A^{2}Flow: Automating Agentic Workflow Generation via Self-Adaptive Abstraction Operators

Mingming Zhao[1], **Xiaokang Wei**[1,2*], **Yuanqi Shao**[1,3],
Kaiwen Zhou[1,4], **Lin Yang**[1], **Siwei Rao**[5], **Junhui Zhan**[5], **Zhitang Chen**[1],
[1] Huawei Noah's Ark Lab,
[2]The Hong Kong Polytechnic University,
[3]State Key Laboratory of Multimodal Artificial Intelligence Systems, CASIA,
[4] The Chinese University of Hong Kong,
[5] Huawei Technologies Co., Ltd
zhaomingming9@huawei.com

Abstract

Large language models (LLMs) have shown strong potential in automating the design of agentic workflows. However, existing methods still rely heavily on manually predefined operators, limiting generalization and scalability. To address this issue, we propose **A^{2}Flow**, a fully automated framework for agentic workflow generation based on *self-adaptive abstraction operators*. **A^{2}Flow** employs a three-stage operator extraction process: 1) Case-based Initial Operator Generation: leveraging expert demonstrations and LLM reasoning to generate case-specific operators; 2) Operator Clustering and Preliminary Abstraction: grouping similar operators across tasks to form preliminary abstractions; and 3) Deep Extraction for Abstract Execution Operators: applying long chain-of-thought prompting and multi-path reasoning to derive compact and generalizable execution operators. These operators serve as reusable building blocks for workflow construction without manual predefinition. Furthermore, we enhance node-level workflow search with an *operator memory mechanism*, which retains historical outputs to enrich context and improve decision-making. Experiments on general and embodied benchmarks show that **A^{2}Flow** achieves a 2.4% and 19.3% average performance improvement and reduces resource usage by 37% over state-of-the-art baselines.

Code — https://github.com/pandawei-ele/A2FLOW

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse domains, including code generation, data analysis, decision-making, question answering, autonomous driving, and even embodied-ai reasoning (Zhu et al. 2024; Zhuge et al. 2024; Liu et al. 2024a; Xie et al. 2024b; Zhong et al. 2024; Cao et al. 2025; Madaan et al. 2023). The emergence of LLM agents—autonomous systems that leverage LLMs to perceive, reason, and act further extends their potential across complex tasks and environments. However, these agents often rely on manually crafted agentic workflows, which are structured sequences of LLM invocations guided by human-designed instructions (Brown 2020). The process of designing and refining such workflows, requiring substantial human effort, constrains the scalability of LLMs, their adaptability to novel domains, and their generalization ability across diverse tasks (Tang et al. 2023; Zhang et al. 2024a; Xie et al. 2024a; Wang et al. 2024c).

Recent research seeks to automate the generation and optimization of effective agentic workflows. DSPy (Khattab et al. 2024) introduces a structured framework that formalizes language model workflows as learnable and optimizable text transformation graphs, enabling automated construction and tuning of LLM pipelines without relying on

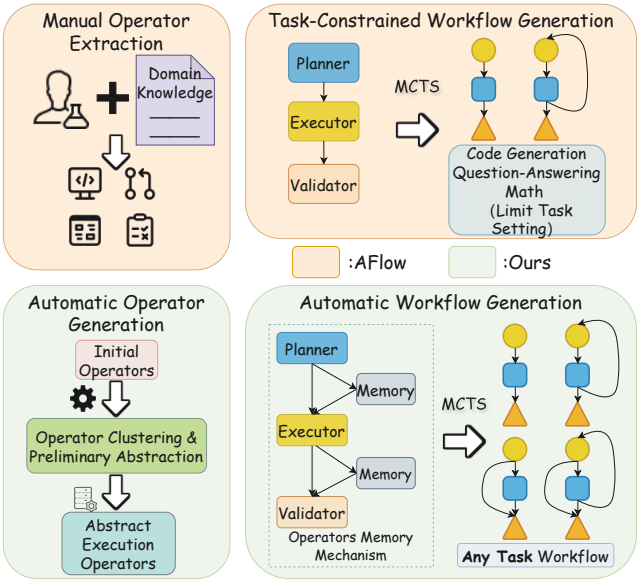

Figure 1: Compared with existing methods, A^{2}Flow improves performance by: (1) Self-Adaptive Abstraction Operators module that optimizes the search process and improves efficiency, and (2) Operators Memory Mechanism that leverages enables more context-aware execution and improving workflow search performance.

*Corresponding author.

manually crafted prompt templates. ADAS (Hu, Lu, and Clune 2024) designs agentic systems through code-based workflows, but its linear heuristic search mechanism lacks efficiency, making it difficult to produce optimal workflows within a restricted number of iterations. AFLOW (Zhang et al. 2024a) introduces an automated workflow generation framework that formulates workflow optimization as a code-based search problem and efficiently explores he vast search space using a variant of MCTS. It introduces Operators: reusable bundles that wrap common agent actions (e.g., Ensemble, Review, Revise) by packaging nodes and edges into a simple interface. Given a predefined set of Operators and the edge space, AFLOW can build and refine workflows more efficiently. However, AFLOW still depends on manually crafted, task-specific operators, which limit the automation of workflow construction and hinder generalization to open-world or embodied tasks, and cannot guarantee optimality within the search space, as illustrated at the top of FigureDebFlow (Su et al. 2025) leverages a debate mechanism and integrates reflection to optimize workflows and reduce resource usage; however, it still relies on the same predefined operators as AFLOW, making it prone to redundant operators and unnecessary overhead. MermaidFlow (Zheng et al. 2025) introduces domain-aware evolutionary operators that maintain semantic correctness while improving workflow diversity. However, its predefined operator initialization limits generalization to open-world tasks. Thus, full automation remains an open challenge.

To address the limitations of existing agentic workflows and realize fully automated workflow generation, we introduce **A^{2}Flow**, an agentic workflow generation framework equipped with *Self-Adaptive Abstraction Operators* to optimize the search process and improve efficiency. By removing the dependency on manually predefined, task-specific operators, A^{2}Flow reduces reliance on human expertise in workflow design and mitigates search inefficiencies caused by manually introduced biases, thereby enhancing the scalability, adaptability, and generalization ability of LLMs across diverse and complex tasks. Additionally, we integrate an *Operators Memory Mechanism* that reuses agents' thought processes, reducing interaction steps during inference and improving generation efficiency. Specifically, A^{2}Flow is designed to autonomously derive optimal abstract operators directly from raw expert demonstrations, eliminating manual predefinition through a novel pipeline (shown at the bottom of Figure 1). This process consists of three key stages: 1) *Initial Operator Generation*: Using expert data and the LLM, prompts generate case-aware operators for each task type; 2) *Operator Clustering and Abstraction*: Operators with similar functions are clustered across cases to form preliminary abstract operators; 3) *Operator Refinement and Selection*: Preliminary operators undergo deep reasoning via Long Chain-of-Thought (COT) (Wei et al. 2022) and feasibility checks, followed by a self-consistency-based multi-path search to select final task-aware abstract operators. Besides, when deploying these adaptive operators for workflow search, the *Operator Memory Mechanism* enhances information sharing and strengthens task comprehension capabilities.

The key contributions of this work are as follows:

- We propose **A^{2}Flow**, a highly automated framework for agentic workflow generation that extracts *Self-Adaptive Abstraction Operators* to reduce computational costs and significantly improve generalization across diverse tasks.
- We introduce an *Operators Memory Mechanism*, which augments each operator with access to accumulated historical outputs, enabling more context-aware execution and improving workflow search performance.
- Extensive experiments on eight benchmark datasets across five distinct domains, including code generation, mathematical reasoning, reading comprehension, games, and embodied tasks, demonstrate that A^{2}Flow consistently outperforms state-of-the-art methods and exhibits strong generalization capabilities.

2 Related Work

Agentic Workflow Agentic workflows and autonomous agents are two main ways to use LLMs. Autonomous agents make decisions dynamically using feedback from the environment (Wang et al. 2023a; Zhuge et al. 2023; Zhang et al. 2024b; Kim et al. 2024; Song et al. 2023; Zhou et al. 2024a). In contrast, agentic workflows follow predefined multi-step procedures designed through human expertise and iterative refinement (Wang et al. 2023b; Yue et al. 2024; Wang et al. 2024b; Zheng et al. 2023; Wang et al. 2024a). Workflows are often easier to build and more adaptable because they do not require complex action spaces or control strategies. Existing workflows fall into two categories: general-purpose and domain-specific. General-purpose workflows use simple and reusable structures to support a wide range of tasks (Wei et al. 2022; Wang et al. 2023b; Madaan et al. 2023; Liu et al. 2024b). Domain-specific workflows target particular applications, such as code generation (Hu, Lu, and Clune 2024; Ridnik, Kredo, and Friedman 2024), data analysis (Xie et al. 2024b), mathematical reasoning (Zhong et al. 2024), and complex question answering (Zhou et al. 2024b). However, manually designed workflows still face challenges. General workflows often fall short on complex reasoning, while domain-specific workflows struggle to transfer to new tasks (Hu et al. 2024; Yao et al. 2022).

Automated Agentic Optimization Recent work to automate agentic workflow design fall into three categories: prompt optimization, hyperparameter tuning, and full workflow optimization (Qiao et al. 2024; Huang et al. 2023a; Zhong et al. 2024; Yang et al. 2024; Huang et al. 2023b). Prompt optimization (Fernando et al. 2023; Yu, He, and Ying 2023; Sumers et al. 2023) improves prompts within fixed workflows using LLMs, but often lacks generalization and requires manual effort. Hyperparameter tuning (Saad-Falcon et al. 2024; Liu et al. 2023; Zhou, Li, and Liu 2023) adjusts preset parameters but has a limited scope. In contrast, automated workflow optimization aims to optimize the entire workflow structure and offers greater potential for fully automatic and generalizable workflow generation. GPTSwarm (Zhuge et al. 2024) adopts graph structures combined with reinforcement learning, but its design struggles to represent workflows involving conditional states,

limiting its expressiveness. ADAS (Hu, Lu, and Clune 2024) represents workflows as code and stores past workflows in a linear list, which aligns with our goals. However, its simple experience representation limits the search efficiency, making it hard to find effective workflows. AFlow (Zhang et al. 2024a) searches workflows using MCTS over code-based nodes and edges. It depends on predefined operators that fix specific node combinations. These operators are manually created for each task, which limits automation and requires redesign in new settings. As a result, AFlow transfers poorly to unseen or open-world tasks such as embodied control and WebShop, and there is no evidence that its operator design is close to optimal. DebFlow (Su et al. 2025) introduces a debate mechanism and reflection to improve workflows and reduce resource use. However, it still inherits AFlow's predefined operators, leading to redundant components and extra overhead. MermaidFlow (Zheng et al. 2025) also faces difficulty generalizing to open-world environments, including embodied and everyday scenarios. **A^{2}Flow** addresses this gap by adaptively extracting abstract operators from expert data without manual design or prior knowledge, enabling automatic discovery of effective operator sets that improve workflow automation and generalization.

3 Method

3.1 Preliminary

Recently, AFLOW (Zhang et al. 2024a) exhibits strong performance and efficiency in automated agentic optimization, which leverages Large Language Models (LLMs) as optimizers within a variant of Monte Carlo Tree Search (MCTS) to search for optimal workflows. To efficiently explore this vast search space, AFLOW (Zhang et al. 2024a) introduces Operators, which encapsulate common agentic operations (e.g., Ensemble, Review, Revise) by unifying N and E into a single interface, enabling AFlow to perform faster search and streamlined workflow generation. Specifically, AFlow defines an *agentic workflow* $\mathcal{W}$ as a series of LLM-invoking nodes connected by edges to define the execution orders, denoted as $\mathcal{N} = \{N_1, N_2, \ldots, N_i, \ldots\}$. Each node N_i represents a specific operation performed by an LLM and is characterized by the following parameters.

A workflow node N_i is defined by a language model M, an input prompt P, a temperature parameter τ controlling output randomness, and an output format F (e.g., XML, JSON, Markdown, raw).

Edge E represents abstract structures defining node relationships, governing the sequence of execution.

Given a task T and an evaluation function G, workflow optimization aims to find a workflow $\mathcal{W}$ that maximizes $G(\mathcal{W}, T)$. AFLOW (Zhang et al. 2024a) formulates this as a search problem where an algorithm A explores the search space $\mathcal{S}$, which includes all possible configurations of node parameters and edge structures. To make this exploration efficient, AFLOW (Zhang et al. 2024a) introduces *Operators* that encapsulate common agentic operations (e.g., Ensemble, Review, Revise) by unifying nodes and edges into concise interfaces, enabling AFLOW (Zhang et al. 2024a) to achieve faster search and streamlined workflow generation.

$$\mathcal{S}_{\text{AFLOW}} = \{(P_1, \ldots, P_n, E, O_1, \ldots, O_n)\}, \quad (1)$$

$$W^* = \text{AFLOW}(\mathcal{S}_{\text{AFLOW}}, G, T), \quad (2)$$

where $\mathcal{P}$, $\mathcal{E}$, $\mathcal{O}$ representing the sets of possible prompts, output formats, and edge configurations, respectively, and $P_i \in \mathcal{P}$, $E \in \mathcal{E}$, $O_i \in \mathcal{O}$.

3.2 Overview

Our A^{2}Flow enhances the automation of generating agentic workflows by employing Large Language Models (LLMs) as optimizers within **Self-Adaptive Abstraction Operators** (Figure 2). Before initiating workflow search, we automatically extract these operators without relying on predefined human expertise. The extraction consists of three stages: (1) *Case-based Initial Operator Generation*, where prompts use expert training data and LLM reasoning to generate case-aware operators for each task type; (2) *Operator Clustering and Abstraction*, where operators with similar functions across cases are grouped into preliminary abstract operators; and (3) *Deep Extraction for Abstract Execution Operators*, where Long CoT (Wei et al. 2022) and self-consistency enable deep reasoning and multi-path search to refine task-aware abstract operators. To support the workflow search process, we also propose an *Operator Memory Mechanism* (Sec. 3.4) that promotes information sharing across operators and improves their task comprehension.

3.3 Self-Adaptive Abstraction Operators

We observe that recent studies rely heavily on manually predefined operators, which limits workflow automation and hinders fast generalization to open-world tasks. To address this, we propose an automatic generation approach for code-based and self-adaptive abstract operators, which directly derives highly integrated operators from raw expert data, enhancing the ability to rapidly and automatically generate workflows for diverse tasks.

Case-based Initial Operators Generation Following AFLOW (Zhang et al. 2024a), we partition the expert dataset for the specific task into a 20% validation set and an 80% test set, with the random seed fixed at 42. Using the validation set, we design initial operator-extraction prompts and leverage the reasoning capability of the LLM to perform initial case-aware operator extraction. Formally, the initial operator set O is defined as

$$O^{(e)} = \{\, o_{i,j} = E(C_i, P_e, M) \mid i = 1, \ldots, n; j = 1, \ldots, m_i \,\}, \quad (3)$$

where n denotes the total number of cases, and m_i is the number of operators generated for C_i. E is the operator extraction function that, given a specific case C_i, the prompt P_e, and LLM M, generates a set of initial case-aware operators $O^{(e)}$.

Notably, we design prompts P_c to convert each case into a code-based workflow, where each operator in $O^{(e)}$ is defined as a basic block, similar to standard program code. Each block has a single input and output, with no intermediate jumps or branching. The details of the prompts P_c as shown in Figure 4 and Appendix.

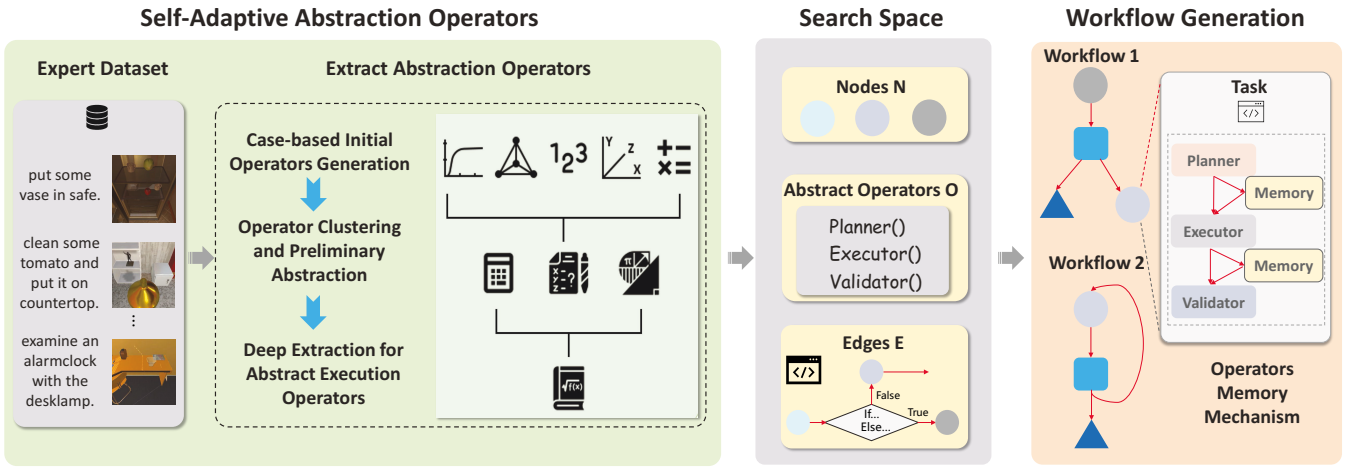

Figure 2: Overview of our framework. Left: Our method synthesizes abstract execution operators through a three-phase generation process, leveraging expert data for iterative refinement. Middle: setting a search space composed of nodes, a abstract operator set, and a code representing edge. Right: Illustration of the MCTS-based evolutionary workflow process. Through integrating an Operators Memory Mechanism, A^{2}Flow enhance the workflow search capability at each node.

Operator Clustering and Preliminary Abstraction Since the initial operator set $O^{(e)}$ often contains operators with overlapping or similar functionalities, we further apply a functional clustering process guided by the LLM. Formally, the preliminary abstract operator set is defined as

$$O^{(a)} = \mathcal{C}(O^{(e)}, P_a, M), \tag{4}$$

where P_a is the clustering prompt, and $\mathcal{C}$ denotes the functional clustering function that abstracts similar operators into higher-level representations. The details of the prompts P_a as shown in Figure 4 and Appendix.

Deep Extraction for Abstract Execution Operators After clustering, the number of case-based operators is reduced, but they remain insufficiently abstract and general. To achieve deeper abstraction and merge the preliminary abstract operator set $O^{(a)}$ into universal execution operators (e.g., execute, verify), we design a *multi-path parallel generation* framework with Long chain-of-thought (CoT) prompting to obtain the final task-aware abstract operators $O^{(t)}$. By adjusting the temperature and randomness of the LLM, we generate m independent reasoning paths $\{\mathcal{P}_p\}_{p=1}^{m}$, where $m = 6$ in our implementation. For each path $\mathcal{P}_p$, we perform three iterative refinement steps guided by introducing CoT prompting:

1. The task instructions I, the prompt P_t, and the preliminary abstract operator set $O^{(a)}$ are provided to the LLM model to generate the initial abstract operator o_1.
2. The tuple (I, P_t, o_1) is fed into the LLM model with CoT prompting to produce the refined operator o_2.
3. The tuple (I, P_t, o_1, o_2) is used with another round of CoT reasoning to obtain the final abstract operator o_3.

The final task-specific operator set is obtained by aggregating the final operators from all paths:

$$O^{(t)} = \mathcal{A}_t\big(\{o_{p,3}\}_{p=1}^{6}, P_t, M\big), \tag{5}$$

where $O^{(t)}$ denotes the final set of task-aware abstract operators, $\{o_{p,3}\}_{p=1}^{6}$ represents the final candidate operators generated from each of the six reasoning paths after three-step iterative refinement, $\mathcal{A}_t(\cdot)$ refers to the LLM-based aggregation function that merges and generalizes the operators across multiple paths, P_t is the prompt used during the aggregation stage.

Reflection Mechanism To ensure the correctness of code-based reasoning, we incorporate a reflection mechanism inspired by self-correction and error-feedback. At each step, the LLM is prompted to output Python code, which is then validated by a Python executor for syntax and executability. If the code fails the check, the executor raises an error signal that triggers the LLM to reflect and regenerate the code until a valid result is obtained. This iterative correction improves the reliability of the operator extraction process.

3.4 Operators Memory Mechanism

We enhance the workflow search capability at each node by integrating an *Operators Memory Mechanism* that retains and accumulates intermediate operator outputs. Unlike the sequential AFLOW design, where each operator o_k only depends on the immediate output of o_{k-1}, our mechanism introduces a memory space $\mathcal{M}_k$ to store all previous results. The computation of the k-th operator is thus defined as:

$$o_k = f_k(\text{input}_k, P_k, \mathcal{M}_{k-1}), \tag{6}$$

where P_k is the instruction prompt and $\mathcal{M}_{k-1}$ contains all outputs up to step $k-1$. In each node, f_k denotes the corresponding code-based operator execution function. After each step, the memory is updated as:

$$\mathcal{M}_k = \mathcal{M}_{k-1} \cup \{o_k\}, \tag{7}$$

By incorporating both the original task context and historical information, this design empowers the operator to perform more accurately and generalize better across tasks.

3.5 Automated Workflow Optimization

Our workflow search and optimization mechanism builds upon the AFLOW framework, which follows a structured cycle of *initialization, selection, expansion, evaluation, and backpropagation*. We start from a template workflow W_0, partition the dataset into a validation set (20%) and a test set (80%) with a fixed random seed, and employ a mixed probability selection strategy to balance exploration and exploitation. The LLM-based optimizer expands workflows by generating or modifying nodes, while each candidate workflow is executed multiple times on the validation set to compute robust performance metrics, which are then backpropagated to update scores and historical experience.

Unlike AFLOW, which uses predefined operators, we inject our self-adaptive abstract operator set $\{O^{(t)}\}$ (Section 3.3) and Operators Memory M(Section 3.4) into the workflow search. During the expansion phase, the LLM-based optimizer directly leverages these adaptive operators, which are capable of dynamically generalizing and merging execution steps, as described in our three-step extraction process. The final optimized workflow is defined as:

$$W^* = \mathcal{S}\big(W_0, \{O^{(t)}\}, G, D_V, \mathcal{M}\big), \tag{8}$$

where $\mathcal{S}$ denotes the AFLOW-based search and optimization procedure, $\{O^{(t)}\}$ is our self-adaptive operator set, G is the evaluator, and D_V is the validation dataset. This integration allows the search to jointly optimize workflow structures and operator representations, yielding more compact and task-specific solutions.

4 Experiments

4.1 Experiments Setting

Datasets Our experiments used eight public benchmarks across five different domains: 1) **code generation** (HumanEval (Chen et al. 2021), MBPP (Austin et al. 2021)), 2) **math reasoning** (GSM8K (Cobbe et al. 2021), MATH (Hendrycks et al. 2021)), 3) **reading comprehension** (HotpotQA (Yang et al. 2018), DROP (Dua et al. 2019)), 4) **embodied task** (ALFWorld (Shridhar et al. 2020)), 5) **game** (textcraft (Prasad et al. 2023)). For the MATH benchmark, we adhered to established protocols (AFLOW (Zhang et al. 2024a)), utilizing identically curated problem sets derived from four prototypical problem types at complexity level 5. Details are provided in the Appendix.

Baseline We benchmark A^{2}Flow against two categories of agent-based methodologies: (1) Manually engineered workflows comprising: IO(direct LLM invocation), Chain-of-Thought (CoT) (Wei et al. 2022), Self-Consistency (SC) with 5-answer sampling (Wang et al. 2022), Multi-Persona debate frameworks (Wang et al. 2023b), (2) Autonomous workflow optimizers:ADAS(Automated workflow synthesis via LLM agents) (Hu, Lu, and Clune 2024), AFLOW (Zhang et al. 2024a)

Implementation Details A^{2}Flow utilizes different models for generation, optimization. and execution. We use Claude-3.5-sonnet (Anthropic 2024) as the workflows' optimizer and use models: GPT-4o-mini-0718 (OpenAI 2024a), GPT-4o (OpenAI 2024b) and deepseek-v3 (Deepseek 2024) as executors. we use models deepseek-v3 (Deepseek 2024) as abstract operations' generator.

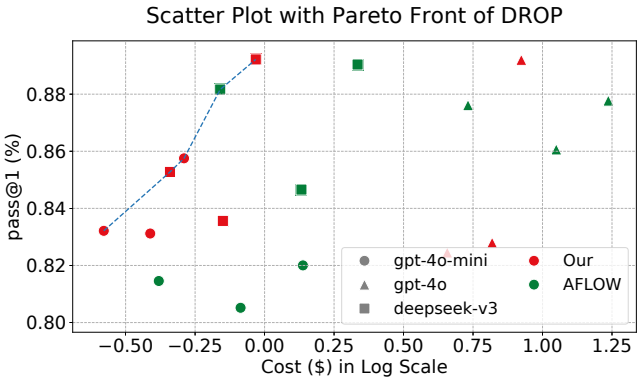

Figure 3: Total execution costs for the partitioned DROP test set are shown. A^{2}Flow and AFLOW-generated workflows (execution model) were evaluated using the same model. Legend colors denote the LLM executing each workflow, with exact values provided in Appendix.

Evaluation Metrics We employ distinct evaluation metrics tailored to each benchmark. ALFWorld and TextCraft are indicated via binary rewards, signifying overall success or failure. For the mathematical reasoning datasets, GSM8K and MATH $_{lv5}$, the Solve Rate serves as the primary performance measure. Code generation capability is assessed on HumanEval and MBPP using the pass@1 metric (Chen et al. 2021). Performance on the question answering benchmarks, HotpotQA and DROP, is quantified using the F1 Score. To evaluate both efficiency and accuracy, token usage across datasets is tracked to estimate cost, which is then analyzed with performance metrics to construct a Pareto front illustrating the trade-offs between methods.

4.2 Experiments Results and Analysis

Main Result As shown in Table 1, we compare A^{2}Flow against 8 baselines on the math reasoning, code generation and reading comprehension benchmarks. The experimental results indicate that A^{2}Flow achieves SOTA performance across all primary task metrics, with the sole exception of the HumanEval benchmark (Chen et al. 2021). Compared to methods that pre-defined operators in workflow, our approach outperforms them by an average margin of 2.4%. On the DROP benchmark specifically, A^{2}Flow exceeds the suboptimal method AFLOW by 4.5%. On the MATH benchmark, it surpasses the SOTA baseline AFLOW by 4.1%. For certain benchmarks, performance is primarily constrained by the inherent limitations of abstract operator implementation. In HumanEval and MBPP benchmarks, baselines utilizing pre-defined operators (including Python interpreter invocation) establish a strong comparison point. Consequently, optimizations relying solely on abstract operators demonstrate significantly lower effectiveness. Conversely, on the GSM8K benchmark, where baseline performance is already high, abstract operator optimization yields only modest gains. We are preliminarily exploring the gen-

Task Method	Reading HotpotQA	 DROP	Coding HumanEval	 MBPP	Reasoning GSM8K	 MATH	Avg.
IO (GPT-4o-mini)	68.1	68.3	87.0	71.8	92.7	48.6	72.8
CoT (Wei et al., 2022)	67.9	78.5	88.6	71.8	92.4	48.8	74.7
CoT SC (5-shot) (Wang et al., 2022)	68.9	78.8	91.6	73.6	92.7	50.4	76.0
MedPrompt (Nori et al., 2023)	68.3	78.0	91.6	73.6	90.0	50.0	75.3
MultiPersona (Wang et al., 2024a)	69.2	74.4	89.3	73.6	92.8	50.8	75.1
Self Refine (Madaan et al., 2023)	60.8	70.2	87.8	69.8	89.6	46.1	70.7
ADAS (Hu et al., 2024)	64.5	76.6	82.4	53.4	90.8	35.4	67.2
AFLOW(Zhang et al., 2025)	73.5	80.6	90.9	83.4	93.5	56.2	79.6
Ours	**74.1**	**85.1**	**92.4**	**85.0**	**93.8**	**58.5**	**81.5**

Table 1: We compare the performance of pre-defined operators and manually designed workflows against the fully automated framework for agentic workflow generation across reading comprehension, code generation, and mathematical reasoning scenarios. All experiments utilize the GPT-4o-mini model as executor on test set. Performance metrics are reported as the mean over three independent trials.

Method	ALFWorld Seen	 UnSeen	TextCraft	Avg.
ReAct	22.0	22.9	33.0	25.9
AFLOW	17.1	26.6	53.0	32.2
Our	**25.0**	**31.3**	**59.0**	**38.4**

Table 2: Comparative experiments on A^2Flow vs. agentic workflow-based baselines. The best results are marked in **bold**. All experiments utilize the DeepSeek-v3 model as the executor on the test set.

Operators Type	Score	Δ Score (%) $\uparrow$
w/o Abstraction Operators & Operators Memory	56.2	-
w/o Operators Memory	53.9	-4.1
Our	58.5	4.1
w/o Initial Operators	49.6	-11.7
w/o Operator Clustering	54.5	-3.0
w/o Deep Extraction Operators	51.6	-8.2

Table 3: Ablation study w.r.t. four components: initialization operators, clustering operators, abstract execution operators, and the operator memory mechanism.

eralization of this approach in embodied and game tasks. Results in Table 2 demonstrate a statistically significant performance improvement of 19.3%. This enhancement primarily stems from the workflow's adaptive suitability for target tasks, enabled by automated search algorithms that derive abstract execution operators from limited training data.

Cost Analysis Using GPT-4o-mini as the execution LLM, we compare the performance and cost of baseline methods with the top three workflows discovered by A^2Flow. As shown in Figure 3, A^2Flow uncovers workflows that allow less capable models to outperform stronger ones on the cost-effectiveness Pareto front. This significantly reduces the barriers to deploying agentic workflows by automating effective workflow design and eliminating the need for manual effort. Remarkably superior cost-performance further enhances the model's potential for widespread adoption.

4.3 Ablation Study

To quantify each component's contribution, we conduct ablations on four A^2Flow variants and report both absolute *Score* and Δ*Score* on MATH (Table 3). The full model attains 58.5% (Δ+4.1). Removing *Operators Memory* reduces performance to 53.9% ($\Delta-4.1$). Disabling the *Self-Adaptive Abstraction Operators*—w/o Initial Operators, w/o Operator Clustering, and w/o Deep Extraction—yields 49.6% ($\Delta-11.7$), 54.5% ($\Delta-3.0$), and 51.6% ($\Delta-8.2$), respectively. The baseline without both modules scores 56.2%. These results show that both modules are vital, with the Self-Adaptive Abstraction Operators offering slightly larger gains.

4.4 Case Study

Figure 4 illustrates the self-adaptive operator generation and automated workflow optimization in A^2Flow, showing the evolution from embodied task samples to execution operators and workflows. In the Self-Adaptive Abstraction Operators stage, we first split the expert dataset into a 20% validation set (33 samples). Using initial operator-extraction prompts, the LLM analyzes example cases and proposes initial abstraction operators, such as ObserveEnvironment() and CreatePlan(). In the second step, we further apply a functional clustering process guided by the LLM and achieve operator clustering and preliminary abstraction. For the third step, we use a multi-path parallel generation framework with Long chain-of-thought (CoT) prompting to obtain the final alfworld embodied task abstract operators(*Planner()*, *Executor()*, *Validator()*). In the MCTS for Workflow part, it evolves from an initial *Executor()* operator to the final workflow presented. In each iteration, the optimizer dynamically integrates new operators or adjusts workflow edges to enhance performance. When a workflow

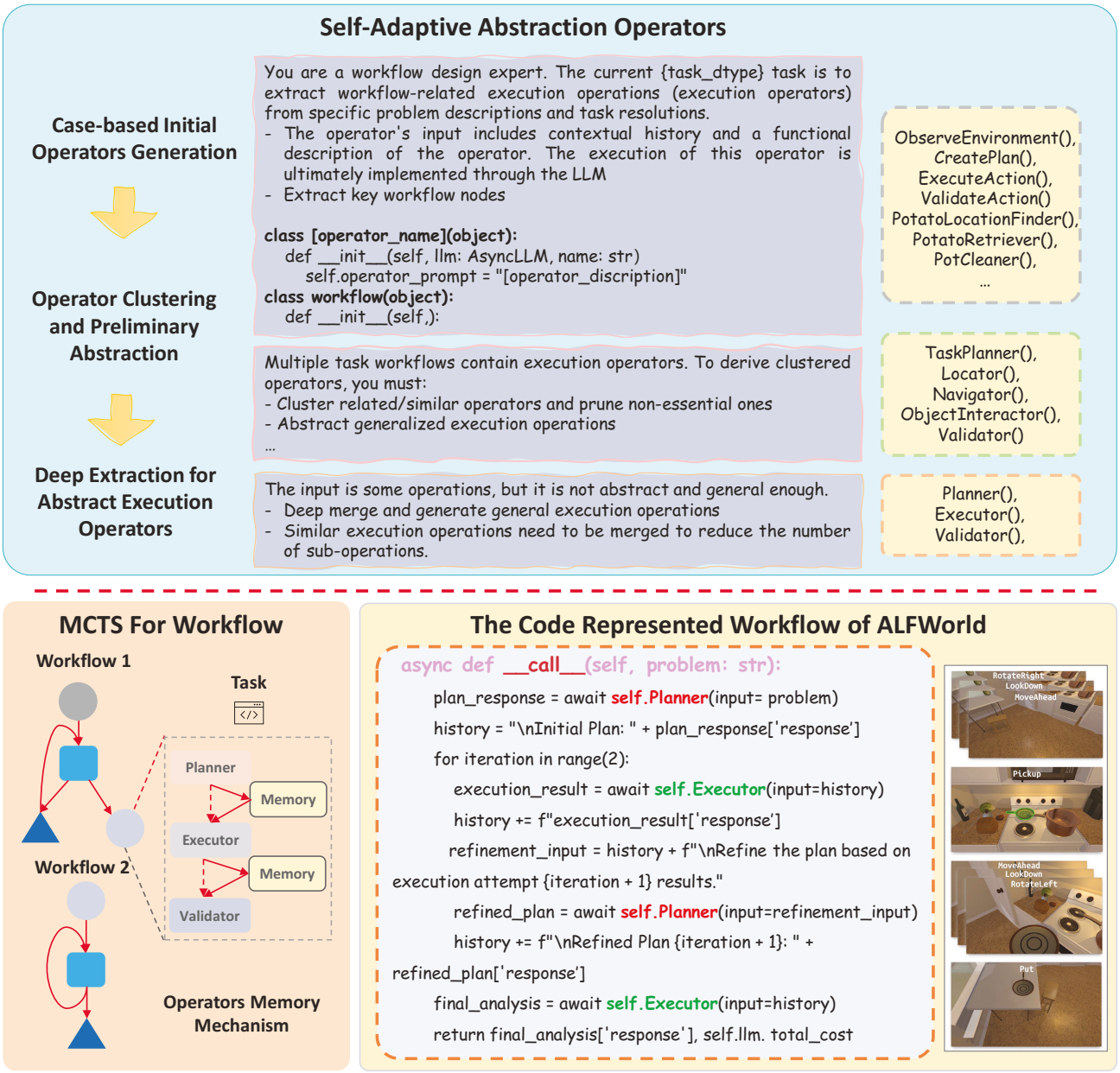

Figure 4: Self-adaptive process of operator generation and automated workflow optimization. The ALFWorld case study illustrates how A^{2}Flow abstracts embodied tasks into executable operators through three-phase generation, then navigates the search space via MCTS to converge on optimal structured agentic workflows.

contains more than two operators, node-to-node transformation messages are stored in operator memory, enabling context information sharing. Finally, we present the code for the optimal workflow discovered by A^{2}Flow on the ALFWorld.

5 Conclusion

In this work, we propose **A^{2}Flow**, a novel fully automated framework for agentic workflow generation via self-adaptive abstraction operators. Specifically, **A^{2}Flow** uses a three-step method to extract operators. It starts with case-specific operator generation, combining expert examples and LLM reasoning. Then, it performs operator clustering, grouping similar ones to form early abstractions. Finally, it applies deep reasoning with chain-of-thought prompts to refine these into compact, general-purpose execution operators. These reusable components enable workflow construction without manual definitions. A^{2}Flow also includes an operator memory mechanism that stores past outputs to improve node-level decision-making. Across general and embodied tasks, A^{2}Flow improves performance by 2.4%, 19.3%, and reduces resource usage by 37% over leading baselines.

References

Anthropic. 2024. Introducing Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet.

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Brown, T. B. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Cao, Z.; Wang, R.; Yang, Y.; Ma, X.; Zhu, X.; Zheng, B.; and Zhao, H. 2025. PGPO: Enhancing Agent Reasoning via Pseudocode-style Planning Guided Preference Optimization. *arXiv preprint arXiv:2506.01475*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Deepseek. 2024. DeepSeek-V2.5. https://huggingface.co/deepseek-ai/DeepSeek-V2.5.

Dua, D.; Wang, Y.; Dasigi, P.; Stanovsky, G.; Singh, S.; and Gardner, M. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*.

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*.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Hu, M.; Zhao, P.; Xu, C.; Sun, Q.; Lou, J.; Lin, Q.; Luo, P.; Rajmohan, S.; and Zhang, D. 2024. AgentGen: Enhancing Planning Abilities for Large Language Model based Agent via Environment and Task Generation. *arXiv preprint arXiv:2408.00764*.

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

Huang, D.; Bu, Q.; Zhang, J. M.; Luck, M.; and Cui, H. 2023a. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Huang, D.; Zhang, J. M.; Luck, M.; Bu, Q.; Qing, Y.; and Cui, H. 2023b. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Khattab, O.; Singhvi, A.; Maheshwari, P.; Zhang, Z.; Santhanam, K.; Haq, S.; Sharma, A.; Joshi, T. T.; Moazam, H.; Miller, H.; et al. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*.

Kim, J.; Paranjape, B.; Khot, T.; and Hajishirzi, H. 2024. Husky: A Unified, Open-Source Language Agent for Multi-Step Reasoning. *arXiv preprint arXiv:2406.06469*.

Liu, X.; Shen, S.; Li, B.; Ma, P.; Jiang, R.; Zhang, Y.; Fan, J.; Li, G.; Tang, N.; and Luo, Y. 2024a. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; et al. 2024b. AgentBench: Evaluating LLMs as Agents. In *International Conference on Learning Representations*.

Liu, Z.; Zhang, Y.; Li, P.; Liu, Y.; and Yang, D. 2023. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*.

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. *Advances in Neural Information Processing Systems*, 36: 46534–46594.

OpenAI. 2024a. GPT-4o mini: Advancing Cost-Efficient Intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

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

Prasad, A.; Koller, A.; Hartmann, M.; Clark, P.; Sabharwal, A.; Bansal, M.; and Khot, T. 2023. Adapt: As-needed decomposition and planning with language models. *arXiv preprint arXiv:2311.05772*.

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 via self-planning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Ridnik, T.; Kredo, D.; and Friedman, I. 2024. Code generation with alphacodium: From prompt engineering to flow engineering. *arXiv preprint arXiv:2401.08500*.

Saad-Falcon, J.; Lafuente, A. G.; Natarajan, S.; Maru, N.; Todorov, H.; Guha, E.; Buchanan, E. K.; Chen, M.; Guha, N.; Ré, C.; et al. 2024. Archon: An architecture search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*.

Shridhar, M.; Yuan, X.; Côté, M.-A.; Bisk, Y.; Trischler, A.; and Hausknecht, M. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Song, C. H.; Sadler, B. M.; Wu, J.; Chao, W.-L.; Washington, C.; and Su, Y. 2023. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2986–2997. IEEE Computer Society.

Su, J.; Xia, Y.; Shi, R.; Wang, J.; Huang, J.; Wang, Y.; Shi, T.; Jingsong, Y.; and He, L. 2025. Debflow: Automating agent creation via agent debate. *arXiv preprint arXiv:2503.23781*.

Sumers, T.; Yao, S.; Narasimhan, K.; and Griffiths, T. 2023. Cognitive Architectures for Language Agents. *Transactions on Machine Learning Research*.

Tang, N.; Yang, C.; Fan, J.; Cao, L.; Luo, Y.; and Halevy, A. 2023. VerifAI: verified generative AI. *arXiv preprint arXiv:2307.02796*.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023a. Voyager: An open-ended embodied agent with large language models, 2023. *URL https://arxiv. org/abs/2305.16291*.

Wang, X.; Li, C.; Wang, Z.; Bai, F.; Luo, H.; Zhang, J.; Jojic, N.; Xing, E. P.; and Hu, Z. 2024a. PromptAgent: Strategic Planning with Language Models Enables Expert-level Prompt Optimization. In *International Conference on Learning Representations*.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; and Zhou, D. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Wang, Y.; Ma, X.; Zhang, G.; Ni, Y.; Chandra, A.; Guo, S.; Ren, W.; Arulraj, A.; He, X.; Jiang, Z.; et al. 2024b. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*.

Wang, Y.; Wu, Z.; Yao, J.; and Su, J. 2024c. Tdag: A multi-agent framework based on dynamic task decomposition and agent generation. *arXiv preprint arXiv:2402.10178*.

Wang, Z.; Mao, S.; Wu, W.; Ge, T.; Wei, F.; and Ji, H. 2023b. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*.

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.

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

Xie, Y.; Luo, Y.; Li, G.; and Tang, N. 2024b. Haichart: Human and AI paired visualization system. *arXiv preprint arXiv:2406.11033*.

Yang, L.; Yu, Z.; Zhang, T.; Cao, S.; Xu, M.; Zhang, W.; Gonzalez, J. E.; and Cui, B. 2024. Buffer of Thoughts: Thought-Augmented Reasoning with Large Language Models. *arXiv preprint arXiv:2406.04271*.

Yang, Z.; Qi, P.; Zhang, S.; Bengio, Y.; Cohen, W. W.; Salakhutdinov, R.; and Manning, C. D. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Yao, S.; Chen, H.; Yang, J.; and Narasimhan, K. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35: 20744–20757.

Yu, J.; He, R.; and Ying, Z. 2023. THOUGHT PROPAGATION: AN ANALOGICAL APPROACH TO COMPLEX REASONING WITH LARGE LANGUAGE MODELS. In *International Conference on Learning Representations*.

Yue, X.; Ni, Y.; Zhang, K.; Zheng, T.; Liu, R.; Zhang, G.; Stevens, S.; Jiang, D.; Ren, W.; Sun, Y.; et al. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9556–9567.

Zhang, J.; Xiang, J.; Yu, Z.; Teng, F.; Chen, X.; Chen, J.; Zhuge, M.; Cheng, X.; Hong, S.; Wang, J.; et al. 2024a. Aflow: Automating agentic workflow generation. *URL https://arxiv. org/abs/2410.10762*.

Zhang, J.; Zhao, C.; Zhao, Y.; Yu, Z.; He, M.; and Fan, J. 2024b. Mobileexperts: A dynamic tool-enabled agent team in mobile devices. *arXiv preprint arXiv:2407.03913*.

Zheng, C.; Chen, J.; Lyu, Y.; Ng, W. Z. T.; Zhang, H.; Ong, Y.-S.; Tsang, I.; and Yin, H. 2025. Mermaid-Flow: Redefining Agentic Workflow Generation via Safety-Constrained Evolutionary Programming. *arXiv preprint arXiv:2505.22967*.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623.

Zhong, Q.; Wang, K.; Xu, Z.; Liu, J.; Ding, L.; and Du, B. 2024. Achieving¿ 97% on gsm8k: Deeply understanding the problems makes llms better solvers for math word problems. *arXiv preprint arXiv:2404.14963*.

Zhou, A.; Yan, K.; Shlapentokh-Rothman, M.; Wang, H.; and Wang, Y.-X. 2024a. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models. In *International Conference on Machine Learning*.

Zhou, W.; Ou, Y.; Ding, S.; Li, L.; Wu, J.; Wang, T.; Chen, J.; Wang, S.; Xu, X.; Zhang, N.; et al. 2024b. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*.

Zhou, X.; Li, G.; and Liu, Z. 2023. Llm as dba. *arXiv preprint arXiv:2308.05481*.

Zhu, J.-P.; Cai, P.; Xu, K.; Li, L.; Sun, Y.; Zhou, S.; Su, H.; Tang, L.; and Liu, Q. 2024. Autotqa: Towards autonomous tabular question answering through multi-agent large language models. *Proceedings of the VLDB Endowment*, 17(12): 3920–3933.

Zhuge, M.; Liu, H.; Faccio, F.; Ashley, D. R.; Csordás, R.; Gopalakrishnan, A.; Hamdi, A.; Hammoud, H. A. A. K.; Herrmann, V.; Irie, K.; et al. 2023. Mindstorms in natural language-based societies of mind. *arXiv preprint arXiv:2305.17066*.

Zhuge, M.; Wang, W.; Kirsch, L.; Faccio, F.; Khizbullin, D.; and Schmidhuber, J. 2024. Gptswarm: Language agents as optimizable graphs. In *International Conference on Machine Learning*.