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

Ji Wang<sup>1,3</sup>, Kashing Chen<sup>1,4</sup>, Xinyuan Song<sup>2</sup>, Ke Zhang<sup>1,5</sup>, Lynn Ai<sup>1</sup>, Eric Yang<sup>1</sup>, Bill Shi<sup>1\*</sup>

Gradient

- <sup>2</sup> Dpartment of Computer Science, Emory University
- Department of Industrial Engineering and Oprations Research, Columbia University
   The Chinese University of Hong Kong
- <sup>5</sup> Department of Computer Science and Computer Engineering, Waseda University

## **Abstract**

Most existing Large Language Model (LLM)-based agent frameworks rely on centralized orchestration, incurring high deployment costs, rigid communication topologies, and limited adaptability. To address these challenges, we introduce **Symphony**, a decentralized multi-agent system which enables lightweight LLMs on consumer-grade GPUs to coordinate. Symphony introduces three key mechanisms: (1) a decentralized ledger that records capabilities, (2) a Beacon-selection protocol for dynamic task allocation, and (3) weighted result voting based on CoTs. This design forms a privacy-saving, scalable, and fault-tolerant orchestration with low overhead. Empirically, Symphony outperforms existing baselines on reasoning benchmarks, achieving substantial accuracy gains and demonstrating robustness across models of varying capacities. Our code is available at https://github.com/GradientHQ/Symphony.git.

# 1 Introduction

Large Language Models (LLMs) have demonstrated strong performance in natural language understanding, reasoning [1], planning [2], and tool use [3], powering diverse domains, such as education [4], healthcare [5], autonomous systems [6], software engineering [7], and scientific discovery [8]. Driven by LLMs, machine learning systems are increasingly dependent on agent-based orchestration to address complicated tasks. Resent frameworks such as AutoGen [9], MetaGPT [10], CAMEL [11], Voyager [12], and CrewAI [13] have demonstrated that agent cooperation enhances task solving. However, most LLM-based systems adopt a centralized architecture where a single agent manages task allocation, orchestrates message routing, and monitors whole workflow, resulting in scalability bottlenecks [14], rigid pipeline [15, 16], and reliance on expensive server-grade GPUs [17].

Meanwhile, edge computing resources (e.g., RTX 4090, Jetson boards, and Apple M-series) are becoming more powerful and available. These devices provide opportunities to bring ML workloads from cloud cluster to decentralized systems. Although decentralized systems have been studied for robustness and coordination - such as wireless sensor networks [18], distributed robotics [19], and blockchain [20] - their integration with heterogeneous LLM agents has not been widely addressed.

Thus, a decentralized orchestration runtime which is able to efficiently allocate tasks, operate on heterogeneous devices, and maintain robustness without a centralized orchestrator is urgently needed. In this work, we present **Symphony**, a decentralized multi-agent system in which lightweight LLMs on edge devices coordinate to achieve intelligence across heterogeneous environments. With three key mechanisms, **Symphony** is capable of completely decentralized workflow:

<sup>\*</sup>Corresponding author: tianyu@gradient.network

- a **ledger** which dynamically records device availability and agent capabilities
- a Beacon-based selection protocol that allocate tasks dynamically and precisely to bestmatch agents via Beacon [21]
- weighted result voting that aggregates results from diverse Chain-of-Thoughts (CoTs) [22].

Together, the mechanisms enable **Symphony** to orchestrate ML workloads on heterogeneous edge devices in a privacy saving, scalable, and fault-tolerant way.

# 2 Methodology

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

# 2.1 System Components

**Decentralized Ledger** A decentralized ledger stores records each agent's capability and availability.

**Worker Nodes** Worker nodes are edge devices equipped with a quantized LLM(e.g., Mistral-7B [23]), a set of **stage-specific prompts**, and a lightweight **Communicator**.

**Gateways** Gateways provide standardized APIs for agent registration, communication, and participation in the task execution process.

Due to space constraints, the details of system components are provided in Appendix A.

## 2.2 Execution Pipeline

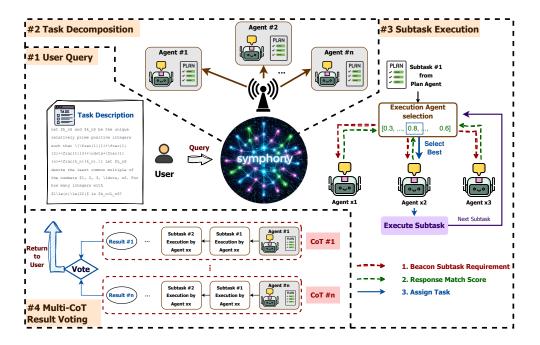


Figure 1: Overview of **Symphony**. 1. A query from user is decomposed into multiple sub-tasks by planning agents. 2. Sub-task execution leverages Beacon-based agent selection to choose appropriate agents. 3. Final response is generated through result voting across multiple reasoning paths.

**User Query** The process begins when the user submits a task description to Symphony. This query is broadcast to multiple planning agents, each responsible for producing a unique CoT.

**Planning Phase** The original description  $T_0$  is broadcast to a set of planning agents, each operating independently. Upon receiving  $T_0$ , a planning agent  $E_p$  first extracts the background information  $\mathcal{B}$ , and then independently generates a decomposition plan composed of a logically ordered sequence of smaller, executable sub-tasks, forming multiple distinct chains-of-thought (CoTs) [22].

**Sub-task Execution** Suppose there are M CoT  $\tau_i$  consisting of  $K_i$  sub-tasks  $\tau_i = \{t_{i,1}, \ldots, t_{i,K_i}\}$ . For each sub-task  $t_{i,k}$ , the plan agent broadcast a  $Beacon\ B_{i,k}$  [21] that describes the sub-task requirements to all available agents  $\mathcal{E} = \{E_1, \ldots, E_N\}$ . Upon receiving this beacon  $B_{i,k}$ , each agent  $E_j$  evaluates its own capability vector and computes a C

$$s_j(t_{i,k}) = \phi(\mathbf{c}_j, \mathbf{r}(t_{i,k})) \in [0,1], \tag{1}$$

where  $\mathbf{c}_j$  is the capability vector of  $E_j$ ,  $\mathbf{r}(t_{i,k})$  is the requirement of  $t_{i,k}$ , and  $\phi(\cdot,\cdot)$  is a similarity function (e.g., cosine similarity). These scores are then returned to the current agent, which compares them and selects the executor as the agent with the highest score. The selected agent  $E_{j^*}$  receives the sub-task  $t_{i,k}$  along with relevant context from previously completed sub-tasks, executes it locally, and sends the output to the next executor.

**Result Voting** The planning agents independently produce distinct CoTs, each representing a complete reasoning trajectory. After all sub-tasks in  $\tau_i$  are executed, the final executer  $E_f$  outputs a final answer  $a_i$  along with an aggregated *confidence score* which is computed as the average of capability match scores along all  $K_i$  sub-tasks. Once all CoTs complete execution, their final results are collected into a candidate set  $\mathcal{A} = \{a_1, \dots, a_M\}$ . The final answer  $\hat{a}$  is determined by a weighted majority vote:

$$\hat{a} = \arg\max_{a \in \mathcal{A}} \sum_{i=1}^{M} \mathbb{I}(a_i = a) \cdot w_i, \tag{2}$$

where  $\mathbb{I}(\cdot)$  is the indicator function. This voting scheme exploits diversity in reasoning paths to mitigate the impact of individual errors or biases in any single CoT.

# 3 Experimental Evaluation

We evaluate Symphony with the five goals: Effectiveness, Scalability across models, Robustness, and Orchestration Overhead.

## 3.1 Experimental Setup

To reduce demonstrate the advantages of our Symphony framework, we registered three agents in the main experiments. Each task required three distinct chains-of-thought (CoT) via different task decompositions, and final answers were obtained through majority voting. The generation window was set to 512 tokens, with the sampling temperature of 0.5 and a nucleus sampling threshold of p=0.9. The decoding process was implemented through the SamplingParams API of the vLLM backend. The runtime environment was composed of three physical servers: Server A equipped with 4 NVIDIA RTX 4090 GPUs (24 GB VRAM), while Server B and C with a single NVIDIA RTX 4090 GPU. All nodes were connected via both public internet and private intranet channels.

Workloads include: (1) **Big-Bench-Hard** [24]: For each of the 23 types of tasks, 6 questions are randomly chosen, and (2) **AMC** [25]: 83 competition-style math questions. For Symphony, each task is decomposed into three CoTs, and aggregated via voting.

# 3.2 Effectiveness

Symphony outperformed LLM-only framework and centralized orchestrations (AutoGen and CrewAI). On BBH, Symphony achieves absolute accuracy gains ranging from 6.5% to 41.6% compared with Direct Solving, and 6.5% to 29.1% over AutoGen, showing that dynamic decentralized orchestration yields significant improvements even over existing multi-agent frameworks.

On AMC, which is a more challenging benchmark with lower absolute scores, Symphony still surpasses all baselines, achieving up to 4.46% higher accuracy than AutoGen and up to 7.41% higher than Direct Solving. These results show that this decentralized orchestration is not only feasible but also more effective than centralized ones.

Table 1: Accuracy (%) of Symphony and simplified variants on BBH and AMC benchmarks.

Benchmark	Model	<b>Direct Solving</b>	AutoGen	CrewAI	Symphony
ввн	Deepseek-7B-instruct	57.24	72.46	66.67	79.71
	Mistral-7B-instruct-v0.3	36.23	48.56	50.72	78.26
	Qwen2.5-7B-instruct	73.19	79.71	77.54	86.23
AMC	Deepseek-7B-instruct	10.84	8.43	7.22	13.25
	Mistral-7B-instruct-v0.3	6.02	1.79	2.40	3.61
	Qwen2.5-7B-instruct	16.87	21.69	18.07	25.30

# 3.3 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 1, Symphony benefits all LLMs across the two benchmarks. Meanwhile, while direct solving demonstrate 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.

Table 2: Effect of CoT Voting: Comparison between Single CoT and 3-CoT Voting

Benchmark	Model	1 CoT	3 CoT Voting	Improvement
BBH	Deepseek-7B	75.36	79.71	+4.25
	Mistral-7B	71.74	78.26	+6.52
	Qwen2.5-7B	81.16	86.23	+5.07
AMC	Deepseek-7B	11.45	13.25	+1.80
	Mistral-7B	2.89	3.61	+0.72
	Qwen2.5-7B	22.67	25.30	+2.63

Table 3: Effect of Beacon Score-based Selection: Random vs. Score Selection

Benchmark	Model	Random	Score	Improvement
BBH	Deepseek-7B	76.09	79.71	+3.62
	Mistral-7B	73.91	78.26	+4.35
	Qwen2.5-7B	82.61	86.23	+3.62
AMC	Deepseek-7B	11.85	13.25	+1.40
	Mistral-7B	3.01	3.61	+0.60
	Qwen2.5-7B	23.12	25.30	+2.18

# 3.4 Robustness

Ablation experiments were conducted to study the impact of Beacon Selection and CoT Voting. As shown in Table 2,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 3 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.

# 3.5 Orchestration Overhead

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

#### 4 Conclusion

We presented **Symphony**, a decentralized multi-agent framework that enables scalable collaboration across heterogeneous edge devices through self-play, sparse parameter sharing, and role-specific cooperation. Experiments show that Symphony achieves competitive performance with significantly lower communication and infrastructure costs, while enhancing accessibility, preserving privacy, and supporting the emergence of decentralized agent economies.

#### References

- [1] Ruike Zhu, Hanwen Zhang, Tianyu Shi, Chi Wang, Tianyi Zhou, and Zengyi Qin. The 4th dimension for scaling model size. *arXiv preprint arXiv:2506.18233*, 2025.
- [2] Miao Zhang, Zhenlong Fang, Tianyi Wang, Shuai Lu, Xueqian Wang, and Tianyu Shi. Ccma: A framework for cascading cooperative multi-agent in autonomous driving merging using large language models. *Expert Systems with Applications*, page 127717, 2025.
- [3] Imad Eddine Toubal, Aditya Avinash, Neil Gordon Alldrin, Jan Dlabal, Wenlei Zhou, Enming Luo, Otilia Stretcu, Hao Xiong, Chun-Ta Lu, Howard Zhou, et al. Modeling collaborator: Enabling subjective vision classification with minimal human effort via llm tool-use. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17553–17563, 2024.
- [4] Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhanxin Hao, Jianxiao Jiang, Jie Cao, Huiqin Liu, Zhiyuan Liu, et al. Simulating classroom education with Ilmempowered agents. *arXiv preprint arXiv:2406.19226*, 2024.
- [5] Charlene H Chu, Simon Donato-Woodger, Shehroz S Khan, Rune Nyrup, Kathleen Leslie, Alexandra Lyn, Tianyu Shi, Andria Bianchi, Samira Abbasgholizadeh Rahimi, and Amanda Grenier. Age-related bias and artificial intelligence: a scoping review. *Humanities and Social Sciences Communications*, 10(1):1–17, 2023.
- [6] Chen Yang, Yangfan He, Aaron Xuxiang Tian, Dong Chen, Jianhui Wang, Tianyu Shi, Arsalan Heydarian, and Pei Liu. Wcdt: World-centric diffusion transformer for traffic scene generation. *arXiv preprint arXiv:2404.02082*, 2024.
- [7] Xinying Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John C. Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*, 2023.
- [8] Yanzheng Wang, Boyue Wang, Tianyu Shi, Jie Fu, Yi Zhou, and Zhizhuo Zhang. Sample-efficient antibody design through protein language model for risk-aware batch bayesian optimization. *bioRxiv*, pages 2023–11, 2023.
- [9] Zhen Wu, Yuzhong Li, Yanjun Deng, Liang Li, Yizhou Song, Wendi Zhu, Zhou Yu, Diyi Yang, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.
- [10] 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. Metagpt: Meta programming for a multi-agent collaborative framework, 2023. https://github.com/geekan/MetaGPT.
- [11] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In *NeurIPS*, 2023.
- [12] J. Wang et al. Voyager: An open-ended embodied agent with large language models. *arXiv* preprint arXiv:2305.16291, 2023.
- [13] CrewAI Contributors. Crewai: Multi-agent orchestration framework. https://github.com/joaomdmoura/crewai, 2024. Accessed: 2025-08-16.
- [14] R. M. Aratchige and W. M. K. S. Ilmini. Llms working in harmony: A survey on the technological aspects of building effective llm-based multi agent systems. arXiv preprint arXiv:2504.01963, 2025.
- [15] Significant Gravitas Torantine. Autogpt: The autonomous gpt-4 experiment. *GitHub repository*, 2023. Accessed: 2025-08-11.
- [16] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

- [17] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv* preprint arXiv:2402.01680, 2024.
- [18] Alexandros G. Dimakis, Soummya Kar, José M. F. Moura, Michael G. Rabbat, and Anna Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98 (11):1847–1864, 2010. doi: 10.1109/JPROC.2010.2052531.
- [19] Gabriele Oliva, Stefano Panzieri, Roberto Setola, and Andrea Gasparri. Gossip algorithm for multi-agent systems via random walk. *Systems & Control Letters*, 128:34–40, 2019. doi: 10.1016/j.sysconle.2019.04.008.
- [20] Ronghua Shi, Yiou Liu, Xinyu Ying, Yang Tan, Yuchun Feng, Lynn Ai, Bill Shi, Xuhui Wang, and Zhuang Liu. Hide-and-shill: A reinforcement learning framework for market manipulation detection in symphony-a decentralized multi-agent system. arXiv preprint arXiv:2507.09179, 2025.
- [21] First Author and Second Author. Lightweight fault detection for distributed systems using periodic beaconing. In *Proceedings of the International Conference on Distributed Computing Systems*, 200X.
- [22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837, 2022.
- [23] Mistral AI. Mistral-7b-instruct-v0.3. Instruction-fine-tuned version of Mistral-7B-v0.3, 2023. See model card on Hugging Face.
- [24] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. arXiv preprint arXiv:2206.04615, 2022.
- [25] Mirac Suzgun, Nathan Scales, Nathanael Schärli, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.
- [26] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv preprint arXiv:2308.08155, Aug 2023. v2 updated October 3, 2023.
- [27] U.S. Congress. Health insurance portability and accountability act of 1996 (hipaa). https://www.hhs.gov/hipaa/, 1996. Public Law 104-191, U.S. Statutes at Large.
- [28] European Parliament and Council of the European Union. General data protection regulation (gdpr). https://eur-lex.europa.eu/eli/reg/2016/679/oj, 2016. Regulation (EU) 2016/679.

# A System Component

**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 [23]) 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 post. 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.

# **B** 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.

## **B.0.1** 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 multi-choice question of daily life issue, requiring context understanding and logistic capability.

#### **B.0.2** Background Extraction and Task Decomposition

In the experiment, 3 CoT were required, so 3 agents with planning capability were chosen from all 8 agents based on capability match and response speed. As shown in Figure 2, upon receiving the original task description, the 3 plan agents extract the background information, and then decompose the task into Chain-of-Thought independently, ensuring diversity of solutions.

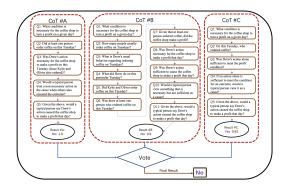


Figure 2: Illustration of the **Symphony** pipeline on a BBH case. Three independent planning agents generate different CoTs to enhance diversity of solutions.

#### **B.0.3** Sequential Cooperation in Execution

Unlike isolated single-agent learning, **Symphony** generates collaborative intelligence through sequential cooperation across agents. When assigned a sub-task, the executor also receives the background information and results of previous sub-tasks, ensuring continuity of reasoning and alignment of thoughts across agents.

For instance, in CoT A, the second sub-task (*Did at least one person order coffee on this Tuesday?*) is executed by a logic-specialized agent. Its output "Yes", along with the result of sub-task 1, is then passed to the following executors in structured context. Based on the previous results, the third sub-task (*Was Drew's action necessary for the coffee shop to make a profit on this Tuesday (since Kylie and Oliver also ordered)?*) was answered. This chaining process guarantees that every answer is not isolated, but co-constructed by multiple agents.

## **B.0.4** Result Aggregation

The three CoTs produce final answers with a confidential score:

```
CoT A \rightarrow No (1.0)
CoT B \rightarrow No (0.9)
CoT C \rightarrow Yes (0.92)
```

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.

# C 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.

# **D** Deployment and Societal Implications

**Symphony** advances existing multi-agent LLM frameworks such as AutoGen [26] and MetaGPT [10] 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 [27] or GDPR [28], 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  $x^4 + \frac{1}{x^2}$  in terms of  $x^2 + \frac{1}{x^2}$ ? Answer:  $\frac{1}{x^2} - 2$ . Based on the background information, solve the sub-task: "Q2: Given that  $23 = x^4 + \frac{1}{x^2}$ , what is the value of  $x^2 + \frac{1}{x^2}$ ?". Provide the final answer formatted as  $\frac{1}{x^2}$ . 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?
Output:
  "original_question": "One root of the equation 5x^2+kx=4 is 2. What is
    the other?",
  "subtasks": [
    "Q1: What is the equation rewritten in standard quadratic form?",
    "Q2: What is the product of the roots of this quadratic equation?",
    "Q3: Given one root is 2, what is the other root?"
 1
}
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:
```