AgentSwift: Efficient LLM Agent Design via Value-guided Hierarchical Search

Yu Li[1], Lehui Li[3], Zhihao Wu[2], Qingmin Liao[1], Jianye Hao[2], Kun Shao[2], Fengli Xu[1]*

[1]Tsinghua University
[2]Huawei Noah's Ark Lab
[3]Shandong University
liyu24@mails.tsinghua.edu.cn, {fenglixu, liaoqm}@tsinghua.edu.cn, {shaokun2, haojianye}@huawei.com,
202200300096@mail.sdu.edu.cn

Abstract

Large language model (LLM) agents have demonstrated strong capabilities across diverse domains, yet automated agent design remains a significant challenge. Current automated agent design approaches are often constrained by limited search spaces that primarily optimize workflows but fail to integrate crucial human-designed components like memory, planning, and tool use. Furthermore, these methods are hampered by high evaluation costs, as evaluating even a single new agent on a benchmark can require tens of dollars. The difficulty of this exploration is further exacerbated by inefficient search strategies that struggle to navigate the large design space effectively, making the discovery of novel agents a slow and resource-intensive process. To address these challenges, we propose AgentSwift, a novel framework for automated agent design. We formalize a hierarchical search space that jointly models agentic workflow and composable functional components. This structure moves beyond optimizing workflows alone by co-optimizing functional components, which enables the discovery of more complex and effective agent architectures. To make exploration within this expansive space feasible, we mitigate high evaluation costs by training a value model on a high-quality dataset, generated via a novel strategy combining combinatorial coverage and balanced Bayesian sampling for low-cost evaluation. Guiding the entire process is a hierarchical Monte Carlo Tree Search (MCTS) strategy, which is informed by uncertainty to efficiently navigate the search space. Evaluated across a comprehensive set of seven benchmarks spanning embodied, math, web, tool, and game domains, AgentSwift discovers agents that achieve an average performance gain of 8.34% over both existing automated agent search methods and manually designed agents. Moreover, our framework exhibits steeper and more stable search trajectories. By enabling the efficient, automated composition of workflow with functional components, AgentSwift provides a scalable methodology to explore complex agent designs. Our framework serves as a launchpad for researchers to rapidly prototype and discover powerful agent architectures without the impediment of prohibitive evaluation costs.

Code — https://github.com/Ericccc02/AgentSwift

Introduction

The recent rise of large language models (LLMs) (Brown et al. 2020; Radford et al. 2018, 2019) has sparked an explosion of interest in agentic systems. Early forms of such systems, like Chain-of-Thought (Wei et al. 2022), Tree-of-Thought (Yao et al. 2023), Debate (Du et al. 2023) and Self-Refine (Madaan et al. 2023), exemplify the agentic workflow paradigm. These agentic workflows significantly boosted performance on reasoning-intensive tasks, such as mathematical problem (Romera-Paredes et al. 2024) and logical deduction (Shang et al. 2024a). Subsequently, more advanced systems like Voyager (Wang et al. 2024) and AutoAct (Qiao et al. 2024) incorporated structured components such as planning, tool use, and memory. These functional enhancements allowed agents to handle a broader range of tasks—such as web interaction (Nakano et al. 2021), open-ended exploration (Wang et al. 2024), and planning (Xie et al. 2024)—further extending their capability. These developments highlight the importance of agent design, yet building high-performing agents remains manual and labor-intensive, motivating the need for automated agent search.

Despite recent progress, the design of agentic systems remains largely manual and heuristic. Early efforts focused on prompt optimization (Yang et al. 2024b; Khattab et al. 2023) or agent profiling (Yuan et al. 2024), while graph-based approaches (Zhuge et al. 2024; Zhang et al. 2024a) explored communication topology. These methods typically target isolated subsystems such as prompts, roles, or message flow. More recent works like AFlow (Zhang et al. 2024b), ADAS (Hu, Lu, and Clune 2024), and AgentSquare (Shang et al. 2024b) formulate agent design as a search problem over agentic workflows, aiming to discover effective end-to-end configurations. While these advances mark a shift toward agent search, the search of agent remains inefficient.

This inefficiency stems from three major challenges. First, there is an under-exploitation of proven human designs: most existing methods restrict search to specific parts of the agent, such as prompts, profiles, or workflows. As a result, they fail to incorporate or discover critical functional components like planning, tool use, and memory—elements essential for building agents capable of tackling complex, multi-stage tasks. Second, the evaluation cost of agent search remains prohibitively high. According to AgentSquare (Shang et al. 2024b), evaluating a simple CoT

[*]Corresponding author.

agent based on GPT-4o in ALFWorld (Shridhar et al. 2021) requires around $60. In most existing methods, each newly generated agent must be fully evaluated on benchmark tasks to obtain feedback. This results in a large number of unnecessary evaluations for poorly performing agents, leading to wasted computation and prolonged search cycles. Third, in large design spaces, search efficiency suffers. While methods like AFlow and ADAS aim to optimize entire workflows based on performance histories, they often employ search strategies that explore the vast design space inefficiently. Addressing these limitations is crucial to unlocking the full potential of agentic system search.

In this work, we propose a comprehensive framework that addresses these inefficiencies through three key innovations. **First**, we construct a hierarchical search space that includes both the agentic workflow and three functional components—*memory*, *tool use*, and *planning*—that can be modularly attached to the agentic workflow. This search space extends the formulation of AFlow, enabling richer design possibilities beyond fixed workflow structures. This structured design space not only broadens the range of agent designs but also facilitates more meaningful performance modeling, making it well-suited for learning a predictive model. **Second**, we develop a value model that predicts the performance of a candidate agent given its design and a task description. To support effective learning, we construct a high-quality training dataset by combining pairwise covering arrays, which ensure comprehensive coverage of interactions between workflows and components, with balanced Bayesian sampling, which selects agent candidates from both high- and low-performing regions of the search space. This enables the model to generalize across a broad design space and provide accurate, low-cost predictions, effectively guiding the search process while avoiding unnecessary real-world evaluations. **Third**, we design an uncertainty-guided hierarchical expansion strategy based on Monte Carlo Tree Search (MCTS). During the MCTS expansion phase, the agent is iteratively improved through three operations—*recombination*, *mutation*, and *refinement*—applied hierarchically to both the agentic workflow and functional components. In the recombination step, new candidates are sampled from a library of possible workflow structures or component implementations to replace existing ones. Mutation explores new candidates based on existing components and workflows, guided by the performance of previously evaluated agents. Refinement adjusts the agentic workflow and components based on feedback from failure cases. These modifications are guided by the value model's predicted performance, ensuring the search explores promising directions efficiently. By comparing predicted and actual performance, we obtain a natural measure of uncertainty, which is integrated into the MCTS selection strategy to guide which to expand during the search. This integration of predictive modeling and uncertainty allows us to prioritize promising agent candidates, avoid unproductive regions, and conduct more targeted, efficient exploration of the design space. The overview of this work is illustrated in Figure 1.

We validate our framework across seven widely-used benchmark datasets spanning domains such as math, web, tool, and game. Experimental results show that our method achieves an average performance improvement of 8.34% over state-of-the-art baselines. The discovered agents generalize well across LLM backbones, demonstrating strong model-agnosticity. Additionally, our approach exhibits a steeper search trajectory, discovering high-performing agent designs with significantly fewer agent evaluations. Beyond final performance, our value model demonstrates high predictive accuracy and strong transferability to unseen tasks with minimal fine-tuning.

The key contributions of this work are as follows:

- We formalize the agentic system optimization as a hierarchical search over agentic workflow and functional components, establishing a general framework that extends prior approaches.
- We train a value model that predicts agent performance from agentic system and task description, enabling low-cost, model-driven evaluation during the search process.
- We propose an uncertainty-guided hierarchical expansion strategy based on MCTS, incorporating recombination, mutation, and refinement steps over both workflow and components.
- We empirically demonstrate the effectiveness of our method on seven diverse benchmarks, showing consistent improvements over state-of-the-art baselines.

Related work

LLM agent

Recent advances in LLM agents have introduced diverse agentic workflows that support multi-step reasoning via reflection and debate (Wei et al. 2022; Madaan et al. 2023; Du et al. 2023). These workflows are often complemented by functional components that extend agent capability: memory supports long-term coherence and retrieval (Wang et al. 2024; Wen et al. 2024; Park et al. 2023), tool use enable interaction with external APIs (Schick et al. 2023; Qin et al. 2023; Du, Wei, and Zhang 2024), and planning facilitates subgoal decomposition and control (Ge et al. 2024; Wang et al. 2024; Shen et al. 2023). However, most agents are still manually designed for specific tasks, lacking a unified framework that can systematically search and optimize across workflow and component design choices.

Automated agentic workflow design

Early work on automating agentic workflows has largely focused on optimizing specific subsystems such as prompts (Yang et al. 2024b; Fernando et al. 2023), agent profiles (Chen et al. 2023a,b), and communication topologies (Zhuge et al. 2024; Qian et al. 2024; Niu et al. 2025). While these methods improve local components, they do not consider the agentic workflow as a whole. More recent approaches have attempted end-to-end agentic workflow search (Zhang et al. 2024b; Hu, Lu, and Clune 2024; Shang et al. 2024b; Zhang et al. 2025a). Extending this direction, MaAS (Zhang et al. 2025b) shifts from searching for a single optimal workflow to learning a query-conditioned distribution over agentic architectures, enabling

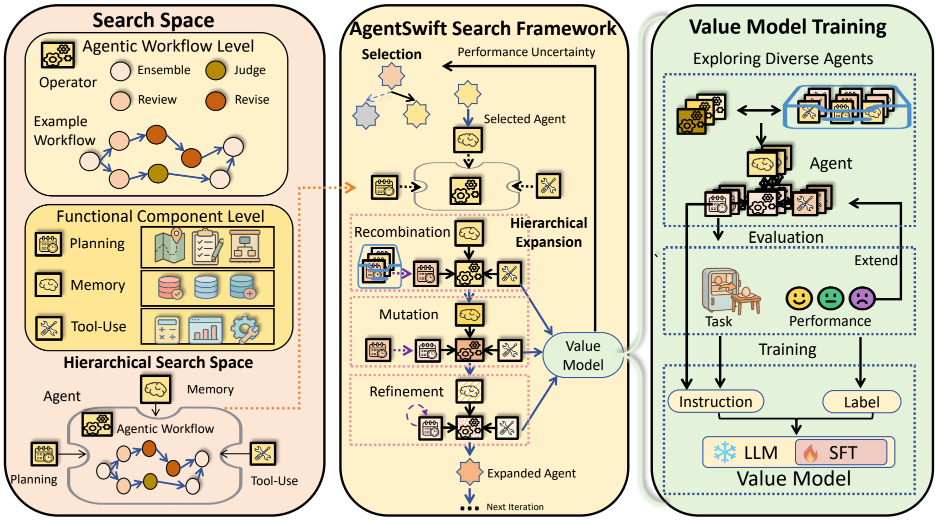

Figure 1: Overview of our framework. The framework integrates (a) hierarchical search space (b) uncertainty-guided MCTS with hierarchical expansion (c) value model training

adaptive deployment. However, these approaches still operate within predefined workflow primitives and often overlook the broader agent design space that includes functional components, limiting adaptability and extensibility.

Performance predictor in AutoML

The development of performance predictors in Neural Architecture Search (NAS) provides a valuable blueprint for progress in agentic system search. Early NAS efforts primarily focused on optimization strategies (Zoph and Le 2016; Real et al. 2019; Maziarz et al. 2018). While effective, these methods required costly evaluations of many candidate architectures. To overcome this limitation, the NAS community gradually introduced performance predictors (Kandasamy et al. 2018; White, Neiswanger, and Savani 2021; Qin et al. 2025). This shift in NAS, from pure search to search guided by learned predictors, has led to significant improvements in efficiency. Notably, the research paradigm in NAS is closely aligned with the goals of agentic system design, as both involve navigating large design spaces under expensive evaluation constraints. Motivated by this connection, we incorporate a value model into agentic system search, enabling performance prediction for candidate agents and guiding the search process more efficiently.

Search space

More recent efforts like AFlow (Zhang et al. 2024b) and ADAS (Hu, Lu, and Clune 2024) treat the agentic workflow as a whole and perform end-to-end search over its structure. Despite their broader scope, these methods do not support flexible integration of functional components such as memory, planning, or tool use. Although AgentSquare (Shang et al. 2024b) introduces these components into its design space, they are combined under a fixed agentic workflow template with rigid interfaces, and its search process remains prompt-centric. In contrast, we propose a hierarchical search space that jointly explores both the agentic workflow and composable functional components.

Agentic workflow

Following AFlow (Zhang et al. 2024b), we define an agentic workflow $\mathbf{W}$ as a series of LLM-invoking nodes connected by edges to specify execution order. Formally, an agentic workflow $\mathbf{W}$ consists of a set of nodes N and a set of edges E, written as $\mathbf{W} = (N, E)$. Each node $N_i \in N$ represents a single execution step and is characterized by the following parameters:

$$N_i = (M_i,\ P_i,\ \tau_i,\ F_i), \tag{1}$$

where $M_i \in \mathcal{M}$ is the language model used at this node, $P_i \in \mathcal{P}$ is the prompt provided to the model, $\tau_i \in \mathcal{T}$ is the decoding temperature, and $F_i \in \mathcal{F}$ specifies the output format. The edges $E \subseteq N \times N$ define the control and data flow between nodes, specifying the execution order.

The agentic workflow search space is defined as:

$$\begin{aligned} \mathcal{S}_{\text{workflow}} = \Big\{ (N, E) \;\Big|\; & N_i = (M_i, P_i, \tau_i, F_i), \\ & M_i \in \mathcal{M},\ P_i \in \mathcal{P},\ \tau_i \in \mathcal{T}, \\ & F_i \in \mathcal{F},\ E \subseteq N \times N \Big\}. \end{aligned} \tag{2}$$

Functional components

In addition to the agentic workflow, we extend the search space to include composable functional components that provide essential agentic capabilities. Specifically, we consider three component types: *memory*, *tool use*, and *planning*. These components are designed to be plug-and-play and can be integrated at specific points within the agentic workflow—for instance, a memory component may interact with a node to retrieve or store context, tool use can augment a node with external API calls, and planning can precede downstream execution steps.

Memory. The memory component allows agents to retrieve and incorporate information. It is defined as $\mathrm{M} = (m,\ \tau,\ d)$, where m is the prompt used to query or update memory, τ is the decoding temperature for memory-related LLM calls, and d denotes the external memory backend, such as a vector database.

Tool Use. The tool use component enables the agent to interact with external APIs or environments. It is defined as

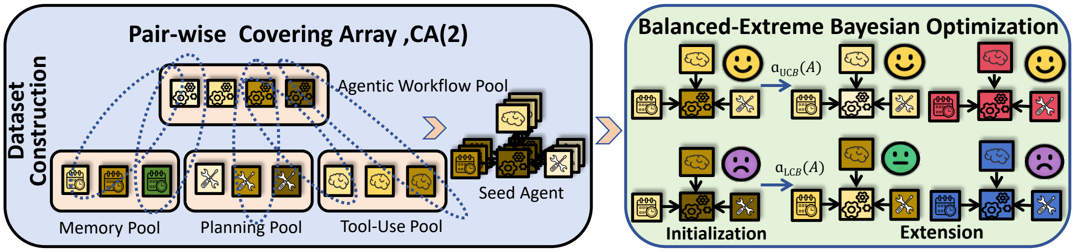

Figure 2: Overview of dataset construction

$\mathbf{T} = (t,\ \tau,\ u)$, where t is the tool invocation prompt, τ is the decoding temperature, and u represents the accessible toolset.

Planning. The planning component supports task decomposition and hierarchical control. It is defined as $\mathbf{P} = (p,\ \tau)$, where p is the prompt for generating subgoals or plans, and τ is the temperature used during plan generation.

The component search spaces are defined as:

$$\begin{aligned} \mathcal{S}_{\text{memory}} &= \{(m,\ \tau,\ d) \mid m \in \mathcal{P},\ \tau \in \mathcal{T},\ d \in \mathcal{D}\}, \\ \mathcal{S}_{\text{tool}} &= \{(t,\ \tau,\ u) \mid t \in \mathcal{P},\ \tau \in \mathcal{T},\ u \in \mathcal{U}\}, \\ \mathcal{S}_{\text{planning}} &= \{(p,\ \tau) \mid p \in \mathcal{P},\ \tau \in \mathcal{T}\}. \end{aligned} \tag{3}$$

Here, $\mathcal{P}$ is the prompt space, $\mathcal{T}$ is the temperature space, $\mathcal{D}$ is the space of memory backends, and $\mathcal{U}$ is the space of available tools.

Hierarchical search space

We define an agent $\mathbf{A}$ as a combination of an agentic workflow and a set of functional components. Formally, the agent is represented as:

$$\mathbf{A} = (\mathbf{W},\ \mathbf{M},\ \mathbf{T},\ \mathbf{P}). \tag{4}$$

The full agent search space is given by:

$$\mathcal{S}_{\text{agent}} = \Big\{\mathbf{W},\ \mathbf{M},\ \mathbf{T},\ \mathbf{P}\ \Big|\ \mathbf{W} \in \mathcal{S}_{\text{workflow}}, \mathbf{M} \in \mathcal{S}_{\text{memory}}, \mathbf{T} \in \mathcal{S}_{\text{tool}}, \mathbf{P} \in \mathcal{S}_{\text{planning}}\Big\}. \tag{5}$$

This formulation defines a hierarchical search space where both the structure of the agentic workflow and the configurations of its functional components are jointly optimized, enabling flexible composition, deeper architectural variations, and the reuse of classical human-designed modules. It subsumes existing methods such as AFlow (Zhang et al. 2024b) and AgentSquare (Shang et al. 2024b) as special cases within a more expressive and extensible design space.

AgentSwift framework

Overview

Given a task description d and a performance evaluation function $\text{Eval}_d(\cdot)$, our objective is to find the agent design $\mathbf{A}^*$ from the joint search space $\mathcal{S}_{\text{agent}}$ that maximizes expected task performance. The optimization problem is defined as:

$$\mathbf{A}^* = \underset{\mathbf{A} \in \mathcal{S}_{\text{agent}}}{\arg\max}\, \text{Eval}_d(\mathbf{A}) = \underset{(\mathbf{W},\mathbf{P},\mathbf{T},\mathbf{M})}{\arg\max}\ \text{Eval}_d(\mathbf{W},\mathbf{P},\mathbf{T},\mathbf{M}). \tag{6}$$

To address the challenges posed by expensive evaluation and inefficient exploration in large agent design spaces, we propose a unified search framework that integrates a predictive *value model* with an *uncertainty-guided hierarchical expansion strategy* based on MCTS. The value model serves as a surrogate evaluator, estimating the performance of candidate agents based on their architecture and task description. This significantly reduces reliance on costly real-world evaluations by allowing the search to be guided by low-cost predictions. To cope with the vast combinatorial search space defined by agentic workflows and functional components, we propose a hierarchical expansion that operates over two levels of abstraction: agentic workflow and functional components. The expansion process includes three operations-*recombination*, *mutation*, and *refinement*—each applied to both levels. Crucially, we incorporate uncertainty estimation from the value model to prioritize exploration of regions where performance predictions are both high and uncertain. Together, the predictive modeling and uncertainty-aware MCTS enable scalable, sample-efficient discovery of high-performing LLM agents. The algorithm is presented in Appendix.

Value model

To efficiently guide the agent search process and reduce the reliance on expensive real-world evaluations, we propose a predictive value model that estimates the performance of a candidate agent $\mathbf{A} = (\mathbf{W}, \mathbf{M}, \mathbf{T}, \mathbf{P})$ on a given task d. The model is trained to approximate the evaluation function $\text{Eval}_d(\cdot)$ via supervised learning:

$$\hat{v} = f_\theta(\mathbf{A}, d), \tag{7}$$

where f_θ denotes the learned value model and $\hat{v}$ is the predicted performance score.

Prior works have leveraged powerful LLM like GPT-4o as in-context predictors for this task, where historical agent performance data is fed directly into the prompt to estimate the success of a new agent (Shang et al. 2024b). However, such in-context evaluation requires repeated invocation of large models during search, resulting in high computational overhead. In contrast, our approach distills this predictive capability into a lightweight, task-generalized value model, enabling fast and scalable inference with significantly lower cost.

Dataset construction. To construct a high-quality training dataset $\mathcal{D} = \{(\mathbf{A}_i, d_i, v_i)\}_{i=1}^{N}$, we employ a two-stage process designed to ensure both broad coverage and discriminative diversity(Figure 2):

1. ***t*-way combinatorial coverage**: We begin by generating an initial dataset using a $t = 2$ covering array to exhaustively sample combinations of pairwise interactions among the four key elements of the agent design: $\mathbf{W}$, $\mathbf{M}$, $\mathbf{T}$, and $\mathbf{P}$. This ensures that all pairwise component

interactions are represented at least once, promoting coverage.

2. **Balanced Bayesian sampling**: We augment the initial dataset using a Balanced-Extreme Bayesian Optimization strategy. We fit a Gaussian Process (GP) surrogate over the discrete agent space, using a Hamming kernel. The posterior mean $\mu(\mathbf{A})$ and standard deviation $\sigma(\mathbf{A})$ are used to define two acquisition functions:
$$\begin{aligned} a_{\text{UCB}}(\mathbf{A}) &= \mu(\mathbf{A}) + \kappa \cdot \sigma(\mathbf{A}), \\ a_{\text{LCB}}(\mathbf{A}) &= -\mu(\mathbf{A}) + \kappa \cdot \sigma(\mathbf{A}). \end{aligned} \tag{8}$$
where κ is an exploration coefficient. In each sampling round, we select a batch of q new agent designs from the candidate pool $\mathcal{S}$:
$$q_{\text{high}} = \left\lceil \frac{q}{2} \right\rceil, \quad q_{\text{low}} = q - q_{\text{high}}, \tag{9}$$
where q_{high} maximizes a_{UCB} to explore high-performing regions and q_{low} maximizes a_{LCB} to explore potentially underperforming yet uncertain configurations. This dual exploration yields a diverse and discriminative dataset. We repeat this process until a total of 220 labeled samples are obtained, which are then randomly split into training, validation, and test sets with a ratio of 8:1:1.

Model architecture and training. We implement the value model using a pre-trained 7B language model augmented with lightweight adapter modules, enabling robust generalization across diverse tasks. The entire model is fine-tuned end-to-end on the constructed dataset using mean squared error (MSE) loss.

Uncertainty-guided MCTS

Initialization. To warm-start the search and improve early-stage efficiency, we initialize a global experience pool $\mathbb{E} = \{(\mathbf{W}, \mathbf{M}, \mathbf{T}, \mathbf{P}, v)\}$, where v is the measured performance of an agent. This pool is seeded using well-designed baseline agents adapted from the AgentSquare (Shang et al. 2024b) codebase. The pools $\{\mathbb{W}, \mathbb{M}, \mathbb{T}, \mathbb{P}\}$ are extracted from these baselines and standardized.

Selection. We adopt a soft mixed probability selection strategy that integrates observed performance and model uncertainty, encouraging balanced exploration and exploitation. Given a set of n candidate agents, the selection probability for agent i is computed as:
$$E(s_j, u_j) = \alpha \cdot \left((1-\beta) \cdot s_j + \beta \cdot u_j - s_{\max}\right), \tag{10}$$
$$P_{\text{mixed}}(i) = \lambda \cdot \frac{1}{n} + (1-\lambda) \cdot \frac{\exp\left(E(s_i, u_i)\right)}{\sum_{j=1}^{n} \exp\left(E(s_j, u_j)\right)}. \tag{11}$$
where s_i denotes the actual task performance of agent i, u_i is the uncertainty, and $s_{\max}$ is the maximum composite score across all candidates. λ, α, and β control the trade-off between uniform exploration, sensitivity to performance differences, and the contribution of uncertainty, respectively. This formulation extends AFlow's (Zhang et al. 2024b) approach by incorporating epistemic uncertainty, thus encouraging the search to explore candidates that are either high-performing or insufficiently evaluated.

Expansion. Starting from the parent agent returned by selection, we perform a top-down *hierarchical expansion* composed of three operations—*recombination*, *mutation*, and *refinement*. These operations utilize task-agnostic prompts, where only the task description varies for new applications. Each operation generates a small batch of new agents and the value model scores every candidate and the best one advances to the next operation.

1. **Recombination** An LLM proposer π_θ (adapted from AgentSquare) replaces one subsystem—*agentic workflow*, *planning*, *tool use*, or *memory*—with an alternative sampled from the corresponding pool. Given a current agent $(\mathbf{W}_0, \mathbf{M}_0, \mathbf{T}_0, \mathbf{P}_0)$ and experience pool $\mathbb{E}$, π_θ produces N candidate agents. For example, $(\mathbf{W}_0, \mathbf{M}_0, \mathbf{T}', \mathbf{P}_0)$ denotes a recombination where the tool use component is replaced with a new $\mathbf{T}' \in \mathbb{T}$. The value model ranks all candidates, and the top one is passed to the next phase.
2. **Mutation** An LLM programmer π_ξ generates a new implementation of the selected subsystem by leveraging task description, existing subsystems, and prior agent performance from $\mathbb{E}$. This yields N mutated agents; for instance, $(\mathbf{W}_0, \mathbf{M}_0, \mathbf{T}_0, \mathbf{P}^*)$ represents a mutated variant where a new planning $\mathbf{P}^*$ is synthesized. The value model ranks all candidates, and the top one is passed to next phase. Newly generated subsystems are appended to the global pools so future searches can reuse them.
3. **Refinement** An LLM refiner π_ϕ applies fine-grained adjustments to the selected agent by modifying a single subsystem in light of failure cases. These refinements include prompt edits, temperature nudges or control-flow modifications. For example, $(\mathbf{W}^{`}, \mathbf{M}_0, \mathbf{T}_0, \mathbf{P}_0)$ denotes a refined variant with an updated workflow $\mathbf{W}^{`}$. Among the refined candidates, the one with the highest predicted performance is inserted into the MCTS tree.

This three-step pipeline simultaneously broadens exploration (via recombination), unlocks novel behaviour (via mutation), and polishes promising designs (via refinement)

Evaluation. Inspired by classic works on probabilistic forecasting and sequential decision-making (Brier 1950; Auer, Cesa-Bianchi, and Fischer 2002; Kocsis and Szepesvári 2006), the child agent is evaluated on the target task to obtain its actual performance score s_{real}. To quantify the epistemic uncertainty of the value model's prediction, we define the uncertainty as the absolute deviation between the predicted score $\hat{s}$ and the true performance:
$$u = \left|s_{\text{real}} - \hat{s}\right|. \tag{12}$$
This uncertainty metric, rooted in the principles of forecast calibration (Brier 1950), enables the search algorithm to balance exploitation of high-performing configurations with exploration of under-evaluated regions.

Backpropagation. After evaluation, the node records its actual score s_{real} together with the uncertainty u. These statistics are then propagated upward, where each ancestor node increments its visit count. Finally, node $\left(\mathbf{W}, \mathbf{M}, \mathbf{T}, \mathbf{P}, s_{\text{real}}\right)$ is attached to the global experience pool $\mathbb{E}$, enlarging the candidate set for subsequent iterations.

Table 1: Performance comparison of our method against hand-crafted agents and agent search methods across seven diverse benchmarks using GPT-4o-mini. The results are averaged over three independent runs. Our method consistently achieves the best performance across all benchmarks.

Baseline Type	**Method**	**Embodied**		**Math**	**Web**	**Tool**		**Game**
		Alfworld	**SciWorld**	**MATH**	**WebShop**	**M3Tool**	**Travel**	**PDDL**
Hand-crafted Agents	COT	0.512±0.009	0.398±0.005	0.532±0.004	0.490±0.011	0.427±0.008	0.433±0.003	0.427±0.011
	CoTSC	0.545±0.006	0.412±0.004	0.543±0.002	0.488±0.006	0.451±0.012	0.410±0.001	0.410±0.009
	TOT	0.530±0.008	0.384±0.004	0.547±0.005	0.462±0.009	0.463±0.014	0.407±0.007	0.433±0.007
	FoA	0.587±0.005	0.427±0.008	0.556±0.003	0.509±0.012	0.488±0.009	0.474±0.006	0.472±0.007
	TP	0.373±0.010	0.195±0.009	0.543±0.001	0.343±0.013	0.402±0.007	0.387±0.008	0.440±0.005
	SelfRefine	0.575±0.007	0.375±0.006	0.551±0.004	0.425±0.010	0.463±0.010	0.047±0.015	0.412±0.008
	Dilu	0.451±0.009	0.358±0.008	0.545±0.003	0.492±0.008	0.476±0.011	0.360±0.009	0.417±0.006
	Voyager	0.336±0.011	0.389±0.005	0.517±0.006	0.423±0.012	0.317±0.014	0.517±0.004	0.337±0.010
	DEPS	0.493±0.007	0.435±0.007	0.513±0.005	0.308±0.015	0.329±0.013	0.523±0.003	0.463±0.007
	Stepback	0.470±0.008	0.314±0.009	0.530±0.002	0.459±0.011	0.488±0.009	0.033±0.012	0.403±0.009
Agent Search	AgentSquare	0.701±0.07	0.475±0.005	0.556±0.004	0.520±0.009	0.561±0.010	0.553±0.004	0.577±0.008
	AFlow	0.619±0.006	0.452±0.007	0.562±0.003	0.497±0.011	0.524±0.012	0.497±0.006	0.528±0.008
	ADAS	0.567±0.009	0.463±0.006	0.543±0.005	0.436±0.013	0.500±0.011	0.453±0.007	0.509±0.009
	MaAS	0.612±0.007	0.437±0.008	0.597±0.001	0.485±0.010	0.537±0.010	0.403±0.005	0.564±0.007
	AgentSwift	**0.806±0.007**	**0.509±0.006**	**0.628±0.000**	**0.562±0.010**	**0.634±0.013**	**0.573±0.002**	**0.614±0.008**

Experiments

Experimental setup

Task setup. We evaluate our framework on seven benchmark spanning five representative task domains commonly used in LLM evaluation (Ma et al. 2024; Xi et al. 2024). More details are presented in Appendix.

Baselines. We compare our framework against two categories of baselines including manually designed agent and automated agent search methods. More details are presented in Appendix.

Implementation details. We conduct experiments using closed-source LLMs (`gpt-4o (OpenAI 2024)`, `gpt-4o-mini (Achiam et al. 2023)`) and an open-source one (`DeepSeek-v3 (Liu et al. 2024)`). For the value model, we adopt two backbone LLMs: `Mistral-7B-v0.3 (Jiang et al. 2024)` and `Qwen2.5-7B (Yang et al. 2024a)`. To ensure fair comparison across agent search methods, the evaluation budget is capped at 60 agents per method, by which point all baselines converge. The value model is trained on a server equipped with 3 A100 GPUs.

Experimental results

Main results.

- **Our method consistently discovers the best-performing agents.** Across all tasks, our framework reliably identifies agent designs that outperform both manually constructed baselines and agents discovered by existing search methods. As shown in Table 1, our best-found agents achieve substantial improvements over the strongest competing methods. These consistent gains highlight the advantage of searching over both agentic workflows and composable functional components in a unified hierarchical design space. Our formulation enables richer architectural compositions beyond fixed or manually selected modules, yielding more generalizable and effective agent behaviors.

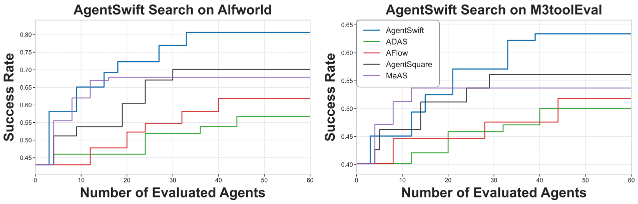

Figure 3: AgentSwift search trajectory on Alfworld and M3ToolEval.

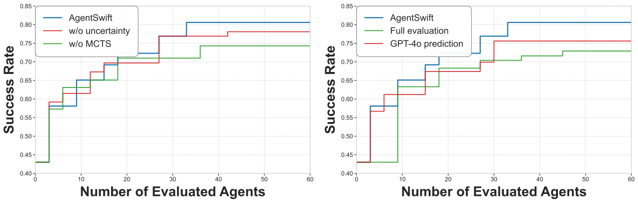

Figure 4: Left: search trajectory of different search strategies on Alfworld: AgentSwift, w/o uncertainty, and w/o MCTS. Right: search trajectory of different evaluate method on Alfworld: AgentSwift, gpt-4o prediction, and full evaluation.

- **Our method enables steeper and more efficient search trajectories.** Figures 3 present the search trajectories of our method and strong baselines. Our method demonstrates a noticeably steeper and more stable performance curve across tasks, indicating faster discovery of high-performing agent. In contrast, methods such as AFlow and ADAS either stagnate due to limited agentic workflow variation or require significantly more iterations and time to escape local optima. These results validate the synergistic effectiveness of predictive modeling and structured, uncertainty-aware search in accelerating agent discovery. For brevity, search trajectories for other tasks, alongside detailed analyses of wall-clock time and the cost-performance Pareto front, are provided in the Appendix.

Table 2: Performance comparison of different surrogate models on all benchmarks. Our method consistently achieves the best performance across all metrics.

Method	MSE	MAE	R²	Spearman
AgentSwift$_{\text{mistral}}$	**0.0060**	**0.0530**	**0.8068**	**0.9026**
AgentSwift$_{\text{qwen}}$	**0.0054**	**0.0547**	**0.8275**	**0.8987**
vanilla	0.1572	0.3593	-4.0590	0.2467
gpt-4o few shot	0.0162	0.0893	0.4793	0.7654
gpt-4o zero shot	0.0675	0.2067	-1.1708	0.0563
gpt-4o-mini few shot	0.0307	0.1179	0.0114	0.5410
gpt-4o-mini zero shot	0.0820	0.2370	-1.6403	-0.0774

Analysis of value model. We evaluate the effectiveness of our value model by comparing it with several baseline predictors trained on the same dataset, including a vanilla supervised model and in-context learning methods using GPT-4o and GPT-4o-mini in both zero-shot and few-shot settings. As shown in Table 2, our approach achieves the best performance across all metrics—MSE, MAE, R^2, and Spearman correlation—demonstrating superior accuracy in both absolute prediction and ranking quality.

Analysis of search strategy. We analyze the effect of our search design by comparing variants of our method on the AlfWorld. As shown in Figure 4 (Left), removing MCTS or uncertainty guidance significantly flattens the search trajectory. Without MCTS, the algorithm lacks hierarchical exploration and becomes overly local, while without uncertainty, the search tends to exploit familiar regions and misses promising but uncertain candidates. On the right, we compare different evaluation strategies. Our value model enables faster improvement than GPT-4o few-shot due to its higher prediction accuracy. In contrast, full evaluation progresses the slowest, as it exhausts much of the evaluation budget on low-performing agents. These results highlight the importance of accurate value estimation and selective evaluation in enabling efficient and targeted agent discovery.

Model-agnostic. To assess the transferability of discovered agents across different LLMs, we perform agent search using `gpt-4o-mini` and then directly evaluate the resulting agent architectures on other models. Our framework demonstrates strong cross-model transferability, as detailed in the table presented in the Appendix.

Hyperparameter Sensitivity. We analyze the sensitivity of our search strategy's key hyperparameters: α, λ, and β. Our default configuration uses $\alpha = 3.0$, $\lambda = 0.3$, and $\beta = 0.4$. As shown in Table 3, we varied each parameter individually while holding the others constant, evaluating performance on the Alfworld benchmarks. The results demonstrate that AgentSwift is robust to variations in these hyperparameters, maintaining strong performance across a range of values.

Generalization analysis

We evaluate the generalization ability of our value model by adapting it to the unseen M3ToolEval benchmark using a varying number of labeled examples for few-shot adaptation. As shown in Figure 5, which plots MSE against the number of adaptation samples, our value model demonstrates remarkable sample efficiency. With as few as 30 labeled examples, our model's performance already approaches the oracle performance achieved when trained on the full dataset. This strong generalization capability is attributed to the highly structured agent representation: the hierarchical design of agentic workflow and functional components forms a compositional abstraction that is both interpretable and transferable across tasks. This allows the model to learn the relationship between agent and performance effectively, even with minimal supervision on a new task.

Table 3: Sensitivity analysis of hyperparameters α, λ, and β on Alfworld benchmarks. Performance is robust across different settings.

Hyperparameter	Alfworld
$\alpha = 2.0$	0.784
$\alpha = 3.0$ (default)	0.806
$\alpha = 4.0$	0.813
$\alpha = 5.0$	0.799
$\lambda = 0.1$	0.768
$\lambda = 0.2$	0.795
$\lambda = 0.3$ (default)	0.806
$\lambda = 0.4$	0.784
$\beta = 0.2$	0.793
$\beta = 0.3$	0.785
$\beta = 0.4$ (default)	0.806
$\beta = 0.5$	0.801

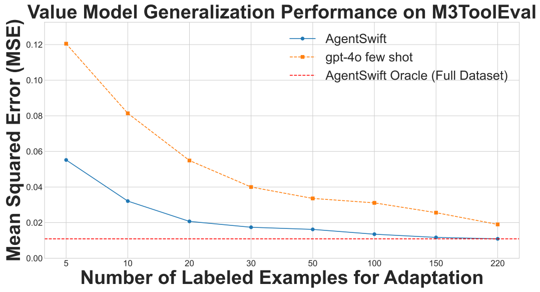

Figure 5: Performance comparison on M3ToolEval under few-shot adaptation.

Ablation study

To assess the contribution of each stage in our hierarchical search strategy, we conduct ablations by individually removing the recombination, mutation, and refinement stages. More details are presented in Appendix.

Conclusion

In this work, we propose a unified framework for automated agentic system search that combines a hierarchical search space with a value model and an uncertainty-guided hierarchical MCTS strategy. Our formulation captures both the structural workflow and functional components of agents, enabling rich architectural variation and compositional reasoning. The value model provides accurate and low-cost performance prediction, while the uncertainty-aware search strategy efficiently explores the vast design space by prioritizing promising candidates.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47: 235–256.

Brier, G. W. 1950. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1): 1–3.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, G.; Dong, S.; Shu, Y.; Zhang, G.; Sesay, J.; Karlsson, B. F.; Fu, J.; and Shi, Y. 2023a. AutoAgents: A Framework for Automatic Agent Generation. *arXiv preprint arXiv:2309.17288*.

Chen, W.; Su, Y.; Zuo, J.; Yang, C.; Yuan, C.; Chan, C.-M.; Yu, H.; Lu, Y.; Hung, Y.-H.; Qian, C.; et al. 2023b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

Du, Y.; Li, S.; Torralba, A.; Tenenbaum, J. B.; and Mordatch, I. 2023. Improving Factuality and Reasoning in Language Models through Multiagent Debate. *arXiv preprint arXiv:2305.14325*.

Du, Y.; Wei, F.; and Zhang, H. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*.

Fernando, C.; Banarse, D.; Michalewski, H.; Osindero, S.; and Rocktäschel, T. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Ge, Y.; Hua, W.; Mei, K.; Tan, J.; Xu, S.; Li, Z.; Zhang, Y.; et al. 2024. Openagi: When llm meets domain experts. *Advances in Neural Information Processing Systems*, 36.

Hu, S.; Lu, C.; and Clune, J. 2024. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Jiang, A.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D.; Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. 2024. Mistral 7B. arXiv 2023. *arXiv preprint arXiv:2310.06825*.

Kandasamy, K.; Neiswanger, W.; Schneider, J.; Poczos, B.; and Xing, E. P. 2018. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31.

Khattab, O.; Singhvi, A.; Maheshwari, P.; Zhang, Z.; Santhanam, K.; Vardhamanan, S.; Haq, S.; Sharma, A.; Joshi, T. T.; Moazam, H.; et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Ma, C.; Zhang, J.; Zhu, Z.; Yang, C.; Yang, Y.; Jin, Y.; Lan, Z.; Kong, L.; and He, J. 2024. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. *arXiv preprint arXiv:2401.13178*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Maziarz, K.; Khorlin, A.; de Laroussilhe, Q.; and Gesmundo, A. 2018. Evolutionary-Neural Hybrid Agents for Architecture Search. *arXiv preprint arXiv:1811.09828*.

Nakano, R.; Hilton, J.; Balaji, S.; Wu, J.; Ouyang, L.; Kim, C.; Hesse, C.; Jain, S.; Kosaraju, V.; Saunders, W.; et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Niu, B.; Song, Y.; Lian, K.; Shen, Y.; Yao, Y.; Zhang, K.; and Liu, T. 2025. Flow: A Modular Approach to Automated Agentic Workflow Generation. *arXiv preprint arXiv:2501.07834*.

OpenAI. 2024. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/.

Park, J. S.; O'Brien, J.; Cai, C. J.; Morris, M. R.; Liang, P.; and Bernstein, M. S. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, 1–22.

Qian, C.; Xie, Z.; Wang, Y.; Liu, W.; Dang, Y.; Du, Z.; Chen, W.; Yang, C.; Liu, Z.; and Sun, M. 2024. Scaling large-language-model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*.

Qiao, S.; Zhang, N.; Fang, R.; Luo, Y.; Zhou, W.; Jiang, Y. E.; Lv, C.; and Chen, H. 2024. AutoAct: Automatic agent learning from scratch for QA via self-planning. *arXiv preprint arXiv:2401.05268*.

Qin, S.; Kadlecová, G.; Pilát, M.; Cohen, S. B.; Neruda, R.; Crowley, E. J.; Lukasik, J.; and Ericsson, L. 2025. Transferrable Surrogates in Expressive Neural Architecture Search Spaces. *arXiv preprint arXiv:2504.12971*.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; et al. 2018. Improving language understanding by generative pre-training.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, 4780–4789.

Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; et al. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995): 468–475.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36: 68539–68551.

Shang, Y.; Li, Y.; Xu, F.; and Li, Y. 2024a. DefInt: A Default-interventionist Framework for Efficient Reasoning with Hybrid Large Language Models. *arXiv preprint arXiv:2402.02563*.

Shang, Y.; Li, Y.; Zhao, K.; Ma, L.; Liu, J.; Xu, F.; and Li, Y. 2024b. Agentsquare: Automatic llm agent search in modular design space. *arXiv preprint arXiv:2410.06153*.

Shen, Y.; Song, K.; Tan, X.; Li, D.; Lu, W.; and Zhuang, Y. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36: 38154–38180.

Shridhar, M.; Yuan, X.; Cote, M.-A.; Bisk, Y.; Trischler, A.; and Hausknecht, M. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *International Conference on Learning Representations*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837.

Wen, L.; Fu, D.; Li, X.; Cai, X.; MA, T.; Cai, P.; Dou, M.; Shi, B.; He, L.; and Qiao, Y. 2024. DiLu: A Knowledge-Driven Approach to Autonomous Driving with Large Language Models. In *The Twelfth International Conference on Learning Representations*.

White, C.; Neiswanger, W.; and Savani, Y. 2021. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 10293–10301.

Xi, Z.; Ding, Y.; Chen, W.; Hong, B.; Guo, H.; Wang, J.; Yang, D.; Liao, C.; Guo, X.; He, W.; et al. 2024. AgentGym: Evolving Large Language Model-based Agents across Diverse Environments. *arXiv preprint arXiv:2406.04151*.

Xie, J.; Zhang, K.; Chen, J.; Zhu, T.; Lou, R.; Tian, Y.; Xiao, Y.; and Su, Y. 2024. TravelPlanner: A Benchmark for Real-World Planning with Language Agents. In *Forty-first International Conference on Machine Learning*.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Yang, C.; Wang, X.; Lu, Y.; Liu, H.; Le, Q. V.; Zhou, D.; and Chen, X. 2024b. Large Language Models as Optimizers. In *The Twelfth International Conference on Learning Representations*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Yuan, S.; Song, K.; Chen, J.; Tan, X.; Li, D.; and Yang, D. 2024. EvoAgent: Towards Automatic Multi-Agent Generation via Evolutionary Algorithms. *arXiv preprint arXiv:2406.14228*.

Zhang, G.; Chen, K.; Wan, G.; Chang, H.; Cheng, H.; Wang, K.; Hu, S.; and Bai, L. 2025a. EvoFlow: Evolving Diverse Agentic Workflows On The Fly. *arXiv preprint arXiv:2502.07373*.

Zhang, G.; Niu, L.; Fang, J.; Wang, K.; Bai, L.; and Wang, X. 2025b. Multi-agent Architecture Search via Agentic Supernet. *arXiv preprint arXiv:2502.04180*.

Zhang, G.; Yue, Y.; Sun, X.; Wan, G.; Yu, M.; Fang, J.; Wang, K.; Chen, T.; and Cheng, D. 2024a. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*.

Zhang, J.; Xiang, J.; Yu, Z.; Teng, F.; Chen, X.; Chen, J.; Zhuge, M.; Cheng, X.; Hong, S.; Wang, J.; et al. 2024b. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Zhuge, M.; Wang, W.; Kirsch, L.; Faccio, F.; Khizbullin, D.; and Schmidhuber, J. 2024. GPTSwarm: Language Agents as Optimizable Graphs. In *Forty-first International Conference on Machine Learning*.

Zoph, B.; and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.