Agent Learning: Execution-Aware Policy Optimization for LLM Tool Systems

Opeyemi Bamigbade *
South East Technological University, Waterford, Ireland

Stephen Oni
African Institute for Mathematical Sciences (AIMS), Mbour, Senegal

Abstract

Large language model (LLM) agents increasingly rely on external tools and structured workflows to accomplish complex tasks. While recent work has emphasized improving reasoning quality and prompting strategies, the orchestration layer responsible for tool selection, execution sequencing, escalation, and constraint handling remains largely heuristic. In many modern agent systems, these decisions are either implemented through hand-designed control flows or implicitly delegated to the language model's generative reasoning process, without explicit optimization for execution-level objectives such as latency, reliability, or cost. This work introduces `Agent Learning` [2], a framework that formalizes orchestration in tool-augmented LLM systems as an execution-aware policy optimization problem. An agent is modeled as a triplet $(\mathcal{M}, \mathcal{T}, \pi)$, where $\mathcal{M}$ denotes a fixed stochastic reasoning module, $\mathcal{T}$ a set of external tools, and π an orchestration policy mapping system states to actions. An execution-aware cost functional captures latency, monetary cost, constraint violations, execution failures, and plan–outcome divergence. The orchestration policy is optimized with respect to this cost while keeping the reasoning module and tool environment fixed. Empirical evaluation in a controlled synthetic tool environment demonstrates that learned policies consistently outperform heuristic baselines in minimizing execution cost and improving constraint satisfaction across multiple random seeds. These results establish orchestration as a well-posed learning problem at the runtime systems layer and provide a formal framework for execution-aware orchestration in LLM-based tool systems.

1 Introduction

Large language model (LLM) agents increasingly operate in tool-augmented environments (e.g., [1, 2]), invoking APIs, databases, search systems, and external services to accomplish complex tasks. Progress in this area has largely focused on improving reasoning quality through better prompting strategies, larger models, and hybrid architectures. In contrast, the orchestration layer that governs tool selection, execution sequencing, escalation decisions, and termination remains predominantly heuristic.

In typical systems, orchestration logic is implemented either through static routing rules, retry heuristics, and hand-designed control flows, or through reasoning-mediated tool invocation in which the language model itself determines which tool to call during generation. While effective in practice, both approaches treat orchestration as an implicit or heuristic process rather than an explicit

*Corresponding author: opeyemi.bamigbade@postgrad.setu.ie
[2]Code available at https://github.com/OBA-Research/agent-learning.

Preprint.

optimization problem. As a result, system-level objectives such as latency, monetary cost, reliability, and constraint satisfaction are often handled indirectly or post hoc.

This work introduces *Agent Learning*, a framework that formalizes orchestration in tool-augmented LLM systems as an execution-aware policy optimization problem. An agent is modeled as a triplet $(\mathcal{M}, \mathcal{T}, \pi)$, where $\mathcal{M}$ denotes a fixed stochastic reasoning module, $\mathcal{T}$ a set of external tools, and π an orchestration policy mapping system states to actions. A structured execution cost functional captures latency, monetary cost, constraint violations, execution failures, and plan–outcome divergence. The orchestration policy is optimized with respect to this cost while keeping the reasoning module and tool environment fixed.

This formulation isolates runtime orchestration as a control problem over stochastic state transitions. Tool selection, sequencing, and escalation are treated as components of a unified policy defined over system states. Under mild assumptions, the existence of an optimal orchestration policy is established. Empirical evaluation in a controlled synthetic tool environment demonstrates that learned policies consistently outperform heuristic baselines in minimizing execution cost and improving constraint satisfaction across multiple random seeds.

Agent Learning models LLM-based agents as execution-aware control systems. By separating representation learning from orchestration optimization, it provides a structured framework for reliability-aware tool use and cost-sensitive decision-making in tool-augmented language model systems.

1.1 Contributions

This work makes four contributions:

- It formalizes tool-augmented LLM agents as triplets $(\mathcal{M}, \mathcal{T}, \pi)$, isolating orchestration as an independent optimization target.
- It introduces an execution-aware cost functional that captures operational trade-offs in tool-based systems.
- It establishes the Orchestration Optimality Principle, showing that an optimal orchestration policy exists under finite-horizon and bounded-cost assumptions.
- It demonstrates stable multi-seed policy optimization that significantly outperforms heuristic baselines in a controlled tool environment.

2 Background and Related Work

2.1 LLM-Based Tool Agents

Recent work has demonstrated that large language models can be augmented with external tools to improve task performance. Approaches such as ReAct [1] interleave reasoning and action, while Toolformer [2] trains models to invoke tools during generation. In these approaches, the decision of when and how to invoke tools is embedded within the language model's generative reasoning process itself, effectively coupling semantic reasoning with system-level orchestration.

Recent advancements in tool augmentation for language models span both handcrafted prompting strategies and learned tool invocation frameworks; surveys of this space characterize a broad set of techniques for enabling models to learn when and how to call external tools [3]. Benchmarks such as WebArena [4] evaluate agent performance in realistic web environments. These works primarily focus on improving reasoning quality and enabling tool invocation within the generative policy of the language model, rather than formalizing orchestration as a separate optimization problem over execution outcomes.

Unlike prior approaches that integrate tool invocation directly into the language model's generative policy, this work explicitly separates reasoning from orchestration. The reasoning module remains fixed and stochastic, while orchestration is treated as an independent execution-aware control policy defined over system states. This architectural separation shifts the learning objective from improving representation quality to optimizing runtime system behavior under explicit cost constraints.

2.2 Reinforcement Learning and Policy Optimization

Reinforcement learning provides a general framework for optimizing policies with respect to cumulative reward [5]. Modern policy optimization methods, such as proximal policy optimization (PPO) [6], have enabled stable training of parameterized policies under stochastic environments. Policy optimization techniques have also been applied to fine-tune language models and tool-using agents.

However, most existing approaches integrate reward signals heuristically and do not explicitly separate reasoning modules from orchestration policies. The present work instead focuses on execution-aware optimization over fixed reasoning modules and tool environments.

2.3 Control-Theoretic Perspectives

Control theory studies feedback policies that govern system behavior under uncertainty [7]. In stochastic control, policies are optimized with respect to cumulative cost functionals over state-transition systems [8]. Viewing orchestration as a closed-loop control problem aligns tool-augmented agents with this tradition.

Unlike classical control settings, the transition dynamics considered here arise from pre-trained stochastic reasoning modules and semi-deterministic tool responses rather than physically modeled dynamical systems. The execution-aware cost functional introduced in this work parallels standard cumulative cost formulations while adapting them to runtime orchestration in language model tool systems.

2.4 Systems-Level Optimization in ML

Prior work in machine learning systems has addressed cost-aware inference, resource allocation, and latency optimization. Agent Learning differs by treating tool routing and execution sequencing as components of a unified policy optimized directly with respect to an explicit execution cost functional.

3 Agent Learning Framework

This work differs from standard end-to-end RL fine-tuning in that $\mathcal{M}$ and $\mathcal{T}$ are treated as fixed system components and only the orchestration policy π is optimized under an explicit execution cost decomposition.

3.1 Agent Decomposition

A tool-augmented language model agent operating over a finite execution horizon $H \in \mathbb{N}$ is formalized in this section. The agent interacts with an external environment through structured tool calls and receives observable responses.

Agent Definition: An agent is defined as a triplet

$$\mathcal{A} = (\mathcal{M}, \mathcal{T}, \pi) \tag{1}$$

where:

- $\mathcal{M}$ denotes a fixed stochastic reasoning module. Given an input context x_t, it produces a distribution over reasoning outputs.
- $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$ is a finite set of external tools. Each tool τ_k maps a structured input to an observable output and may incur latency, monetary cost, or execution failure.
- π is an orchestration policy mapping system states to actions.

Throughout optimization, $\mathcal{M}$ and $\mathcal{T}$ remain fixed. The orchestration policy π is the only component subject to learning. This decomposition explicitly separates semantic reasoning from execution orchestration, allowing the reasoning module to remain fixed while system-level behavior is governed by a policy optimized with respect to execution outcomes.

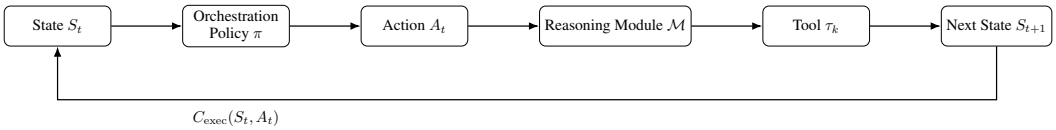

Figure 1: Agent decomposition as a closed-loop control system. The orchestration policy selects actions based on the system state. Execution through the reasoning module and tool environment produces a new state and induces execution cost.

State Space: At each discrete time step $t \in \{0, \dots, H\}$, the agent maintains a state

$$S_t \in \mathcal{S}, \tag{2}$$

where $\mathcal{S}$ denotes the state space.

In general, the underlying tool environment may be partially observable. The state S_t is therefore interpreted as the agent's *information state*, constructed from the task specification, accumulated interaction history, prior tool outputs, resource usage statistics (including total latency and monetary cost), constraint indicators, and execution step index.

This representation is designed to be Markov with respect to the execution process induced by $(\mathcal{M}, \mathcal{T}, \pi)$, in the sense that the distribution of S_{t+1} depends only on (S_t, A_t) and stochastic effects arising from the reasoning module and tool responses. The state space $\mathcal{S}$ is assumed to be measurable and finite-dimensional.

Action Space: At each time step t, the agent selects an action

$$A_t \in \mathcal{U},$$

where $\mathcal{U}$ denotes the action space. An action may correspond to: (a) invoking a tool $\tau_k \in \mathcal{T}$ with structured input; (b) querying the reasoning module $\mathcal{M}$; (c) requesting clarification; (d) escalating to an external supervisor; or (e) terminating execution.

The orchestration policy is defined as

$$\pi : \mathcal{S} \to \mathcal{U}.$$

State Transitions: State evolution follows

$$S_{t+1} = f(S_t, A_t, \xi_t) \tag{3}$$

where f is a transition function and ξ_t represents stochastic effects arising from the reasoning module or tool responses. The random variable ξ_t captures stochastic effects arising from $\mathcal{M}$ and tool execution, including nondeterministic outputs and failures.

This formulation isolates orchestration as the optimization target: the reasoning module $\mathcal{M}$ and tool set $\mathcal{T}$ are fixed, while the policy π determines system-level behavior. Figure 1 illustrates the resulting closed-loop structure. The orchestration policy operates as a controller over state transitions induced by the reasoning module and tool environment.

3.2 Execution-Aware Cost Functional

The objective of Agent Learning is to optimize the orchestration policy π with respect to system-level execution behavior. This requires a cost functional that captures operational trade-offs in tool-augmented LLM systems.

Execution Horizon: Let $H \in \mathbb{N}$ denote a finite execution horizon. A trajectory induced by policy π is the sequence

$$\{(S_0, A_0), (S_1, A_1), \dots, (S_H, A_H)\},$$

where $S_t \in \mathcal{S}$ and $A_t \in \mathcal{U}$.

Instantaneous Execution Cost: Let $C_k : \mathcal{S} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ for $k = 1, \ldots, q$ denote measurable execution cost components. Each component $C_k(S_t, A_t)$ represents a specific operational attribute such as latency, monetary expenditure, constraint violation, execution failure, or plan–outcome divergence. In practice, each component can be normalized or clipped to a fixed range, ensuring bounded per-step cost without changing the optimization structure.

Let

$$\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_q) \in \mathbb{R}^q_{\geq 0}$$

denote a non-negative weight vector.

The instantaneous execution cost is defined as

$$C_{\text{exec}}(S_t, A_t) = \sum_{k=1}^{q} \lambda_k \, C_k(S_t, A_t) \tag{4}$$

Plan–Outcome Divergence: When available, an internal predictor produces a planned or expected next-state summary $\hat{S}_{t+1}$ prior to executing A_t. The divergence term penalizes mismatch between expectation and realized outcome:

$$C_d(S_t, A_t) = \|\hat{S}_{t+1} - S_{t+1}\|.$$

where $\|\cdot\|$ denotes a norm on the state space $\mathcal{S}$. This component is optional and can be omitted when no meaningful predictor is defined.

Total Execution Cost: The execution-aware cost functional is defined as

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{H} C_{\text{exec}}(S_t, A_t)\right] \tag{5}$$

where the expectation is taken over stochastic transitions induced by ξ_t and any stochasticity in the reasoning module $\mathcal{M}$.

Optimization Objective: Agent Learning seeks a policy

$$\pi^* = \arg\min_{\pi} J(\pi) \tag{6}$$

This formulation treats orchestration as a policy optimization problem under the execution-aware objective defined in Eq. 5. The optimization target in Eq. 6 makes explicit that system-level behavior is governed by the orchestration policy rather than the reasoning module.

3.3 Orchestration Optimality Principle

This section establishes the existence of an optimal orchestration policy under the execution-aware cost functional defined in Section 3.2.

Assumptions: The following assumptions are adopted:

(a) The execution horizon H is finite.

(b) The action space $\mathcal{U}$ is finite.

(c) The state space $\mathcal{S}$ is measurable and finite-dimensional.

(d) The instantaneous execution cost $C_{\text{exec}}(S_t, A_t)$ is bounded and measurable.

Under these assumptions, the cost functional

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{H} C_{\text{exec}}(S_t, A_t)\right]$$

is well-defined for any measurable policy $\pi : \mathcal{S} \to \mathcal{U}$.

Theorem 3.1 (Orchestration Optimality Principle) *Let $\mathcal{A} = (\mathcal{M}, \mathcal{T}, \pi)$ denote a tool-augmented language model agent as defined in Section 3.1. Under the assumptions stated above, there exists an orchestration policy π^* such that*

$$\pi^* = \arg\min_{\pi} J(\pi).$$

Proof Sketch. Under a finite horizon and finite action space, the induced decision process admits an optimal policy by standard finite-horizon stochastic control arguments (e.g., dynamic programming; [8]). Bounded per-step cost ensures that $J(\pi)$ is finite for any admissible policy, and an optimal policy attains the minimum.

Implications. The theorem formalizes orchestration as an optimization object independent of representation learning. The reasoning module $\mathcal{M}$ and tool set $\mathcal{T}$ remain fixed, while system-level performance is governed by the choice of policy π. This formalizes orchestration as a well-defined learning objective within LLM tool systems.

3.4 Learning the Orchestration Policy

Theorem 3.1 establishes the existence of an optimal orchestration policy. This section describes how such a policy may be approximated in practice.

Parameterized Policy. Let $\pi_\theta : \mathcal{S} \to \mathcal{U}$ denote a parameterized policy with parameters $\theta \in \Theta \subset \mathbb{R}^p$, where p is the number of trainable parameters. The reasoning module $\mathcal{M}$ and tool set $\mathcal{T}$ remain fixed.

Empirical Objective. Given a dataset of trajectories

$$\mathcal{D} = \{\tau^{(i)}\}_{i=1}^{N},$$

where each trajectory

$$\tau^{(i)} = \{(S_0^{(i)}, A_0^{(i)}), \ldots, (S_H^{(i)}, A_H^{(i)})\}$$

is generated under the system dynamics defined in Section 3.1, the empirical execution cost is defined as

$$\widehat{J}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{A_t^{(i)} \sim \pi_\theta(\cdot|S_t^{(i)})} \left[\sum_{t=0}^{H} C_{\text{exec}}(S_t^{(i)}, A_t^{(i)}) \right]. \tag{7}$$

The training objective in Eq. 7 provides a finite-sample approximation to the population objective defined in Eq. 5. The expectation reflects stochastic action selection during training; evaluation uses the greedy action under π_θ.

The learning objective is

$$\theta^* = \arg\min_{\theta \in \Theta} \widehat{J}(\theta).$$

Learning Regimes. The empirical objective can be optimized under multiple regimes:

(a) **Supervised policy learning**, where trajectories are generated from a baseline policy and used for cost-sensitive classification.

(b) **Offline policy optimization**, where logged interaction data is used to minimize estimated execution cost.

(c) **Online policy refinement**, where parameters are updated using sampled trajectories under the current policy.

In all cases, optimization occurs over orchestration decisions while keeping $\mathcal{M}$ and $\mathcal{T}$ fixed.

Discussion. This formulation separates representation learning from system-level control. The reasoning module provides stochastic semantic outputs, while the orchestration policy governs execution decisions under explicit cost constraints. Agent Learning therefore operates at the runtime systems layer rather than the model pretraining layer.

4 Experimental Instantiation

This section provides a controlled instantiation of Agent Learning in a synthetic tool environment designed to isolate orchestration behavior.

4.1 Synthetic Tool Environment

A discrete-time environment with finite horizon H is constructed. The state space $\mathcal{S}$ consists of: (a) a task vector $z \in \mathbb{R}^d$; (b) accumulated resource usage (latency and cost); (c) binary constraint indicators; and (d) execution step index t.

Each tool $\tau_k \in \mathcal{T}$ is defined by: (a) a deterministic functional effect on the task vector; (b) latency cost $\ell_k \geq 0$; (c) monetary cost $m_k \geq 0$; and (d) failure probability $p_k \in [0, 1]$.

When tool $\tau_k \in \mathcal{T}$ is invoked at state S_t, the next state is given by

$$S_{t+1} = f(S_t, \tau_k, \xi_t),$$

where f is the transition function introduced in Section 3.1. The random variable ξ_t models stochastic execution effects and is sampled independently at each time step. In the synthetic environment, ξ_t is sampled independently at each time step and induces tool failure with probability p_k.

A task is considered successful if the constraint function

$$g(S_H) = 0$$

holds at termination, where $g : \mathcal{S} \to \mathbb{R}$ is a predefined measurable function encoding task feasibility.

4.2 Baseline Orchestration Policy

A fixed heuristic policy π_{base} is defined as a reference point for evaluation. The policy does not optimize the execution-aware cost functional $J(\pi)$ introduced in Section 3.2.

At each state S_t, the baseline policy selects an admissible tool according to a predefined local rule:

(a) Among available tools, select the tool with minimal instantaneous latency cost.

(b) If execution failure occurs, retry the same tool at most once.

(c) Terminate when the constraint function $g(S_t) = 0$ is satisfied or when the horizon H is reached.

This policy is static and does not incorporate accumulated cost, failure probability, or plan–outcome divergence into its decision process. It therefore provides a comparison point for evaluating policies trained to minimize the execution-aware cost functional. Both baselines are myopic with respect to terminal constraint satisfaction, which is required for success and may require paying higher immediate execution costs.

4.3 Learned Policy Implementation

A parameterized orchestration policy $\pi_\theta : \mathcal{S} \to \mathcal{U}$ is instantiated as a finite-dimensional function with parameters $\theta \in \mathbb{R}^p$. The policy outputs a discrete action in $\mathcal{U}$ for each state S_t.

State Representation. Each state S_t is represented as a fixed-length vector comprising:

(a) the task vector z,

(b) accumulated latency,

(c) accumulated monetary cost,

(d) binary constraint indicators,

(e) normalized time index t/H.

All state components are normalized to comparable scales prior to training.

Policy Parameterization. The policy π_θ is implemented as a feedforward network mapping the state vector to a categorical distribution over the action space $\mathcal{U}$. The selected action is the maximizer of this distribution during evaluation.

Training Procedure. Training trajectories are generated by executing the current policy within the synthetic environment defined in Section 4.1. For each trajectory, the execution-aware cost $J(\pi_\theta)$ is estimated using the empirical cost functional defined in Section 3.4.

Policy parameters are optimized via policy-gradient updates to minimize empirical execution cost. The reasoning module $\mathcal{M}$ and tool set $\mathcal{T}$ remain fixed throughout training.

Termination. Execution terminates when either the constraint function $g(S_t) = 0$ is satisfied or the horizon H is reached. The terminal state contributes to the total execution cost through accumulated latency, monetary cost, and any remaining constraint penalties.

4.4 Evaluation Metrics

Evaluation focuses on system-level execution behavior under the cost functional defined in Section 3.2.

Total Execution Cost. For each evaluated policy π, the primary metric is the empirical execution cost

$$\widehat{J}(\pi) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} C_{\text{exec}}(S_t^{(i)}, A_t^{(i)}),$$

computed over N independent trajectories.

Component-wise Cost Analysis. Let $\{C_k\}_{k=1}^{q}$ denote the execution cost components defined in Section 3.2. For a policy π, the empirical average of component k is

$$\widehat{C}_k(\pi) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} C_k\left(S_t^{(i)}, A_t^{(i)}\right), \quad k = 1, \ldots, q.$$

The empirical cost vector is therefore

$$\widehat{\mathbf{C}}(\pi) = \left(\widehat{C}_1(\pi), \ldots, \widehat{C}_q(\pi)\right).$$

Success Rate. A trajectory is considered successful if the constraint function satisfies

$$g(S_H^{(i)}) = 0.$$

The empirical success rate is defined as

$$\text{Success} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\big(g(S_H^{(i)}) = 0\big),$$

where $\mathbb{I}(\cdot)$ denotes the indicator function, equal to 1 if the condition holds and 0 otherwise.

Comparison Protocol. Policies are evaluated under identical initial state distributions and tool stochasticity. All reported metrics are averaged over the same set of evaluation seeds to ensure comparability.

4.5 Results

This section evaluates whether execution-aware policy optimization improves orchestration performance relative to heuristic baselines. Results are aggregated over five independent random seeds. Each configuration is evaluated over $N = 1000$ trajectories per seed, with training performed for 12,000 episodes per seed.

Table 1: Aggregated performance over five seeds (mean $\pm$ standard deviation).

Policy	Total Cost	Success Rate
Latency Baseline	69.5754 ± 0.4427	0.0000 ± 0.0000
Cost-Aware Baseline	69.2179 ± 0.2527	0.0000 ± 0.0000
Learned Policy	$\mathbf{10.2262 \pm 0.0309}$	$\mathbf{0.9164 \pm 0.0063}$

Total Execution Cost and Success Rate. The latency-only heuristic and the instantaneous cost-aware heuristic both fail to satisfy terminal constraints. Despite observing weighted execution costs, the cost-aware baseline remains myopic and does not anticipate long-horizon drift effects.

Table 1 reports aggregated performance across the evaluated policies. The learned orchestration policy substantially reduces total execution cost relative to heuristic baselines while achieving high constraint satisfaction in the synthetic environment. Performance is stable across seeds, with low variance in both metrics.

The heuristic baselines optimize only instantaneous execution costs and therefore fail to select tool sequences required to reach constraint-satisfying terminal states. As a result, terminal constraints are never satisfied under these policies, leading to a success rate of zero across evaluation seeds.

Component-Wise Cost Analysis. To understand behavioral differences, Table 2 reports execution cost components.

Table 2: Component-wise costs (mean $\pm$ standard deviation).

Component	Baseline	Learned Policy
Latency	5.0000 ± 0.0000	25.2705 ± 2.5569
Monetary	1.0000 ± 0.0000	8.2974 ± 0.9205
Constraint	32.5507 ± 0.2378	0.4607 ± 0.2436
Failure	0.9992 ± 0.0317	0.5896 ± 0.0497
Divergence	4.5498 ± 0.0694	3.3124 ± 0.0310

The learned policy increases latency and monetary expenditure relative to the heuristic baseline. However, it substantially reduces constraint violations, execution failures, and plan–outcome divergence. This demonstrates explicit trade-off optimization: short-term efficiency is sacrificed to reduce long-horizon penalties.

Discussion of Stability. Policy optimization is performed using variance-reduced policy gradients with a running baseline. Without baseline subtraction, training exhibits seed-dependent instability. With baseline subtraction, learning converges consistently across seeds.

These results indicate that orchestration behavior can be learned under the execution-aware objective defined in Section 3.2.

5 Discussion

The current experiments validate learnability in a controlled setting; evaluation on standardized agent benchmarks and real tool environments is left to future work.

5.1 Escalation as Optimal Stopping

Escalation decisions, such as deferring to a supervisor or terminating execution early, can be interpreted as optimal stopping problems under the execution-aware cost functional. Escalation becomes a learned decision rather than a fixed heuristic.

5.2 Multi-Agent Extensions

The formulation naturally extends to multi-agent systems in which multiple policies coordinate over shared tools or resources. In such settings, execution-aware optimization may involve joint cost functionals and coordination constraints.

5.3 Reliability-Aware Systems

Constraint violations and execution failures are explicitly modeled in the cost functional. This enables reliability-aware orchestration in which policies trade efficiency against robustness. The framework provides a structured approach for reliability-aware optimization in LLM tool systems.

5.4 Limitations

The experimental validation is conducted in a synthetic environment designed to isolate orchestration behavior. While this allows controlled analysis, real-world tool environments introduce additional complexities, including partial observability and distribution shift. Extending Agent Learning to standardized web-based benchmarks remains an important direction for future work.

6 Conclusion

This work formalizes orchestration in tool-augmented LLM systems as a policy optimization problem under an explicit execution-aware cost functional. By isolating orchestration from representation learning and modeling it as a control problem over stochastic state transitions, Agent Learning provides a principled framework for runtime decision-making over fixed reasoning modules and tools.

Empirical results in a controlled synthetic environment demonstrate stable multi-seed learning and consistent improvements over heuristic orchestration strategies. These findings suggest that orchestration can be treated as a learnable systems component rather than solely as static control logic. Future work includes evaluation on standardized agent benchmarks, extension to multi-agent coordination, and investigation of deployment considerations in real-world tool environments.

References

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[2] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[3] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with Large Language Models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

[4] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents, 2024.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017.

[7] Karl J. Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[8] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2017.