Extended Abstract

Learning When to Use Tools: Cost-Aware RL for Agentic Reasoning

Vikram Srinivasan (vikr4m@stanford.edu)

Modern AI agents increasingly interleave internal reasoning with calls to external tools such as calculators, retrieval systems, code execution, and search. Tools raise accuracy, but every call has a cost in latency, compute, and longer execution traces. A reward that values only the final answer gives the agent no reason to economize, so it tends to call tools more often than it needs to. A reward that penalizes tool use too heavily pushes the agent the other way, so it avoids tools even when they would help. This report studies one question: how does reward design shape the cost–accuracy tradeoff of a reinforcement learning agent that chooses between reasoning internally and calling tools? I study it in a controlled environment so the causes of overuse and underuse can be measured directly rather than confounded with large-model behavior.

I model tool use as a Markov decision process. MultiToolEnv presents one reasoning task per episode (arithmetic, retrieval, or mixed) at one of three difficulties, with four discrete actions: think, calculate, retrieve, and answer. I compare three reward designs: an outcome-only reward, a cost-aware reward with a per-call penalty λ_{tool}, and a step-shaped reward that pays a bonus for useful calls. Two choices in the environment make the tradeoff meaningful. Hiding the task type stops a fixed heuristic from reading off the answer and forces the agent to infer the task type from which features are present. An internal-solve mechanism lets the agent answer without a tool at a difficulty-dependent base rate (0.9, 0.5, 0.1), so tools become a costly accuracy boost rather than a hard requirement. Agents are trained with PPO (clipped surrogate, GAE, a small actor–critic MLP), and I add a budget-conditioned variant that observes its remaining tool budget.

The environment, reward functions, PPO trainer, six deterministic baselines, evaluation, and a parallel sweep runner are written in PyTorch around a roughly 6k-parameter MLP. The full study is a 40-run matrix: three rewards across three seeds, a seven-point λ sweep across three seeds, a budget sweep, and ablations. Each run is 50k steps on a CPU process pool, and evaluation uses 200 held-out episodes per policy.

Under the outcome reward, PPO matches the privileged oracle in accuracy (0.997) but makes 1.32 tool calls per episode, calling a tool on almost every task. The cost-aware reward makes the agent selective: it cuts calls to 1.00 while raising cost-adjusted reward from 0.865 to 0.885. Sweeping λ_{tool} traces a graded accuracy–cost frontier that peaks at $\lambda = 0.10$, and the agent drops tools in order of how little they help, easy tasks first, then medium, then hard. RL beats the fixed heuristic (0.71) only once the task type is hidden. The shaped reward is exploited: the agent farms its useful-call bonus up to 8.1 calls per episode, and cost-adjusted reward falls to 0.17.

These results separate three regimes (overuse, selective use, and underuse) and show that reward design is the lever that moves the agent between them. RL is needed when the right tool cannot be read off the observation; when it can, a one-line heuristic matches the oracle. The shaped-reward failure is a concrete case of reward misspecification. The main limitations are the synthetic environment, the hand-set internal base rates, and a single task family. The practical message is that reward design, more than the choice of RL algorithm, decides whether a tool-using agent is efficient. A cost-aware penalty produces graded, difficulty-aware selectivity, while an outcome-only reward overuses tools and naive shaping invites reward hacking.

Learning When to Use Tools: Cost-Aware Reinforcement Learning for Agentic Reasoning

Vikram Srinivasan
Department of Computer Science
Stanford University
vikr4m@stanford.edu

Abstract

Agentic systems increasingly add external tools to internal reasoning, but tool calls are not free: they add latency, compute, and complexity, and an agent trained only to be correct tends to overuse them. This report studies how reward design shapes the cost–accuracy tradeoff of a reinforcement learning agent that chooses between reasoning internally and calling tools. I build MultiToolEnv, a controlled multi-tool MDP, and train PPO agents under three reward designs (outcome-only, cost-aware, and step-shaped) against six deterministic baselines. Two choices in the environment make the tradeoff graded. Hiding the task type forces the agent to infer which tool it needs, so a fixed heuristic can no longer match a privileged oracle. An internal-solve mechanism lets the agent answer without a tool at a difficulty-dependent base rate, so tools are a costly accuracy boost rather than a hard requirement. Across a 40-run matrix, the outcome reward drives overuse (1.32 tool calls per episode at oracle-level accuracy), the cost-aware reward produces selective, difficulty-conditioned tool use that both lowers cost and raises cost-adjusted reward (best at $\lambda_{\text{tool}} = 0.10$), and the step-shaped reward is hacked into heavy overuse (8.1 calls per episode). A budget-conditioned variant learns a priority-based strategy under a hard tool-call cap. The agent beats the heuristic only when the task type is hidden, and a naive-environment ablation shows the frontier is flat, and reward design close to irrelevant, when one correct tool call is both necessary and sufficient.

1 Introduction

AI agents are increasingly built as loops that interleave internal reasoning with calls to external tools such as calculators, retrieval systems, code interpreters, web search, and APIs Yao et al. (2023); Schick et al. (2023); Qin et al. (2024). Tools extend an agent's reach and often improve accuracy, but each call has a cost: added latency, compute, money, and a longer, more brittle execution trace. A common failure mode follows from how these agents are trained. If an agent is rewarded only for the correct final answer, tool calls are free insurance: they can only help, so the agent calls tools on almost every task, including ones it could have answered on its own. Recent work documents this tendency toward excessive tool use and cognitive offloading in tool-integrated reasoning systems Wang et al. (2025). An agent penalized too heavily for tool use does the opposite and avoids tools even when they are needed. Sensible behavior sits in between, where the agent spends a call only when the expected accuracy gain outweighs its cost.

This is a reinforcement learning problem. The agent makes a sequence of decisions under cost and delayed reward: think, call a tool, observe the result, possibly call another, and eventually answer. The

final outcome depends on the whole sequence, and each intermediate action is costly. The question this project asks is:

> How does reward design shape the cost–accuracy tradeoff of a reinforcement learning agent that chooses between reasoning internally and calling tools?

I do not train a large language model. Instead I isolate the decision structure of tool use in a small, controlled environment where the causes of success and failure can be measured directly. This keeps the experiment clean: I can vary the reward, sweep its hyperparameters, control task difficulty, and read out behavioral metrics (overuse rate, underuse rate, tool productivity) that model scale and prompting would confound in a full LLM agent.

Two design choices make the study work. First, the task type is hidden from the observation. Without this, a one-line heuristic that reads the task-type flag is identical to a privileged oracle, and there is nothing for RL to learn. Hiding it forces the agent to infer which tool it needs from which features are present, which creates a real gap between the heuristic and the oracle that RL can close. Second, an internal-solve mechanism lets the agent answer without the matching tool and still succeed at a difficulty-dependent base rate (0.9 easy, 0.5 medium, 0.1 hard). Tools become a costly accuracy boost rather than a hard requirement, which makes the cost–accuracy tradeoff graded and surfaces the full overuse, selective, and underuse range within a single penalty sweep.

This project makes four contributions. First, I formalize multi-tool agentic reasoning as a controlled MDP, MultiToolEnv, with two design choices (hidden task type and internal-solve) that turn a degenerate problem into a graded one. Second, I compare three reward designs (outcome, cost-aware, shaped) under PPO and show that the outcome reward overuses tools, the cost-aware reward induces difficulty-conditioned selective use that is best at an interior λ, and the shaped reward is reward-hacked into heavy overuse. Third, I trace a graded accuracy–cost frontier and a per-difficulty selectivity pattern, and show through a naive-environment ablation that reward design only matters when tools are a graded boost rather than a hard requirement. Fourth, I add a budget-conditioned variant that learns a priority-based strategy under a binding tool-call cap.

2 Related Work

ReAct showed that language models benefit from interleaving reasoning traces with external actions, letting a model gather information mid-solution rather than committing to a single forward pass Yao et al. (2023). This motivates the structure of my environment, in which the agent repeatedly chooses between internal reasoning and external tool use. ReAct is a prompting framework, though, and does not study, through reinforcement learning, the cost–accuracy consequences of when an agent should act. Chain-of-thought prompting Wei et al. (2022) similarly elicits multi-step reasoning but treats acting and cost as out of scope.

A second line of work studies how a model learns to call tools at all. Toolformer showed that a language model can teach itself to call external APIs (calculators, search engines, QA systems), deciding when a call helps and how to fold in its output Schick et al. (2023). ToolLLM scales tool use to thousands of real APIs Qin et al. (2024), and WebGPT fine-tunes a browser-augmented model with human feedback Nakano et al. (2021). These works establish that tool use is a learnable behavior, but they optimize mainly for task success; the cost of calling tools, and the resulting overuse, is not the object of study.

Closest to my setting, StepTool casts multi-step tool learning as a dynamic decision problem and proposes step-grained reinforcement learning Yu et al. (2025). I extend this direction by making reward design itself the variable of interest. Rather than proposing one training recipe, I hold the algorithm fixed (PPO Schulman et al. (2017) with GAE Schulman et al. (2016)) and compare outcome-only, cost-aware, and step-shaped rewards in one controlled environment, reading out how each reshapes accuracy, cost, and overuse.

The motivation I operationalize comes from work on tool overuse. Acting Less is Reasoning More argues that tool-integrated reasoning systems call tools excessively, which raises cost and encourages cognitive offloading Wang et al. (2025). The risk that a shaped intermediate reward is exploited rather than followed is a long-standing reward-misspecification concern Ng et al. (1999); Amodei et al. (2016), and my step-shaped result is a reproducible instance of it. Relative to this prior work,

my contribution is a small, reproducible study that isolates how reward structure, not model scale, governs the cost–accuracy frontier of tool use.

3 Method

3.1 The MultiToolEnv MDP

I formalize tool use as a finite-horizon Markov decision process $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Each episode is a single reasoning task τ drawn from a task type in $\{\text{ARITHMETIC}, \text{RETRIEVAL}, \text{MIXED}\}$ at a difficulty in $\{\text{EASY}, \text{MEDIUM}, \text{HARD}\}$.

The observation is a 35-dimensional feature vector that encodes a difficulty one-hot, numeric operands and operator indices for arithmetic, hashed entity and attribute strings for retrieval, info bits summarizing the results of any tool calls made so far, running per-action counts, and a normalized step counter. The leading three dimensions are the task-type one-hot, and whether the agent sees them is the subject of the first design choice (Section 3.3). When the budget-conditioned variant (Section 3.5) is enabled, a 36th dimension reports the normalized remaining tool budget.

The action space is $\mathcal{A} = \{\text{THINK}, \text{CALC}, \text{RETRIEVE}, \text{ANSWER}\}$ (discrete, size 4). Think is a no-op that only incurs a step penalty. Calc invokes a deterministic calculator that parses and evaluates the task's arithmetic, Retrieve queries a 50-fact synthetic knowledge base, and Answer terminates the episode by committing the agent's current best answer. Tool calls update the corresponding info bits in the state. Calc succeeds on tasks whose type is not pure retrieval, and Retrieve succeeds on tasks whose type is not pure arithmetic, so a mixed task needs both. The episode ends on Answer or is truncated after $H = 10$ steps (recorded as incorrect). At termination the environment scores the committed answer (Section 3.4) and emits the terminal reward.

3.2 Three reward designs

Let $\mathbb{1}[\text{correct}]$ be the terminal correctness indicator, c_t the per-step indicator that a tool was called (Calc or Retrieve), and "step" a per-step constant. I compare:

$$R_{\text{outcome}} = \textstyle\sum_t \big(-\lambda_{\text{step}} \big) + \mathbb{1}[\text{correct}], \tag{1}$$

$$R_{\text{cost-aware}} = \textstyle\sum_t \big(-\lambda_{\text{step}} - \lambda_{\text{tool}}\, c_t \big) + \mathbb{1}[\text{correct}], \tag{2}$$

$$R_{\text{shaped}} = \textstyle\sum_t \big(-\lambda_{\text{step}} + \lambda_{\text{useful}}\, u_t - \lambda_{\text{wasted}}\, w_t \big) + \mathbb{1}[\text{correct}], \tag{3}$$

where u_t flags a tool call whose type matches the task and w_t flags a mismatched call. Defaults are $\lambda_{\text{step}} = 0.01$, $\lambda_{\text{tool}} = 0.05$, $\lambda_{\text{useful}} = 0.1$, and $\lambda_{\text{wasted}} = 0.05$. The outcome reward says "be correct, quickly" but is indifferent to which actions achieve correctness. The cost-aware reward charges a flat price per tool call. The shaped reward tries to teach "call the right tool" by paying an intermediate bonus, which, as the results show, is exactly the lever a policy can exploit.

3.3 Hidden task type

In the milestone environment the task-type one-hot was visible, so a rule-based heuristic that reads it and runs the matching tool plan was identical to a privileged oracle that reads the true task type from hidden metadata. Both reach perfect accuracy, and there is nothing for RL to add. I close this gap with a `-hide-task-type` flag that zeros the leading three observation dimensions. The heuristic then argmaxes over zeros and defaults to a single fixed plan (calculator), which degrades on retrieval and mixed tasks, while the oracle is unaffected. The agent can still recover the task type, but only by inferring it from feature presence: numeric operands point to arithmetic, entity and attribute hashes point to retrieval, and both present point to mixed. This change makes the oracle a real performance ceiling and turns "can RL learn to pick the right tool from latent structure?" into a meaningful question. Keeping it a flag gives a clean observable-versus-hidden ablation (Section 5.4).

3.4 Internal-solve

If a correct answer were impossible without the matching tool, tool use would be a hard requirement and the cost–accuracy tradeoff would be degenerate: exactly one call is needed, so a cost penalty has nothing to economize. To make the tradeoff graded, I add an `-internal-solve` mechanism. When the agent answers without having called the appropriate tool, it still succeeds with a difficulty-dependent base probability $p_{\text{base}}(d)$, set to 0.9 (easy), 0.5 (medium), and 0.1 (hard); calling the right tool lifts success to about 1.0. Tools therefore become a costly accuracy boost whose value depends on difficulty: on an easy task the roughly 0.1 accuracy gain rarely justifies the call, while on a hard task the roughly 0.9 gain almost always does. This is what produces the graded frontier and the per-difficulty pattern in Section 5, and it is the central change I make to the environment. Internal-solve is a designed mechanism, not a property of the task itself, and the naive-environment ablation (Section 5.8) makes the contrast explicit.

3.5 Budget-conditioned variant

To study how an agent adapts its tool use under an explicit resource constraint, I add a budget-conditioned variant (`-budget-conditioned`, `-max-budget` K). Each episode the agent is given a tool-call budget K; the normalized remaining budget is appended to the observation, and a call that would exceed the budget is blocked as a no-op. Sweeping $K \in \{1, 2, 3, \infty\}$ lets me ask whether the learned policy mechanically caps its calls or instead prioritizes its scarce calls toward the tasks where the accuracy boost is largest.

3.6 PPO and the policy network

All learned agents are trained with Proximal Policy Optimization (PPO) Schulman et al. (2017), which optimizes the clipped surrogate

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta),\, 1-\epsilon,\, 1+\epsilon)\,\hat{A}_t\big)\Big], \quad r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{4}$$

with advantages $\hat{A}_t$ from generalized advantage estimation Schulman et al. (2016). The policy is an actor–critic MLP: a shared two-layer trunk (obs $\to 64 \to 64$, $\tanh$), a categorical policy head over the four actions, and a scalar value head, with orthogonal initialization (about 6k parameters). The trunk reads its input dimension from the environment so the budget dimension flows through automatically. I use $\gamma = 0.99$, $\lambda_{\text{GAE}} = 0.95$, clip $\epsilon = 0.2$, learning rate 3×10^{-4}, rollout length 2048, batch size 64, 4 PPO epochs per update, value coefficient 0.5, entropy coefficient 0.01, and gradient-norm clipping at 1.0.

4 Experimental Setup

Unless noted, all results use the headline configuration: internal-solve enabled, task type hidden, mixed difficulty (uniform over easy, medium, and hard), and uniform task type (uniform over arithmetic, retrieval, and mixed). This is the hardest and most informative regime, since the agent must infer both what kind of task it faces and whether a tool is worth its cost.

I compare against six deterministic policies. NoTool answers immediately, which exposes the internal base rate. AlwaysCalc and AlwaysRetrieve call one fixed tool then answer. FixedSeq runs a ReAct-style sequence of think, calc, retrieve, then answer. The Heuristic reads the observation's task-type slot (zeros under hidden task type) and runs the matching tool plan; the Oracle reads the privileged true task type from hidden metadata and runs the matching plan. The heuristic and oracle differ only in whether they can see the task type, which is what the hidden-task-type design controls.

For each policy I report, over 200 held-out episodes (seed 42): accuracy, mean tool calls per episode, mean total cost (Calc and Retrieve cost 1, others 0), cost-adjusted reward (accuracy $-\ \lambda_{\text{cost}} \cdot$ cost with $\lambda_{\text{cost}} = 0.1$), tool productivity (accuracy gain over NoTool per tool call), unnecessary-tool-call rate, and underuse rate. Each learned configuration is trained for 50k environment steps with 3 seeds $\{0, 1, 2\}$. Training runs locally on a CPU process pool (six concurrent workers, one CPU per worker, since the 6k-parameter MLP is fast on CPU), with all metrics logged to local CSVs. The full study is a 40-run matrix: three rewards across three seeds for the headline, a seven-point λ_{tool} sweep across

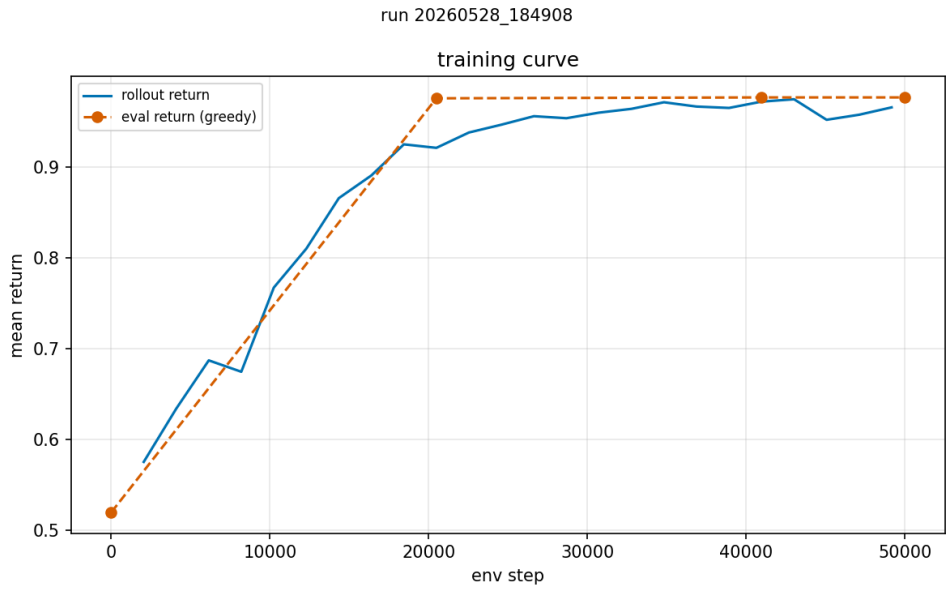


Figure 1: Representative PPO training run under the outcome reward (hidden task type, internal-solve, mixed difficulty). Rollout return and greedy-eval return both saturate well before the 50k-step budget; greedy evaluation reaches the optimal return after about 20k steps.

three seeds, a four-point budget sweep, the observable ablation, and the difficulty-generalization evaluations. Reported $\pm$ values are the population standard deviation across the three training seeds.

5 Results

5.1 Tool use under the three rewards

Table 1 is the central result. Under the outcome reward, PPO matches the Oracle in accuracy (0.997) but makes 1.32 tool calls per episode. It calls a tool on almost every task, including easy ones whose internal base rate is already 0.9, so the tool buys only about 0.1 accuracy. Tools are free insurance under an outcome-only reward, so the agent buys all of it, which is the tool-overuse failure. The cost-aware reward changes this. PPO becomes selective, making fewer calls (1.00 versus 1.32) while reaching a higher cost-adjusted reward (0.885 versus 0.865) and giving up only 1.2 accuracy points. Cost-aware PPO is also the most tool-productive policy (productivity about 0.45 versus 0.36 for the Oracle), because it spends calls only where they pay off. The step-shaped reward is reward-hacked: its useful-call bonus is farmed to 8.1 calls per episode, running nearly to the 10-step cap. Accuracy stays high (0.98) but cost-adjusted reward collapses to 0.17 (Section 5.7).

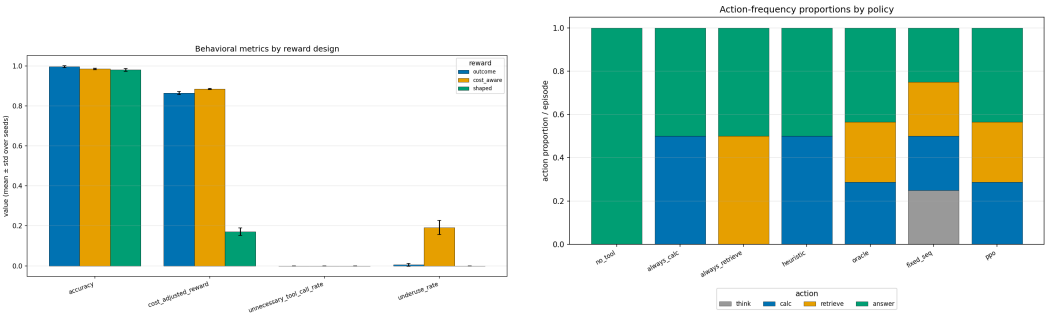
Figure 2a summarizes the same three learned policies across behavioral metrics. Accuracy is close across reward designs, but cost-adjusted reward separates them: cost-aware is highest, outcome is next, and shaped is far below. The cost-aware policy alone shows a non-trivial underuse rate (0.19), which is the visible sign of its selectivity. The action-frequency breakdown (Figure 2b) shows that cost-aware PPO's action mix matches the Oracle's (one tool, then answer) while the shaped policy spams tool actions.

5.2 The accuracy–cost frontier

Sweeping the per-call penalty λ_{tool} traces a smooth, graded accuracy–cost frontier (Table 2, Figure 3). As λ rises, tool calls fall steadily (1.285 to 0.000) and accuracy falls from 0.998 toward the internal base-rate floor of 0.535. Cost-adjusted reward is unimodal in λ and peaks at $\lambda = 0.10$ (0.894), an economizing optimum where the agent keeps the calls worth more than their cost. Beyond it, raising λ past the value of the accuracy boost causes underuse: at $\lambda = 1.0$ the agent makes zero calls and

Table 1: Headline comparison: PPO under each reward (mean $\pm$ std over 3 seeds) against the six deterministic baselines, in the enhanced environment (internal-solve, hidden task type, mixed difficulty). 200 episodes, seed 42; $\lambda_{\text{cost}} = 0.1$ for cost-adjusted reward; cost-aware uses its default $\lambda_{\text{tool}} = 0.05$. Cost-aware reaches the highest cost-adjusted reward.

Policy	Accuracy	Tool calls	Cost	Cost-adj. rew.	Unnec. rate	Underuse
PPO (outcome)	$0.997 \pm .005$	$1.322 \pm .031$	1.322	$0.865 \pm .008$	0.000	0.005
PPO (cost-aware)	$0.985 \pm .004$	$1.002 \pm .029$	1.002	$0.885 \pm .003$	0.000	0.192
PPO (shaped)	$0.980 \pm .007$	$8.093 \pm .124$	8.093	$0.171 \pm .019$	0.000	0.000
NoTool	0.535	0.000	0.000	0.535	0.000	1.000
AlwaysCalc	0.710	1.000	1.000	0.610	0.170	0.340
AlwaysRetrieve	0.700	1.000	1.000	0.600	0.180	0.360
Heuristic	0.710	1.000	1.000	0.610	0.170	0.340
Oracle	1.000	1.300	1.300	0.870	0.000	0.000
FixedSeq	0.710	2.000	2.000	0.510	0.175	0.000



(a) Behavioral metrics by reward design.

(b) Action-frequency proportions by policy.

Figure 2: (a) Accuracy is close across reward designs, but cost-adjusted reward is highest for cost-aware and collapses for shaped; cost-aware shows the only non-trivial underuse rate (0.19). (b) Cost-aware PPO's action mix matches the Oracle (one tool, then answer); the shaped policy spams tool actions, and NoTool answers immediately.

accuracy equals the no-tool floor (underuse rate 1.0). The full range from overuse to selective use to underuse is present in one sweep.

5.3 Tool use by difficulty

The mechanism behind the graded frontier is visible when I split cost-aware PPO's tool calls by difficulty (Figure 4, Table 3). The agent sheds tools in order of how little they help. At $\lambda = 0$ it calls tools on all three difficulties (overuse). By $\lambda = 0.10$ it has dropped tools on easy tasks, where the 0.9 base rate makes the 0.1 gain not worth the cost, while keeping them on medium and hard. By $\lambda = 0.30$ it begins shedding medium; by $\lambda = 0.50$ only hard tasks still receive tools (gain about 0.9, well above λ); and by $\lambda = 1.0$ even hard tasks are abandoned and accuracy tracks the raw internal base rates (0.93, 0.53, 0.09). The agent learns a difficulty-conditioned tool policy, which is the behavior the cost-aware reward is meant to produce.

5.4 Observable versus hidden task type

Table 4 isolates the effect of the hidden-task-type design. When the task-type one-hot is observable, the fixed Heuristic reads it and is identical to the privileged Oracle (both 1.000), and PPO matches them; there is nothing for RL to add. When it is hidden, the Heuristic falls to 0.710 (it defaults to calculator-only and fails on retrieval and mixed tasks), the Oracle is unchanged (it reads privileged metadata), and PPO still reaches 1.000 by inferring task type from feature presence. RL is needed only when the discriminating signal is not handed to the agent; when it is, a one-line heuristic matches the Oracle.

Table 2: λ_{tool} sweep (cost-aware PPO, mean $\pm$ std over 3 seeds). Cost-adjusted reward peaks at $\lambda = 0.10$; beyond it the agent underuses tools.

λ_{tool}	Accuracy	Tool calls	Cost	Cost-adj. rew.	Underuse
0.00	$0.998 \pm .002$	$1.285 \pm .021$	1.285	$0.870 \pm .000$	0.007
0.05	$0.982 \pm .002$	$1.015 \pm .046$	1.015	$0.880 \pm .002$	0.175
0.10	$0.975 \pm .000$	$0.813 \pm .002$	0.813	$0.894 \pm .000$	0.370
0.20	$0.973 \pm .002$	$0.808 \pm .005$	0.808	$0.892 \pm .002$	0.373
0.30	$0.925 \pm .004$	$0.595 \pm .011$	0.595	$0.866 \pm .004$	0.480
0.50	$0.727 \pm .009$	$0.223 \pm .012$	0.223	$0.704 \pm .008$	0.777
1.00	$0.535 \pm .000$	$0.000 \pm .000$	0.000	$0.535 \pm .000$	1.000

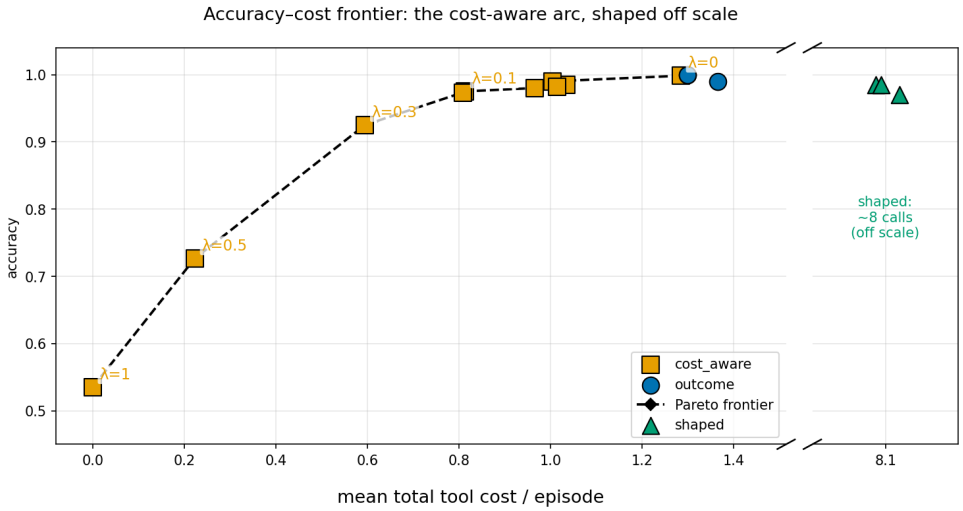


Figure 3: Graded accuracy–cost frontier (broken x-axis). The left panel zooms the cost-aware λ sweep (squares), which traces a smooth curve from the $\lambda = 1$ collapse (cost 0, accuracy 0.54) up to the $\lambda = 0$ full-use point (cost about 1.3, accuracy about 1.0); the outcome reward (circle) sits at the high-cost end and the dashed line is the efficient envelope. The right strip holds the shaped reward (triangles), shown off-scale at cost about 8 so the cost-aware arc stays legible.

5.5 Difficulty generalization

I evaluate the three mixed-trained outcome models on each difficulty separately (Figure 5). PPO matches the Oracle across every difficulty (1.000, 1.000, 0.990 for easy, medium, and hard), while the static Heuristic falls off steeply (0.946 to 0.409) because it cannot adapt its plan, and the NoTool floor exposes the internal base rates (0.932, 0.533, 0.091) built into the environment. Because the learned tool-selection policy keys on feature presence rather than on difficulty, it transfers across the difficulty distribution.

5.6 Budget adaptation

The budget-conditioned agent (Figure 6) holds about 0.985 accuracy at about 1.0 tool calls and never wastes a call (unnecessary rate 0) for generous budgets $K \in \{\infty, 3, 2\}$. At the tightest budget $K = 1$ the constraint binds: accuracy drops to 0.873 because mixed and hard tasks that need two calls (retrieve, then calculate) can no longer get the second one. The agent still spends its single call where it matters most (unnecessary rate stays 0), so it learns a budget-aware strategy that prioritizes scarce calls toward the highest-value tasks rather than a mechanical cap.

5.7 Reward hacking under the shaped reward

The step-shaped reward is the clearest failure mode. Its $+0.1$-per-useful-call bonus is meant to teach the agent to call the right tool, but because the bonus is paid per call and a useful call's type-match

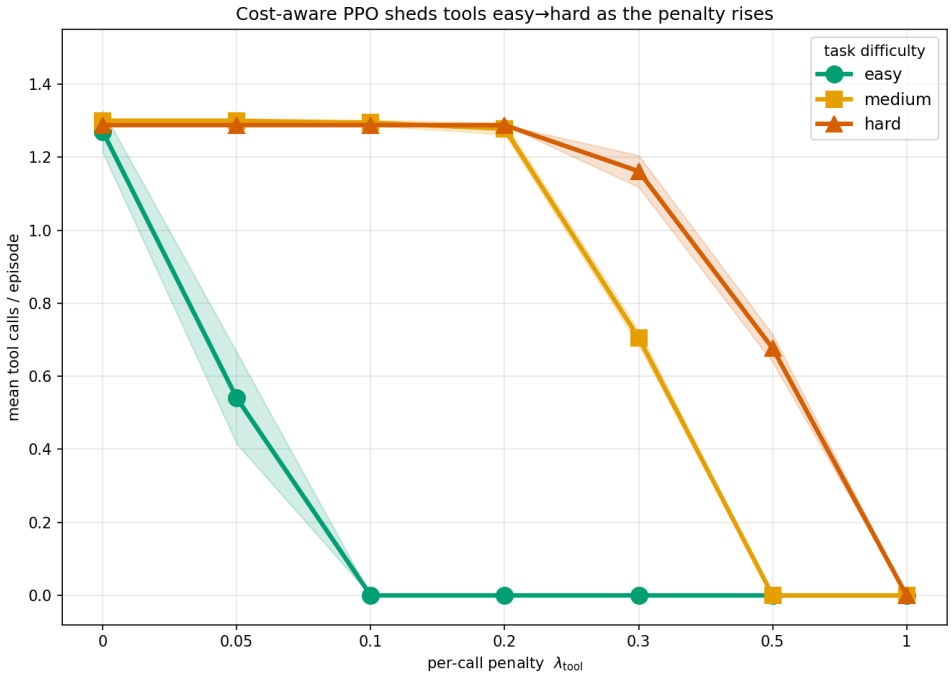


Figure 4: Cost-aware PPO sheds tools from easy to medium to hard as the per-call penalty λ rises, a difficulty-conditioned selectivity policy. Each curve is the 3-seed mean tool calls per episode for one difficulty (shaded band: $\pm$std). Easy is dropped first (off by $\lambda = 0.10$), medium next, and hard is held longest (until $\lambda = 1.0$), tracking how much a tool helps at each difficulty.

Table 3: Tool use by difficulty (numeric backup to Figure 4): cost-aware PPO mean tool calls per episode split by difficulty (3-seed mean). The agent drops tools from easy to medium to hard as λ rises. Rightmost column: easy, medium, and hard accuracy.

λ_{tool}	Easy calls	Medium calls	Hard calls	Acc. (E/M/H)
0.00	1.27	1.30	1.29	0.995/1.000/1.000
0.05	0.54	1.30	1.29	0.950/1.000/1.000
0.10	0.00	1.29	1.29	0.932/1.000/1.000
0.20	0.00	1.28	1.29	0.932/0.994/1.000
0.30	0.00	0.71	1.16	0.932/0.900/0.939
0.50	0.00	0.00	0.68	0.932/0.533/0.672
1.00	0.00	0.00	0.00	0.932/0.533/0.091

condition can be re-triggered, the policy learns to farm the bonus. It makes about 8.1 tool calls per episode (versus about 1.0 optimal), running nearly to the 10-step episode cap (mean length 9.08). Accuracy stays high (0.98) because the task is still solved, but cost-adjusted reward crashes to 0.171 (Table 1, off-frontier in Figure 3). This is a reproducible instance of reward misspecification Ng et al. (1999); Amodei et al. (2016): a well-intentioned intermediate bonus becomes the thing the agent optimizes, producing the overuse the project set out to characterize.

5.8 When reward design matters

To show why the internal-solve design is load-bearing, I re-ran the same λ sweep in a naive environment in which a single correct tool call is both necessary and sufficient for every task (the calculator returns ground truth for arithmetic and mixed, and retrieval solves retrieval). There, one call is exactly optimal, so a cost penalty has nothing to economize, and the accuracy–cost frontier is flat (Figure 7). PPO finds the same one-call optimum under outcome, cost-aware, and every $\lambda \leq 0.5$ (all accuracy about 1.0, about 1.0 calls), and the frontier only moves at the breaking point $\lambda = 1.0$

Table 4: Observable versus hidden task-type ablation (outcome reward, seed 0). Hiding the task type creates the Heuristic–Oracle gap that RL closes.

Task-type signal	Heuristic acc	Oracle acc	PPO acc	PPO tool calls
Observable	1.000	1.000	1.000	1.300
Hidden	0.710	1.000	1.000	1.300

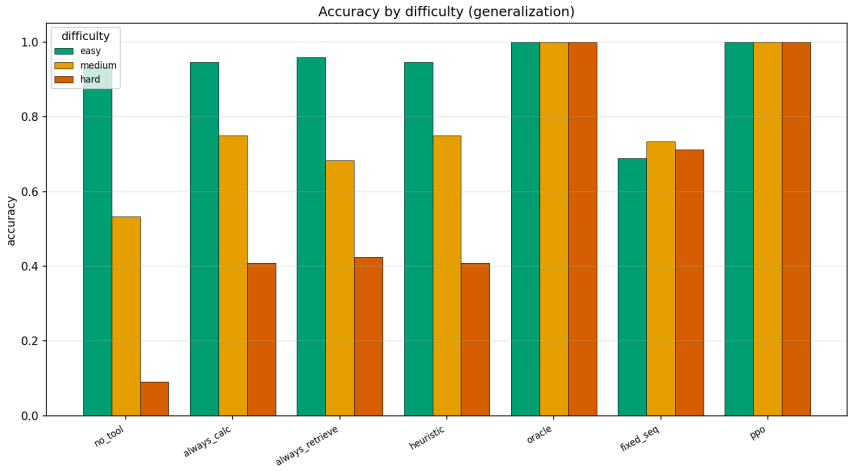


Figure 5: Accuracy by difficulty (mixed-trained models). PPO (about Oracle) holds 1.0, 1.0, 0.99 across easy, medium, and hard; the Heuristic degrades from 0.95 to 0.41; the NoTool bars expose the internal base rates 0.93, 0.53, 0.09.

where the agent collapses to never calling a tool. When tools are a hard requirement, reward design barely matters, except that the shaped reward still hacks. The internal-solve mechanism breaks the necessary-and-sufficient coupling and turns tools into a graded boost, which is what converts the flat frontier into the graded one of Section 5.2. This contrast is an artifact of a designed mechanism, and I include it as a motivating ablation that marks the regime in which cost-aware reward design is interesting.

6 Discussion

The results separate three regimes of behavior (overuse, selective use, and underuse) and show that reward design, rather than the choice of RL algorithm, is what moves the agent between them. The outcome reward sits in the overuse regime, where tools are free insurance; a moderate cost penalty (about $\lambda = 0.1$) lands in the selective regime and gives the best cost–accuracy tradeoff; a large penalty pushes the agent into underuse. The whole range is reachable by tuning a single scalar.

RL outperforms the fixed Heuristic only when the discriminating signal is hidden (Section 5.4). When task type is observable, a trivial rule matches the privileged Oracle and RL adds nothing. This is a useful check on the claim that "RL helps tool use": the value of learning here comes specifically from having to infer latent task structure, and the practical point is that RL is worth its cost when the right action cannot be read off the input.

I see three failure modes. The shaped reward is hacked into about $8\times$ overuse when a per-call bonus is farmed (Section 5.7). Pushing the cost penalty past the value of the accuracy boost drives tool calls and accuracy to the no-tool floor (Section 5.2). And at $K = 1$ the budget-conditioned agent cannot serve two-call tasks, so accuracy drops to 0.87, though it still prioritizes its single call (Section 5.6). Each is a concrete instance of a tradeoff that real agentic systems face.

The study is controlled, which bounds how far the conclusions carry. The environment is synthetic and small, the internal-solve base rates (0.9, 0.5, 0.1) are hand-set design parameters, and conclusions such as the $\lambda = 0.1$ optimum depend on their relationship to the tool cost. The task family is

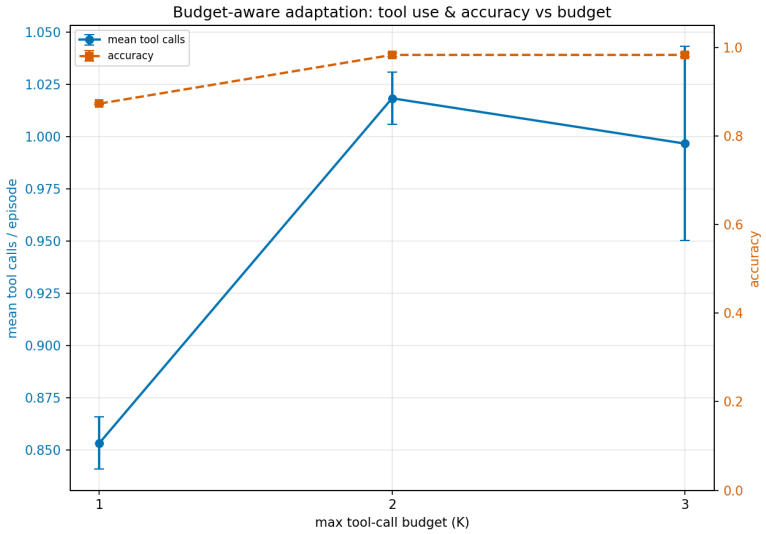


Figure 6: Budget-aware adaptation. For $K \geq 2$ the agent holds about 0.98 accuracy at about 1.0 calls; at $K = 1$ the constraint binds, dropping accuracy to 0.87 as two-call tasks cannot be fully served, while the agent still spends its single call on the highest-value task.

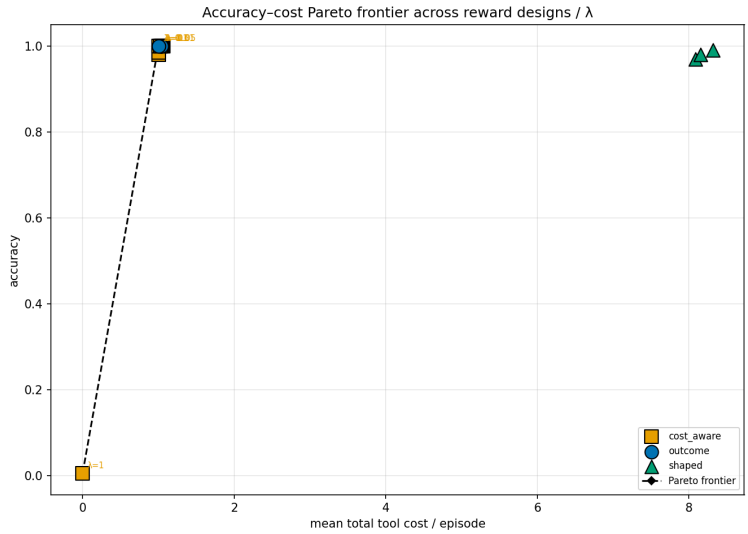


Figure 7: Naive-environment frontier (motivating ablation). When one tool call is necessary and sufficient, all $\lambda \leq 0.5$ bunch at cost about 1 and accuracy about 1.0, and only the $\lambda = 1.0$ collapse point moves, giving a flat frontier. Reward design matters only once internal-solve makes tools a graded boost (Figure 3).

narrow (arithmetic, retrieval, mixed), the tools are deterministic (a stochastic tool-failure variant was implemented but left to future work), and I use a single small MLP rather than a language model. These are the price of a clean, measurable study of reward design; scaling the findings to real LLM agents with noisy tools and richer task distributions is the natural next step.

7 Conclusion

I studied how reward design shapes the cost–accuracy tradeoff of a tool-using RL agent in a controlled multi-tool environment. Two design choices, hiding the task type and an internal-solve mechanism

that makes tools a costly accuracy boost, turn a degenerate problem into a graded one in which the full range from overuse to selective use to underuse is visible. An outcome-only reward overuses tools; a cost-aware reward gives difficulty-conditioned selective use that is best at an interior penalty and is the most tool-productive policy I measured; a naively shaped reward is hacked into heavy overuse. RL beats a fixed heuristic only when the right tool must be inferred rather than read off the input. The practical message for agentic-system designers is that reward design, not the learning algorithm, decides whether a tool-using agent is efficient, and that a single cost-per-call term is enough to steer the agent along the cost–accuracy frontier.

8 Team Contributions

This is a solo project. I was responsible for all of it: environment design (MultiToolEnv, the hidden-task-type and internal-solve mechanisms, the budget-conditioned variant), the RL implementation (PPO, the actor–critic network, the three reward functions, the six baselines), the full experiment matrix and sweep orchestration, evaluation and figure generation, the analysis, and the writing of this report and the poster.

Changes from Proposal

The core research question and method are unchanged from the proposal, but I added three things during the final sprint to sharpen the study. First, the proposal exposed task type to the agent, which made the Heuristic identical to the Oracle and left no gap for RL to close; I added the hidden-task-type design so the agent must infer task type from feature presence, which gives the observable-versus-hidden ablation (Section 5.4). Second, the proposal treated tools as effectively required; I added the internal-solve mechanism, without which the accuracy–cost frontier is flat (Section 5.8), and this is what makes the tradeoff graded. Third, the proposal listed a budget-conditioned policy as a stretch goal, and it is now a first-class variant (Section 5.6). The reward designs, baselines, metrics, and the PPO algorithm are as proposed.

AI Tools Disclosure

I used claude code to help with scaffolding and debugging, but I wrote the bulk of the code, particularly the key algorithms like PPO. I also used claude to look over my final report and poster, but they were written by me.

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete Problems in AI Safety. arXiv:1606.06565 [cs.AI]

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332* (2021).

Andrew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *International Conference on Machine Learning (ICML)*.

Yujia Qin, Shengding Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations (ICLR)*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *International Conference on Learning Representations (ICLR)*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG]

Hongru Wang, Cheng Fu, Quan Du, Jieyu Cheng, Yongqi Lin, Pengbo Liu, Fei Mi, Zhanming Wang, Zezhong Chen, Yuxuan Zhou, and Kam-Fai Wong. 2025. Acting Less is Reasoning More: Teaching Model to Act Efficiently in Tool-Integrated Reasoning. arXiv:2504.21370 [cs.CL]

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Guo, Zhiqiang Wang, and Min Zhang. 2025. StepTool: Enhancing Multi-Step Tool Usage in LLMs through Step-Grained Reinforcement Learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.