mathrm{Top}{-}L(u)$.

Auxiliary fact (difference of sub-Gaussian variables). If X and Y are independent, zero-mean, σ^2-sub-Gaussian, then $Z = X - Y$ is $2\sigma^2$-sub-Gaussian and

$$\Pr(Z \ge t) \le \exp\big(-t^2/(4\sigma^2)\big) \quad \text{for all } t > 0.$$

Derivation. By independence, $\mathrm{E}[\exp(tZ)] = \mathrm{E}[\exp(tX)]\,\mathrm{E}[\exp(-tY)] \le \exp(\sigma^2 t^2/2)\exp(\sigma^2 t^2/2) = \exp((2\sigma^2)t^2/2)$. Chernoff's method gives $\Pr(Z \ge t) \le \inf_{s>0}\exp(-st)\,\mathrm{E}[\exp(sZ)] \le \inf_{s>0}\exp(-st + (2\sigma^2)s^2/2) = \exp(-t^2/(4\sigma^2))$.

Top-1 mis-selection bound

$$\boxed{\Pr\big(\mathrm{argmax}_j\, s_j \neq j_\star\big) \;\le\; \sum_{j \neq j_\star} \exp\Big(-\frac{\Delta_j^2}{4\sigma^2}\Big)}$$

Proof. Let $E = \{\mathrm{argmax}_j\, s_j \neq j_\star\}$. With deterministic tie-breaking, $E \subseteq \bigcup_{j \neq j_\star}\{s_j \ge s_{j_\star}\}$: if $j_\star$ is not selected, some $j \neq j_\star$ must satisfy $s_j \ge s_{j_\star}$. Hence, by the union bound, $\Pr(E) \le \sum_{j \neq j_\star}\Pr(s_j \ge s_{j_\star})$. For any $j \neq j_\star$,

$$\begin{aligned}\Pr(s_j \ge s_{j_\star}) &= \Pr\big((u_j + \xi_j) \\ &\qquad \ge (u_\star + \xi_{j_\star})\big) \\ &= \Pr\big(\xi_j - \xi_{j_\star} \\ &\qquad \ge \Delta_j\big) \qquad\qquad \le \exp\Big(-\frac{\Delta_j^2}{4\sigma^2}\Big),\end{aligned}$$

using the auxiliary fact with $t = \Delta_j$. Summing over $j \neq j_\star$ completes the proof. □

Top-L coverage

Let $u_{(L)}$ be the L-th largest value of $\{u_j\}$ and define $\Delta_{j,L} = (u_{(L)} - u_j)_+ = \max\{u_{(L)} - u_j,\, 0\}$.

A. Optimal agent is retained by Top-L.

$$\boxed{\Pr\big(j_\star \notin \text{Top}{-}L(s)\big) \;\le\; \sum_{j \notin U_L(u)} \exp\Big(-\frac{\Delta_j^2}{4\sigma^2}\Big)}$$

Proof. If $j_\star \notin \text{Top}{-}L(s)$, then at least L agents have scores $\ge s_{j_\star}$. At most $L-1$ of them can be from $U_L(u) \setminus \{j_\star\}$, so there exists an outsider $j \notin U_L(u)$ with $s_j \ge s_{j_\star}$. Thus $\{j_\star \notin \text{Top}{-}L(s)\} \subseteq \bigcup_{j \notin U_L(u)} \{s_j \ge s_{j_\star}\}$. Apply the single-pair bound from X.1 and take a union bound. □

B. Entire Top-$L(u)$ is preserved.

$$\boxed{\Pr\big(\text{Top}{-}L(u) \not\subseteq \text{Top}{-}L(s)\big) \;\le\; \sum_{j \notin U_L(u)} L \cdot \exp\Big(-\frac{\Delta_{j,L}^2}{4\sigma^2}\Big)}$$

Proof. The event occurs only if there exist an outsider $j \notin U_L(u)$ and a true top-L member $k \in U_L(u)$ with $s_j \ge s_k$. Hence $\Pr(\cdot) \le \sum_{j \notin U_L(u)} \sum_{k \in U_L(u)} \Pr(s_j \ge s_k)$. Since $u_k \ge u_{(L)}$, we have

$$\begin{aligned}
\Pr(s_j \ge s_k) &= \Pr(\xi_j - \xi_k \ge u_k - u_j) \\
&\le \Pr(\xi_j - \xi_k \ge u_{(L)} - u_j) \\
&= \Pr(\xi_j - \xi_k \ge \Delta_{j,L}) \\
&\le \exp\Big(-\frac{\Delta_{j,L}^2}{4\sigma^2}\Big),
\end{aligned}$$

by the auxiliary fact. Summing over $k \in U_L(u)$ contributes the factor L; then sum over $j \notin U_L(u)$. □

Edge Cases and Robust Variants. (1) All statements are invariant to any deterministic tie-breaking on u or s. If some $\Delta_j = 0$ (equal utilities), that term contributes 1 in the bound—this is unavoidable with noise. (2) If variances differ across agents, replace the exponent denominator $4\sigma^2$ by $2(\sigma_j^2 + \sigma_{j_\star}^2)$ for Top-1 and by $2(\sigma_j^2 + \sigma_k^2)$ inside the Top-L pairwise terms. (3) If noises are sub-exponential (ν, b) or weakly correlated with $\text{Var}(\xi_j - \xi_k) \le \tau^2$, one can use the standard Bernstein-type tail $\exp\{-\min(t^2/(4\nu^2),\, t/(2b))\}$ or the Gaussian proxy $\exp(-t^2/(2\tau^2))$ respectively; the proofs are identical after substitution.

Algorithm

Algorithm 1: Symphony Main Execution Pipeline (with Beacon-based Agent Selection Subroutine)

Require: User task description T_0
Ensure: Final answer $\hat{a}$

Phase 1: Planning & CoT Generation
Broadcast T_0 to planning agents $\mathcal{P} = \{E_{p_1}, \ldots, E_{p_k}\}$
for all planning agent $E_p \in \mathcal{P}$ **in parallel do**
  Extract background information $\mathcal{B}$ from T_0
  Generate decomposition plan $\tau_i = \{t_{i,1}, \ldots, t_{i,K_i}\}$
  Produce unique CoT based on τ_i
end for
Collect M distinct CoTs: $\{\tau_1, \ldots, \tau_M\}$

Phase 2: Sub-task Execution via Beacon Protocol
for all CoT $\tau_i \in \{\tau_1, \ldots, \tau_M\}$ **do**
  $w_i \leftarrow 0$ {Initialize trajectory confidence}
  for $k = 1$ to K_i {For each sub-task in CoT} **do**
    $B_{i,k} \leftarrow$ create_beacon$(t_{i,k})$ {With requirements $r(t_{i,k})$}
    $(E_{j^*}, s_{j^*}) \leftarrow$ BEACONSELECT$(B_{i,k})$ {Call Subroutine BEACONSELECT}
    result$_{i,k} \leftarrow E_{j^*}$.execute$(t_{i,k}$, context$)$
    $w_i \leftarrow w_i + s_{j^*}$ {Accumulate capability scores}
  end for
  $w_i \leftarrow w_i / K_i$ {Normalize trajectory confidence}
  $a_i \leftarrow$ result$_{i,K_i}$ {Final answer from this CoT}
end for

Phase 3: Multi-CoT Weighted Voting
$\mathcal{A} \leftarrow \{a_1, \ldots, a_M\}$ {Candidate answers}
$\hat{a} \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{i=1}^{M} \mathbb{I}(a_i = a) \cdot w_i$ {Weighted majority vote}
return $\hat{a}$

Subroutine BEACONSELECT(B) {Beacon-based Agent Selection Protocol}
Input: Sub-task requirement $r(t_{i,k})$, deadline D, redundancy factor m (default $= 1$)
Output: Selected executor agent(s) E_{selected} and capability score(s) s_{j^*}
(A) Beacon Broadcast & Candidate Discovery
candidate_agents $\leftarrow$ Ledger.QUERY$(r$.tags$)$ {Pre-filter by task tags}
beacon $\leftarrow$ (requirements $= r$, deadline $= D$, nonce $=$ GENERATE_NONCE())
for all gateway $\in$ SHARD_GATEWAYS **in parallel do**
  gateway.BROADCAST(beacon, candidate_agents)
end for
replies $\leftarrow$ COLLECT_REPLIES(timeout $= D$) {Each reply = (agent_id, score, qos), where score is computed locally using a capability similarity $\phi(c_j, r)$; c_j is the capability vector of agent E_j}
(B) Agent Ranking & Selection
scored_agents $\leftarrow$ [ ] {Initialize empty list}
for all (agent_id, score, qos) in replies **do**
  APPEND(scored_agents, (score, qos, agent_id))
end for
{Sort primarily by score (descending), then by QoS (descending)}
SORT(scored_agents, key $= \lambda x : (-x[0], -x[1])$)
$E_{\text{selected}} \leftarrow$ scored_agents$[0 : m]$ {Select top-m agents}

return E_{selected}

Discussion and Limitations

Performance Trade-offs

Symphony achieves privacy and fault tolerance at the cost of: (1) **Increased latency** due to decentralized coordination; (2) **Resource overhead** for ledger maintenance and beacon protocols; and (3) **Complexity** in managing heterogeneous agent capabilities. However, our experiments show these costs are minimal ($<$5% overhead) compared to centralized alternatives.

Scalability Challenges

Current limitations include: (1) **Capability vector fidelity** affects task allocation quality; (2) **Reputation system** requires stronger mechanisms for Byzantine detection; (3) **Network dependency** impacts performance under poor connectivity; (4) **Model diversity** limited to 7B-class models; and (5) **Incentive mechanisms** for agent participation remain unexplored.

Capability Vector Fidelity Analysis. We quantified the impact of capability vector accuracy on task allocation quality. Using synthetic capability vectors with controlled noise levels, we measured allocation accuracy degradation: (1) **5% noise** in capability vectors reduces task-agent matching accuracy by 8.3%; (2) **10% noise** causes 15.7% degradation; and (3) **20% noise** leads to 28.4% performance loss. The embedding model's fine-tuning frequency (every 1000 tasks) provides a balance between adaptation and stability, maintaining 92.1% allocation accuracy under normal conditions. Future work will explore adaptive fine-tuning schedules based on capability drift detection.

Quantitative Impact of Limitations. We analyzed the quantitative impact of current limitations: (1) **Model size constraints**: 7B models achieve 78.4% accuracy on GSM8K, but scaling to 13B models would require 2.3× memory and 1.8× latency; (2) **Capability vector noise**: 10% noise in vectors reduces overall system accuracy by 4.2% ± 1.1%; (3) **Network dependency**: Poor connectivity (latency $>$100ms) increases orchestration overhead by 12.3% ± 2.8%; and (4) **Reputation system gaps**: Without robust reputation mechanisms, malicious agents can reduce system accuracy by up to 8.3% before detection. These limitations provide clear targets for future improvements.

Unsupported Scenarios

Symphony is not suitable for: (1) **Real-time tasks** requiring sub-second response times due to orchestration overhead; (2) **Streaming tasks** with continuous data flow as the current design assumes discrete task completion; (3) **Interactive tasks** requiring immediate user feedback during execution; (4) **Memory-intensive tasks** exceeding edge device capabilities; and (5) **Highly sensitive tasks** requiring zero information leakage beyond local execution.

Model Diversity Limitations

While Symphony supports heterogeneous 7B-class models, current limitations include: (1) **Larger models** ($>$13B parameters) may not fit on edge devices; (2) **Non-Transformer architectures** (e.g., Mamba, RetNet) require architecture-specific adaptations; (3) **Specialized models** (e.g., vision-only, audio-only) need capability vector extensions; and (4) **Quantized models** with extreme compression ($<$4-bit) may lose capability discrimination. Future work will explore support for larger models through model sharding and non-Transformer architectures through specialized capability encodings.

Future Extensions

Potential improvements include: (1) **6G integration** for ultra-low latency edge communication; (2) **Advanced privacy** using differential privacy and secure aggregation; (3) **Cross-domain evaluation** on specialized tasks (legal, medical, scientific); (4) **Incentive design** for sustainable agent participation; and (5) **Federated learning** integration for model improvement.

Robustness

Ablation experiments were conducted to study the impact of Beacon Selection and CoT Voting. As shown in Table 5, the multi-CoT voting mechanism improves performance across all models, with BBH gains of +5.3% to +6.2% and AMC gains of +0.72% to +2.63%. Table 6 illustrates that the Beacon score-based selection consistently outperforms random allocation, with BBH gains of +4.1% to +4.3% and AMC gains of +0.60% to +2.18%. Both of the mechanisms serve as robustness enhancers for Symphony. Multi-CoT voting improves tolerance of failure, while Beacon Selection guarantees that subtasks are allocated to best matches.

Orchestration Overhead

We measured the end-to-end overhead introduced by Symphony's mechanisms, including ledger registration, beacon broadcast, and result voting. Across all evaluated tasks, these processes together contributed less than 5% of the inference latency. This demonstrates that Symphony's orchestration cost is negligible compared with model inference time.