LoRA-Guided PPO for Cost-Aware and Compute-Efficient Agent Orchestration

Aneesh Durai[1] **Joshua Cong Hu**[2] **Kevaan Buch**[2] **Kevin Zhu**[3]
Vasu Sharma[4] **Aishwarya Balwani**[5]
[1]Columbia University [2]University of Toronto [3]Algoverse AI Research
[4]FAIR at Meta [5]St. Jude Children's Research Hospital

Abstract

A fundamental challenge in multi-agent reasoning systems is budget-aware allocation: deciding which sub-agents to invoke across multiple steps while balancing success against computational and monetary cost. We formalize this setting as a cost-constrained sequential decision problem and propose a hybrid policy that integrates parameter-efficient pretraining with reinforcement learning. Specifically, a LoRA adapter captures cost-sensitive priors from heuristic traces, and Proximal Policy Optimization (PPO) fine-tunes only this low-rank subspace. Restricting updates to the adapter stabilizes optimization, improves sample efficiency, and preserves allocation thrift while enabling sequential credit assignment. Empirical evaluation suggests that this framework can add value in efficiency–accuracy trade-offs. On a ToolBench-style benchmark, the hybrid achieves perfect success while reducing cost-per-success (CPS) by 12% relative to PPO (21.40 vs. 24.30). In the synthetic FlightPlanner setting, it achieves the lowest CPS (7.71) among high-success methods, compared with Rule-based (10.58) and supervised LoRA (11.70). Our results demonstrate that combining parameter-efficient fine-tuning with RL yields controllers that are both adaptive and budget-aware, providing a practical recipe for efficient reasoning under real-world resource constraints.

1 Introduction

Reasoning in complex domains often requires orchestrating multiple sub-agents such as retrievers, planners, calculators, and validators – across several turns. This poses the challenge of requiring reasoning capabilities both within and across models: deciding which agents to invoke, in what sequence, and at what cost. Each additional call consumes FLOPs, increases latency, and incurs monetary cost, and in multi-step settings these costs compound quickly [Schick et al., 2023, Qin et al., 2023a]. System designers therefore face a fundamental trade-off: invoke more agents to maximize reliability, or conserve resources and risk incomplete or incorrect solutions. Balancing task success against resource use is thus central to making multi-agent reasoning practical at scale.

Existing allocation strategies, unfortunately, leave much to be desired. Heuristic and rule-based methods are lightweight but brittle, unable to adapt when task distribution or agent capabilities shift [Schick et al., 2023, Qin et al., 2023a]. Reinforcement learning (RL) methods, such as Proximal Policy Optimization (PPO), can in principle adaptively balance multiple objectives [Schulman et al., 2017], but in practice they demand heavy training budgets, are unstable in high-variance environments, and often overspend on expensive agents [Nakano et al., 2021]. Parameter-efficient fine-tuning techniques like LoRA [Hu et al., 2021] reduce training overhead, yet by themselves they lack mechanisms for sequential credit assignment across turns.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

This tension motivates our central question: **How can we design allocators that simultaneously achieve high task success and strong cost efficiency, without incurring prohibitive training budgets or sacrificing adaptability?**

We argue that combining parameter-efficient fine-tuning with RL offers a promising path. Supervised fine-tuning with LoRA adapters can encode cost-sensitive allocation strategies from traces, while restricting PPO updates to these low-rank adapters anchors RL in a stable subspace. This hybrid – LoRA-Guided PPO – leverages the sample efficiency of supervised fine-tuning and the adaptability of RL, yielding allocators that are both efficient and robust.

Subsequently, the contributions of our work are threefold:

- We formalize cost-aware orchestration as a constrained allocation problem, balancing task success against resource use.
- We compare three allocators: a PPO baseline, a supervised LoRA-based allocator, and our proposed hybrid LoRA-Guided PPO.
- We evaluate across synthetic and benchmark environments, showing that parameter-efficient updates can meaningfully improve efficiency without sacrificing success.

2 Related Work

Research on efficient reasoning agents has evolved along two main lines. Heuristic allocation methods—such as static routing, rule-based schedulers, or capability matching—are simple and interpretable but fail to adapt when task distributions or resource constraints change [Schick et al., 2023, Qin et al., 2023a]. These methods treat efficiency as an afterthought, assuming costs are either negligible or fixed.

Reinforcement learning approaches adapt allocations dynamically, optimizing success rates in multi-step tasks [Schulman et al., 2017, Nakano et al., 2021]. While flexible, they require extensive training budgets, exhibit instability in high-variance environments, and often overuse costly agents when left unconstrained. This makes them difficult to deploy under strict compute or monetary budgets.

Parameter-efficient fine-tuning offers an alternative. LoRA and its variants update only a small number of low-rank parameters [Hu et al., 2021, Dettmers et al., 2023], drastically reducing training cost while retaining the expressiveness of the base model. Though widely applied in classification and instruction tuning [Ding et al., 2023, Zhang et al., 2023], their use in allocation tasks has been limited. On their own, such supervised methods capture strong priors but cannot address sequential credit assignment across turns.

Our work bridges these threads. We propose a hybrid allocator that leverages LoRA to encode cost-sensitive priors and PPO to adapt within that low-rank subspace, combining the efficiency of supervised fine-tuning with the adaptability of reinforcement learning. This situates our contribution at the intersection of efficient training algorithms and cost-aware reasoning.

3 Methodology

We view cost-aware allocation as a sequential decision problem where an allocator must balance success against compute and monetary cost. At each step t, the allocator observes state $s_t \in \mathbb{R}^{d_s}$ (dialogue features, intermediate outputs, and summaries of the m candidate agents) and selects a subset $z_t \in \{0,1\}^m$ of agents to invoke. Each agent i incurs cost $c_i > 0$ and has capability embedding $u_i \in \mathbb{R}^{d_u}$, producing stochastic outcomes: success $S_t \in \{0,1\}$ and quality $Q_t \geq 0$.

Reward and objective. We define a reward that makes efficiency a first-class optimization target:

$$r_t = \alpha\, S_t + \beta\, Q_t - \lambda \sum_{i=1}^{m} c_i z_{t,i} - \gamma \|z_t\|_0, \tag{1}$$

where $\alpha, \beta, \lambda, \gamma \geq 0$ trade off success, quality, and cost. The return $R = \sum_{t=1}^{T} \gamma_{\text{disc}}^{t-1} r_t$ (with discount $\gamma_{\text{disc}} \in [0,1]$) captures both accuracy and thrift. This formulation is equivalent to a Lagrangian relaxation of a constrained MDP, where varying (λ, γ) traces an efficiency–accuracy Pareto frontier.

3.1 PPO baseline

We first train a reinforcement learning allocator with Proximal Policy Optimization (PPO) [Schulman et al., 2017]. The policy π_θ predicts independent Bernoulli logits for each agent:

$$\pi_\theta(z_t \mid s_t) = \prod_{i=1}^{m} \mathrm{Ber}(z_{t,i}; p_{t,i}), \qquad p_t = \sigma(W_\ell\, \phi_\theta(s_t)), \tag{2}$$

where ϕ_θ encodes the state and W_ℓ projects to agent scores. A critic head $V_\theta(s_t)$ estimates expected return, stabilizing training. The PPO objective balances clipped policy updates, entropy regularization, and critic loss:

$$\mathcal{L}_{\mathrm{PPO}}(\theta) = \mathbb{E}\Big[\min\big(r_t(\theta)\hat{A}_t,\ \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big] - \eta\, \mathbb{H}(\pi_\theta) + \kappa\, \big(V_\theta(s_t) - \hat{R}_t\big)^2.$$

Because cost terms appear in r_t, PPO learns not just to succeed but to succeed *efficiently*.

3.2 Supervised allocator with LoRA

To avoid the high budget of RL, we train a supervised allocator with Low-Rank Adaptation (LoRA) [Hu et al., 2021]. Labels y_t are generated by a greedy heuristic that selects the minimal set of agents meeting a similarity threshold with task embeddings. LoRA adapters (A, B with rank $r \ll d, k$) are inserted into frozen weights W:

$$W' = W + \Delta W, \quad \Delta W = AB.$$

We optimize a cost-aware loss:

$$\mathcal{L}_{\mathrm{sup}} = \mathrm{BCE}(p_t, y_t) + \lambda_1 \|p_t\|_1 + \lambda_c \sum_i c_i p_{t,i} + \lambda_{\mathrm{cov}} \Big(\tfrac{1}{m}\sum_i p_{t,i} - \rho\Big)^2,$$

encouraging sparsity and penalizing expensive calls. This provides a strong efficiency-aware prior with minimal training cost.

3.3 LoRA-Guided PPO

Finally, we combine both approaches: PPO is warm-started from the supervised LoRA model, and updates are restricted to the LoRA subspace. This design retains cost-efficient allocation priors while allowing RL to adapt to sequential credit assignment:

$$\max_{\theta \in \Theta_{\mathrm{LoRA}}} \mathbb{E}\Big[\min\big(r_t(\theta)\hat{A}_t,\ \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big] - \eta\, \mathbb{H}(\pi_\theta) + \kappa\, (V_\theta - \hat{R}_t)^2 - \lambda_c \mathbb{E}[\textstyle\sum_i c_i p_{t,i}].$$

This hybrid makes RL training both *efficient* and *stable*, while retaining modular adapters for deployment.

4 Experiments and Results

We evaluate our allocators in two complementary environments designed to stress efficiency under sequential decision-making. The first, a synthetic *FlightPlanner* setting, requires assembling valid multi-leg itineraries under a fixed budget. Decisions unfold across planner, search, and validator calls, making early missteps costly as the horizon extends up to 7 turns. This setting highlights how compounding dependencies magnify inefficiencies.

The second, a ToolBench-style environment, involves orchestrating heterogeneous tools such as a retriever, calculator, or browser [Qin et al., 2023b]. Tasks typically resolve over 3–5 turns, and the allocator must weigh the high cost of strong tools against cheaper but weaker alternatives. This mirrors deployment realities where efficiency bottlenecks arise not from a single call, but from repeated interactions under tight budgets.

To capture efficiency, we report success rate, cost-per-success (CPS), and FLOPs-per-success. Success ensures accuracy is preserved, while CPS and FLOPs/Success directly quantify monetary and compute efficiency. Each result aggregates over 400 sampled trajectories per environment. Hyperparameters (e.g., PPO clip, entropy/value weights, LoRA ranks) follow validation-tuned defaults (Appendix A).

4.1 FlightPlanner results

Table 1 shows the synthetic FlightPlanner results. Among high-success methods, LoRA-Guided PPO achieves the lowest CPS and FLOPs/Success, reflecting strong efficiency gains. Supervised LoRA is competitive with Rule-based while requiring minimal training budget, but the hybrid consistently provides the best balance between thrift and adaptability.

Table 1: **FlightPlanner (synthetic)**. LoRA-Guided PPO yields the strongest efficiency among high-success methods.

Method	Success Rate ↑	CPS ↓	FLOPs/Success ↓
Random	0.0975	57.88	5.47×10^{12}
CapabilityMatch	0.0000	960.90	8.60×10^{13}
RuleBased	0.7200	10.58	9.80×10^{11}
CostOpt-Oracle	0.0000	660.50	4.67×10^{13}
LoRA (supervised)	0.7900	11.70	1.18×10^{12}
PPO	**0.8025**	15.23	1.46×10^{12}
LoRA-Guided PPO	0.6425	**7.71**	$\mathbf{4.73 \times 10^{11}}$

4.2 ToolBench results

Table 2 summarizes the ToolBench benchmark with an 8-agent pool. All high-performing methods reach perfect success, but efficiency varies widely. LoRA-Guided PPO achieves the lowest CPS and FLOPs/Success, outperforming PPO in thrift while matching its accuracy. This illustrates that constraining RL updates to the LoRA subspace preserves allocation efficiency while enabling sequential adaptation.

Table 2: **ToolBench benchmark** (default pool size m=8). LoRA-Guided PPO achieves perfect success at the lowest efficiency cost.

Method	Success Rate ↑	CPS ↓	FLOPs/Success ↓
Random	0.1925	118.23	2.84×10^{12}
CapabilityMatch	0.0800	218.25	4.72×10^{12}
RuleBased	1.0000	36.43	8.35×10^{11}
CostOpt-Oracle	0.1625	69.13	1.44×10^{12}
LoRA (supervised)	1.0000	67.51	1.73×10^{12}
PPO	1.0000	24.30	4.38×10^{11}
LoRA-Guided PPO	**1.0000**	**21.40**	$\mathbf{4.38 \times 10^{11}}$

5 Conclusion

Overall, our results highlight that supervised LoRA provides an economical allocator when training budgets are constrained, while LoRA-Guided PPO consistently achieves the most favorable efficiency–accuracy trade-off. By restricting RL updates to a lightweight adapter space, we avoid PPO's tendency to overspend as horizons and action sets grow, aligning with the workshop's focus on efficient reasoning under real-world resource budgets.

We presented a cost-aware approach to reasoning-agent allocation, comparing supervised LoRA, PPO, and a hybrid LoRA-Guided PPO. Our results across two environments show that parameter-efficient supervision establishes strong allocation priors at minimal training cost, while the hybrid method achieves the best trade-off between success and efficiency by constraining RL updates to a low-rank adapter space. This demonstrates that reinforcement learning, when guided by lightweight fine-tuning, can deliver controllers that are both adaptive and resource-aware.

Several open directions remain. Our benchmarks, while controlled, simplify the dynamics of real-world deployments where noisy outputs, dynamic budgets, and heterogeneous hardware constraints complicate allocation. Future work should extend to live API settings, richer tool ecosystems, and system-level metrics such as throughput under concurrency. Further, integrating efficiency-aware pretraining signals or curriculum strategies may help scale hybrid allocators to larger pools of

agents without prohibitive exploration cost. Taken together, our findings underscore the opportunity for principled, efficiency-oriented algorithms that make reasoning systems practical under tight computational and financial budgets.

References

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, et al. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Yujia Qin, Shihao Liang, Yining Ye, et al. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023a.

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

Reiichiro Nakano, Jacob Hilton, Shantanu Balaji, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Edward J Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Lu Wang, and Weizhu Wang. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Ning Ding, Xiaojie Chen, Beichen Xu, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 2023.

Aston Zhang, Xiaodong Liu, Yichong Wang, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023.

Chengwei Qin, Aston Zhang, Zhuosheng Zhang, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023b.

A Additional Experimental Details

A.1 Environments

FlightPlanner (synthetic). This environment simulates assembling multi-leg itineraries under budget constraints. The allocator must invoke planner, search, and validator sub-agents, each with heterogeneous costs. Trajectories typically unfold over 5–7 turns, where early allocation errors propagate downstream and may block feasibility. Costs are drawn from fixed distributions to approximate API latency and pricing, inducing natural trade-offs between robustness and thrift.

ToolBench-style environment. We adapt a ToolBench-style generator [Qin et al., 2023b], providing a pool of $m{=}8$ heterogeneous tools such as retrievers, calculators, and browsers. Tasks average 3–5 turns, requiring the allocator to decide whether to pay the higher cost of a strong tool or rely on cheaper but weaker options. Costs are normalized against FLOPs estimates, ensuring expensive tools proportionally reflect higher compute or latency demands.

A.2 Baselines

We compare against a range of allocation strategies:

- **Random**: uniform sampling over agent subsets.
- **CapabilityMatch**: selects agents based on similarity between task embedding and agent capability vectors.
- **RuleBased**: invokes a fixed heuristic sequence of planner, search, and validator agents.
- **CostOpt-Oracle**: an oracle with access to ground-truth task–agent utilities, selecting the minimum-cost subset that guarantees success whenever possible.
- **LoRA (supervised)**: parameter-efficient supervised allocator trained on greedy heuristic traces with low-rank adapters.
- **PPO**: reinforcement learning baseline trained from scratch using cost-aware reward.
- **LoRA-Guided PPO**: our hybrid method, warm-started from supervised LoRA and fine-tuned under PPO in the LoRA subspace.

A.3 Metrics

We evaluate our orchestrators on the following measures:

Success Rate. The fraction of tasks for which the final output satisfies all hard constraints of the environment. Formally, if $S_j \in \{0, 1\}$ indicates whether task j is solved, then

$$\text{Success Rate} = \frac{1}{N} \sum_{j=1}^{N} S_j$$

Note that the success of a task does not depend on the constrained allocation of agents or resources.

Quality. Each task j is assigned a continuous score $q_j \in [0, 10]$ reflecting the overall utility of the produced solution, even if the task fails. The definition of q_j depends on the environment:

ToolBench: Quality is proportional to token-level F1 against the reference answer:

$$q_j^{\text{TB}} = 10 \cdot \text{F1}_j,$$

where $\text{F1}_j \in [0, 1]$. This allows partial credit for near misses.

FlightPlanner: Quality is a weighted combination of cost slack and hop optimality:

$$s_{\text{cost},j} = \text{clip}\left(\frac{B_j - C_j}{B_j},\, 0,\, 1\right), \quad s_{\text{hop},j} = \text{clip}\left(\frac{h_j^\star}{h_j},\, 0,\, 1\right),$$

where B_j is the budget, C_j the plan cost, h_j the produced number of hops, and $h_j^\star$ the optimal minimum.

With weights w_{cost}=0.7, w_{hop}=0.3, we set

$$q_j^{\text{FP}} = 10 \cdot \big(w_{\text{cost}} s_{\text{cost},j} + w_{\text{hop}} s_{\text{hop},j}\big).$$

Infeasible plans receive $q_j = 0$.

Average Quality. We report the mean quality across all tasks:

$$\text{Avg. Quality} = \frac{1}{N} \sum_{j=1}^{N} q_j.$$

Cost and Cost-per-Success (CPS). Total cost is the cumulative monetary or FLOPs proxy cost across tasks. CPS normalizes by success:

$$\text{CPS} = \frac{\text{Total Cost}}{\max(1, \sum_{j=1}^{N} S_j)}.$$

FLOPs-per-Success. The total floating point operations divided by the number of successful tasks, serving as a proxy for computational efficiency.

A.4 Note on hyperparameters

Hyperparameters for PPO (clip $\epsilon = 0.2$, entropy weight $\eta = 0.01$, value weight $\kappa = 0.5$, GAE $\lambda = 0.95$) and LoRA (rank $r = 8$, learning rate $5{\times}10^{-5}$) follow common practice and were tuned on validation splits.