LEARNING TO CONFIGURE AGENTIC AI SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Configuring LLM-based agent systems involves choosing workflows, tools, token budgets, and prompts from a large combinatorial design space, and is typically handled today by fixed large templates or hand-tuned heuristics. This leads to brittle behavior and unnecessary compute, since the same cumbersome configuration is often applied to both easy and hard input queries. We formulate *agent configuration* as a query-wise decision problem and introduce **ARC** (**A**gentic **R**esource & **C**onfiguration learner), which learns a light-weight hierarchical policy using reinforcement learning to dynamically tailor these configurations. Across multiple benchmarks spanning reasoning and tool-augmented question answering, the learned policy consistently outperforms strong hand-designed and other baselines, achieving up to 25% higher task accuracy while also reducing token and runtime costs. These results demonstrate that learning per-query agent configurations is a powerful alternative to "one size fits all" designs.

1 INTRODUCTION

Large Language Models (LLMs) have evolved from simple answer predictors to being the backbone of complex *multi-agent systems* capable of iterative planning, tool usage, and multi-step reasoning. In this new paradigm, the performance of an agentic system is governed not only by the underlying LLM but also by the architecture that wraps it: workflows (e.g. voters, verifiers, optimizers), tool availability, information routing, and context management.

Current approaches to agentic architecture design largely rely on static heuristics or "kitchen sink" strategies that flood the context window with extensive history and retrieved evidence. However, this is suboptimal for two reasons. First, performance degrades in long contexts due to the "lost-in-the-middle" phenomenon, where models fail to attend to relevant information Liu et al. (2024); Hong et al. (2025). Second, static systems are inefficient; they fail to adapt their computational footprint to task difficulty. For example, an agentic system designed for multi-hop reasoning might trigger expensive web-search tools and iterative verification loops even for a basic arithmetic query. This "one-size-fits-all" approach leads to wasted compute and unnecessary latency by applying the same heavy resources to trivial and complex inputs.

Ideally, an agent system should *adapt its configuration to each query at hand*. While intuitive, learning such dynamic configurations is difficult because the design space is combinatorial. For instance, consider a system that must answer questions of varying difficulty: Should it invoke a single-shot Chain-of-Thought? Deploy parallel voters to cross-check answers? Allocate a generous token budget, or conserve compute with a minimal response? Even a simple 3-agent system with 5 workflow patterns, 3 independently selectable tools for two agents, and 3 budget levels per agent yields $5 \times (2^3)^2 \times 3^3 = 8{,}640$ structural configurations before prompt selection. Adding only 20 prompt choices already expands this to over 10^5 possible configurations, rendering brute-force search intractable and manual tuning impractical.

To address this, we cast agent configuration as a query-wise decision making problem. A key insight of our work is that agent workflows are not atomic actions, but rather structured processes with varying computational footprints. For instance, a simple inference strategy, such as direct prompting, consumes a single LLM invocation. In contrast, iterative paradigms like an Evaluator-Optimizer loop Madaan et al. (2023) may require multiple sequential calls and variable cost. Standard MDP formulations fall short here because they assume actions have a fixed, discrete duration. We argue that this is fundamentally the wrong abstraction for learning to chose agentic configurations. Instead, we

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

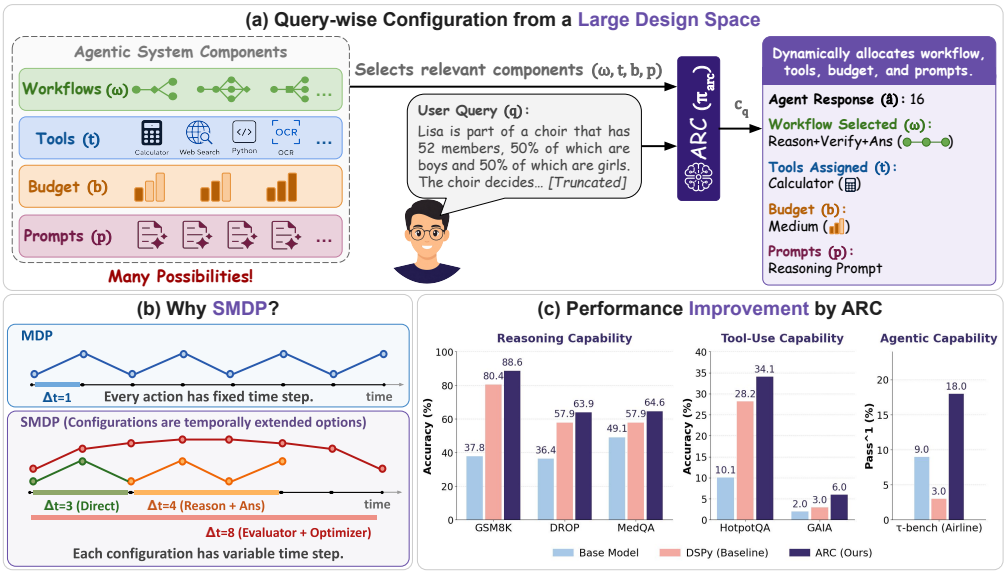

Figure 1: **Overview of ARC.** (a) ARC learns to select a query-specific agent configuration from a large combinatorial design space. (b) Unlike standard MDP actions with fixed duration, agent configurations induce temporally extended executions with variable numbers of LLM calls, motivating our SMDP formulation. (c) Across reasoning, tool-use, and agentic benchmarks, ARC improves performance over budget-matched base models and strong baselines for Qwen 2.5 7B.

model each configuration as an *option* Sutton et al. (1999) within a SMDP. This formulation naturally handles actions with variable durations and costs, and ultimately enables a lightweight controller on top of a frozen agent system to jointly reason about *what* to deploy and *how much compute it will consume*, balancing task accuracy against efficiency in a principled way.

Our contributions are as follows:

1. We formulate agent configuration as SMDP (see Figure 1b), and introduce ARC, a **Hierarchical Reinforcement Learning (HRL) framework** where a high-level policy selects workflows, tools and, budget while a low-level policy composes prompt instructions. Importantly, the learning operates over a lightweight network, without updating the backbone LLM.
2. We propose a **hybrid training pipeline** that combines masked RL with supervised fine-tuning (SFT) on elite trajectories to stabilize learning under sparse rewards.
3. We demonstrate through **extensive experiments** across reasoning, tool-use, and agentic benchmarks that our method achieves significant gains while optimizing cost over static, flat RL-policies and configuration optimization baselines (see Figure 1c).

2 Related Work

We situate our work at the intersection of three research threads: LLM-based agents, prompt and workflow optimization, and hierarchical reinforcement learning.

Large Language Agents LLM-based agents extend language models with iterative decision-making, tool use, and multi-step interaction Yao et al. (2022); Schick et al. (2023). Recent frameworks provide abstractions for tool calling and multi-agent coordination Chase (2022); Wu et al. (2024), while evaluation suites stress realistic long-horizon behavior across general agent benchmarks and web-interaction settings Liu et al. (2023); Zhou et al. (2023). Tool-centric datasets further broaden the space of available actions and APIs Qin et al. (2023), and agents have been instantiated in software engineering interfaces where actions include editing code and navigating repositories Yang et al. (2024b). Despite this progress, most systems rely on manually specified or heuristically tuned workflows with fixed sequencing of reasoning, retrieval, and tool calls. In contrast, our work treats agent configuration as a learnable decision problem, enabling query-wise adaptive workflow and budget selection under resource constraints.

Prompt and Workflow Optimization Early work on adapting LLM behavior focused on the input prompt optimization. Automated frameworks such as OPRO Yang et al. (2023), DSPy Khattab et al. (2023), and GEPA Agrawal et al. (2025) treat the prompt as an optimization variable, iteratively searching for improved instructions. More recently, the optimization target has shifted from static prompts to agentic workflows and context management. Methods like LLMLingua Jiang et al. (2023) and LongLLMLingua Jiang et al. (2024) compress input prompts to fit context windows, while cost-aware frameworks have evolved from model routing Chen et al. (2023) to token-budget-aware reasoning Han et al. (2025). Our work unifies these directions by learning a hierarchical policy that jointly optimizes workflow structure and context budget end-to-end.

Hierarchical Planning and Control While LLM agents are typically viewed through the lens of prompt engineering, our work connects to the rich literature on Hierarchical Reinforcement Learning (HRL). Classic frameworks such as the Options framework Sutton et al. (1999) and Feudal RL Dayan & Hinton (1993) decompose complex tasks into high-level goals and low-level actions, enabling more efficient exploration in large state-action spaces. Recent work has applied HRL to language model fine-tuning and multi-step reasoning Pang et al. (2024), though typically within fixed architectural constraints. We adapt this paradigm by treating workflow selection as the high-level "option" and the selection and ordering of prompt components as the low-level sub-policy, allowing structured search over the combinatorial configuration space that flat RL approaches cannot efficiently navigate.

3 METHODOLOGY

We aim to learn a policy π_{arc} that adapts agent configurations to each input query, balancing correctness against computational cost. Formally, let $\mathcal{D} = \{(q_i, a_i)\}_{i=1}^{N}$ be a dataset of queries q_i and ground-truth answers a_i. As shown in Figure 1a, for each query q in a session, π_{arc} selects a configuration $c_q = (\omega, \mathbf{t}, \mathbf{b}, \mathbf{p})$, where ω represent how "LLMs are connected", $\mathbf{t}$ encodes the enabled tools, $\mathbf{b}$ denotes token-budget tiers, and $\mathbf{p}$ specifies prompt instructions for each agent. Executing the configured system on q produces a response $\hat{a}$, a correctness signal $\mathbb{I}[\hat{a} = a]$, and cost statistics. Through which we define the per-turn utility as $U(q, c) = \mathbb{I}[\hat{a} = a] - \lambda\, C_{\text{cost}}(c)$, where $C_{\text{cost}}(c)$ aggregates token usage and runtime, and $\lambda \geq 0$ controls the accuracy efficiency trade-off.

The challenge here is that different workflows impose fundamentally different computational commitments: a direct call consumes one LLM invocation, while an evaluator-optimizer (Appendix A) loop may require up to seven. A standard MDP treats all configuration choices as instantaneous actions and cannot represent this variation. Therefore, we model agent configurations as an *option* (Sutton et al., 1999) i.e., a temporally extended action whose duration depends on the chosen workflow. This yields a semi-Markov decision process (SMDP) in which the controller reasons about both what to deploy and what temporal cost each configuration commits the session to.

3.1 MODELING THE DECISION PROCESS OVER OPTIONS

The agent configuration problem is a semi-Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{O}, P, R, \gamma)$ where:

- **State space** $(\mathcal{S})$: For query q, the state $s_q = [\phi(q); \mathbf{f}_q]$ concatenates a semantic embedding $\phi(q)$ (MetaCLIP-H/14; Xu et al. (2024)) with hand-crafted features $\mathbf{f}_q$ encoding query length, numerical density, and binary indicators for multi-step reasoning and tool use. See Appendix D for the embedding selection ablation.
- **Options** $(\mathcal{O})$: Each option $o = (\mathcal{I}_o, \pi_{arc}, \beta_o)$ represents a complete act of configuring and executing the agent system on a query.:
  - *Initiation set* $\mathcal{I}_o \subseteq \mathcal{S}$: the set of states in which option o may be invoked. After action masking prunes structurally invalid configurations (Section 3.2), all surviving options share $\mathcal{I}_o = \mathcal{S}$.
  - *Policy* π_{arc}: given state s_q, selects a structural configuration $(\omega, \mathbf{t}, \mathbf{b})$ and autoregressively composes prompt instructions $\mathbf{p}$ from a library of semantic fragments $\mathcal{P}$, yielding the full configuration $c = (\omega, \mathbf{t}, \mathbf{b}, \mathbf{p})$.
  - *Termination* β_o: deterministic for fixed-structure workflows (e.g., Direct terminates after one LLM call, Reason+Answer after two) and stochastic for iterative workflows (e.g., Evaluator-Optimizer terminates upon convergence or at $K_{\max}$ iterations).
- **Transition dynamics** $P(s', k \mid s, o)$: joint transition-duration kernel giving the probability of reaching state s' after k steps under option o from state s. Executing option o in state s_q runs the full workflow, consuming $k_o \in \{1, \ldots, K_{\max}\}$ LLM calls and $n_{\text{tokens}}(o)$ tokens.

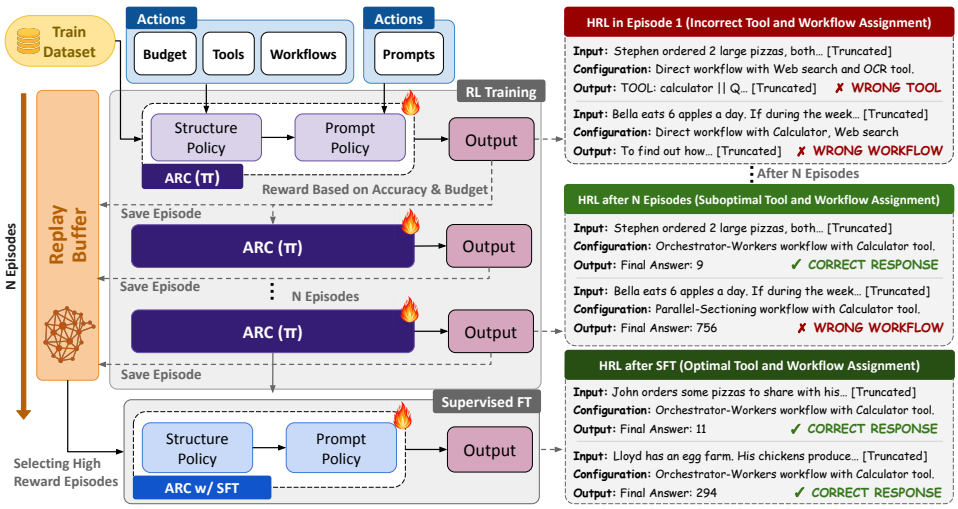

Figure 2: **Training pipeline.** The structure policy selects workflows, tools, and budgets while the prompt policy composes instructions. During RL training, episodes are stored in a memory buffer. After RL converges, high-reward episodes are filtered and used for supervised fine-tuning (SFT), which consolidates successful strategies and improves consistency.

- **Reward** $R(s, o)$: scalar reward received upon option termination, incorporating correctness and duration-dependent costs (Eq. 3).
- **Discount factor** $\gamma = 0.9$: applied per primitive step (γ^{k_o}), so heavier workflows incur steeper discounting on subsequent turns.

The combinatorial size of the configurations makes direct optimization of π_{arc} intractable. Therefore, π_{arc} is decomposed into 1) a *structure policy* π_{struct} that selects the workflow, tools, and budget, and 2) a *prompt policy* π_{prompt} that composes instructions conditioned on the structural choice:

$$\pi_{arc}(c \mid s_q) \;=\; \pi_{\text{struct}}(\omega, \mathbf{t}, \mathbf{b} \mid s_q) \;\cdot\; \pi_{\text{prompt}}(p \mid s_q, \omega, \mathbf{t}, \mathbf{b}) \tag{1}$$

This hierarchy replaces a single joint decision over $|\mathcal{O}|$ configurations with sequential structural and prompt decisions. Action masking (Section 3.2) further prunes invalid branches during training.

Structure Policy. The structure policy π_{struct} selects one option per query based on the current state s_q, where each option corresponds to a composite action $(\omega, \mathbf{t}, \mathbf{b})$. Workflows are drawn from established agentic patterns (Anthropic, 2024), spanning direct, sequential (Chain-of-Thought), parallel (Voting), and iterative (Evaluator-Optimizer) topologies. Tools such as calculators, web search, and code execution can be independently enabled or disabled per agent, and output budget tiers control the token allocation for each agent in the pipeline.

Prompt Policy. The prompt policy π_{prompt} operates as a sequential decision-making process, conditioned on the structural choice $(\omega, \mathbf{t}, \mathbf{b})$. Its role is to operationalize the selected option by defining the instructions for each agent. At each step, π_{prompt} selects a fragment from $\mathcal{P} \cup \text{STOP}$. The composed instructions complete the configuration $c = (\omega, \mathbf{t}, \mathbf{b}, \mathbf{p})$, which is then executed. We use a separate discount $\gamma_{\text{prompt}} = 0.5$ for the prompt composition steps: since the reward is observed only after execution, the aggressive discount ensures that credit concentrates on the first few fragment selections and penalizes long, redundant instruction sequences. To ensure high-quality instructions, $\mathcal{P}$ is augmented with dataset-specific fragments generated via meta-prompting. Our ablations (Appendix E) demonstrate that generation using GPT-5.2 outperforms other language models.

3.2 Training Procedure

We optimize both policies end-to-end using Proximal Policy Optimization (PPO) (Schulman et al., 2017) as shown in Figure 2. Each policy maintains a separate value network for advantage estimation, and we apply per-batch advantage normalization to stabilize learning across the heterogeneous reward landscape. The following objective is applied independently to each factor of π_{arc}:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi(a_t|s_t)}{\pi^{\text{old}}(a_t|s_t)}\hat{A}_t,\;\; \text{clip}\left(\frac{\pi(a_t|s_t)}{\pi^{\text{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{2}$$

where $\hat{A}_t$ is the estimated advantage, with $\pi = \pi_{\text{struct}}$ operating over Options and $\pi = \pi_{\text{prompt}}$ operating over prompt fragments. We add entropy regularization to encourage exploration in the early stages of training.

Reward Design. Designing rewards for variable-duration actions is non-trivial: optimizing correctness alone leads to over-provisioning (allocating maximum resources), while pure efficiency metrics sacrifice accuracy. We decompose the reward R into three interpretable terms:

$$R(s, o) = \alpha \cdot \mathbb{I}[\text{correct}] \; - \; \beta_s \cdot k_o - \beta_t \cdot n_{\text{tokens}}/T_{\max} \; + \; \eta \cdot \mathcal{R}_{\text{tool}}, \tag{3}$$

where k_o is the number of LLM calls taken by option o, n_{tokens} is total token consumption normalized by the maximum budget $T_{\max}$, and $\alpha, \beta_s, \beta_t, \eta$ are weighting coefficients. The duration penalty $\beta_s \cdot k_o$ penalizes each option proportionally to its actual temporal commitment,for selecting shorter options when they suffice without penalizing complex workflows that require difficult queries.

The tool shaping term $\mathcal{R}_{\text{tool}}$ addresses a key challenge: the structure policy allocates tools, but the downstream LLM decides whether to invoke them. Naively rewarding tool allocation creates a mismatch: tools may be provisioned but never used. We design an asymmetric reward:

Action Masking. The raw option space ($|\mathcal{O}| = 9 \times 16^2 \times 3^3 = 62{,}208$) contains structurally invalid configurations. For example, the Direct workflow employs a single agent, yet $\mathcal{O}$ permits allocation of tools and budgets to a second agent dimension. We employ action masking to prune such invalid combinations, reducing $|\mathcal{O}|$ to $41{,}904$ valid options (a 32.6% reduction).

During action sampling, once π_{struct} selects a workflow ω, we apply a conditional mask that excludes incompatible choices by setting their logits to $-\infty$ prior to the softmax. Formally, for action dimension i with logits $\mathbf{z}_i$, we apply a mask $\mathbf{m}_i(\omega) \in \{0, 1\}^{|\mathcal{A}_i|}$ conditioned on the selected workflow, yielding masked logits $\tilde{\mathbf{z}}_i = \mathbf{z}_i + \log\big(\mathbf{m}_i(\omega)\big)$. The same mask is applied during policy updates, ensuring that ρ_t in Eq. 2 is computed over consistent, valid distributions.

Algorithm 1: Training Pipeline (SMDP → SFT)

Require: Dataset $\mathcal{D}$, policies π_{struct}, π_{prompt}, buffer $\mathcal{B} = \emptyset$
Initialize policies and value networks V_{struct}, V_{prompt}
for $t = 1$ to T_{RL} **do**
  for each $q \sim \mathcal{D}$ **do**
    Encode $s_q = [\phi(q); \mathbf{f}_q]$; sample $o \sim \pi_{\text{struct}}(\cdot|s_q)$
    Apply mask based on workflow ω
    Compose prompts $\mathbf{p}$ via π_{prompt}
    Execute workflow $c = (\omega, \mathbf{t}, \mathbf{b}, \mathbf{p})$ for k_o steps
    Compute reward R; store (s_q, o, k_o, R) in $\mathcal{B}$
  end for
  Compute advantages $\hat{A}_t$; normalize per batch
  for $e = 1$ to E **do**
    Update θ via PPO clipped objective with entropy regularization
  end for
end for
Extract elite: $\mathcal{D}_{\text{elite}} = \{(s_i, o_i^*) \in \mathcal{B} : \text{correct}_i \wedge R_i \geq \tau\}$
for $e = 1$ to E_{SFT} **do**
  Update θ via SFT: $\max \mathbb{E}_{(s,o^*)\sim\mathcal{D}_{\text{elite}}}[\log \pi_{\text{struct}}(o^*|s)]$
end for
return $\pi_{\text{struct}}^{\text{SFT}}$, $\pi_{\text{prompt}}^{\text{SFT}}$

3.3 Post-Training Refinement

Although RL effectively explores the combinatorial design space, high-variance gradient estimates can leave the final policy with residual stochasticity. We introduce a supervised fine-tuning (SFT) refinement phase that runs after the RL completion. **SFT is computationally inexpensive**: it fine-tunes only the lightweight policy networks (not the LLM) via supervised learning on a subset of the RL buffer, avoiding the expensive rollouts required during RL. We filter $\mathcal{B}$ to obtain elite trajectories that are both correct and exceed a reward threshold: $\mathcal{D}_{\text{elite}} = \{(s_i, o_i^*) \in \mathcal{B} : \text{correct}_i \wedge R(s_i, o_i^*) \geq \tau\}$, where τ retains the top 30% of episodes by reward. We fine-tune both policies via maximum likelihood on the elite demonstrations: $\max_\theta \; \mathbb{E}_{(s,o^*)\sim\mathcal{D}_{\text{elite}}}[\log \pi_{\text{struct}}(o^*|s)]$. This distillation concentrates the policy on configurations that succeeded during training, providing formal guarantees (Theorem 3.2).

3.4 Theoretical Guarantees

We establish two theoretical results: the SMDP formulation admits a well-defined unique optimal policy that RL can recover (Theorem 3.1), and SFT concentrates the final policy on high-reward configurations with a formal performance guarantee (Theorem 3.2).

Theorem 3.1 (SMDP Convergence). *Let $\mathcal{T}$ be the SMDP Bellman optimality operator:*

$$(\mathcal{T}Q)(s,o) \;=\; \mathbb{E}\Big[R(s,o)+\gamma^{k_o}\max_{o'\in\mathcal{O}}Q(s',o')\;\Big|\;s,o\Big].$$

Then: (i) *$\mathcal{T}$ is a γ-contraction in ℓ_∞ norm, so a unique fixed point Q^* exists; and* (ii) *under Robbins–Monro step sizes ($\sum_n \alpha_n = \infty$, $\sum_n \alpha_n^2 < \infty$), infinite visitation of all (s,o) pairs, and bounded durations $1 \le k_o \le K_{\max}$, SMDP Q-learning iterates converge: $Q_n \to Q^*$ almost surely.*

Proof sketch. Since every Option executes at least one LLM call ($k_o \ge 1$), we have $\gamma^{k_o} \le \gamma < 1$. The contraction follows: $\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty \le \gamma\|Q_1 - Q_2\|_\infty$. Existence and uniqueness of Q^* is immediate from the Banach fixed-point theorem. Convergence of the stochastic iterates follows from the extension of Jaakkola et al. (1993) to SMDPs Sutton et al. (1999). Full proof in Appendix C. □

Theorem 3.2 (Policy Concentration). *Let $\mathcal{C}_{\text{elite}}(s) = \{c : (s,c) \in \mathcal{D}_{\text{elite}}\}$. Under sufficient model capacity, the refined policy π_{arc}^{SFT} satisfies:* (i) $\pi_{arc}^{\text{SFT}}(c|s) > 0 \Rightarrow c \in \mathcal{C}_{\text{elite}}(s)$ (support restriction)*, and* (ii) $\mathbb{E}_{c\sim\pi_{arc}^{\text{SFT}}}[R(s,c)] \ge \tau$ (reward floor).

Proof sketch. The MLE objective minimizes $D_{\text{KL}}(\hat{p}_{\text{elite}}\|\pi_{arc})$, attaining zero when $\pi_{arc} = \hat{p}_{\text{elite}}$. Support restriction follows because $\hat{p}_{\text{elite}}(c|s) = 0$ for $c \notin \mathcal{C}_{\text{elite}}(s)$. The reward floor holds because every $(s,c) \in \mathcal{D}_{\text{elite}}$ satisfies $R(s,c) \ge \tau$ by construction. Full proof in Appendix C. □

Theorem 3.1 establishes that the SMDP has a unique optimal value function Q^*, so RL targets a well-defined optimum and the elite buffer contains near-optimal configurations. Theorem 3.2 guarantees that SFT then concentrates π_{arc} onto these configurations, establishing a performance floor at τ. We also validate this empirically in Section 4.5

4 Experiments

Our experiments investigate three key aspects of the proposed framework ARC: **(RQ1)** Does the learned configuration outperform fixed architectures? **(RQ2)** Does adaptive allocation improve efficiency? **(RQ3)** Can these learned configuration be transferred across task and parameters?

4.1 Experimental Setup

Benchmarks. We evaluate on six benchmarks spanning across three primary capability axes: *Reasoning Capability*, which includes GSM8k Cobbe et al. (2021), DROP Dua et al. (2019), and MedQA Jin et al. (2021); *Tool-Use Capability*, comprising HotPotQA Yang et al. (2018) and GAIA Mialon et al. (2023); and *Agentic Capability* on τ-Bench (Airline) Yao et al. (2024). We employ standard training split for policy learning and evaluate on the test (or validation) set. For GAIA, we partitioned the validation set, using the first 65 samples for training and the rest for evaluation. More details are provided in Appendix F.

Baselines. We compare our framework and its non SFT-refined variant against several baselines: (1) *Base models with tools*, utilizing off-the-shelf LLMs evaluated under max-token constraints and restricted to non-iterative tool calls; (2) *Search-based methods*, employing Grid and Greedy search to establish upper bounds on fixed strategies; (3) *Workflow/Prompt optimization frameworks*, including AutoGen Wu et al. (2024), DSPy Khattab et al. (2023), GEPA Agrawal et al. (2025), AFlow Zhang et al. (2025), and LLM as A Policy (LAP); and (4) *RL baselines*, which treat configuration as either a bandit problem (RL Bandits) or a sequential decision process (RL Episodes) optimized via PPO.

Implementation. We provide a modular framework supporting arbitrary tool registries, custom workflows, and n-agent topologies. Our default experimental setup uses 9 workflow patterns (details in Appendix A), 4 tools, and 3 agents. We evaluate both Qwen 2.5 7B Instruct Yang et al. (2024a) and Gemini 2.5 Flash Lite Comanici et al. (2025), focusing on Qwen in the main text. Full training details and additional results are included in Appendix F.

4.2 RQ1: Does Learning Configuration Improve Performance?

Table 1 and 2 summarizes the performance across the reasoning, tool-use, and agentic benchmarks. We observe consistent improvements with ARC over baseline methods.

Table 1: **Performance comparison across reasoning and tool-use benchmarks.** We compare our approach (ARC and ARC w/o SFT) against base models with tools, search-based methods (Grid/Greedy Search), optimization frameworks (AutoGen, DSPy, GEPA, LLM as a policy (LAP)), and flat PPO policy baselines (RL Bandits, RL Episodes). Our method achieves the best results on most tasks. **Bold** and underline indicates best and second best performance over each task.

		Reasoning Capability (↑)				Tool-Use Capability (↑)		
Method	**Model**	**GSM8k**	**DROP**	**MedQA**	**Avg / Δ**	**HotpotQA**	**GAIA**	**Avg / Δ**
Base Model w/ Tools *	Qwen	37.8%	36.4%	49.1%	41.1%	10.1%	2.0%	6.1%
	Gemini	62.3%	45.9%	10.0%	39.4%	18.8%	3.0%	10.9%
Grid Search	Qwen	74.0%	54.3%	53.1%	60.5% (+19.4)	27.9%	1.0%	14.45% (+8.35)
Greedy Search	Qwen	78.2%	57.5%	54.9%	63.5% (+22.4)	28.6%	3.0%	15.8% (+9.7)
AutoGen	Qwen	74.8%	58.0%	70.5%	67.8% (+26.7)	**34.1%**	1.0%	17.55% (+11.45)
DsPy	Qwen	80.4%	57.9%	57.9%	65.4% (+24.3)	28.2%	3.0%	15.6% (+9.5)
GEPA	Qwen	83.6%	39.3%	**87.1%**	70.0% (+28.9)	27.4%	4.0%	15.7% (+9.6)
Aflow	Qwen	86.5%	56.2%	60.3%	67.7% (+26.3)	30.3%	1.0%	15.65% (+9.55)
LAP	Qwen	38.3%	38.4%	49.0%	41.9% (+0.8)	10.2%	0.0%	5.1% (−1.0)
LAP - FS	Qwen	46.1%	44.9%	49.4%	46.8% (+5.7)	10.5%	1.0%	5.75% (−0.35)
RL Bandits	Qwen	80.0%	54.3%	56.9%	63.7% (+22.6)	27.3%	2.0%	14.65% (+8.55)
	Gemini	74.7%	54.5%	58.8%	62.7% (+23.3)	33.4%	3.0%	18.2% (+7.3)
RL Episodes	Qwen	85.2%	54.3%	57.3%	65.6% (+24.5)	27.8%	2.0%	14.9% (+8.8)
	Gemini	71.0%	59.0%	60.9%	63.6% (+24.2)	30.5%	1.0%	15.75% (+4.85)
ARC w/o SFT	Qwen	87.6%	62.3%	58.4%	69.4% (+28.3)	33.7%	5.0%	19.35% (+13.25)
	Gemini	85.5%	61.3%	62.8%	69.9% (+30.4)	33.8%	3.0%	18.4% (+7.5)
ARC	Qwen	**88.6%**	**63.9%**	64.6%	**72.4%** (+31.3)	**34.1%**	**6.0%**	**20.05%** (+13.95)
	Gemini	**88.5%**	**65.6%**	**64.7%**	**72.9%** (+33.5)	**35.7%**	**4.0%**	**19.85%** (+8.95)

**Base model constrained to match the maximum per-query budget available to learned policies.*

Reasoning Tasks. On GSM8K, ARC achieves 88.6% accuracy with Qwen, outperforming GEPA (83.6%) and RL Episodes (85.2%). On DROP, ARC reaches 63.9% with Qwen and 65.6% with Gemini, improving over the base models by 27.5 and 19.7%, respectively. On MedQA, GEPA performs best (87.1%), while ARC reaches 58.4%, still improving over the base model by 15.5 points. We attribute this gap to *prompt content*; GEPA uses a domain-specific system prompt with ∼1,100 tokens of medical reasoning heuristics, whereas our prompt library uses general-purpose fragments each averaging ∼50 tokens. This suggests that in knowledge-intensive domains, prompt semantics can dominate structural choices. Nevertheless, ARC still outperforms RL and search baselines, supporting the value of hierarchical decision-making; combining structural optimization with domain-specific prompts is a promising direction.

Tool-Use and Agentic Tasks. On HotpotQA, ARC achieves 34.1% with Qwen, matching AutoGen (34.1%) and outperforming the strongest RL baseline by 6.3%. On GAIA, ARC reaches 6.0% with Qwen and 4.0% with Gemini, improving over the base models by 4.0 and 1.0 points, respectively. Notably on τ-Bench (Airline), ARC doubles base-model Passˆ1 performance (9.0% → 18.0%), while DSPy reduces performance, likely because fixed prompt instructions hinder adaptation in interactive tasks. Overall, these gains show that learned configurations can better allocate tools and budgets for retrieval and agentic tasks.

Table 2: **Performance on τ-Bench (Airline).** We evaluate using Qwen 2.5. Each result reports Passˆk, where larger k measures whether the agent succeeds consistently across trials runs.

	Airline Task (↑)		
Method	**Passˆ1**	**Passˆ2**	**Passˆ3**
Base Model *	9.0%	4.5%	1.5%
DSPy	3.0%	0.5%	0.0%
ARC w/o SFT	16.0%	7.5%	3.0%
ARC	**18.0%**	**11.0%**	**6.5%**

**Base model constrained to match the ARC's per-query budget.*

Model Agnostic. As shown in Table 1, we observe consistent gains across Qwen 2.5 (open-weights) and Gemini 2.5 (proprietary) validates the generalizability of our framework.

4.3 RQ2: Does Adaptive Allocation Improve Efficiency?

A key advantage of ARC is *resource allocation*: our policy can deploy expensive workflows (e.g., multi-agent verification, iterative refinement) when necessary, while using lightweight strategies for simpler queries. We quantify this in terms of token consumption and workflow complexity.

Table 3: **Workflow diversity across datasets.** We report the number of unique workflows (UW), entropy, and Gini coeff. for different configuration strategies. Higher entropy and lower Gini indicate more diverse exploration of the workflows.

Dataset	Method	UW (↑)	Entropy (↑)	Gini
HotpotQA	Grid Search	5	0.48	0.74
	Greedy Search	2	0.24	0.46
	ARC (w/o SFT)	9	2.13	0.58
GSM8k	Grid Search	3	0.98	0.45
	Greedy Search	2	0.24	0.46
	ARC (w/o SFT)	9	1.90	0.67
MedQA	Grid Search	3	1.00	0.44
	Greedy Search	2	0.24	0.46
	ARC (w/o SFT)	9	2.95	0.30

Table 4: **Policy transfer across datasets for Qwen.** $S_T \rightarrow S_N$: accuracy on training dataset transferring to zero-shot accuracy on new dataset; D_{sim}: dataset similarity measured via embedding cosine distance. Policies show moderate transfer to reasoning tasks, with performance degradation of $6-7\%$, while transfer for tool-use tasks is more weaker.

Capability	Train → New	$S_T \rightarrow S_N$	D_{sim}
Reasoning	GSM8k → DROP	88.6 → 63.0	0.79
	GSM8k → MedQA	88.6 → 57.0	0.81
	DROP → GSM8k	63.9 → 76.0	0.79
Tool-Use	HotpotQA → GAIA	34.1 → 2.0	0.93
	GAIA → HotpotQA	6.0 → 19.0	0.93
	HotpotQA → MedQA	34.1 → 57.0	0.76

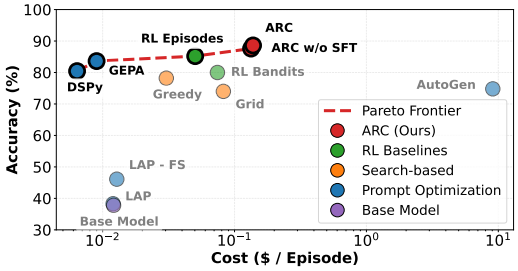

Figure 3: **Accuracy Vs. Cost trade-off on GSM8K.** Each point shows average accuracy vs. inference cost. The dashed line denotes the Pareto frontier, representing methods that achieve the best possible accuracy for a given cost.

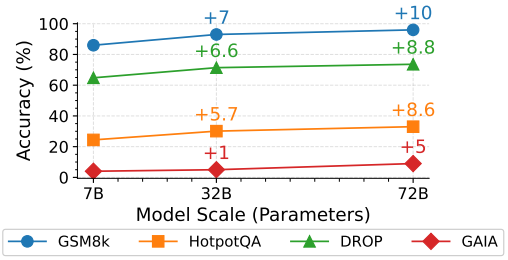

Figure 4: **Scaling trends of model accuracy with capacity.** Accuracy as a function of model size for the Qwen 2.5 family (7B, 32B, 72B) across four benchmarks. Performance improves consistently with scale, with gains varying by task complexity.

Figure 3 shows the accuracy–cost trade-off across methods. Cost is computed as token-based API cost per episode using OpenRouter rates for Qwen2.5 7B Instruct. The dashed curve denotes the Pareto frontier, computed via pairwise dominance. Fixed and search-based baselines lie along this frontier, reflecting the expected trade-off between accuracy and token expense. In contrast, ARC occupies Pareto-optimal regions that strictly improve upon prior methods, achieving high accuracy at lower cost. This indicates that instance-specific adaptation yields more efficient accuracy–cost trade-offs than uniform resource allocation or naive search strategies. Consistent with this observation, Table 3 shows that ARC explores a diverse set of workflows.

4.4 RQ3: Can Policies Transfer Across Tasks and Model Capacity?

We investigate the generalization capabilities of our learned configurations across two dimensions: distinct task domains (to test structural adaptability) and varying model sizes (to test scalability).

Task Specificity Shapes Cross-Domain Transfer. Table 4 reports zero-shot transfer of policies trained on a source dataset (S_T) and evaluated on a target subset dataset of (S_N). For reasoning tasks, transfer preserves most in-domain performance: ARC trained on GSM8k achieves 63.0% on DROP (vs. 63.9% in-domain) and 57.0% on MedQA (vs. 64.7%), indicating only moderate degradation. In contrast, tool-use transfer depends strongly on tool overlap. ARC trained on HotpotQA transfer reasonably to MedQA (57.0% vs. 64.7%), where both rely on web retrieval, but perform poorly on GAIA (2.0% vs. 6.0%), which requires fundamentally different multimodal tools. This suggests that policy transfer is governed more by workflow and tool structure than by semantic similarity alone.

Scaling with Model Capacity. In contrast to task transfer, we observe zero-shot generalization across model scales. We trained the ARC using the Qwen 2.5 7B backbone and evaluated it directly on the 32B and 72B variants of the same family. As shown in Figure 4, performance improves with model capacity across tasks (Avg. Kendall's $\tau = 1.0$ and Pearson's $r = 0.94$). This indicates that the structural priors learned on smaller models are largely invariant to scale.

Table 5: **Comparison of training objectives across models.** PPO w/ shaped rewards outperforms GRPO, while SFT provides better generalization than DPO.

	Method	Model	GSM8k (↑)	GAIA (↑)	Avg (↑)
RL	PPO	Qwen	87.6%	5.0%	46.3%
		Gemini	85.5%	3.0%	44.2%
	GRPO	Qwen	82.3%	3.0%	42.6%
		Gemini	88.9%	2.0%	45.4%
post-train	SFT	Qwen	88.6%	6.0%	47.3%
		Gemini	88.5%	4.0%	46.2%
	DPO	Qwen	85.7%	4.0%	44.85%
		Gemini	80.1%	2.0%	41.1%

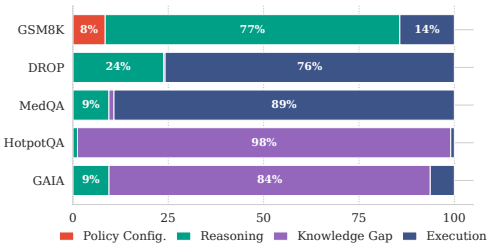

Figure 5: **Error distribution.** Reasoning tasks exhibit primarily reasoning errors, while tool-use tasks are dominated by knowledge gap errors. Policy configuration errors remain minimal (<10%) across all datasets.

4.5 Ablation Studies

Why SFT refinement? Comparing ARC and ARC w/o SFT rows in Table 1 and 2, the SFT phase consistently yields 1–3% accuracy gains. Beyond accuracy, **SFT improves average episode reward by ≈ 5–35%** across all datasets and models, validating Theorem 3.2: distillation to elite trajectories raises the performance floor, this refinement distills successful strategies from the RL buffer.

Alternative training objectives. We compared our PPO-based training against GRPO. PPO with shaped rewards outperformed GRPO: GRPO struggled with the sparse reward signal from correctness alone. We did not compare against value-based methods due to the high-dimensional action space and the need for efficient exploration. For post-training refinement, we evaluated DPO (direct preference optimization) as an alternative to SFT. Although pairwise comparisons can be constructed from the RL buffer for DPO, this approach exhibits overfitting on the training set, leading to degraded generalization performance compared to SFT. Details are in Table 5 and Appendix H.

Search-based alternatives. Table 1 includes grid and greedy search baselines. Grid search achieves 74.0%, substantially below ARC (88.5%). Greedy search performs even worse (36.0%), confirming that naive exploration of the configuration space is insufficient and shows the necessity of learning-based approaches for tractably navigating the combinatorial design space.

4.6 Error Analysis

We categorize errors across five benchmarks (reasoning and tool-use) into four types: (1) *policy configuration errors*, where ARC selects suboptimal workflows or tools; (2) *reasoning errors*, where the LLM applies incorrect logic; (3) *knowledge gap*, where the LLM fails to retrieve correct information; and (4) *execution errors*, including arithmetic mistakes and extraction failures. (see Appendix I for detection criteria)

Figure 5 reveals distinct error profiles of ARC. On reasoning tasks (GSM8k), 77% of errors are reasoning failures with only 9% from policy misconfiguration. On tool-use tasks (HotpotQA, GAIA), knowledge gap errors dominate (84-98%). Critically, policy configuration errors remain below 10% across all benchmarks, indicating the learned policy effectively adapts to query requirements.

5 Conclusion

We introduced ARC, a hierarchical RL framework for query-adaptive configuration of LLM-based agents. ARC formulates agent configuration as an SMDP, where workflows, tools, budgets, and prompts define temporally extended options, and decomposes this large combinatorial space into a tractable two-level policy. By combining masked reinforcement learning with supervised fine-tuning on high-reward trajectories, ARC learns to efficiently select query-specific configurations without updating the backbone LLM. Across reasoning, tool-use, and agentic benchmarks, ARC consistently improves over budget-matched tool-augmented LLMs, increasing average reasoning accuracy by 31.3%, tool-use accuracy by 13.95%, and doubling τ-Bench (Airline) Pass^1 success from 9.0% to 18.0%. These results highlight the value of learning hierarchical, query-wise agent configurations for improving both performance and resource allocation. More broadly, ARC shows that adaptive architectural decision-making is a practical direction for building flexible and compute-efficient LLM-based agent systems.

AI USE STATEMENT

We used generative AI tools only to polish the writing for readability and to check for typographical and grammatical errors in author-written text. All edits were reviewed by the authors, and no technical claim, result, or citation was altered. We did not use generative AI tools for any task on the required-disclosure list. We take responsibility for the final content of this work.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Anthropic. Building effective agents. `https://www.anthropic.com/research/building-effective-agents`, 2024. Accessed: 2025-01-14.

Harrison Chase. Langchain, 2022. URL `https://github.com/langchain-ai/langchain`. Accessed: 2025-01-12.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1993.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, 2025.

Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts llm performance. Technical report, Chroma, July 2025. URL `https://research.trychroma.com/context-rot`.

Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems*, 6, 1993.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677, 2024.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Andreas Koukounas, Georgios Mastrapas, Michael Günther, Bo Wang, Scott Martens, Isabelle Mohr, Saba Sturua, Mohammad Kalim Akram, Joan Fontanals Martínez, Saahil Ognawala, Susana Guzman, Maximilian Werk, Nan Wang, and Han Xiao. Jina clip: Your clip model is also your text retriever, 2024.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In *Findings of the association for computational linguistics: ACL 2022*, pp. 1864–1874, 2022.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization. In *arXiv preprint arXiv:2404.19733*, 2024.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

Amanpreet Singh, Ronghang Hu, Vedanuj Goswami, Guillaume Couairon, Wojciech Galuba, Marcus Rohrbach, and Douwe Kiela. Flava: A foundational language and vision alignment model. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15638–15650, 2022.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 33: 16857–16867, 2020.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Hu Xu, Saining Xie, Xiaoqing Tan, Po-Yao Huang, Russell Howes, Vasu Sharma, Shang-Wen Li, Gargi Ghosh, Luke Zettlemoyer, and Christoph Feichtenhofer. Demystifying clip data. In *International Conference on Learning Representations*, volume 2024, pp. 47812–47831, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986, 2023.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A AGENTIC WORKFLOWS

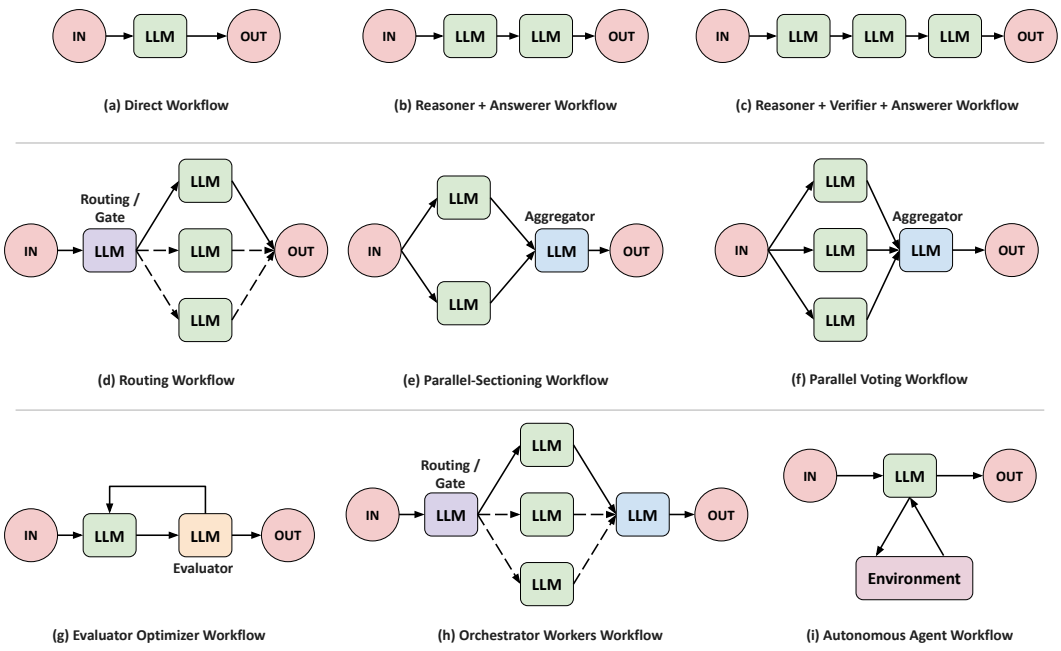

Figure 6: Overview of the nine agentic workflows: Direct (0), Reason+Ans (1), Reason+Verify+Ans (2), Routing (3), Parallel-Sectioning (4), Parallel-Voting (5), Orchestrator-Workers (6), Evaluator-Optimizer (7), and Autonomous-Agent (8). Each workflow defines a distinct pattern of LLM calls and agent interactions.

Our framework supports nine agentic workflows, ranging from single-call baselines to multi-agent orchestration patterns. Each workflow specifies a computation graph over LLM calls, with configurable tool access and token budgets per agent role. Below we summarize all workflows concisely.

Workflow 0: Direct. A single LLM call produces the final answer directly. Tools may optionally be used. **LLM Calls:** 1, **Agent2 Tools:** Not used.

Workflow 1: Reason + Answer. The Reasoner generates intermediate reasoning, which the Answerer converts into a final response. **LLM Calls:** 2, **Agent2 Tools:** Not used.

Workflow 2: Reason + Verify + Answer. An explicit verification step critiques the reasoning before answer synthesis. **LLM Calls:** 3, **Agent2 Tools:** Verifier.

Workflow 3: Routing.A router dispatches the query to one of two specialized reasoners with different tool configurations. **LLM Calls:** 3, **Agent2 Tools:** Conditional (Reasoner2).

Workflow 4: Parallel Sectioning. The question is decomposed into independent subtasks solved in parallel and then aggregated. **LLM Calls:** 4, **Agent2 Tools:** Worker1.

Workflow 5: Parallel Voting. Multiple independent attempts are aggregated via majority voting. **LLM Calls:** 4, **Agent2 Tools:** Not used.

Workflow 6: Orchestrator-Workers. The Orchestrator delegates subtasks to workers, separating planning from execution. **LLM Calls:** 4, **Agent2 Tools:** Workers.

Workflow 7: Evaluator-Optimizer. An iterative generate–evaluate–refine loop runs until convergence or a maximum number of iterations. **LLM Calls:** 4-7, **Agent2 Tools:** Evaluator.

Workflow 8: Autonomous Agent. The agent iteratively reasons and invokes tools without an explicit evaluator. **LLM Calls:** 4, **Agent2 Tools:** Iterations 2+.

B Computational Resources

This section summarizes the computational resources required to reproduce our experiments (training, evaluation, and plotting). We support two execution modes: *local* (running an open-weight LLM on your own GPU) and *API* (calling an external provider such as OpenRouter[1]). The codebase works on a Python 3.10+ installation and a CUDA-capable machine when using local models.

For local (HuggingFace) inference and training, it is sufficient to have a single modern GPU for 7B-class models such as Qwen 2.5 7B-Instruct, together with at least 64 GB of CPU RAM to store datasets and logs. Models in the 1.5B–3B range can typically be trained and evaluated on commodity GPUs with 24 GB of VRAM, which is adequate for ablations and scaling experiments. For larger models (14B parameters and above), practical training and evaluation usually require either multiple GPUs with $\geq$40 GB VRAM each or aggressive quantization; in such cases, it is often preferable to access hosted checkpoints via an API. Under these conditions, reproducing the GSM8K and DROP results with a 7B-class model is feasible on a single A100/V100/4090-class GPU with 40 GB VRAM (or 24 GB with 4-bit quantization) and 64 GB CPU RAM.

In API mode no accelerator is needed; the limiting factors are latency, concurrency limits, and monetary budget. Our evaluation scripts support parallel workers so that, when the provider permits concurrent requests, hundreds to thousands of evaluation episodes can be processed in a few hours from a single machine.

C Proofs of Theoretical Guarantees

C.1 Proof of Theorem 3.1

Proof. **(i) Contraction.** Let Q_1, Q_2 be bounded. For any (s, o):

$$|(\mathcal{T}Q_1)(s,o) - (\mathcal{T}Q_2)(s,o)| = \left|\mathbb{E}\Big[\gamma^{k_o}\Big(\max_{o'} Q_1(s',o') - \max_{o'} Q_2(s',o')\Big)\Big|s,o\Big]\right| \tag{4}$$

$$\leq \mathbb{E}\Big[\gamma^{k_o} \max_{o'} |Q_1(s',o') - Q_2(s',o')|\Big|s,o\Big] \tag{5}$$

$$\leq \mathbb{E}[\gamma^{k_o} \mid s,o]\, \|Q_1 - Q_2\|_\infty. \tag{6}$$

Since $k_o \geq 1$, we have $\mathbb{E}[\gamma^{k_o} \mid s,o] \leq \gamma < 1$. Taking the supremum:

$$\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty \leq \gamma\, \|Q_1 - Q_2\|_\infty.$$

By the Banach fixed-point theorem, $\mathcal{T}$ has a unique fixed point Q^*.

(ii) Convergence. The SMDP Q-learning update is:

$$Q_{n+1}(s,o) = (1-\alpha_n)\, Q_n(s,o) + \alpha_n\Big[R_n + \gamma^{k_n} \max_{o'} Q_n(s'_n, o')\Big].$$

Write this as $Q_{n+1}(s,o) = Q_n(s,o) + \alpha_n[(\mathcal{T}Q_n)(s,o) - Q_n(s,o) + w_n]$, where w_n has zero conditional mean. The conditions of Theorem 1 of Jaakkola et al. (Jaakkola et al. (1993)) hold: (a) Robbins–Monro step sizes by assumption, (b) all pairs visited infinitely often by assumption, (c) $\mathcal{T}$ is a γ-contraction by Part (i), (d) $\mathrm{Var}[w_n \mid Q_n, s, o] \leq C(1 + \|Q_n\|_\infty)^2$ since rewards and durations are bounded. Therefore $Q_n(s,o) \to Q^*(s,o)$ a.s. for all (s,o). □

C.2 Proof of Theorem 3.2

Proof. Let $\hat{p}_{\text{elite}}(c|s) = n(s,c)/n(s)$ be the empirical distribution over $\mathcal{D}_{\text{elite}}$.

(i) Support restriction. The SFT objective $\max_\theta \mathbb{E}_{(s,c^*)\sim\mathcal{D}_{\text{elite}}}[\log \pi_\theta(c^*|s)]$ is equivalent to minimizing $D_{\mathrm{KL}}(\hat{p}_{\text{elite}} \| \pi_\theta) + H(\hat{p}_{\text{elite}})$. Under sufficient capacity, the minimum is attained at $\pi^{\text{SFT}}_{arc} = \hat{p}_{\text{elite}}$. Since $\hat{p}_{\text{elite}}(c|s) = 0$ for $c \notin \mathcal{C}_{\text{elite}}(s)$, we have $\pi^{\text{SFT}}_{arc}(c|s) > 0 \Rightarrow c \in \mathcal{C}_{\text{elite}}(s)$.

(ii) Reward floor. Every $(s,c) \in \mathcal{D}_{\text{elite}} = \{(s_i, c_i^*) \in \mathcal{B} : \text{correct}_i \wedge R(s_i, c_i^*) \geq \tau\}$ satisfies $R(s,c) \geq \tau$. Combined with (i):

$$\mathbb{E}_{c\sim\pi^{\text{SFT}}_{arc}}[R(s,c)] = \sum_{c\in\mathcal{C}_{\text{elite}}(s)} \pi^{\text{SFT}}_{arc}(c|s)\, R(s,c) \geq \tau \sum_{c\in\mathcal{C}_{\text{elite}}(s)} \pi^{\text{SFT}}_{arc}(c|s) = \tau. \qquad \square$$

[1] https://openrouter.ai/

Theorem 3.1 The contraction result establishes that the SMDP has a unique optimal value function Q^*, meaning RL has a well-defined target to converge toward. Without this, it would be unclear whether the optimization is even searching for something that exists. The γ^{k_o} discounting is central: because each Option consumes at least one LLM call ($k_o \geq 1$), the contraction modulus is at most $\gamma < 1$, ensuring Q^* is unique. Options with longer durations (e.g., Evaluator-Optimizer at $k_o \approx 5$) contribute a tighter contraction factor ($\gamma^5 \approx 0.59$ for $\gamma = 0.9$) than short ones, reflecting that heavier Options are discounted more aggressively.

Theorem 3.2 Support restriction ensures that π_{arc}^{SFT} only proposes configurations that were successful during training, preventing the policy from inventing novel, untested configurations at deployment time. The reward guarantee provides a performance floor: since all elite configurations achieve reward at least τ, and π_{arc}^{SFT} samples exclusively from this set, expected performance is guaranteed to be at least τ. In our experiments, τ corresponds to the 70th percentile of rewards in the RL buffer, meaning π_{arc}^{SFT} is guaranteed to perform at least as well as the top 30% of RL trajectories in expectation.

These theoretical properties complement the empirical findings in Section 4.5, where we observe that adding the SFT phase consistently improves performance and reduces variance across runs.

D Ablation: Identifying the Best Embedding for State Representation

The structure policy π_{struct} conditions on a state representation $s_q = [\phi(q); \mathbf{f}_q]$, where $\phi(q)$ is a learned embedding and $\mathbf{f}_q$ contains hand-crafted features. The choice of embedding model $\phi(\cdot)$ is critical: if the representation fails to capture task-relevant semantics, the policy cannot learn to differentiate queries that require different configurations.

We conduct a comprehensive ablation across 19 embedding models spanning text-only encoders (Sentence-T5 Ni et al. (2022), E5 Wang et al. (2022), MiniLM, MPNet Song et al. (2020)), vision-language models (CLIP, MetaCLIP, SigLIP Zhai et al. (2023), FLAVA Singh et al. (2022)), and hybrid approaches (Jina-CLIP Koukounas et al. (2024)). For each model, we evaluate embeddings in two modes: *native* (using the model's original dimensionality) and *projected* (linearly projecting to a fixed 768-dimensional space for fair comparison). We design four evaluation tasks that directly measure properties relevant to RL policy learning:

Clustering Quality (ARI). We evaluate how well embeddings group semantically similar questions using Adjusted Rand Index (ARI). RL policies rely on embeddings that place similar problems close together so that similar inputs lead to consistent decisions. We cluster embeddings using k-means and compare against ground-truth labels (dataset $\times$ tool type). ARI ranges from -1 to 1, where 1.0 indicates perfect alignment with ground truth.

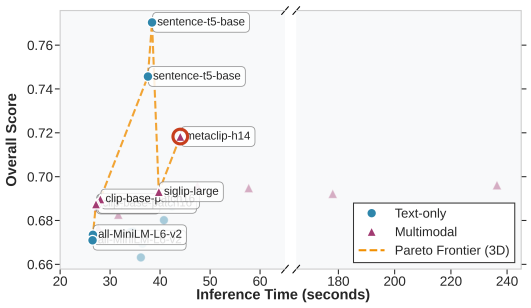

Figure 7: Overall score vs. runtime vs. multimodality with the Pareto frontier under a three-dimensional dominance criterion (runtime ↓, score ↑, multimodality ↑). Circles denote text-only models and triangles denote multimodal models. The selected multimodal model (MetaCLIP-H14) is highlighted.

Classification Accuracy (Cls. Acc). We test whether embeddings distinguish problem types and tool requirements, which is essential for the policy to select correct workflows and tools. We train linear classifiers for two tasks: (1) *dataset classification* predicting the source benchmark (GSM8k, HotpotQA, AIME, MedQA), and (2) *tool requirement classification* predicting the required tool (calculator, web_search, python, none). The reported score is the average cross-validation accuracy across both tasks.

Complexity Ranking (Complexity). We assess whether embeddings capture problem difficulty, which is crucial for budget allocation decisions. We define a complexity score using heuristics (question length, answer length, difficulty-indicating keywords) and train a Ridge regressor to predict complexity from embeddings. We report the Spearman correlation between predicted and true

complexity rankings, which measures monotonic relationship quality. Higher values indicate better difficulty ordering.

Decision Prediction (Decision). This task most directly simulates the RL objective. We train a multi-head MLP to simultaneously predict: (1) workflow ID, (2) required tool, and (3) compute budget tier (Low/Mid/High). The Decision Score is the average accuracy across all three prediction heads, weighted by a combined accuracy metric that requires all three predictions to be correct simultaneously.

Overall Score. We compute a weighted average across all four metrics: Overall $= 0.15 \cdot$ ARI $+$ $0.25 \cdot$ Cls. Acc $+ 0.25 \cdot$ Complexity $+ 0.35 \cdot$ Decision The weights reflect the relative importance of each property for downstream RL performance, with Decision Prediction receiving the highest weight as it most closely matches the policy's objective.

Table 6: Evaluation of embedding models for downstream RL tasks. The highlighted *sentence-t5-base* (text-only) and *MetaCLIP-H14* (multimodal) are the final candidates, with their performance–efficiency trade-off shown as a Pareto frontier in Fig. 7.

Embedder	Mode	ARI	Cls. Acc	Complexity	Decision	Overall	Time (s)
sentence-t5-base (768D)	native	0.5603 ± 0.1019	0.8733 ± 0.0048	0.9261 ± 0.0099	0.7221 ± 0.0070	0.7704 ± 0.0305	38.39 ± 1.24
sentence-t5-base (768D)	projected	0.4867 ± 0.1017	0.8635 ± 0.0084	0.9189 ± 0.0109	0.7138 ± 0.0018	0.7457 ± 0.0229	37.56 ± 0.89
MetaCLIP-H14 (1024D)	native	0.4074 ± 0.0525	0.8514 ± 0.0043	0.8944 ± 0.0112	0.7198 ± 0.0139	0.7183 ± 0.0197	44.04 ± 1.07
MetaCLIP-2-Worldwide-L14 (768D)	native	0.3470	0.8553	0.9131	0.7451	0.7151	43.91
MetaCLIP-2-Worldwide-L14 (768D)	projected	0.3272	0.8470	0.9114	0.7245	0.7025	49.20
Jina-CLIP-v2	projected	0.3736 ± 0.0415	0.8476 ± 0.0028	0.8833 ± 0.0124	0.6798 ± 0.0027	0.6961 ± 0.0129	236.38 ± 2.23
SigLIP-Large	projected	0.3790 ± 0.0026	0.8048 ± 0.0132	0.8857 ± 0.0088	0.7135 ± 0.0030	0.6958 ± 0.0057	47.45 ± 1.58
SigLIP-Large (1024D)	native	0.3469 ± 0.0412	0.8081 ± 0.0132	0.8911 ± 0.0095	0.7264 ± 0.0037	0.6931 ± 0.0090	39.74 ± 1.19
CLIP-Base-Patch16 (512D)	native	0.3370 ± 0.0036	0.8454 ± 0.0094	0.8894 ± 0.0141	0.6872 ± 0.0121	0.6898 ± 0.0061	28.03 ± 0.87
CLIP-Large (768D)	native	0.3406 ± 0.0067	0.8416 ± 0.0082	0.8811 ± 0.0116	0.6955 ± 0.0096	0.6897 ± 0.0050	33.74 ± 5.19
SigLIP-Base	projected	0.3264 ± 0.0447	0.8251 ± 0.0102	0.8797 ± 0.0131	0.7187 ± 0.0035	0.6875 ± 0.0115	27.17 ± 1.10
all-mpnet-base-v2 (768D)	native	0.3356 ± 0.0181	0.8426 ± 0.0160	0.8894 ± 0.0076	0.6784 ± 0.0071	0.6865 ± 0.0097	42.37 ± 0.92
CLIP-Large	projected	0.3293 ± 0.0064	0.8366 ± 0.0113	0.8744 ± 0.0111	0.7029 ± 0.0051	0.6858 ± 0.0064	30.08 ± 0.87
siglip-base (768D)	native	0.3025 ± 0.0183	0.8284 ± 0.0121	0.8863 ± 0.0087	0.7262 ± 0.0128	0.6858 ± 0.0084	27.54 ± 1.13
flava-full	projected	0.3526 ± 0.0294	0.8476 ± 0.0086	0.8898 ± 0.0047	0.6513 ± 0.0041	0.6853 ± 0.7577	31.11 ± 0.01
flava-full (768D)	native	0.3362 ± 0.0086	0.8509 ± 0.0090	0.8923 ± 0.0081	0.6513 ± 0.0051	0.6827 ± 0.0031	31.64 ± 0.97
all-MiniLM-L12-v2 (384D)	native	0.3304 ± 0.0195	0.8432 ± 0.0034	0.8766 ± 0.0219	0.6497 ± 0.0115	0.6750 ± 0.0118	34.55 ± 1.24
all-MiniLM-L6-v2	projected	0.3250 ± 0.0028	0.8399 ± 0.0246	0.8624 ± 0.0113	0.6670 ± 0.0148	0.6735 ± 0.0071	26.56 ± 1.26
e5-base (768D)	native	0.3225 ± 0.0003	0.8432 ± 0.0054	0.8705 ± 0.0133	0.6412 ± 0.0056	0.6694 ± 0.0042	36.48 ± 1.05

E Ablation: Identifying the Best Prompt Generator

The prompt policy π_{prompt} selects instructions from a predefined library of prompts, semantic instruction fragments such as "Decompose the problem" or "Verify intermediate steps". To ensure high-quality instructions, we augment base templates with dataset-specific prompts generated via meta-prompting. This ablation identifies the optimal LLM for generating these prompts.

Table 7: Ablation study comparing LLM models for generating prompt atoms. Each row is a model-dataset combination, ranked by combined score (40% diversity + 60% quality). **Diversity metrics**: Uniqueness ($1 -$ mean pairwise cosine similarity), Coverage (fraction of 6 strategy types), Semantic (silhouette score from k-means). **Quality metrics**: Coherence and Clarity (GPT-5 rated 1–10), Specificity (distance from base atoms).

			Diversity			Quality			
Model	**Size**	**Dataset**	**Unique**	**Cover**	**Sem.**	**Coher.**	**Spec.**	**Clar.**	**Score**
GPT-5.2	Closed	gsm8k	0.18	0.83	0.08	8.76	0.24	5.50	**0.48**
GPT-5.2	Closed	medqa	0.24	0.67	0.19	8.94	0.26	5.50	**0.49**
GPT-5.2	Closed	gaia	0.19	0.83	0.10	8.56	0.25	5.50	**0.48**
GPT-5.2	Closed	hotpotqa	0.17	0.83	0.05	8.88	0.27	5.50	**0.48**
GPT-5.2	Closed	aime25	0.15	0.83	0.18	8.81	0.23	5.50	**0.49**
Llama 3.1 70B	70B	gaia	0.18	0.83	0.29	5.50	0.21	5.50	0.44
Llama 3.1 8B	8B	gaia	0.24	0.67	0.30	5.53	0.22	5.50	0.42
GPT-4o	Closed	medqa	0.19	0.67	0.30	5.50	0.19	5.76	0.42
Claude 3.5 Haiku	Closed	hotpotqa	0.17	0.67	0.25	5.50	0.25	5.50	0.41
Claude 3.5 Haiku	Closed	aime25	0.15	0.67	0.19	5.71	0.25	5.71	0.41
Llama 3.1 8B	8B	hotpotqa	0.23	0.67	0.27	5.50	0.22	5.50	0.41
GPT-4 Turbo	Closed	medqa	0.19	0.67	0.29	5.50	0.20	5.50	0.40
Llama 3.1 70B	70B	hotpotqa	0.16	0.67	0.25	5.50	0.23	5.50	0.40
Qwen 2.5 7B	7B	gaia	0.25	0.50	0.28	5.50	0.21	5.50	0.40
Qwen 2.5 7B	7B	hotpotqa	0.22	0.67	0.25	5.50	0.23	5.50	0.40

We generate atoms for 5 datasets (GSM8k, HotpotQA, GAIA, MedQA, AIME) using 12 models spanning OpenAI (GPT-4o, GPT-4 Turbo, GPT-5.2), Anthropic (Claude 3.5 Haiku/Sonnet), Meta (Llama 3.1 8B/70B), Mistral (Large), Google (Gemini 2.5 Pro), and Qwen (2.5 7B/72B). Each model generates atoms for three agent roles: reasoner, verifier, and answerer.

We evaluate generated atoms on two axes:

- **Diversity (40% weight)**: Uniqueness ($1 -$ mean pairwise cosine similarity), strategy coverage (ratio of covered reasoning strategies), and semantic diversity (silhouette score from k-means clustering).
- **Quality (60% weight)**: Coherence (GPT-5 rating 1–10), specificity (cosine distance from base atoms), and clarity (GPT-5 rating 1–10).

The combined score is $0.4 \times$ Diversity $+ 0.6 \times$ Quality.

Table 7 shows results aggregated by model. GPT-5.2 achieves the highest combined score (0.487 average) across all datasets, driven by superior quality metrics (coherence 8.79, clarity 5.50). Notably, GPT-5.2 also achieves the highest strategy coverage (0.80), indicating diverse reasoning approaches. Based on these results, we use GPT-5.2-generated atoms in our prompt library.

F Training Details

PPO Training. We use the following hyperparameters:

- Learning rate: 3×10^{-4} (structure), 5×10^{-5} (prompt)
- Batch size: 32 episodes
- PPO clip epsilon: $\epsilon = 0.2$
- Discount factor: $\gamma = 0.95$
- Entropy coefficient: $\beta_{\text{ent}} = 0.05$
- Value loss coefficient: 0.5
- Gradient clipping: max norm 0.5
- PPO epochs per batch: $E = 4$
- Total training episodes: 4,000 per dataset minimum

Reward shaping coefficients:

- Task success weight: $\alpha = 5.0$
- Step penalty: $\beta_s = 0.02$
- Token penalty: $\beta_t = 0.03$
- Tool shaping: $\delta_1 = 0.1$, $\delta_2 = 0.2$, $\delta_3 = 0.3$

SFT Post-training:

- Structure LR: 1×10^{-4}, Prompt LR: 5×10^{-6}
- Entropy regularization: 0.01
- Reward threshold τ: 4.0 (top 30% of episodes)
- Epochs: 3

For the SFT refinement phase:

- Learning rate: 1×10^{-4}
- Elite threshold: τ set to retain top 30% by reward
- Number of SFT epochs: $E_{\text{SFT}} = 10$
- Batch size: 32

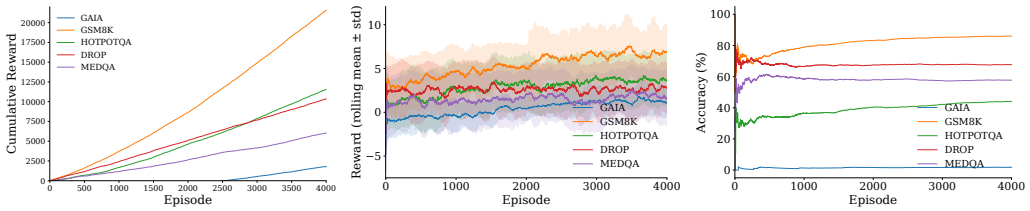

Figure 8: **Training dynamics of ARC across datasets.** Left: cumulative reward over episodes, showing steady improvement as the policy discovers higher-value configurations on GSM8K, DROP, MedQA, HotpotQA, and GAIA. Middle: rolling mean ± standard deviation of per-episode reward, indicating reduced variance and stabilization over time. Right: running validation accuracy, demonstrating that reward gains translate into improved task performance with dataset-specific convergence levels.

Figure 8 summarizes training behavior across datasets. Cumulative and smoothed rewards increase steadily, while running accuracy converges to stable plateaus, indicating that the learned structure and prompt policies make consistent progress rather than overfitting to early episodes.

Figure 10 illustrates how the structure policy's workflow distribution evolves over the course of training. Early on, the agent explores a broad mix of patterns, but as learning proceeds the distribution sharpens and different datasets either converge to distinct dominant workflows or maintain a small mixture of high-value patterns, reflecting task-dependent preferences (e.g., more verification-heavy patterns on GSM8K versus coordination-heavy patterns on GAIA). This behavior confirms that ARC is not merely memorizing a single "best" architecture, but actively specializing (and, when needed, mixing) structural choices to match the demands of each domain.

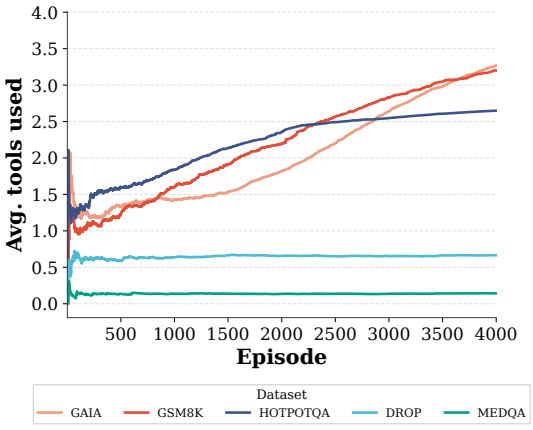

Figure 9: **Tool usage during training.** Running average number of tools used per episode for each dataset. ARC quickly learns sparse tool usage and gradually adjusts invocation patterns, with different steady-state levels reflecting task-specific reliance on tools.

Figure 9 tracks the running average number of tools invoked per episode during training. Across datasets, the policy initially explores a wider range of tool configurations and then converges to stable, task-specific usage levels, indicating that ARC learns when tools are actually helpful rather than indiscriminately calling them. Notably, tool-centric benchmarks such as HotpotQA and GAIA converge to higher usage than primarily textual reasoning tasks like GSM8K, suggesting that the learned structure policy adapts its reliance on tools to the demands of each domain.

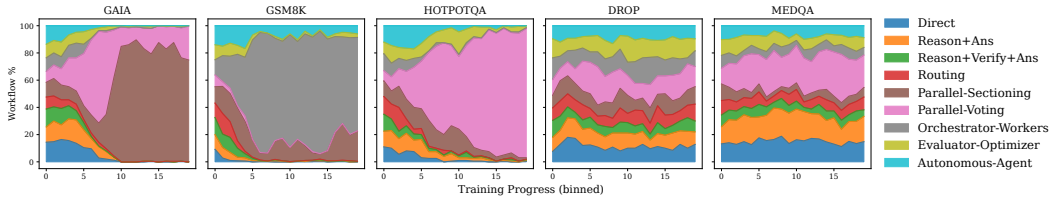

Figure 10: **Evolution of workflow selection during training.** Stacked area plots show, for GSM8K, HotpotQA, and GAIA, the fraction of episodes assigned to each workflow as training progresses. The structure policy quickly prunes suboptimal patterns and concentrates mass on a small set of task-appropriate workflows (e.g., Evaluator–Optimizer on GSM8K, Orchestrator–Workers on HotpotQA).

G Additional Analysis

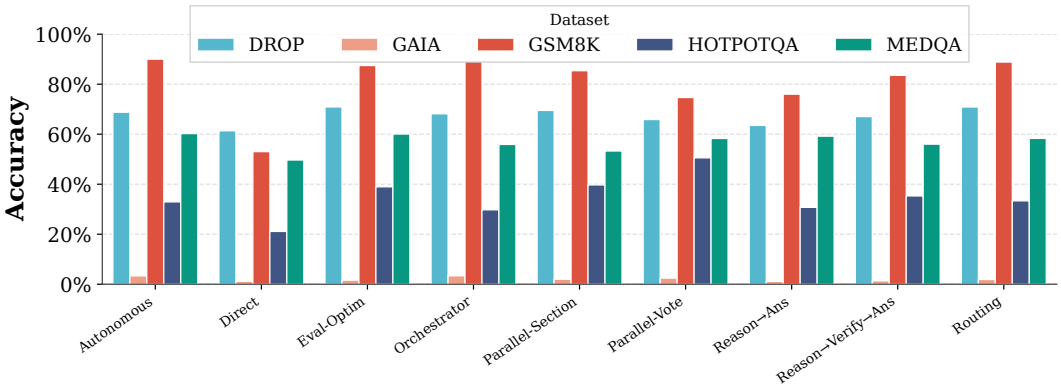

Figure 11: **Accuracy by workflow and dataset.** Each bar shows the average accuracy of a fixed workflow on a given benchmark. Performance varies substantially across workflows and tasks no single workflow is uniformly optimal—highlighting the importance of learning query-adaptive configurations rather than relying on a fixed architecture.

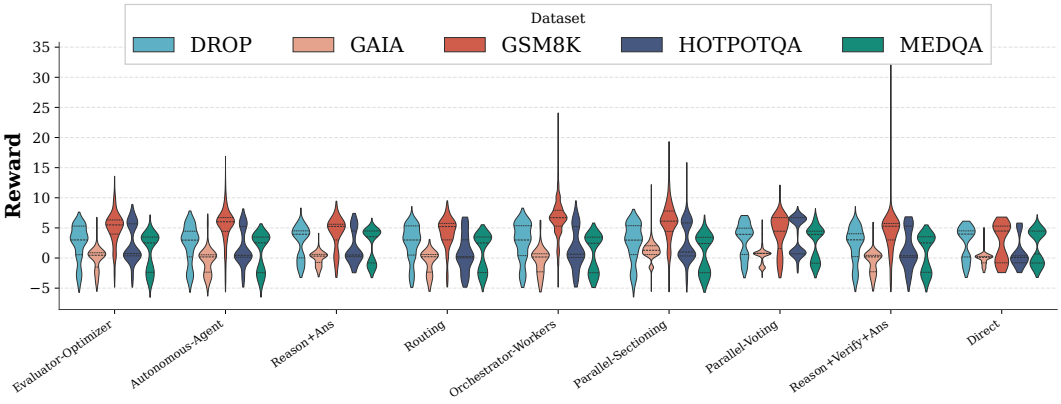

Figure 12: **Reward distribution by workflow.** Violin plots show the distribution of per-episode rewards for each workflow across datasets. Higher-performing workflows exhibit both higher central reward and tighter spread, illustrating that certain structural patterns not only achieve better returns but also yield more stable behavior during training.

Figure 11 breaks down accuracy by workflow and dataset, revealing how different structural choices contribute to performance. Within each benchmark, accuracy varies substantially across workflows, with multi-step patterns (e.g., Reason→Ans or Reason→Verify→Ans) typically outperforming

simple Direct execution, and more complex coordination patterns (such as routing or parallel voting) being beneficial only on some tasks. No single workflow dominates across all datasets, underscoring that the optimal configuration is highly task-dependent and motivating the need for a learned policy that can adaptively select workflows rather than relying on a fixed template.

Figure 12 visualizes the reward distribution for each workflow across datasets. We observe that workflows favored by the learned policy (e.g., Reason→Ans and Reason→Verify→Ans) concentrate mass at higher rewards with lower variance, whereas rarely selected patterns exhibit broader, lower-reward distributions, indicating that the policy systematically avoids structurally inefficient configurations.

H Alternative Training Objectives

We compared PPO against two alternative RL algorithms:

GRPO (Group Relative Policy Optimization): A variant of PPO that uses group-based advantage estimation to reduce variance. On GSM8K, GRPO achieved 81.2% accuracy after 2,000 episodes, compared to PPO's 85.7%. GRPO struggled with the sparse binary reward signal from task correctness, as group normalization dampened learning signals.

We used the following hyperparameters for GRPO:

- Learning rate: 3×10^{-4}
- Batch size: 64 episodes
- Clip epsilon: $\epsilon = 0.2$
- Discount factor: $\gamma = 0.99$
- Entropy coefficient: 0.05
- KL coefficient: 0.0 (no KL regularization)
- Update epochs per batch: 4
- Max gradient norm: 0.5
- Advantage computation: $A_i = \frac{R_i - \bar{R}}{\sigma_R + 10^{-8}}$ (group-relative)

DPO (Direct Preference Optimization): A preference-based method requiring pairwise configuration comparisons. We sampled pairs of configurations and labeled preferences based on reward differences. DPO achieved 79.8% accuracy but required $3\times$ more environment interactions to collect pairwise data. Additionally, the preference labeling was noisy when configurations had similar rewards, leading to unstable training.

DPO hyperparameters:

- Learning rate (structure policy): 1×10^{-4}
- Learning rate (prompt policy): 1×10^{-5}
- Batch size: 16
- DPO temperature: $\beta = 0.05$
- Entropy coefficient: 0.05
- Training epochs: 3
- Max gradient norm: 0.5
- Preference pair thresholds: correct episodes with reward ≥ 4.0, incorrect episodes with reward ≤ 2.0

I Error Categorization Methodology

We automatically classify errors into four categories based on heuristic analysis of the episode data. Each incorrect prediction is analyzed as follows:

I.1 Policy Configuration Errors

We detect policy misconfiguration by checking if the selected workflow, tools, and token budget are appropriate for the query:

- **Workflow mismatch**: Simple workflows (Direct) assigned to multi-step problems (detected via question length $>$100 words or presence of multiple sub-questions).
- **Tool mismatch**: Calculator missing when query contains arithmetic keywords ("calculate", "sum", "total"); web search missing when query asks about real-world facts.
- **Budget under-allocation**: Low token budget ($<$256) assigned to complex queries (word count $>$50 or contains "explain", "step-by-step").

I.2 Reasoning Errors

We detect reasoning failures by comparing the prediction structure against ground truth:

- **Wrong operation**: Prediction uses addition when ground truth uses subtraction (detected via operation keywords: "add"/"plus" vs "subtract"/"minus"/"remaining").
- **Missing steps**: Ground truth contains $\geq$3 reasoning steps but prediction is a single textbf.
- **Comprehension failure**: Query contains critical constraints ("remaining", "left", "after") that are absent from prediction.

I.3 Knowledge Gap Errors

We detect knowledge/retrieval failures for tool-use tasks:

- **Retrieval failure**: Prediction contains phrases like "could not find", "no information available", "cannot determine" while using retrieval tools (web search, code execution).
- **Factual error**: Uses retrieval tools but provides a confident answer that differs from ground truth without explicit "not found" indicators (hallucination).

I.4 Execution Errors

We detect execution failures where the approach is correct but output is wrong:

- **Arithmetic error**: Ground truth contains intermediate calculations (e.g., $\langle\langle$5*3=15$\rangle\rangle$) and prediction produces a different numeric answer.
- **Answer extraction error**: Correct answer appears in prediction text but a different value is extracted as the final answer.

Priority. Errors are assigned to the first matching category in order: policy configuration $\rightarrow$ answer extraction $\rightarrow$ reasoning $\rightarrow$ arithmetic $\rightarrow$ knowledge gap $\rightarrow$ unclassified. This ensures policy failures are surfaced first, as they are most actionable for our framework.

I.5 Failure Case Examples

We provide concrete examples illustrating each error category:

Policy Configuration Error:

- Query: "Find x such that $\log_2(x) + \log_2(x-7) = 3$"
- Selected: Direct workflow, Low budget
- Issue: Multi-step algebra requires Reason+Verify workflow with high budget.

Reasoning Error:

- Query: "John has 5 apples. He gives 2 to Mary. How many does John have left?"
- Prediction: "John has 5 + 2 = 7 apples."
- Issue: Used addition instead of subtraction despite "gives" and "left" keywords.

Knowledge Gap Error:

- Query: "Who directed the 2014 film Big Stone Gap?"
- Prediction: "Based on the search results, I cannot find information about the director."
- Ground truth: "Adriana Trigiani"

- Issue: Retrieval failed to find available information.

Execution Error:

- Query: "What is $\frac{1}{2} \times 5 \times 12$?"
- Prediction: "$\frac{1}{2} \times 5 \times 12 = 25$"
- Ground truth: 30
- Issue: Correct formula, wrong arithmetic.