# On Learning to Think with Action Process Reward Models

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Large language models (LLMs) and reinforcement learning (RL) with verifiable task completion show exciting promise for tool calling tasks. However, extending this paradigm to multi-turn, long-horizon workflows results in sparse rewards and credit-assignment problems. Meanwhile, providing more frequent learning signal—either via step-level process feedback or supervised finetuning (SFT) on full reasoning traces—requires expensive human annotation that limits scalability. We study an alternative paradigm of learning from easy-to-collect, action-only demonstrations—successful trajectories on2ly containing actions recordable without additional effort while a human completes a task. We first find that this action-only setting results in significant performance gaps across text-based games and function-calling benchmarks compared to training on full reasoning traces. To address this, we thus introduce Action Process Reward Models (Act-PRMs). We hypothesize that action-only traces only provide part of the context necessary to generate high-return future steps. This motivates Act-PRM as an expectation-maximization (EM) approach: we treat actions as observed data behind "latent" thoughts, and train LLMs to generate thoughts that maximize the likelihood of next-step observed actions. We further show that this EM objective can be implemented as policy gradient with dense rewards via the action likelihoods—enabling LLMs to iteratively generate high-likelihood reasoning traces without human annotation. Finally, we empirically show that Act-PRMs consistently turn action-only logs into effective training signal. Across multi-turn text-based games and tool-calling benchmarks, Act-PRMs match or exceed SFT on fully-annotated reasoning traces on end-to-end task completion. They also deliver substantial improvements over action-only behavioral cloning.

## 1 Introduction

LLM agents for multi-turn tool-calling and long-horizon workflows are an increasingly important training setting. Here, a single rollout can involve many turns of thinking, calling tools, and reacting to environment feedback before producing a verifiable success or failure [1–3]. As such tasks can be checked for completion, reinforcement learning (RL) emerges as a natural training recipe [4].

However, extending this RL paradigm to long multi-turn workflows runs into three coupled challenges. First, terminal-only rewards via task completion are sparse: a single $r_T = 1$ for success and $0$ otherwise at the end of a rollout makes credit assignment difficult, where tasks with little success can result in zero learning signal. Second, while in response we may design dense per-step rewards, aligning these with global success is hard: engineered intermediate proxies are typically biased or hackable, and RL can readily exploit this gap. Finally, we may hope to circumvent these challenges with high-quality intermediate feedback. However, this can be tough to scale. Prior approaches such as process reward models (PRMs) trained on dense human-written step labels [5, 6] can be feasible

Submitted to ICML 2026 Workshop on RL from World Feedback.

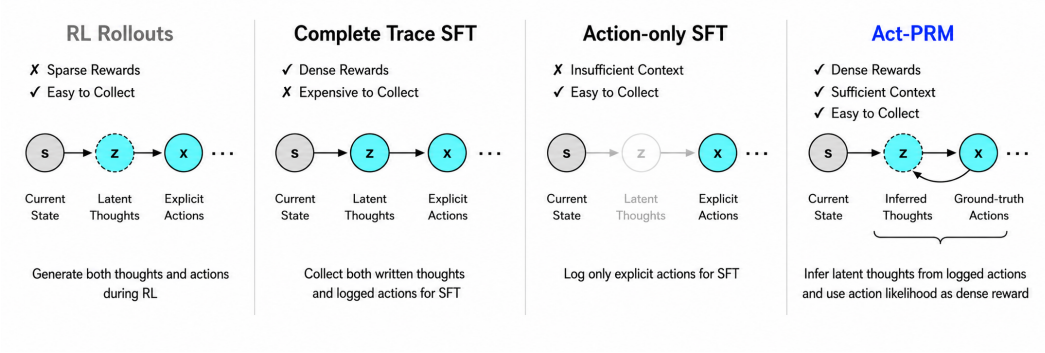


Figure 1: **Learning Signals for Long-Horizon Tool Use**. RL rollouts provide sparse rewards, while full-trace supervision gives dense signal but requires costly annotations. Action-only logs are cheap but omit latent reasoning. ACT-PRMs treats thoughts as latent variables and uses observed-action likelihood as dense feedback, turning action-only demonstrations into process-level training signal.

for grade-school math, but collecting comparable annotations for tool-calling agents can require costly expert review at every step. Automated PRM-supervision recipes such as Math-Shepherd [7] rely on domain structure that does not transfer cleanly to long agent rollouts.

We instead study an alternative paradigm of learning from *easy-to-collect, action-only demonstrations*: successful trajectories whose explicit actions—*e.g.,* tool calls or chat responses—are automatically captured in process logs without additional human annotation. The simplest way to use such data is supervised finetuning (SFT) on the action sequence. In a motivating study spanning text-world games and function-calling benchmarks (Section 2.2), however, we find that this is insufficient: across the three tasks we evaluate, action-only SFT trails SFT on full reasoning-and-action traces by 9–14 percentage points of next-step action prediction accuracy.

We hypothesize that this gap arises because action-only traces only provide *part of the context* necessary to predict future steps: missing from them are the human's implicit thoughts between recorded actions, which would otherwise supply the connecting context. Rather than asking humans to write these thoughts down, we propose to recover them using the same LLM being trained. We introduce *Action Process Reward Models* (ACT-PRMs), an expectation-maximization (EM) approach that treats actions as observed data behind latent thoughts: the E-step samples candidate thoughts $z$ at each step; the M-step updates the model to maximize the joint likelihood of the sampled thought and the observed action $x$. We show this EM objective reduces to a standard policy gradient whose per-step reward is the *action likelihood* $p_\theta(x \mid z, s)$ — a thought is rewarded exactly to the extent that conditioning on it makes the human's observed next action more probable (Section 3). The resulting signal is dense, requires no PRM annotation, and grounded in successful trajectories (Figure 1).

We empirically validate ACT-PRMs on three real-world tool-calling benchmarks and two embodied text-game tasks: $\tau^2$-bench Airline and $\tau^2$-bench Retail [8], Snorkel Finance Reasoning [9], and TextWorld Coin Collector and Treasure Hunter [10] (Section 4.2). Across these tasks, SFT on ACT-PRMs-generated thought-and-action traces *matches or exceeds* SFT on traces written by frontier LLMs as an analogue to human experts, and substantially outperforms action-only behavioral cloning.

**Contributions.**

- We introduce ACT-PRMs, a method for turning action-only demonstrations into thought-and-action training data without human reasoning annotations. We derive ACT-PRMs as the policy-gradient form of an EM objective that treats thoughts as latents behind observed actions (Section 3).
- We empirically establish a *context gap* between action-only and thought-and-action SFT: across text-world games and tool-calling benchmarks, action-only SFT consistently trails thought-and-action SFT in next-step action prediction accuracy and perplexity (Section 2.2).
- We show ACT-PRMs-generated thoughts close this gap: SFT on ACT-PRMs-relabelled traces matches or exceeds SFT on traces written by frontier LLMs and far outperforms action-only behavioural cloning across four real-world tool-calling benchmarks (Section 4.2).

## 2 Background and motivating study

### 2.1 Preliminaries

**Multi-turn LLM agents.** We consider an LLM agent policy $\pi_\theta$ that completes a task over a sequence of turns. At each timestep $t$, the model processes a state $s_t$ as input context, generates an action $a_t \sim \pi_\theta(\cdot \mid s_t)$, and receives a reward $r_t = r(s_t, a_t, o_{t+1})$ together with a new observation $o_{t+1} \sim p(\cdot \mid s_t, a_t)$. We model the state as the LLM's input context, *i.e.,* growing as $s_1 = (o_1, )$, $s_2 = (o_1, a_1, r_1, o_2)$, and in general,

$$s_t \triangleq (o_1, a_1, r_1, \ldots, a_{t-1}, r_{t-1}, o_t), \tag{1}$$

and continue until the rollout ends at horizon $T$. Following standard LLM representations, $s$, $a$, and $o$ are all text: the system prompt and initial task instruction form $o_1$, the model's first tool call is $a_1$, and the tool response is $o_2$. We denote the full rollout trajectory by

$$\tau = (o_1, a_1, r_1, o_2, a_2, r_2, \ldots, o_T, a_T, r_T, o_{T+1}), \tag{2}$$

and write $\tau_{i:j}$ for the sub-trajectory between time $i$ and $j$. We let $\tau_{i:}$ and $\tau_{:j}$ be the corresponding right- and left-truncated halves.

**Sparse terminal-success reward.** Rewards are sparse and tied to terminal task completion. Letting $y \in \{0, 1\}$ denote task success, we set $r_T = \mathbf{1}[y = 1]$ and $r_t = 0$ for $t < T$. With discount $\gamma = 1$, the step-wise return collapses to

$$R(\tau_{t:}) \triangleq \sum_{t'=t}^{T} \gamma^{T-t'} r_{t'} = y, \tag{3}$$

so every step shares the same indicator return $y$. For brevity, we omit $r_t$ in trajectories when $r_t = 0$.

**Thoughts, explicit actions, and action-only demonstrations.** We further decompose each action into a sequence of (possibly empty) *latent reasoning* tokens $z_t$, and the *explicit, observable action* tokens $x_t$ (*e.g.,* a tool call or model reply to a user):

$$a_t = (z_t, x_t), \tag{4}$$

This matches a ReAct-style interleaving of "thoughts" and "actions" [1]. However, in this work we aim to infer optimal thoughts from observable actions. We assume access to a dataset of successful task demonstrations, but—following the practical reality of process logs—we observe only the explicit actions $x_t$ at each step, not an expert's thoughts. We call this the *action-only* setting and write a partial trajectory as

$$\dot{\tau} = (o_1, \cdot, x_1, \ldots, o_T, \cdot, x_T), \tag{5}$$

where $\cdot$ denotes a missing or unobserved $z_t$.

### 2.2 Motivating study: action-only demonstrations are insufficient context

A natural first attempt at using action-only logs is to train an LLM to predict each explicit action $x_t$ from the preceding partial context $\dot{\tau}_{:t}$,—standard supervised finetuning (SFT) over action tokens As these actions lead to success, generating the same target actions at each step should result in successful task completion. Before introducing our method, we ask empirically if this is sufficient: do action-only trajectories carry enough context to predict the next action in a successful demonstration?

**Experimental setup.** For each task we collect a corpus of successful trajectories and construct two SFT variants of the same base LLM:

1. **Action-only SFT**, supervising only the explicit $x_t$ tokens given a partial-trajectory context $\dot{\tau}_{:t}$.
2. **Thoughts + Actions SFT**, supervising both $z_t$ and $x_t$ from a full-trajectory context $\tau_{:t}$ that includes the (oracle or frontier-LM-written) thoughts up to time $t$.

We evaluate each model on held-out successful trajectories using two inference-time metrics computed over the ground-truth explicit-action tokens: **action-token perplexity** and **action-token prediction accuracy** (top-1 next-token match). Unless otherwise noted, the base LLM is Qwen3-4B-Instruct-2507 [11] with LoRA ($r$=16, $\alpha$=32, on $\{q, k, v, o\}$). Full hyperparameters appear in Section A.1.

Table 1: **Motivating study: action-only SFT consistently lags Thoughts + Actions SFT on action-token prediction.** Both conditions are SFT-trained on the same successful trajectories with the same base model (Qwen3-4B-Instruct-2507 + LoRA $r$=8, $\alpha$=16); the only difference is whether the supervised target sequence contains the human's thoughts $z_t$ in addition to the explicit action $x_t$. Metrics are computed on held-out trajectories over the explicit-action tokens only (action-token perplexity, lower is better; action-token next-step prediction accuracy, higher is better). Across both text-world games and the tool-calling task, the Thoughts + Actions condition wins on both metrics.

| | Action-token PPL ↓ | | Action-token Acc ↑ | |
|---|---|---|---|---|
| Task | Action-only | Thoughts + Actions | Action-only | Thoughts + Actions |
| Treasure Hunter (TextWorld) | 1.03 | **1.01** | 0.563 | **0.661** |
| Coin Collector (TextWorld) | 1.07 | **1.02** | 0.531 | **0.674** |
| Snorkel Finance Reasoning | 1.25 | **1.14** | 0.053 | **0.144** |

**Tasks.** Following the analysis in our earlier deck, we span two families of multi-turn tasks:

- *Text-world games* built on TextWorld [10]: **Treasure Hunter** (easy) and **Coin Collector** (easy), both featuring long-horizon planning under sparse rewards.
- *Tool-calling agents*: **Snorkel Finance Reasoning**, featuring structured tool-call actions, SQL-style queries against task-specific databases, and multi-step decision workflows.

**Finding: action-only SFT consistently lags Thoughts + Actions SFT.** Table 1 reports the action-token accuracy and perplexity for both training variants on all three tasks. Across the board, the action-only model trails the Thoughts + Actions model by a substantial margin: gaps of 10–14 percentage points of action-prediction accuracy on the text-world games and roughly 9 points on the tool-calling task. Action-token perplexity of the action-only model is correspondingly higher across the board. The implication is that the explicit-action sequence alone is missing information that a model could exploit if it had access to the interleaved thoughts. This motivates recovering those thoughts from the model itself, the subject of Section 3.

### 2.3 Related work

**Process reward models and step-level supervision.** Process reward models (PRMs) replace a single outcome reward with per-step labels, typically collected by hand on math solutions [5, 6]. Math-Shepherd [7] sidesteps the labelling cost by deriving step-level reward targets from rollout statistics, but the recipe is specialized to math and assumes a verifiable per-step notion of correctness. ACT-PRMs avoids both requirements: the per-step reward is the LM's own likelihood over an action that has already been verified to lie on a successful trajectory.

**Reasoning + acting agents.** ReAct [1], Reflexion [2], Self-Refine [12], Voyager [3], ExpeL [13], Agent Workflow Memory [14], and ExpRAG [15] all show LLM agents benefit from interleaving reasoning with external actions or from accumulating experience over trajectories. Most rely on prompting alone, on human-written reasoning traces for SFT, or on outcome-based RL. We instead recover the reasoning traces themselves from action-only logs, and use this recovery as the training signal.

**Behavioural cloning and SFT on agent trajectories.** Behavioural cloning on thoughts-and-action traces is the dominant recipe for training agents on procedural data [4], but writing reasoning is the costliest part of building such datasets. Action-only BC has appeared in offline-RL framings of agent training [16–18], but as Section 2.2 demonstrates, it suffers from a significant action-prediction gap relative to thoughts-and-action SFT.

**RL as inference and EM for language models.** Viewing policy optimization as inference over latent variables has a long history [19–21]. Our derivation in Section 3 instantiates this view with the particular latent structure of thoughts behind observed actions, yielding a reward function—the action likelihood—that is computable by the same model being trained.

**In-context policy improvement.** Algorithm Distillation [22], in-context RL [23, 24], in-context decision transformers [25], and self-distillation approaches for language models [26, 27] train LLMs to absorb a policy-improvement operator into context. ACT-PRMs shares this spirit: each iteration uses the current model's in-context reasoning to produce a strictly better policy, supervised by an intrinsic, action-likelihood reward.

# 3 Action Process Reward Models

We now begin the steps to back out an RL algorithm for our setting. The Section 2.2 study showed that an action-only trajectory $\dot{\tau} = (o_1, \cdot, x_1, \ldots, o_T, \cdot, x_T)$ is an impoverished training signal: predicting the next explicit action from the previous explicit actions alone is hard, because the human who produced the demonstration was reasoning between actions in ways that $\dot{\tau}$ does not record. We view $\dot{\tau}$ as "half the picture"; the other half is the sequence of latent thoughts $z_1, \ldots, z_T$ that the human did not write down. The rest of this section derives a training algorithm for an LLM that recovers these latent thoughts itself. The resulting per-step reward model is what we call an *Action Process Reward Model* (ACT-PRM).

## 3.1 Training to infer thoughts given successful demonstrations

To recap, we are given access to a sequence of explicit actions $x_t$ and outcomes $o_{t+1}$ from successful human task completions, i.e., trajectories $\tau$ for which $p(\tau_{\text{succeeds}} \mid o_1, a_1, \ldots, o_T, a_T) = 1$. However, for each action, rather than the full $a_t = (z_t, x_t)$ we only observe $a_t = (x_t, \ )$, resulting in "partial" trajectories

$$\dot{\tau} \;=\; (o_1,\ \cdot,\ x_1,\ \ldots,\ o_T,\ \cdot,\ x_T). \tag{6}$$

We could SFT LLMs over $\dot{\tau}$ directly, as generating the explicit actions $x_t$ is enough to reach a success state. However, just $x_1, \ldots, x_t$ alone may not provide enough context to predict $x_{t+1}$ (Section 2.2).

So we are interested in learning to generate the best interweaving *inferred thoughts* $\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_T$ to fill in the complete trajectory

$$\hat{\tau} \;=\; (o_1,\ \hat{z}_1,\ x_1,\ \ldots,\ o_T,\ \hat{z}_T,\ x_T), \tag{7}$$

where "best" is determined by the $\hat{z}$ that make the observed sequence of observations and explicit actions $\hat{\tau}$ most likely.

## 3.2 An EM viewpoint with action-likelihood rewards

We view recovering thoughts $z$ from observed actions $x$ as a latent- variable problem and solve it with expectation–maximization (EM) [19, 28]. Fix a step $t$ with state $s_t$ and observed action $x_t$, and drop the $t$ subscript. At iteration $n$, with parameters $\theta^{(n)}$, define the variational distribution $q^{(n)}(z) \triangleq p_{\theta^{(n)}}(z \mid x, s)$ and the surrogate

$$Q(\theta;\, \theta^{(n)}) \;\triangleq\; \mathbb{E}_{z \sim q^{(n)}(z)}[\, \log p_\theta(x, z \mid s)\,]\,. \tag{8}$$

The E-step computes $q^{(n)}$; the M-step sets $\theta^{(n+1)} \in \arg\max_\theta Q(\theta;\, \theta^{(n)})$.

Two standard facts motivate this choice, summarised here as propositions; the proofs (Jensen's inequality and a single application of Bayes' rule, respectively) appear in Section A.2.

**Proposition 1** (EM lower-bounds the action log-likelihood)**.** *For any $\theta$ and any reference $\theta^{(n)}$,*

$$\log p_\theta(x \mid s) \;\geq\; \mathbb{E}_{z \sim q^{(n)}(z)}[\, \log p_\theta(x, z \mid s)\,] \;-\; \mathbb{E}_{z \sim q^{(n)}(z)}\Big[\, \log q^{(n)}(z)\,\Big]\,, \tag{9}$$

*with equality when $\theta = \theta^{(n)}$. Maximizing $Q(\theta;\, \theta^{(n)})$ over $\theta$ (the M-step) therefore monotonically increases $\log p_\theta(x \mid s)$, the log-likelihood of the observed successful action.*

Proposition 1 is the reason we choose EM: maximizing the joint $p(x, z \mid s)$ under the M-step is, up to a $\theta$-independent entropy term, equivalent to maximizing the marginal $p(x \mid s)$ that we actually care about. The second proposition rewrites the M-step gradient in policy-gradient form, exposing the per-step reward that ACT-PRM optimizes.

**Proposition 2** (Policy-gradient form of the M-step)**.** *The gradient of the M-step objective is*

$$\nabla_\theta Q(\theta;\, \theta^{(n)}) \;=\; \mathbb{E}_{z\sim p_{\theta^{(n)}}(z|s)}\big[\, r(z)\, \nabla_\theta \log p_\theta(x, z \mid s)\,\big], \qquad r(z) \;\triangleq\; \frac{p_{\theta^{(n)}}(x \mid z, s)}{p_{\theta^{(n)}}(x \mid s)}. \tag{10}$$

In words, the M-step gradient is a REINFORCE-style policy gradient over thoughts sampled from the current policy $p_{\theta^{(n)}}(z \mid s)$, with per-sample reward equal to the importance ratio between the thought-conditioned and marginal action likelihoods.

**Interpretation.** $r(z)$ has an immediate operational reading. Ignoring the $z$-independent denominator, $r(z)$ scores a thought by how much it raises the model's probability of the observed action $x$: thoughts that make $x$ more likely are reinforced; thoughts that make $x$ less likely are suppressed.

### 3.3 Implementation as policy gradient

The reward $r(z)$ in Equation (10) would be exact if we could compute its denominator, but $p_{\theta^{(n)}}(x \mid s) = \int p_{\theta^{(n)}}(x \mid z, s)\, p_{\theta^{(n)}}(z \mid s)\, dz$ marginalises over all thoughts $z$ and is intractable. We replace it with a Monte-Carlo estimate over the same batch of $G$ thoughts sampled from $p_{\theta^{(n)}}(z \mid s)$ in the E-step. Define the unnormalised action-likelihood reward $\tilde{r}(z) \triangleq p_{\theta^{(n)}}(x \mid z, s)$; then $p_{\theta^{(n)}}(x \mid s) \approx \frac{1}{G}\sum_g \tilde{r}(z^{(g)})$, which yields the empirical group-normalised reward

$$\bar{r}(z^{(g)}) \;\triangleq\; \frac{\tilde{r}(z^{(g)})}{\sum_{g'=1}^{G} \tilde{r}(z^{(g')})} \;=\; \frac{p_{\theta^{(n)}}(x \mid z^{(g)}, s)}{\sum_{g'=1}^{G} p_{\theta^{(n)}}(x \mid z^{(g')}, s)}, \tag{11}$$

and the policy-gradient estimator

$$\widehat{\nabla_\theta Q}(\theta;\, \theta^{(n)}) \;=\; \sum_{g=1}^{G} \bar{r}(z^{(g)})\, \nabla_\theta \log p_\theta(x, z^{(g)} \mid s). \tag{12}$$

**The training loop.** Equation (12) is just policy gradient; it implements directly as a standard RL algorithm. The Generation and Policy-Update phases correspond to the E- and M-steps from Section 3.2; see Algorithm 1 for full pseudocode.

*Generation (Expectation step).*

1. Starting with the first prompt state, sample thoughts $z_t^{(g)} \sim p_{\theta^{(n)}}(z \mid s_t)$ from the LLM.
2. Compute a per-timestep reward $\tilde{r}(z_t^{(g)}) = p_{\theta^{(n)}}(x_t \mid z_t^{(g)}, s_t)$ for every $g$, then normalise via Equation (11) to obtain $\bar{r}(z_t^{(g)})$.
3. Pick the highest-reward thought $\hat{z}_t = z_t^{(\arg\max_g \tilde{r}(z_t^{(g)}))}$ and transition to the next state $s_{t+1} = (s_t,\, \hat{z}_t,\, x_t,\, r_t,\, o_{t+1})$.
4. Repeat to the final timestep $T$, saving each trajectory step with its sampled $\{z_t^{(g)}\}$ and their rewards.

*Policy update (Maximization step).* Train the LLM via the policy gradient

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\tau_t\sim p_\theta(\tau_t)}\Big[\, \bar{r}(z_t^{(g)})\, \nabla_\theta \log p_\theta(x_t, z_t^{(g)} \mid s_t)\,\Big]. \tag{13}$$

That is, each step's update is weighted by that same step's group- normalized reward $\bar{r}(z_t^{(g)})$ from Equation (11), with no trajectory-level aggregation — a *contextual-bandit* form that follows directly from the EM derivation in Section 3.2. Two natural alternatives — trajectory-sum returns $R_t = \sum_{t'=t}^{T} \bar{r}(z_{t'}^{(g)})$ (motivated because each committed thought $\hat{z}_t$ enters the context of later steps) and group-mean-centered scores in the spirit of GRPO [29, 30] — are discussed but not trained with in this paper; see Section A.3.

**Distilling ACT-PRMs-generated traces back into the model.** After ACT-PRM training, we use the trained model to relabel each action-only trajectory with its highest-reward thoughts, producing a synthetic full-trajectory dataset $\{\hat{\tau}^{(i)}\}_i$. Standard SFT on this dataset then yields a final agent policy. Section 4 shows this recipe matches or beats SFT on thought-and-action traces written by frontier LLMs, at a fraction of the data-collection cost.

**Algorithm 1** ACT-PRM training (single trajectory, one EM iteration).

**Require:** Action-only successful trajectory $\dot{\tau} = (o_1, \cdot, x_1, \ldots, o_T, \cdot, x_T)$; group size $G$; current parameters $\theta^{(n)}$; learning rate $\eta$.

$s_1 \leftarrow o_1$
**for** $t = 1, \ldots, T$ **do** ▷ E-step: sample + score thoughts
  Sample $\{z_t^{(g)}\}_{g=1}^{G} \sim p_{\theta^{(n)}}(z \mid s_t)$
  **for** $g = 1, \ldots, G$ **do**
    $\tilde{r}(z_t^{(g)}) \leftarrow p_{\theta^{(n)}}(x_t \mid z_t^{(g)}, s_t)$ ▷ action-likelihood reward
  **end for**
  $\bar{r}(z_t^{(g)}) \leftarrow \tilde{r}(z_t^{(g)}) \,/\, \sum_{g'=1}^{G} \tilde{r}(z_t^{(g')})$ for all $g$
  $\hat{z}_t \leftarrow z_t^{(\arg\max_g \tilde{r}(z_t^{(g)}))}$ ▷ commit best thought to context
  $s_{t+1} \leftarrow (s_t,\, \hat{z}_t,\, x_t,\, r_t,\, o_{t+1})$
**end for**
$\theta \leftarrow \theta + \eta \sum_{t=1}^{T} \sum_{g=1}^{G} \bar{r}(z_t^{(g)}) \nabla_\theta \log p_\theta(x_t, z_t^{(g)} \mid s_t)$ ▷ M-step: Equation (13)

**Ensure:** Updated parameters $\theta^{(n+1)} \leftarrow \theta$.

# 4 Experiments

We empirically validate ACT-PRMs on five multi-turn agent tasks spanning real-world tool calling and embodied text-game environments. Our central question is whether ACT-PRMs-trained models actually generate thoughts that close the action-only context gap established in Section 2.2 — that is, whether they substantially improve action-token perplexity and action-token prediction accuracy over the same ground-truth expert trajectories. We then provide supporting evidence that the ACT-PRMs action-likelihood reward rises smoothly during training.

## 4.1 Setup

**Tasks.** We evaluate on three real-world tool-calling benchmarks spanning retail, travel, and finance, plus two embodied text-adventure tasks:

- **$\tau^2$-bench Airline** [8]: airline customer-support workflows with structured tool calls.
- **$\tau^2$-bench Retail** [8]: retail customer-support workflows under the same dual-control evaluation protocol as $\tau^2$-bench Airline.
- **Snorkel Finance Reasoning** [9]: multi-turn financial question-answering against per-company SQL databases, requiring reasoning over disclosure tables.
- **TextWorld Coin Collector (easy)** [10]: navigate through a procedurally generated dungeon to pick up a coin, using a fixed set of text-game actions (`go`, `take`, `open`, ...).
- **TextWorld Treasure Hunter (easy)** [10]: traverse a small dungeon to retrieve a single specified treasure object, again using the standard text-game action vocabulary.

**Base model and ACT-PRMs training.** We use Qwen3-4B-Instruct-2507 [11] as the base LLM with LoRA finetuning ($r$=8, $\alpha$=16, on $\{q, k, v, o\}$ projections), trained with Tinker [31] as the rollout and update driver. ACT-PRMs uses group size $G = 8$ thoughts per state and the self-normalised reward $\bar{r}$ from Equation (11). Full hyperparameters are in Section A.1.

**Comparisons.** We compare three downstream agents trained from *the same* corpus of successful action-only trajectories, differing only in the supervision they receive:

- **Action-only BC.** SFT on raw action-only logs $\dot{\tau}$, supervising only the explicit action tokens. This is the "half the picture" lower bound.
- **Frontier-LLM trace SFT.** SFT on full thought-and-action traces in which the thoughts $z_t$ are generated by frontier LLMs conditioned on the observed action sequence. We average over thoughts produced by Claude Opus 4, Claude Sonnet 4, o3, o4-mini, and Gemini 2.5 Pro to control for any single model's idiosyncrasies. This is an expensive upper-bound baseline that assumes dense thought annotations from the strongest available models.

Table 2: **Main results: ACT-PRMs closes the action-only context gap across tool-calling and embodied text-game tasks.** "$\tau^2$-Airline" / "$\tau^2$-Retail" refer to $\tau^2$-bench Airline and Retail [8]; "Snorkel Finance" is Snorkel Finance Reasoning; "TW Coin" and "TW Treasure" are the easy difficulty of the TextWorld Coin Collector and Treasure Hunter games [10].

| | $\tau^2$-Airline | | $\tau^2$-Retail | | Snorkel Finance | | TW Coin | | TW Treasure | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | PPL ↓ | Acc ↑ | PPL ↓ | Acc ↑ | PPL ↓ | Acc ↑ | PPL ↓ | Acc ↑ | PPL ↓ | Acc ↑ |
| Action-only BC | 1.56 | 0.155 | 1.49 | 0.249 | 1.25 | 0.053 | 1.07 | 0.531 | 1.03 | 0.563 |
| Frontier-trace SFT | 1.59 | 0.255 | 1.28 | 0.453 | 1.14 | 0.144 | 1.02 | 0.674 | 1.01 | 0.661 |
| ACT-PRMs SFT | 1.58 | 0.253 | 1.36 | 0.376 | 1.35 | 0.299 | 1.07 | 0.929 | 1.14 | 0.739 |

- **ACT-PRMs (ours).** SFT on full thought-and-action traces whose thoughts are produced by relabelling the action-only data with ACT-PRMs (Algorithm 1).

**Metrics.** We evaluate every method on held-out successful trajectories using two inference-time metrics over the ground-truth explicit-action tokens: **action-token perplexity** (lower is better) and **action-token prediction accuracy** (top-1 next-token match; higher is better). Both metrics are computed with the model's own inference-time context: previous thoughts come from the model, previous explicit actions are forced to the ground-truth.

### 4.2 Main result: ACT-PRMs closes the action-only context gap

**ACT-PRMs substantially beats action-only BC.** Wherever both conditions ran, ACT-PRMs SFT delivers substantially higher final action-token accuracy than action-only BC: $+10.0$ points on $\tau^2$-bench Airline ($0.253$ vs $0.155$), $+12.7$ on $\tau^2$-bench Retail ($0.376$ vs $0.249$), and $+24.6$ on Snorkel Finance Reasoning ($0.299$ vs $0.053$). This is the core claim: when the missing thought context is filled in by ACT-PRMs, the same explicit-action targets become substantially easier to predict.

**ACT-PRMs matches or exceeds frontier-trace SFT on two of three completed tasks.** On $\tau^2$-bench Airline, ACT-PRMs SFT and frontier-LLM-trace SFT finish within $0.002$ of each other on action-token accuracy ($0.253$ vs $0.255$) and within $0.01$ on perplexity ($1.58$ vs $1.59$) — effectively a tie. On Snorkel Finance Reasoning, ACT-PRMs-relabelled traces deliver a $+15.5$-point accuracy gain over the GPT-5-mini frontier-trace baseline ($0.299$ vs $0.144$). This matches our central hypothesis — thoughts whose score is the action likelihood under the training model are at least as useful for SFT as thoughts written by an external frontier model that did not produce the action.

## 5 Discussion

**Limitations and future work.** In this work, we demonstrate how we can train models to infer or generate thoughts behind action-only trajectories, enabling more effective learning from easy-to-collect demonstrations. Scaling base-model size (Qwen3-8B / 32B, non-INSTRUCT variants), running multiple EM iterations, and combining the per-step action-likelihood signal with a sparse verifiable terminal reward during multi-trajectory RL rollouts are the natural next steps.

**Conclusion** We show how to make action-only logs a competitive supervision source, by training a model to infer the latent thoughts via Act-PRMs. We derive how Act-PRMs can be implemented as a policy-gradient form of classifcal EM step against the marginal action log-likelihood (Propositions 1 and 2), yielding a dense per-step signal that is cheap to compute and grounded in successful trajectories. Across $\tau^2$-bench Airline and Retail, Snorkel Finance Reasoning, and the TextWorld games, SFT on ACT-PRM-relabelled traces matches or exceeds SFT on frontier-LLM-authored traces while requiring no annotation beyond the action-only logs that the agent already produces.

## 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, 2023. URL https:

//arxiv.org/abs/2210.03629.

[2] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

[3] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[4] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

[5] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

[6] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

[7] Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

[8] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. $\tau^2$-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

[9] Snorkel AI. Snorkel ai leaderboard, finance reasoning. *Snorkel AI*, 2026. URL https://snorkel.ai/leaderboard/category/financereasoning/.

[10] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. TextWorld: A learning environment for text-based games. In *Computer Games Workshop at IJCAI*, 2018. URL https://arxiv.org/abs/1806.11532.

[11] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[12] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

[13] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

[14] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

[15] Thomas Palmeira Ferraz, Romain Deffayet, Vassilina Nikoulina, Hervé Déjean, and Stéphane Clinchant. Retrieval-augmented LLM agents: Learning to learn from experience, 2026. URL https://arxiv.org/abs/2603.18272.

[16] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021. URL https://arxiv.org/abs/2106.01345.

[17] Thomas Schmied, Josef Dean, Valentin Flunkert, and Sepp Hochreiter. Retrieval-augmented decision transformer, 2024. URL https://arxiv.org/abs/2410.07071.

[18] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

[19] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018. URL https://arxiv.org/abs/1805.00909.

[20] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

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

[22] Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation. In *International Conference on Learning Representations*, 2023.

[23] Yaoqi Ye, Yiran Zhao, Keyu Duan, Zeyu Zheng, Kenji Kawaguchi, Cihang Xie, and Michael Qizhe Shieh. In-context reinforcement learning for tool use in large language models. *arXiv preprint arXiv:2603.08068*, 2026.

[24] Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes, 2022. URL https://arxiv.org/abs/2208.01066.

[25] Sili Huang, Jifeng Hu, Hechang Chen, Lichao Sun, and Bo Yang. In-context decision transformer: Reinforcement learning via hierarchical chain-of-thought, 2024. URL https://arxiv.org/abs/2405.20692.

[26] Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation, 2026. URL https://arxiv.org/abs/2601.20802.

[27] Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models, 2026. URL https://arxiv.org/abs/2601.18734.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1): 1–38, 1977.

[29] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[30] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645:633–638, 2025.

[31] Thinking Machines. Announcing Tinker: An API for post-training frontier models. https://thinkingmachines.ai/blog/announcing-tinker/, 2025. Blog post.

Table 3: **Section 2.2 motivating-study datasets.** Each task's training corpus is the set of successful trajectories (out of all attempted) produced by GPT-5-mini at `reasoning_effort=med` on the task's training split. Both the action-only and Thoughts + Actions SFT conditions train on the same trajectory set; the only difference is whether the human's thoughts $z_t$ are included in the supervision target.

| Task | Eval env config | # Successful / total | # Unique tasks |
|---|---|---|---|
| Treasure Hunter (TextWorld, easy) | `textworld/treasure_hunter_easy` | 219/320 | 67 |
| Coin Collector (TextWorld, easy) | `textworld/coin_collector_easy` | 278/320 | 78 |
| Snorkel Finance Reasoning | `finqa/reasoning` | 103/248 | 34 |

# A Additional details

## A.1 Hyperparameters and training commands

**Shared hyperparameters (motivating study, Section 2.2).** All Section 2.2 SFT runs share the following settings; only the environment configuration (and the `-actions_only` flag for the action-only condition) varies across runs.

- **Base model:** Qwen3-4B-Instruct-2507 [11].
- **LoRA:** rank $r$=8, scaling $\alpha$=16, on $\{q, k, v, o\}$ projections (config: `r8_a16_qkvo`).
- **Optimsizer:** AdamW, learning rate 1e−3.
- **Batch sizes:** mini-batch size 8 with gradient-accumulation steps 8.
- **Data:** for each task, successful trajectories produced by GPT-5-mini at `reasoning_effort=med` are used as the training corpus (the action-only condition strips the thoughts from these trajectories; the Thoughts + Actions condition keeps them). All runs use `success_only=true` and `hide_observations=true` (the step-wise observation-hiding training described in Section 2.1).

**Main-results training (Section 4.2).** All runs share the same Qwen3-4B-Instruct-2507 base model and LoRA configuration (`r8_a16_qkvo`: $r$=8, $\alpha$=16, target modules $\{q, k, v, o\}_{\text{proj}}$, dropout 0.05). All runs use mini-batch size 32 and gradient accumulation 32 (effective batch size 1024). Three deviations are worth flagging:

## A.2 EM-to-policy-gradient derivation

This appendix collects the two derivations omitted from Section 3.2: the variational lower bound that justifies the EM objective (Proposition 1) and the importance-sampling rewrite that turns the M-step into a policy gradient (Proposition 2). We follow the notation of Section 3.2: a single step with state $s$, observed action $x$, latent thought $z$, and a reference parameter $\theta^{(n)}$ from the previous EM iteration. Define the variational distribution $q^{(n)}(z) \triangleq p_{\theta^{(n)}}(z \mid x, s)$ and the surrogate $Q(\theta;\, \theta^{(n)}) = \mathbb{E}_{z\sim q^{(n)}}[\log p_\theta(x, z \mid s)]$.

*Proof of Proposition 1.* Starting from the marginal log-likelihood and introducing $q^{(n)}$,

$$
\begin{aligned}
\log p_\theta(x \mid s) &= \log \int p_\theta(x, z \mid s)\, dz \\
&= \log \int q^{(n)}(z)\, \frac{p_\theta(x, z \mid s)}{q^{(n)}(z)}\, dz \\
&\geq \int q^{(n)}(z)\, \log \frac{p_\theta(x, z \mid s)}{q^{(n)}(z)}\, dz \qquad \text{(Jensen's inequality on } \log) \\
&= \mathbb{E}_{z\sim q^{(n)}}[\log p_\theta(x, z \mid s)] \;-\; \mathbb{E}_{z\sim q^{(n)}}[\log q^{(n)}(z)] \\
&= Q(\theta;\, \theta^{(n)}) \;+\; H(q^{(n)}), \qquad (14)
\end{aligned}
$$

where $H(q^{(n)}) = -\mathbb{E}[\log q^{(n)}]$ does not depend on $\theta$.

The gap between $\log p_\theta(x \mid s)$ and the right-hand side of Equation (14) equals $\mathrm{KL}(q^{(n)} \,\|\, p_\theta(z \mid x, s))$, which vanishes when $\theta = \theta^{(n)}$ since $q^{(n)} = p_{\theta^{(n)}}(z \mid x, s)$ by construction. The bound is therefore

tight at $\theta = \theta^{(n)}$, and any $\theta^{(n+1)}$ that increases the right-hand side — in particular, the M-step maximizer of $Q(\,\cdot\,;\,\theta^{(n)})$ — must satisfy $\log p_{\theta^{(n+1)}}(x \mid s) \geq \log p_{\theta^{(n)}}(x \mid s)$. This is the standard monotonic-improvement property of EM. □

**Consequence.** Maximizing the joint log-likelihood $\log p_\theta(x, z \mid s)$ under the M-step expectation is therefore not a heuristic surrogate for our actual objective: it is a sound variational substitute for maximizing the marginal $\log p_\theta(x \mid s)$, the log-likelihood of the observed successful action that ultimately matters for downstream deployment.

*Proof of Proposition 2.* Expand $Q(\theta;\,\theta^{(n)})$ as an integral against $q^{(n)}(z) = p_{\theta^{(n)}}(z \mid x, s)$ and apply Bayes' rule $p_{\theta^{(n)}}(z \mid x, s) = p_{\theta^{(n)}}(x \mid z, s)\, p_{\theta^{(n)}}(z \mid s) / p_{\theta^{(n)}}(x \mid s)$:

$$
\begin{aligned}
Q(\theta;\,\theta^{(n)}) &= \int p_{\theta^{(n)}}(z \mid x, s)\, \log p_\theta(x, z \mid s)\, dz \\
&= \int \frac{p_{\theta^{(n)}}(x \mid z, s)\, p_{\theta^{(n)}}(z \mid s)}{p_{\theta^{(n)}}(x \mid s)}\, \log p_\theta(x, z \mid s)\, dz \\
&= \mathbb{E}_{z \sim p_{\theta^{(n)}}(z \mid s)}\big[\, r(z)\, \log p_\theta(x, z \mid s)\,\big], \qquad (15)
\end{aligned}
$$

where $r(z) \triangleq p_{\theta^{(n)}}(x \mid z, s) / p_{\theta^{(n)}}(x \mid s)$. The sampling distribution $p_{\theta^{(n)}}(z \mid s)$ and the importance ratio $r(z)$ both depend only on the frozen reference $\theta^{(n)}$, so $r(z)$ is treated as a constant when differentiating with respect to $\theta$:

$$
\nabla_\theta Q(\theta;\,\theta^{(n)}) \;=\; \mathbb{E}_{z \sim p_{\theta^{(n)}}(z \mid s)}\big[\, r(z)\, \nabla_\theta \log p_\theta(x, z \mid s)\,\big], \qquad (16)
$$

which is exactly Equation (10). This is the REINFORCE-style score- function estimator with on-policy sampling from $p_{\theta^{(n)}}(z \mid s)$ and per-sample reward $r(z)$. □

The denominator of $r(z)$, the marginal action likelihood $p_{\theta^{(n)}}(x \mid s) = \int p_{\theta^{(n)}}(x, z \mid s)\, dz$, is not tractable in closed form. Section 3.3 estimates it via the same Monte-Carlo batch used in the score-function update, yielding the group-normalised reward $\bar{r}$ of Equation (11) and the empirical policy-gradient estimator in Equation (12).

### A.3 Reward and return: design space beyond contextual bandit

Section 3.3 commits to a single reward and return choice: each step's policy-gradient update is weighted by that step's group-normalized empirical reward $\bar{r}(z_t^{(g)}) = \tilde{r}(z_t^{(g)}) / \sum_{g'} \tilde{r}(z_t^{(g')})$ (Equation (11)), with no aggregation across timesteps. This *contextual-bandit* form falls out directly of the EM derivation: the M-step objective for step $t$ depends only on the action $x_t$ and its candidate thoughts $\{z_t^{(g)}\}$, not on future steps' rewards. It is also what our trainer code implements (`ActPrmTrainer`'s `action_prompt_act_prm` generator with `reward_method="em"`).

Two natural alternatives sit just outside what we train with. We have implemented neither and we do not claim experimental results for them; they are described here as design choices a future iteration could ablate:

**Unnormalized contextual-bandit return.** Replace $\bar{r}$ with the raw action-likelihood $\tilde{r}(z_t^{(g)}) = p_{\theta^{(n)}}(x_t \mid z_t^{(g)}, s_t)$. This drops the intractable-normalsizer approximation but loses the relative-ranking property of the EM-normalized form. Our codebase supports this via `reward_method="action_probs"`.

**Trajectory-sum return.** Because each committed thought $\hat{z}_t$ enters the context of later steps, summing rewards along the trajectory $R_t = \sum_{t'=t}^{T} \bar{r}(z_{t'}^{(g)})$ would also be motivated. This is not implemented in our default training path; a proof-of-concept lives in `generator/tinker_act_prompt_aprm_nobandit.py` but is not exercised by the main trainer.

**Group-mean-centered return (GRPO-style).** Centring the unnormalized rewards group-wise gives signed scores in the spirit of GRPO [29, 30]: $\tilde{r}_{\text{grpo}}(z^{(g)}) = \tilde{r}(z^{(g)}) - \frac{1}{G} \sum_{g'=1}^{G} \tilde{r}(z^{(g')})$. Not currently implemented.