SYMPHONY: Synergistic Multi-agent Planning with Heterogeneous Language Model Assembly

Wei Zhu **Zhiwen Tang**[*] **Kun Yue**
School of Information Science and Engineering, Yunnan University, Kunming, China
Yunnan Key Laboratory of Intelligent Systems and Computing, Kunming, China
`zhuwei@stu.ynu.edu.cn, {zhiwen.tang, kyue}@ynu.edu.cn`

Abstract

Recent advancements have increasingly focused on leveraging large language models (LLMs) to construct autonomous agents for complex problem-solving tasks. However, existing approaches predominantly employ a single-agent framework to generate search branches and estimate rewards during Monte Carlo Tree Search (MCTS) planning. This single-agent paradigm inherently limits exploration capabilities, often resulting in insufficient diversity among generated branches and suboptimal planning performance. To overcome these limitations, we propose **SY**nergistic **M**ulti-agent **P**lanning with **H**eter**O**geneous la**N**gauge model assembl**Y** (**SYMPHONY** [2]), a novel multi-agent planning framework that integrates a pool of heterogeneous language model-based agents. By leveraging diverse reasoning patterns across agents, SYMPHONY enhances rollout diversity and facilitates more effective exploration. Empirical results across multiple benchmark tasks show that SYMPHONY achieves strong performance even when instantiated with open-source LLMs deployable on consumer-grade hardware. When enhanced with cloud-based LLMs accessible via API, SYMPHONY demonstrates further improvements, outperforming existing state-of-the-art baselines and underscoring the effectiveness of heterogeneous multi-agent coordination in planning tasks.

1 Introduction

The advent of large language models (LLMs) has significantly advanced the development of autonomous agents capable of performing complex tasks across various domains, including question answering, code generation, and web navigation. These LLM-based agents leverage the extensive knowledge and reasoning capabilities inherent in LLMs to make decisions and plan actions. A prevalent approach in this context is the integration of Monte Carlo Tree Search (MCTS) [10] with LLMs, wherein the LLM guides the exploration of potential action sequences to achieve specific goals [14, 46, 28, 12]. This combination has shown promise in enhancing the decision-making processes of autonomous agents.

Despite the recent progress in integrating LLMs with planning algorithms, existing methods [34, 33, 43, 29, 42, 14] predominantly adopt a single-model paradigm in which one LLM is queried multiple times with identical or slightly perturbed prompts to simulate diverse action branches during MCTS. The underlying assumption is that the stochasticity or sampling variance of the model is sufficient to generate rollouts that explore a wide range of potential solutions. However, in practice, this approach suffers from a critical limitation: the outputs tend to exhibit high similarity across calls, often reflecting the same dominant reasoning pattern learned by the model [14, 46, 12]. As a result, the generated rollouts lack meaningful diversity, leading to narrow and redundant search trajectories.

[*]Corresponding author
[2]Code is available at `https://github.com/ZHUWEI-hub/SYMPHONY`

This deficiency severely constrains the agent's exploration capability within the solution space. When the search tree is populated with highly similar branches, the planner becomes susceptible to local optima, and its ability to discover novel or unexpected solutions is greatly diminished. In particularly challenging tasks that require compositional reasoning or multi-step tool use, the agent may fail to identify the correct solution path altogether. Even in cases where the solution is eventually found, the process may involve excessive sampling and token consumption, incurring significant computational overhead. These inefficiencies highlight a fundamental mismatch between the need for diverse exploration in planning and the limited variability achievable by repeatedly sampling from a single, monolithic LLM.

To address the above limitations,we propose **SY**nergistic **M**ulti-agent **P**lanning with **H**eterogene**O**us La**N**guage Model Assembl**Y** (**SYMPHONY**). The framework integrates multiple language models into a unified planning system that enhances multi-step reasoning through diversity-aware search, adaptive coordination, and reflective adaptation.

A central innovation of SYMPHONY is its heterogeneous agent pool, composed of LLMs with diverse pretraining sources and reasoning styles. Instead of relying on a single agent, SYMPHONY assigns different agents to generate candidate actions at each search node, thereby introducing structural diversity into the search tree. This diversity increases the likelihood of generating complementary reasoning paths, reduces model-specific biases, and improves performance on complex, multi-hop tasks. Empirical results show that expanding model diversity leads to more unique branches per node and consistent gains in task accuracy.

In addition to model heterogeneity, SYMPHONY incorporates several complementary components that further enhance planning performance. A UCB-based scheduling strategy dynamically allocates agents based on historical effectiveness, improving coordination across agents. An entropy-modulated confidence scoring (EMCS) mechanism calibrates value estimates using agent-level uncertainty, yielding more stable evaluations. Finally, a pool-wise memory sharing mechanism enables agents to learn from past failures through natural language reflections, which are shared across the agent pool and incorporated into future prompts. These components together support efficient, adaptive, and robust search behavior.

We evaluate SYMPHONY across three distinct environments that represent key capabilities of LLM-based agents: multi-hop reasoning (HotpotQA), sequential decision making (WebShop), and code generation (MBPP). Experimental results show that SYMPHONY consistently outperforms strong baselines. In addition to improved performance, SYMPHONY achieves higher planning efficiency, requiring fewer MCTS node expansions to reach correct solutions. Notably, the framework delivers competitive or superior results even when built upon cost-effective models, demonstrating practical value without relying on high-cost large-scale deployments.

2 Related Work

2.1 LLM-based Planning and Reasoning

Early work on LLM-based reasoning focused on improving consistency and correctness through guided inference. Brown et al. [5] introduced in-context learning with exemplars, while the Chain-of-Thought paradigm [34, 19, 24] encouraged models to generate step-by-step rationales during prediction. Later studies proposed more structured prompting techniques, such as meta-prompting [45] and meta-constraint-guided inference [30], to scaffold the reasoning process with predefined formats or global constraints.

To enhance reasoning adaptability, several methods incorporate dynamic feedback. Yao et al. [43] interleaves environment interactions with reasoning steps, Shinn et al. [29] enables self-correction through natural language reflections, and code-based approaches Qiu et al. [27], Chen et al. [9] iteratively refine outputs based on execution results. Xu et al. [37] further improves performance by prompting LLMs to rearticulate and revise their own reasoning chains.

As tasks grew more complex, researchers began to move beyond linear inference and explore tree-structured reasoning. Tang et al. [31] and Zhang et al. [44] introduced early mechanisms for maintaining and refining multiple hypotheses in dialogue and QA tasks. Wang et al. [32] aggregates diverse reasoning paths through sampling and majority voting, while Pan et al. [25] dynamically adjusts decoding strategy between heuristic and deliberative modes.

A more principled formulation of structured search appears in Tree of Thoughts [42], which organizes reasoning steps into a decision tree with intermediate backtracking. Building on this, Monte Carlo Tree Search (MCTS) has been applied to guide reasoning more systematically: Hao et al. [14] treat the LLM as a world model within a reward-driven search process, and Zhou et al. [46] further integrate reasoning, planning, and reflection within an MCTS-based framework using learned value estimates and external feedback. Shi et al. [28] employ memory-augmented single-agent MCTS to enhance decision-making in text-based games.

While prior work focuses on structured reasoning with a single model, our approach introduces model-level heterogeneity to enhance diversity and robustness in planning.

2.2 Multi-Agent Collaboration with Language Models

Multi-agent frameworks leverage multiple LLMs or specialized modules to improve reasoning diversity, adaptability, and robustness. Early approaches adopt static task division, assigning agents predefined roles and communication protocols. For instance, ChatDev [26] simulates software development by dividing planning, coding, and testing among fixed-role agents, while MetaGPT [15] enforces similar pipelines using hand-crafted coordination logic. AutoAgents [7] automates agent instantiation but still operates under rigid, rule-based interaction patterns. Although effective in structured environments, these systems struggle with dynamic tasks due to their limited flexibility.

More recent work shifts toward dynamic coordination, enabling emergent collaboration and context-aware adaptation. AgentVerse [8] adopts a blackboard architecture where agents communicate freely through shared language-based memory. CAMEL [20] introduces turn-based agent dialogue for zero-shot task-solving, while AutoGen [35] allows agents to negotiate roles and delegate subtasks on the fly. In complex QA settings, WebGPT [23] decomposes queries into search, summarization, and synthesis subtasks, and MAd [21] employs adversarial debate between LLMs to expose reasoning flaws. MASTER [12] integrates multi-agent behavior into MCTS by adapting the UCT formula using reward signals. AgentCoder [16] further demonstrates the utility of functional specialization in code generation, coordinating programmer, test designer, and test executor agents within a feedback loop to ensure correctness and completeness.

Unlike existing frameworks that rely on uniform agents and costly coordination, our method enables lightweight, heterogeneous collaboration through principled search and memory sharing.

3 SYMPHONY

3.1 Background and Overview

A Markov Decision Process (MDP) [4] provides a principled framework for modeling sequential decision-making, defined by the tuple $(S, A, \mathcal{T}, R, \gamma)$, where S is the state space, A is the action space, $\mathcal{T} : S \times A \to \mathcal{P}(S)$ defines the transition dynamics, $R : S \times A \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. At each timestep, the agent observes a state $s \in S$, selects an action $a \in A$, transitions to a new state $s' \sim P(\cdot \mid s, a)$, and receives a reward $R(s, a)$. The objective is to learn a policy that maximizes the expected cumulative discounted return.

LLMs can be naturally integrated into this framework to support high-level reasoning and decision-making. Specifically, an LLM can serve as a *policy* by generating actions conditioned on language-based state representations, as a *value function* by estimating expected returns from textual trajectories, or as a *world model* by predicting future states and rewards through learned knowledge. Unlike traditional reinforcement learning agents that rely on explicit environment modeling and manually designed reward signals, LLM-based agents leverage pretraining on large corpora to internalize commonsense, domain knowledge, and structured reasoning. This allows them to operate effectively in complex, open-ended environments with minimal task-specific engineering.

Monte Carlo Tree Search (MCTS) [10] is a sample-based planning algorithm that incrementally builds a search tree by balancing exploration and exploitation. It has been widely used in sequential decision-making problems and is well-suited for integration with LLM-based agents, as it allows structured reasoning guided by model-generated priors.

Formally, given the MDP setup, MCTS constructs a partial search tree rooted at the initial state s_0, iteratively performing four steps: *selection*, which traverses the tree using an upper confidence bound to choose promising actions; *expansion*, which adds new child nodes for unexplored actions; *simulation* (or rollout), which estimates future rewards using a policy; and *backpropagation*, which updates statistics along the visited path. A detailed description of MCTS can be found in Appendix C.

In this work, we adapt MCTS by incorporating LLMs to guide both the selection and rollout phases, replacing uniform or heuristic strategies with model-informed priors that focus exploration on semantically meaningful regions. Building on this foundation, we introduce **SYMPHONY**, a synergistic multi-agent planning framework designed to enhance both the efficiency and robustness of LLM-based decision-making. SYMPHONY extends classical MCTS through several key innovations: a heterogeneous ensemble of LLM agents with diverse inductive priors, a UCB-driven adaptive agent scheduling strategy, a pool-wise memory sharing protocol enabling decentralized reflective adaptation, and an entropy-aware utility modulation mechanism for confidence-calibrated evaluation. These components collectively promote diverse trajectory generation, context-aware coordination, coherent information propagation and reliable value estimation. The theoretical analysis and complete pseudocode of SYMPHONY can be found in Appendix A.

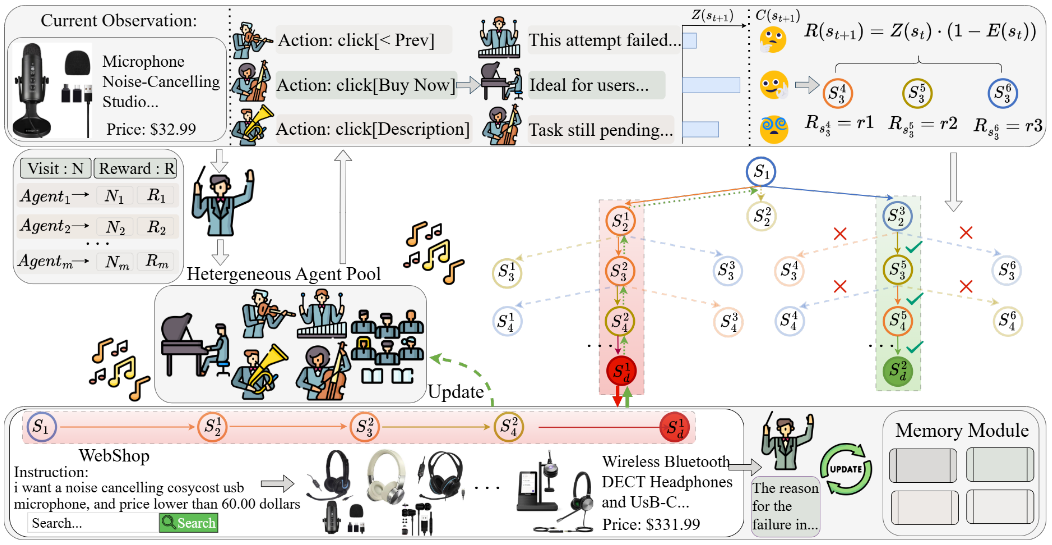

Figure 1: SYMPHONY System Overview.

3.2 Heterogeneous Agent Pool

The heterogeneous agent pool in SYMPHONY is designed to enhance rollout diversity by incorporating multiple language models with varied inductive biases and reasoning behaviors. Unlike traditional MCTS approaches that rely on repeated queries to a single language model, SYMPHONY maintains a collection of distinct language models, each serving as an independent agent that contribute complementary perspectives during search. Formally, the agent pool is represented as $\mathcal{M}^{(k)} = \left\{M_1^{(k)}, \cdots, M_n^{(k)}\right\}$, where M_i is the i^{th} agent based on a language model after the k^{th} memory update.

These agents may be instantiated from either open-source models that are deployable on consumer-grade hardware, or large-scale cloud-based models accessible only via remote API. Different agents exhibit complementary strengths in reasoning depth, factual precision, abstraction ability, and stylistic preferences, which collectively enhance the system's capacity to explore diverse trajectories in the search space.

SYMPHONY employs a uniform input-output interface for agents pool. More specifically, the input to the agent pool at the t^{th} step is $P_\phi(s_t, h_{t-1})$, where $\phi \in \{\text{expansion}, \text{evaluation}, \text{reflection}\}$ is the function indicator of language models , P_ϕ is the corresponding prompt template, and h_t is the interaction history $h_{t-1} = (s_0, a_0, \cdots, s_{t-1}, a_{t-1})$. This design choice facilitates modularity. New models can be added or removed without altering the core planning algorithm. It ensures compatibility with future advances in LLMs and facilitates efficient reuse of available computational resources under different deployment settings. Prompts for each stage can be found in case studies (Appendix L).

3.3 Agent Scheduling

To operationalize the functional heterogeneity of the agent pool, SYMPHONY implements an adaptive dispatch mechanism grounded in the Upper Confidence Bound (UCB) principle, formulating agent selection at each MCTS rollout step as a structured multi-armed bandit problem. Rather than relying on static sampling heuristics or fixed priority weights, the framework dynamically calibrates agent choice based on performance statistics, enabling context-sensitive allocation of reasoning capacity.

Formally, for each agent $M_i^{(k)} \in \mathcal{M}^{(k)}$, the scheduler maintains a cumulative utility estimate $\bar{Q}(M_i^{(k)})$ reflecting empirical rollout effectiveness, Let $S_{M_i^{(k)}}$ denotes the set of nodes generated by agent $M_i^{(k)}$, $S_{M_i^{(k)}} = \{s_{t+1} \sim \mathcal{T}(s_t, M_i^{(k)}(s_t, h_{t-1}))\}$. We record the total invocation count for agent $M_i^{(k)}$ as $N_i^{(k)}$, Similarly, the cumulative average score for agent $M_i^{(k)}$ is defined as $\bar{Q}(M_i^{(k)}) = \sum_{s_t \in S_{M_i^{(k)}}} R(s_t)/|S_{M_i^{(k)}}|$. The selection priority at a search node s_t is governed by the canonical UCB expression:

$$\mathrm{UCB}(M_i^{(k)}) = \bar{Q}(M_i^{(k)}) + \alpha \cdot \sqrt{\frac{\ln N_{total}^{\mathcal{M}^{(k)}}}{N(M_i^{(k)}) + 1}} \tag{1}$$

Here α denotes an exploration–exploitation trade-off hyperparameter, $N_{total}^{\mathcal{M}^{(k)}} = \sum_{j=1}^{n} N(M_j^{(k)})$ represents the total number of scheduling decisions made thus far, and the denominator smoothing term ensures initialization-phase optimism. This formulation favors agents that exhibit either superior historical returns or low invocation frequency, thereby enabling simultaneous exploitation of high-confidence models and exploration of underutilized reasoning modes.

Crucially, this scheduling mechanism is not an isolated module but is tightly interwoven with the recursive structure of MCTS, encompassing action generation and reflective evaluation. Following UCT-guided node traversal, a frontier node s_t is expanded by dispatching an $M_i^{(k)} \in \mathcal{M}^{(k)}$ selected via Equation 1, which is queried using the expansion prompt:

$$a_t = M_i^{(k)}(P_{\text{expansion}}(s_t, h_{t-1})) \tag{2}$$

where h_{t-1} encodes the accumulated interaction trace. The returned actions populate the search frontier with semantically diverse and structurally varied hypotheses.

The agent scheduling mechanism is also used in creating pool-wise reflection memory and node evaluation with EMCS, which will be detailed in the subsequent subsections.

We further establish, from a theoretical perspective, that sampling agents from the ensemble with non-zero probabilities leads to a strictly lower expected error than deterministically selecting a single agent. The detailed proof is provided in Appendix B.

3.4 Pool-wise Memory Sharing

To support continual adaptation without parameter updates, SYMPHONY introduces a pool-wise memory sharing mechanism based on decentralized reflection with natural language. Rather than relying on explicit retraining, agents update their behavior by integrating peer-generated reflections into prompt-level memory.

When a trajectory terminates unsuccessfully, $\tau_{\text{fail}} = (s_0, a_0, \ldots, s_T)$, a UCB-selected agent $M_i^{(k)}$ generates a structured reflection $\mathcal{R}_i^k$ summarizing the failure. This reflection is broadcast to the entire agent pool and treated as a shared memory block. As reflections accumulate from different agents and episodes, they form a diverse collective memory that enhances generalization and coordination.

To manage memory constraints and maintain efficiency, each agent retains a fixed-size buffer updated via a FIFO policy. Reflections are incorporated through prompt-level memory updates:

$$\mathcal{M}^{(k+1)} = \text{Update}(\mathcal{M}^{(k)}, \mathcal{R}^k), \mathcal{R}^k = M_i^{(k)}(P_{\text{reflection}}(s_t, h_{t-1})) \tag{3}$$

This update mechanism enables behavioral adjustment without modifying model parameters, supporting lightweight and scalable adaptation across heterogeneous agents.

3.5 Entropy-Modulated Node Evaluation

To improve value estimation during search, SYMPHONY introduces an entropy-modulated node evaluation strategy that adjusts utility scores based on agent confidence. Upon expanding a new node s_t, a scheduled agent $M_i^{(k)} \in \mathcal{M}^{(k)}$ performs an internal evaluation, producing a value estimate $Z(s_t) \in [0, 1]$ and a confidence score $C(s_t) \in (0, 1)$:

$$Z(s_t), C(s_t) = M_i(P_{\text{evaluation}}(s_t, h_{t-1})) \tag{4}$$

To integrate these outputs, SYMPHONY employs Entropy-Modulated Confidence Scoring (EMCS), which penalizes uncertain predictions by down-weighting value estimates using the entropy of a Bernoulli distribution. Here, the confidence score $C(s_t)$ is interpreted as the success probability of a Bernoulli variable: the entropy is maximal at $C(s_t) = 0.5$, indicating maximum uncertainty, and approaches zero as $C(s_t) \to 0$ or $C(s_t) \to 1$, reflecting high confidence.

$$R(s_t) = Z(s_t) \cdot (1 - E(s_t)) \tag{5}$$

where $E(s_t) = -C(s_t) \ln C(s_t) - (1 - C(s_t)) \ln(1 - C(s_t))$.

This formulation preserves confident evaluations while suppressing uncertain ones, ensuring that nodes with ambiguous outcomes have reduced influence. Compared to fixed heuristics, EMCS offers uncertainty-aware, real-time modulation with minimal overhead, leading to more stable and reliable planning behavior within the MCTS loop.

4 Experiments

We evaluate our approach across three representative tasks spanning reasoning, decision-making, and code generation. Specifically, we conduct experiments on: (1) multi-hop question answering using HotpotQA [40] to assess reasoning capabilities; (2) goal-directed interaction on WebShop [41] to evaluate decision-making and planning; and (3) code generation on MBPP [3] to test the model's ability to reason and produce executable solutions.

4.1 Experiment Settings

SYMPHONY supports flexible agent composition and is compatible with a range of language models under different computational constraints. We evaluate two deployment configurations: **SYMPHONY-S**, designed for consumer-grade hardware, and **SYMPHONY-L**, which leverages large-scale foundation models via cloud-based APIs.

SYMPHONY-S comprises open-source models that can be executed locally, including Qwen2.5-7B-Instruct-1M [39], Mistral-7B-Instruct-v0.3 [18], and Llama-3.1-8B-Instruct [13]. This configuration supports efficient inference with minimal deployment cost. In contrast, **SYMPHONY-L** comprises high-performance models: GPT-4 [1], Qwen-Max (2024-09-19) [38], and DeepSeek-V3 (2025-03-24) [22], which operate through API endpoints within inference-as-a-service infrastructures.

All experiments are carried out under a unified protocol aligned with previous work [29, 46, 12]. To ensure comparability, we apply consistent prompt formats and fixed hyperparameter settings across both configurations, including decoding temperature, planning depth, rollout budget, and number of demonstrations. To mitigate LLM stochasticity, each experiment is repeated 3 times on the same data set, and the mean accuracy is reported. The detailed hyper-parameter settings are described in Appendix D.

Table 1: HotpotQA.

Method	Exact Match ↑
CoT [34]	0.34
CoT-SC [33]	0.38
ReAct [43]	0.39
Reflexion [29]	0.51
ToT [42]	0.55
RAP [14]	0.60
LATS [46]	0.71
Beam Retrieval [44]	0.73
MASTER [12]	0.76
SYMPHONY-S	**0.59**
SYMPHONY-L	**0.79**

Table 2: WebShop.

Method	Score ↑	SR ↑
IL [41]	0.60	0.29
IL+RL [41]	0.62	0.29
ReAct [43]	0.54	0.32
Reflexion [29]	0.64	0.35
Fine-tuning [11]	0.68	0.45
AgentKit [36]	0.70	–
LATS [46]	0.76	0.38
MASTER [12]	0.80	–
Human Expert [41]	0.82	0.60
SYMPHONY-S	**0.82**	**0.56**
SYMPHONY-L	**0.88**	**0.72**

Table 3: MBPP.

Method	Pass@1 (Python) ↑	Pass@1 (Rust) ↑
GPT-4 [29]	0.800	0.710
GPT-4(CoT) [12]	0.683	–
GPT-4(ReAct) [43]	0.710	–
Reflexion [29]	0.771	0.754
RAP [14]	0.714	–
LATS [46]	0.811	–
MetaGPT [15]	0.877	–
AgentVerse [8]	0.890	–
MASTER [12]	0.910	–
AgentCoder [17]	0.918	–
SYMPHONY-S	**0.927**	**0.946**
SYMPHONY-L	**0.965**	**0.974**

Note: Metrics are normalized to the [0,1] range; A dash (–) marks those not reported in the publication.

4.2 Reasoning:HotpotQA

Setup. HotpotQA [40] is a large-scale benchmark for multi-hop question answering, constructed from Wikipedia and containing approximately 113,000 question–answer pairs. In line with prior work [43, 29, 46, 12], we employ an oracle feedback setting, where the environment immediately indicates whether a selected answer is correct. This setup is designed to isolate and evaluate the agent's decision-making capabilities during interaction, rather than its ability to generate final answers. Evaluation on this dataset is based on the exact match (EM) metric.

We compare SYMPHONY against representative baselines from four categories: (1) *Linear reasoning* methods such as CoT [34] and CoT-SC [33]; (2) *Feedback-driven* approaches including ReAct [43] and Reflexion [29]; (3) *Structured reasoning* methods such as ToT [42], RAP [14], LATS [46], and Beam Retrieval [44]; and (4) the *multi-agent framework* MASTER [12], which builds multi-agent from the same LLM. Baseline results are taken from Gan et al. [12], where GPT-4 is used uniformly across all methods.

Results. SYMPHONY demonstrates strong performance across all baseline categories. The lightweight **SYMPHONY-S** outperforms both linear reasoning and feedback-driven baselines, and performs comparably to structured search methods like RAP. The stronger **SYMPHONY-L** surpasses all structured baselines, including MASTER, achieving state-of-the-art performance on HotpotQA. These improvements reflect SYMPHONY's ability to combine model heterogeneity with coordinated compositional reasoning.

4.3 Sequential Decision Making:WebShop

Setup. WebShop [41] is a simulated e-commerce platform featuring over 1.18 million products and 12,000 natural language queries. Agents must navigate the website using browser-like operations (e.g., search, click, select) to identify items that satisfy user constraints. Performance is measured by average score, which reflects partial attribute satisfaction, and success rate (SR), which reflects full constraint satisfaction.

We compare SYMPHONY against a comprehensive set of baselines reflecting five categories: (1) *Task-native methods* including imitation learning (IL), IL+RL, and Human Expert [41]; (2) *Supervised models* such as a fine-tuned LLM [11]; (3) *Feedback-driven reasoning*, including ReAct, Reflexion; (4) *Structured search* methods including LATS and AgentKit; and (5) the *multi-agent framework*: MASTER [12]. All baselines were reproduced under consistent settings by Gan et al. [12] using GPT-4, ensuring fair comparison.

Results. SYMPHONY outperforms all baseline categories. Compared to task-native and supervised approaches, it achieves higher task completion while requiring no domain-specific training. Against feedback-driven and structured search methods, it exhibits stronger planning efficiency and generalization. Finally, SYMPHONY-L surpasses the multi-agent MASTER, establishing a new performance benchmark. These results underscore SYMPHONY's adaptability across different task-specific execution environments.

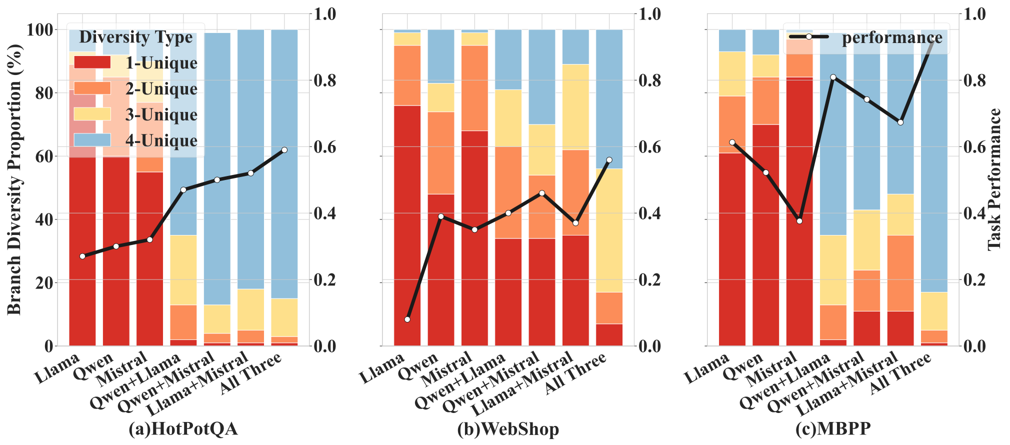

Figure 2: Branch Diversity vs. Task Performance. Bars and left y-axis shows the branch diversity, while lineplot and right y-axis shows the task performance.

4.4 Programming:MBPP

Setup. The Mostly Basic Programming Problems (MBPP) [3] involves multi-step code generation tasks that require condition decomposition, procedural planning, and implementation. Each task provides a description in natural language and a test suite. Success is defined by passing all tests. We follow [29] to evaluate both Python and Rust versions of the datasets using the MultiPL-E compiler suite [6].

Baselines span three major categories: (1) *Single-agent methods*, including GPT-4 and Reflexion [29], representing the performance ceiling of basic prompting and reactive reasoning; (2) *Multi-agent frameworks*, such as MetaGPT [15], AgentVerse [8], and AgentCoder [17], which explore different collaboration strategies; and (3) *Search-based approaches*, including RAP [14], LATS [46], and MASTER [12], which emphasize structured optimization. All baseline results are drawn from or reproduced by Gan et al. [12] under consistent backbone and data settings.

Results. SYMPHONY achieves strong performance across all baseline categories. Compared to single-agent methods, it demonstrates superior reasoning depth and planning efficiency. Against multi-agent frameworks, SYMPHONY provides more effective solution search via the introduction of heterogeneous agent pool. Compared to search-based approaches, it attains state-of-the-art performance in cross-language settings, including Rust, a programming language usually ignored by previous works. These results confirm SYMPHONY's robustness, generality, and computational efficiency in code generation.

4.5 Diversity Analysis

Branch diversity plays a crucial role in effective search. To assess its impact, we evaluate how different agent pool configurations affect both task performance and branch diversity across all three tasks using SYMPHONY-S. The expansion width is fixed at 4, and each node's candidate branches are categorized by output uniqueness: (a) **4-Unique**: all branches distinct, (b) **3-Unique**, (c) **2-Unique**, and (d) **1-Unique**: all branches identical. Higher frequencies of 3-Unique and 4-Unique indicate more diversified and informative exploration.

As shown in Figure 2, increasing agent heterogeneity, from single-agent to pairwise and full-trio configurations (e.g., Qwen+Mistral+Llama), leads to a substantial rise in 4-Unique expansions. On MBPP, for example, this proportion exceeds 80% under the full ensemble, compared to under 20% in the single-agent setting. This increase in structural diversity strongly correlates with improved accuracy, with SYMPHONY outperforming single-agent baselines by over 30% on MBPP and showing similar trends on HotpotQA and WebShop. These findings highlight the critical role of model-level diversity in enhancing search coverage and reasoning robustness.

Table 5: Ablation Study.

Method	HotpotQA(EM)↑	WebShop(SR)↑	MBPP(pass@1)↑
SYMPHONY-S	**0.59**	**0.56**	**0.927**
w/o Agent Scheduling	0.51	0.48	0.906
w/o Memory Sharing	0.45	0.46	0.871
w/o EMCS	0.51	0.49	0.892

We also experimented with alternative diversity-promoting strategies such as adversarial prompting and temperature scaling, but found their effect to be marginal. Detailed comparisons are included in Appendix I.

4.6 Efficiency and Cost Analysis

Table 4: Comparison of the search tree size on HotpotQA.

Method	K	HotpotQA ↑	#Nodes ↓
ToT	10	0.34	33.97
RAP	10	0.44	31.53
LATS	10	0.44	28.42
ToT	50	0.49	84.05
RAP	50	0.54	70.60
LATS	50	0.61	66.65
SYMPHONY-S	**10**	**0.59**	**16.39**
SYMPHONY-L	**10**	**0.79**	**9.47**

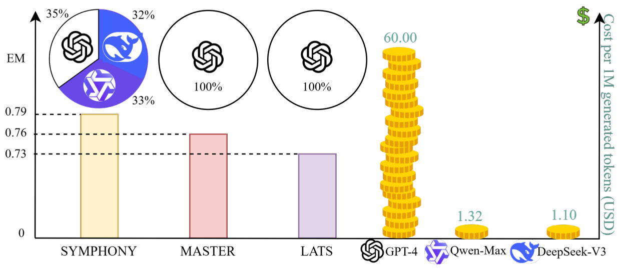

Figure 3: Comparison of model invocation frequency and final performance on HotpotQA.

To evaluate SYMPHONY's practicality, we analyze two key aspects: the size of the search tree and the cost of model inference—both crucial to real-world deployment.

Compared to methods like LATS, which use a large trajectory budget ($K = 50$) and wider expansion ($n = 5$) on HotpotQA and WebShop, SYMPHONY achieves comparable or better results with much smaller values ($K = 10$, $n = 4$), indicating a more compact search process.

We further assess efficiency by measuring average node expansions in MCTS on HotpotQA. As shown in Table 4, SYMPHONY consistently requires fewer expansions and even outperforms LATS with a fraction of its search budget, reflecting strong sample efficiency.

In terms of cost, SYMPHONY-L reduces reliance on expensive models by using a heterogeneous agent pool. As shown in Figure 3, GPT-4 is used in only 40% of calls, yet SYMPHONY-L still outperforms GPT-4-only baselines. Token-level cost details are provided in Appendix E.

Together, these results show that SYMPHONY achieves efficient and cost-effective planning through smaller search trees and more economical model usage.

4.7 Ablation Study and Hyperparameter Tuning

To evaluate the impact of SYMPHONY's core components, we perform a series of ablation studies by selectively disabling key modules, including UCB-based agent scheduling, pool-wise memory sharing, and EMCS scoring. As presented in Table 5, removing any of these components leads to consistent performance degradation across tasks. These results underscore the effectiveness of dynamic agent scheduling, collaborative memory sharing, and uncertainty-aware scoring in enhancing overall system performance.

We further conduct hyperparameter tuning for the UCB exploration coefficient α used in agent scheduling, as well as the MCTS parameters n and K, which jointly determine the search strategy and computational efficiency. Detailed analyses and results are provided in Appendix G and Appendix H. An extended analysis of architectural robustness under varying agent compositions and noise perturbations is also included in Appendix F, offering deeper insights into SYMPHONY's stability and adaptability. Case studies are included in Appendix L.

5 Conclusion and Future Work

We present SYMPHONY, a multi-agent planning framework that combines MCTS with a diverse pool of language models. By leveraging model heterogeneity and incorporating adaptive scheduling, entropy-modulated confidence scoring, and memory sharing, SYMPHONY improves both search diversity and planning effectiveness. Experiments across multiple benchmarks show consistent gains in accuracy and efficiency. Importantly, SYMPHONY performs well even with models that run on consumer-grade hardware, making it a practical and scalable solution.

Future research will focus on extending SYMPHONY to unstructured or noisy environments, reducing reliance on manually tuned hyperparameters, and integrating fairness and robustness considerations into the planning process. We also plan to explore more efficient memory architectures to support scalable, continual adaptation.

Acknowledgements

This work is supported by the Joint Key Project of National Natural Science Foundation of China (U23A20298), Yunnan Fundamental Research Project (202501AT070231), Open Project Program of Yunnan Key Laboratory of Intelligent Systems and Computing (ISC24Y03), and Professional Degree Graduate Practice Innovation Project of Yunnan University (ZC-252514097).

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Anthropic. Introducing claude 3.5 sonnet, 2024. URL `https://www.anthropic.com/news/claude-3-5-sonnet`.

[3] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[4] Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

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

[6] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. Multipl-e: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

[7] Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023.

[8] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2(4):6, 2023.

[9] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[10] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[11] Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned