A Few Teacher Steps Go a Long Way: Cost-Efficient On-Policy Data Augmentation for Agent Post-Training

Junze Ye[1,*] **Jiayi Cheng**[2,*] **Miao Lu**[1,*] **Michal Mankowski**[2] **Jose Blanchet**[1]
Mohsen Bayati[1]
[1]Stanford University [2]New York University

Abstract

For LLM agents, supervised fine-tuning is not only about teacher labels' quality, but also about which interaction contexts those labels condition on. Pure behavioral cloning uses complete teacher rollouts, creating a mismatch between teacher-induced contexts seen in training and student-induced contexts encountered at test time. Recent work addresses this mismatch by querying a teacher at contexts reached by the student, often with increasingly elaborate filtering of the teacher's continuations. We instead frame on-policy data construction as a budget-allocation problem: under matched supervision resources, should teacher output be spent on more complete rollouts, longer continuations, outcome-filtered, or broader coverage of learner-induced contexts? We formalize this design space through the rollout policy, switch-time distribution, continuation horizon, filtering rules, and two complementary costs: teacher inference generated before filtering and teacher supervision retained for SFT. Across HotpotQA, ALFWorld, and Terminal-Bench-Dev, bounded unfiltered teacher continuations at learner-induced contexts improve over pure behavioral cloning at matched budgets. On HotpotQA, where we run the full comparison, few-step continuations match or exceed success-filtered and critical-context-filtered alternatives. Overall, our findings suggest that a few teacher steps, placed at learner-induced contexts, can be a more cost-effective source of supervised signal than longer or more heavily curated teacher completions.

1 Introduction

Supervised fine-tuning (SFT) trains a language-model agent by teacher forcing on labeled assistant turns. In agent post-training, those labels are often generated by a capable but expensive teacher; the simplest dataset is then a corpus of end-to-end teacher demonstrations. We call this instantiation pure behavioral cloning. It is easy to implement, but inherits a familiar weakness from imitation learning: the student is trained on input contexts induced by the teacher, while at test time it acts on contexts induced by its own earlier decisions. For multi-turn agents, an early imperfect action changes the subsequent observation stream, after which the student may see contexts that are poorly represented in the teacher-demonstration corpus. This motivates DAgger-style interactive imitation learning [Ross et al., 2011, Kelly et al., 2019], and recent LLM-agent work has adapted the idea by querying a teacher at states reached by the student [Choudhury and Sodhi, 2025, Lyu et al., 2025, Lauffer et al., 2025, Li et al., 2026].

Once one collects teacher supervision at learner-induced states, however, the central design question is not simply whether to use "on-policy" data. It is how to allocate a limited supervision budget. Additional teacher output could be spent on more end-to-end teacher demonstrations, longer teacher continuations after a student prefix, outcome-filtered continuations, or bounded corrections spread across more learner-induced states. These choices can all be described as adding supervised data,

*Equal contribution. Contact {junze, miaolu}@stanford.edu, jiayicheng@nyu.edu.

Accepted by ICML 2026 Workshop: RLxF: Reinforcement Learning from World Feedback.

but their costs are not comparable under a generic unit such as "one training sample." We therefore track two concrete cost axes: C_{i}, the total teacher inference spent before filtering, and C_{tr}, the teacher-generated supervision retained and trained on per SFT epoch. This separates protocols that are cheap to generate from those that are cheap to train on after filtering.

A second comparability issue is that even the phrase "randomly sample a learner state" is ambiguous. In fixed-horizon settings, time-uniform state sampling is intuitive. But in agentic LLM tasks where episodes terminate at random times, sampling uniformly within each completed trajectory weights episodes equally, while sampling from a fixed global horizon and rejecting invalid switches gives longer rollouts more mass. We make these choices explicit through a unified data-construction framework specifying the rollout policy, switch-time distribution, continuation horizon, filtering rules, and induced context distribution. The goal of this framework is not to introduce a more complex algorithm, but to modularize the design choices of on-policy data augmentation protocols.

We then study a stylized one-round augmentation setting. First, train an initial behavioral-cloning checkpoint on a teacher-demonstration corpus $\mathcal{D}_{\mathrm{BC}}$. Next, roll out this student on training tasks, sample contexts from the learner-induced trajectories, and ask the teacher to continue from those contexts. Finally, reinitialize from the base student and train on $\mathcal{D}_{\mathrm{BC}} \cup \mathcal{D}_{\mathrm{OP}}$. We call this recipe BC+OP. Our main variant, OP-SHORT, keeps every sampled teacher continuation but caps it to a small number K of teacher turns. The continuation length K controls a breadth-depth tradeoff: a few teacher steps can show local recovery behavior after a student-induced prefix, but longer continuations spend more budget on teacher-descendant contexts rather than on covering additional learner-induced states. We compare OP-SHORT with PURE-BC, which spends the budget on additional complete teacher demonstrations, and with filtered variants OP-SUCCESS and OP-CRITICAL, which keep successful teacher continuations and, in the latter case, focus queries on tasks the student fails.

Across HotpotQA (question answering with search tool) [Yang et al., 2018], ALFWorld (text-based embodied planning) [Shridhar et al., 2021], and Terminal-Bench-Dev (command-line coding tasks) [OpenThoughts-Agent Team, 2025], bounded teacher continuations at learner-induced states are a consistently cost-effective form of SFT data augmentation. At matched supervision budgets, OP-SHORT improves over scaling PURE-BC. On HotpotQA, where we run the full comparison, unfiltered few-step continuations match or exceed the filtered alternatives depending on the cost axis. On ALFWorld and Terminal-Bench-Dev, the best continuation length is larger than in HotpotQA, but additional teacher turns still do not monotonically improve performance. On Terminal-Bench-Dev, OP-SHORT with a small fraction of the SFT corpus and no reinforcement-learning stage matches the OpenThoughts-Agent baseline that uses the full corpus followed by reinforcement learning.

Our results suggest a practical rule for agent SFT under constrained supervision budgets: use pure teacher demonstrations to obtain a reasonable initial student, then allocate additional budget to bounded teacher continuations at learner-induced contexts before investing in full rollouts or heavier filtering. The right continuation length is task-dependent, but across our experiments a few teacher steps often go a long way. Our contributions are:

1. We formulate supervised data construction for agent post-training as a budget-allocation problem, separating teacher-inference cost C_{i} from retained SFT training cost C_{tr}.

2. We provide a formal framework for describing on-policy data augmentation that makes explicit the context-generation policy, switch-time distribution, continuation horizon, filtering rules, and how they affect the induced context distribution in variable-length rollouts.

3. We empirically compare bounded unfiltered teacher continuations, filtered continuations, critical-context filtering, and pure behavioral cloning across three agentic benchmarks, finding that a few on-policy teacher steps are often the most cost-effective use of additional teacher supervision.

2 Methods

2.1 Modeling Agentic Interactions as a POMDP

Our experiments involve ReAct-style agents [Yao et al., 2022], each parametrized by an LLM policy $\pi \in \Pi$. Such an agent interacts with a digital environment by generating assistant turns that optionally

contain tool call(s) and receiving environment responses in ChatML format.[2] An agent's goal is to maximize terminally verifiable rewards on a domain-specific collection of tasks $\mathcal{Q}$, split into a training set $\mathcal{Q}_{\text{train}}$ and a test set $\mathcal{Q}_{\text{test}}$ for performance reporting. Specifying the task collection naturally includes a collection of environments and reward functions. We model the aggregate domain-level dynamics as a finite-horizon Partially Observable Markov Decision Process (POMDP, [Smallwood and Sondik, 1973, Kaelbling et al., 1998]), denoting it by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathrm{P_s}, \mathrm{P_o}, \mu, r \mid T_{max}, \mathcal{H} \rangle$. In agentic settings, a task $q \in \mathcal{Q}$ is often identified by its initial state, transition and observation kernels, and reward function; to simplify notation, we mostly suppress this q-dependence below.

$\mathcal{S}$ is the latent *environment state* space. $\mathcal{A}$, or action set, is the text space of producible assistant turns. The observation space $\mathcal{O}$ is the text space of one-step environment responses returned by the digital environment and mediated by an agent scaffold.[3] At turn $t > 0$, a full assistant turn (a token sequence ending at a pre-programmed stop rule) is sampled from the LLM policy,

$$a_t \sim \pi(\cdot \mid h_{t-1}), \tag{1}$$

where $h_{t-1} \in \mathcal{H}$ is the LLM-readable context we define later. The environment then evolves according to the transition kernel while the agent scaffold extracts the environment response:

$$s_t \sim \mathrm{P_s}(\cdot \mid s_{t-1}, a_t), \qquad o_t \sim \mathrm{P_o}(\cdot \mid s_t). \tag{2}$$

This distributional notation also covers the special case of deterministic digital settings where $\mathrm{P_s}$ and/or $\mathrm{P_o}$ are Dirac delta kernels. A rollout terminates at the stopping time $T \leq T_{max}$, either when the assistant turn indicates task completion or when the horizon is reached. We set $o_T \equiv \varnothing$.

Let μ be the empirical distribution over tasks $q \in \mathcal{Q}$. Drawing $q \sim \mu$ fixes that task's initial environment state s_0, its kernels (P_s, P_o), and its reward function r, so a distribution over tasks and a distribution over s_0 carry the same information in our setting; we write $q \sim \mu$ throughout. In the basic setting μ is uniform over the task collection under study, and $\mu_{\text{train}}, \mu_{\text{test}}$ denote its restrictions to $\mathcal{Q}_{\text{train}}, \mathcal{Q}_{\text{test}}$. Reward is terminal: the reward function $r : \mathcal{S} \to [0, 1]$ is read only at the terminal environment state, yielding the realized return $r := r(s_T)$ with no intermediate reward. This is the RLVR setting, where the environment returns a single verifiable outcome at termination.

A *trajectory* (τ) is a turn-level, complete-information representation of an agent's interaction and environment history in a single episode. For a fixed task q and policy π,

$$\tau_0 = (q, s_0), \quad \tau_t = (q, s_0, a_1, s_1, o_1, \ldots, a_t, s_t, o_t), \quad \tau = (q, s_0, \ldots, a_T, s_T, o_T, r), \tag{3}$$

where T is the terminal turn (i.e. horizon). As an aside, we reserve the word "rollout" for the sampling procedure that runs a policy in the environment and realizes a trajectory. Let $\mathbb{P}_q^\pi$ denote τ's probability distribution given a task q. Given a task distribution μ,

$$\mathbb{P}_\mu^\pi(\tau) := \mathbb{E}_{q \sim \mu}\left[\mathbb{P}_q^\pi(\tau)\right]. \tag{4}$$

During the process of rolling out τ, the partial trajectory τ_t at each turn $t < T$ is projected by the agent scaffold to the *context state* $h_t \in \mathcal{H}$ ("context"). We call such a mapping H and define:

$$h_0 = H(\tau_0) := (q), \quad h_t = H(\tau_t) := h_{t-1} + (a_t, o_t) = (q, a_1, o_1, \ldots, a_t, o_t). \tag{5}$$

In addition, given a completed trajectory τ with terminal length T, we also denote the retrospective construction of h_t by $H_t(\tau)$, which equals $H(\tau_t)$ when $t \leq T$, and $\mathrm{N/A} \notin \mathcal{H}$ when $t > T$.

Note that a context h_t's serialized form is the information state the LLM policy π conditions its action on at time t (Eq. 1), not the latent environment state s_t. Therefore, we use the unmodified "state" to alias context hereafter, referring to "state" and "context" interchangeably. Eq. 5 assumes the agent context extends linearly between turns without truncation or compaction, same in related works like Choudhury and Sodhi [2025] and Lauffer et al. [2025]. Appendix G shows a real context in our experiments and its correspondence to this subsection's notation.

2.2 Context Distribution and the SFT Objective

In the original DAgger paper [Ross et al., 2011], the trajectory stopping time was assumed to be a *constant* T, and they instantiated the state distribution (a.k.a. state occupancy measure in the

[2] Chat Markup Language (ChatML; [OpenAI, 2023]) defines a structured role-annotated LLM context formatting template (e.g. `system`, `user`, `assistant` roles), including serialization protocol. It is widely adopted by open-weight LLMs too.

[3] Also referred to as an "agent harness" in the AI agents literature. See for example [Anthropic, 2026].

RL theory literature) as a time-uniform distribution $d^\pi(s) = \frac{1}{T}\sum_{t=1}^{T} d_t^\pi(s)$, where they hid the μ-dependence and described d_t^π as "*the distribution of states at time t if the learner executed policy π from step 1 to $t-1$.*" We observe that using such a notation to describe episodes with varying lengths, such as in agentic LLM tasks, is under-specified: when T is random, how could we instantiate $\mathbb{E}[\frac{1}{T}\sum_t d_t^\pi]$? We try to formalize the inductive biases a practitioner can choose to resolve the "average over what" question.

Recall from (4) and (5) that $\mathbb{P}_\mu^\pi$ is the trajectory distribution marginalized over μ, and that given a finished trajectory τ, $H_t(\tau) \in \mathcal{H} \cup \{\mathrm{N/A}\}$ is the retrospectively constructed context. Define the time-t marginal context distribution $h_t \sim d_{\mu,t}^\pi$, which implicitly assumes that $T(\tau) \geq t$ for the τ that emits h_t, as a trajectory-average counting measure:

$$d_{\mu,t}^\pi(B) := \mathbb{E}_{\tau\sim\mathbb{P}_\mu^\pi}[\,\mathbf{1}\{H_{t-1}(\tau) \in B\}\,], \quad \text{for any set of contexts } B \subseteq \mathcal{H}. \tag{6}$$

We now introduce a general, any-time context distribution $d_\mu^{\pi;w}$, s.t. $h \sim d_\mu^{\pi;w}$ where h's turn count $t(h)$ is jointly sampled. As the POMDP analogue of the state distribution, it is parametrizable by μ, π and a turn-indexed context weighting function $w_t(\tau) \geq 0$ (w for short, when unambiguous):

$$d_\mu^{\pi;w}(B) = \frac{\mathbb{E}_{\tau\sim\mathbb{P}_\mu^\pi}\left[\sum_{t=1}^{T(\tau)}\left(w_t(\tau)\cdot\mathbf{1}\{H_{t-1}(\tau)\in B\}\right)\right]}{\mathbb{E}_{\tau\sim\mathbb{P}_\mu^\pi}\left[\sum_{t=1}^{T(\tau)} w_t(\tau)\right]}, \quad \text{for any set of contexts } B \subseteq \mathcal{H}. \tag{7}$$

This lets Eq. 7 distinguish two state averaging conventions that only coincide when $T(\tau)$ is constant with probability 1 for $\tau \sim \mathbb{P}_\mu^\pi$:

$$\begin{aligned} w_t(\tau) = 1 \quad &\longrightarrow \quad d_\mu^{\pi;\mathbf{1}}(B) := d_\mu^{\pi;\mathbf{pool}}(B) = \frac{\mathbb{E}_\tau\left[\sum_{t=1}^{T(\tau)}\mathbf{1}\{H_{t-1}(\tau)\in B\}\right]}{\mathbb{E}_\tau[T(\tau)]}, \\ w_t(\tau) = \frac{1}{T(\tau)} \quad &\longrightarrow \quad d_\mu^{\pi;\frac{\mathbf{1}}{\mathbf{T}}}(B) := d_\mu^{\pi;\mathbf{traj}}(B) = \mathbb{E}_\tau\left[\frac{1}{T(\tau)}\sum_{t=1}^{T(\tau)}\mathbf{1}\{H_{t-1}(\tau)\in B\}\right]. \end{aligned} \tag{8}$$

The first is pooled-turns randomization: every visited context, regardless of its trajectory, receives equal probability mass after normalization, so longer trajectories contribute more total mass. A fixed-horizon switch sampler with invalid-switch rejection, as in OEC-style random switching, is one implementation that induces this convention; Appendix B.2 gives the derivation. The second is trajectory-uniform randomization: roll out τ until termination, then sample a turn $t \sim \mathrm{Unif}\{1, T(\tau)\}$. In this case, every trajectory (episode) receives equal probability mass. The latter is the randomization induced by our uniform-prefix takeover rule used in §3.1 and §3.2.

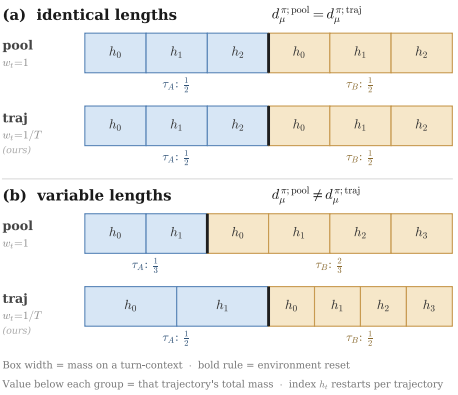

Figure 1: The two state-averaging conventions of Eq. 8, on two sampled trajectories of (a) identical and (b) variable length.

We now define the SFT objective independently of how the context distribution is induced. Let $\pi_e \in \Pi^\star$ be a frozen teacher policy and $\pi_\theta \in \Pi^\Theta$ be the student policy being fine-tuned. For any turn-level context distribution d over $\mathcal{H}$, define the next-turn population SFT objective as the student policy minimizing the forward KL divergence averaged on d:

$$\mathcal{L}_{\mathrm{SFT}}(\theta; d) := \mathbb{E}_{h\sim d}\left[D_{\mathrm{KL}}(\pi_e(\cdot \mid h) \,\|\, \pi_\theta(\cdot \mid h))\right]. \tag{9}$$

Since π_e is fixed, minimizing Eq. 9 is the same as minimizing the d-averaged cross entropy

$$\widetilde{\mathcal{L}}_{\mathrm{SFT}}(\theta; d) := \mathbb{E}_{h\sim d}\left(\mathbb{E}_{a^\star\sim\pi_e(\cdot|h)}\left[-\log\pi_\theta(a^\star \mid h)\right]\right) \tag{10}$$

up to an additive constant independent of θ. Here $a^\star \in \mathcal{A}$ is a complete teacher-generated assistant turn, matching our choice of the SFT objective to be next-turn-level: on an interaction context h,

the student is trained to imitate the teacher's next assistant turn's tokens. The single-teacher-turn assumption of Eq. 10 can be easily extended for the case of continuing multiple teacher turns by autoregressive factorization. Appendix B.1 formalizes this multi-turn extension.

d's parametrization distinguishes supervision regimes. Standard pure behavioral cloning SFT corresponds to a teacher-induced and pooled-turn context distribution $d = d_\mu^{\pi_e;\mathbf{pool}}$, i.e. $d = d_\mu^{\pi_e;w}$ with $w \equiv 1$. A more "on-policy" supervision scheme, we denote by $d = d_\mu^{\rho;w}$, would instead use a context-generating policy $\rho \neq \pi_e$; ρ can be a student π_θ, a teacher-student mixture, or a more complex composition. $w_t(\tau)$ offers another flexible knob to configure d: besides the two t-agnostic instances in Eq. 8, it may be beneficial for future work to instantiate $w_t(\tau)$ as a per-context-turn value function. Lastly, μ may be instantiated as the empirical distribution over a finite task collection.

2.3 The Budget Allocation Problem in Fine-Tuning Dataset Construction

Separate from the sequential decision process of an LLM's rollout, researchers face a different decision problem when post-training it for downstream applications: how to allocate a given fine-tuning resource budget to maximize final performance on the test set. In this paper, the teacher is an advanced LLM whose model size and inference cost are much larger than the student's, but it can be instantiated as a human expert too. We quantify the budget constraint in two separate axes:

1. C_{i}: Total teacher *inference cost*, before custom data filtering
2. C_{tr}: Per-epoch *training cost*, i.e. GPU hours needed on the (filtered) SFT traces

Both C_{i} and C_{tr} are measurable either in tokens or the number assistant turns generated by the teacher. For C_{tr}, the GPU hours needed by an SFT training run is roughly proportional to the number of accepted teacher tokens on an epoch-normalized basis. Whether the spending on C_{i} or C_{tr} dominates a research team's fine-tuning budget concern, we suppose, is often idiosyncratic. For example, a research group with a dedicated allocation on a large GPU cluster faces near-zero marginal training cost, so they would be far more cost-sensitive to external API inference calls (C_{i}) needed to generate teacher completions than to the size of the resulting training corpus (C_{tr}). So rather than assigning an ad-hoc weight, our experiments visualize budget scaling under each extreme point respectively.

Beyond inference and training compute, task diversity in $\mathcal{Q}_{\mathrm{train}}$, which we proxy by $|\mathcal{Q}_{\mathrm{train}}|$, also shapes a student's learning efficiency; recall from §2.1 that $(\mu, \mathrm{P_s}, \mathrm{P_o}, r)$ is often implemented as an empirical distribution over $\mathcal{Q}_{\mathrm{train}}$. To avoid confounding effect from this factor, we generally control $|\mathcal{Q}_{\mathrm{train}}|$ when comparing among OP and pure BC variants, so differences along the cost axes isolate the effect of the data-construction recipe alone. Any variation in $|\mathcal{Q}_{\mathrm{train}}|$ is explicitly indicated in a plot. We treat $|\mathcal{Q}_{\mathrm{train}}|$ as a hyperparameter rather than a third budget axis since it is often much more expensive and inelastic than C_{i} and C_{tr}: the pool of domain-specific training tasks with verifiable answers is typically determined by whatever open-source or proprietary datasets are available at the fine-tuning phase's onset, and meaningfully expanding its coverage requires highly costly human annotation effort that is qualitatively different from scaling teacher inference or GPU compute.

2.4 Design Space of Fine-Tuning Dataset Construction

We present a common recipe to connect implementation-level data curation knobs to the population SFT objective in Eq. 10. In code, these knobs are easily mutable objects compatible with LLM post-training libraries like `verl` or `LlamaFactory`: the policy checkpoint generating prefix contexts, the switch-time sampler, the prefix-context and continuation filters, the teacher-continuation horizon, etc. Connecting the formalism in §2.1-§2.3, choosing these objects instantiates the empirical context distribution corresponding to a population distribution d and to deciding how much teacher inference C_{i} and accepted training signal C_{tr} are spent to approximate it.

Let ρ be the prefix-context policy and π_e the teacher labeler. A general implementation recipe is

$$\mathcal{G} := (\rho, \pi_e, n, \Psi, \mathcal{T}_{\mathrm{switch}}, K, \Phi), \tag{11}$$

where n is the number of SFT context-continuation traces proposed per task, Ψ is a prefix-context filter applied before teacher continuation, $\mathcal{T}_{\mathrm{switch}}$ is a protocol to sample a candidate switch time t', K caps the number of teacher assistant turns in the continuation, and Φ is a post-continuation filter. The realized count of accepted traces N is therefore random and can be smaller than $n \cdot |\mathcal{Q}_{\mathrm{train}}|$. For

each trace proposal on task q, first the context-generating policy is rolled out to produce a trajectory $\tau \sim \mathbb{P}_q^\rho$ and either $t' \sim \mathcal{T}_{\text{switch}}(\tau)$ or $t' \sim \mathcal{T}_{\text{switch}}$ independently of τ.

If $t' \leq T(\tau)$ and $\Psi(\tau, t') = 1$, the proposal is explored further: the (partial) trajectory up to the t'-th turn induces the prefix context $h_{t'-1} = H(\tau_{t'-1})$. The teacher continues the interaction from $H(\tau_{t'-1})$ and its associated state $s_{t'-1}$ for $k' \leq K$ assistant turns before finish. We denote the aggregate trajectory including the continuation, which may be rejected by the filter Φ, by

$$\widetilde{\tau} = (q, s_0, a_1, s_1, o_1, \ldots, a_{t'-1}, s_{t'-1}, o_{t'-1}, a^\star_{t'}, s_{t'}, o_{t'}, \ldots, a^\star_{t'+k'-1}), \tag{12}$$

where starred actions are teacher-generated. Note that under the linear-context, no-format-change convention of Eq. 5, $H(\tau_{t'-1}) = H(\widetilde{\tau}_{t'-1})$. If the teacher and student scaffolds use different chat templates and tokenizers, the teacher will see $H(\tau_{t'-1})$ serialized in its own format, though the accepted continuation would be rendered in the student's format before SFT. Altogether, the post-filter SFT dataset is the collection of *context-continuation traces* with superscript i as index:

$$\mathcal{D}_{\mathcal{G}} := \{(H_{t'_i-1}(\tau^i),\ H(\widetilde{\tau}^i)[t'_i :]) : i = 1, \ldots, N\}, \tag{13}$$

where $[t'_i :]$ is an array slicing notation for the teacher-generated continuation segment. The SFT loss is computed only on teacher-generated assistant turns ($a^\star_\cdot$) in the continuation, excluding environment observations. Eq. 13 cleanly expresses the intended separation: ρ supplies the prefix context and π_e supplies the continuation. Algorithm 1 summarizes the overall procedure of constructing $\mathcal{D}_{\mathcal{G}}$.

The way $\mathcal{T}_{\text{switch}}$ samples t' implicitly defines the context weighting function w in Eq. 7 and 8. If τ is first completed and then $t' \sim \text{Unif}\{1, \ldots, T(\tau)\}$, the induced context distribution is trajectory-uniform, $d_\mu^{\rho;\text{traj}}$ and $w \equiv 1/T(\tau)$. If instead $t' \sim \text{Unif}\{1, \ldots, T_{max}\}$, i.e. a fixed support, and the pair (τ, t') is rejected when $t' > T(\tau)$, then the accepted contexts follow the pooled-turn distribution $d_\mu^{\rho;\text{pool}}$ with $w \equiv 1$; in other words, tasks and rollouts with shorter horizons naturally are less represented in accepted examples. Lauffer et al. [2025]'s implementation resembles $d_\mu^{\rho;\text{pool}}$: it samples a switch index before the rollout, lets the student act until either the switch is reached or the student terminates, and discards trajectories that terminate before the teacher takes over (§B.2).

When $\mathcal{T}_{\text{switch}} \equiv 1$, $(\rho, K, n) = (\pi_e, T_{max}, 1)$, and no filters, we call $\mathcal{D}_{\mathcal{G}}$ "pure BC" and alias by $\mathcal{D}_{\text{BC}}$. We define the policy initialized from a base student checkpoint π_0 then fine-tuned on $\mathcal{D}_{\mathcal{G}}$ as

$$\pi_1 := \text{SFT}(\pi_0, \mathcal{D}_{\text{BC}}). \tag{14}$$

We call $\mathcal{D}_{\mathcal{G}}$ "on-policy" and write it as $\mathcal{D}_{\text{OP}}$ when $\rho \in \Pi_\Theta$ and $\rho \neq \pi_e$, for example with $\rho = \pi_1$. In the BC+OP recipes, the final supervised run is reinitialized from π_0 and trained on $\mathcal{D}_{\text{BC}} \cup \mathcal{D}_{\text{OP}}$; the context-generating policy that generated $\mathcal{D}_{\text{OP}}$ is fixed during this data-generation round.

Algorithm 1 Constructing supervised fine-tuning data from a recipe $\mathcal{G}$.

Require: Training tasks $\mathcal{Q}_{\text{train}}$ and recipe $\mathcal{G} = (\rho, \pi_e, n, \Psi, \mathcal{T}_{\text{switch}}, K, \Phi)$.

```
D_G ← ∅,   (C_i, C_tr) ← (0, 0)
for q ∈ Q_train do
    for ℓ = 1, ..., n do
        Sample one proposal (τ, t'): τ ~ P_q^ρ, t' ~ T_switch        ▷ optionally, t' ~ T_switch(τ).
        if t' > T(τ) or Ψ(τ, t') = 0 then
            continue                                ▷ invalid switch or prefix-context filter triggered.
        end if
        τ̃ ← π_e continues from prefix context H(τ_{t'-1}) and env. state s_{t'-1} for ≤ K turns.
        Δ ← count of teacher tokens outputted in the τ_{t'-1}-to-τ̃ continuation.
        C_i ← C_i + Δ
        if Φ(τ̃) = 1 then
            D_G ← D_G ∪ {(H(τ_{t'-1}), H(τ̃)[t' :])}
            C_tr ← C_tr + Δ
        end if
    end for
end for
return SFT dataset D_G, teacher inference cost C_i, accepted training cost C_tr.
```

Table 1: $\mathcal{G}$ types involved in our experiments (§3). The "OP" rows specify how the data augmentation $\mathcal{D}_{\text{OP}}$ is generated; the final aggregate SFT dataset is $\mathcal{D}_{\text{SFT}} = \mathcal{D}_{\text{BC}} \cup \mathcal{D}_{\text{OP}}$. π_1 defined in (14).

Type	ρ	n proposal attempts	Ψ prefix-context filter	$\mathcal{T}_{\text{switch}}$	K	Φ continuation filter
PURE-BC	π_e teacher	$1, 2, \ldots$ full traj. per task	No filter	$t' = 1$ from scratch	$T_{\max}$ to finish	No filter
OP-SHORT	π_1	constant†	No filter	$\text{Unif}\{1, \ldots, T(\tau)\}$ trajectory-uniform	$1, 2, \ldots$ capped continuation	No filter
OP-SUCCESS	π_1	constant	No filter	$\text{Unif}\{1, \ldots, T(\tau)\}$ trajectory-uniform	$T_{\max}$ to finish	$\mathbb{1}\{r(\widetilde{\tau}) = 1\}$ continuation succeeds
OP-CRITICAL	π_1	constant	$\mathbb{1}\{r(\tau) = 0\}$ when ρ fails	$\text{Unif}\{1, \ldots, T(\tau)\}$ trajectory-uniform	$T_{\max}$ to finish	$\mathbb{1}\{r(\widetilde{\tau}) = 1\}$ continuation succeeds
OEC [Lauffer et al., 2025]	π_1	constant	repetition	$t' \sim \nu^{\ddagger} \wedge t' \leq T(\tau)$ reject invalid time	$T_{\max}$ to finish	$\mathbb{1}\{r(\widetilde{\tau}) = 1\}$ continuation succeeds
LEAP [Choudhury and Sodhi, 2025]	π_1	constant	$\mathbb{1}\{r(\tau) = 0\}$ when ρ fails	$t' \sim \pi_e(\tau)$ Teacher decides	1 next turn only	No filter

† In §3.3 (Terminal Bench), we also try a task-specific n that correlates with task difficulty.
‡ ν denotes a fixed-support switch distribution, e.g. $\text{Unif}\{1, 2, \ldots, T_{\max}\}$.

Table 1 shows four types of $\mathcal{G} = (\rho, \pi_e, n, \Psi, \mathcal{T}_{\text{switch}}, K, \Phi)$ our experiments study. PURE-BC uses n as a data scaling lever; intuitively, more teacher traces per task offer a lower variance estimation of the KL term in Eq. 9. OP-SHORT accepts every sampled prefix context and continuation, but uses a small K. OP-SUCCESS sets $K = T_{max}$ and accepts only teacher continuations with verifiably correct answers; it is similar to OEC's rejection sampling setup, and the success-only filter makes C_{i} larger per accepted trace as a teacher may not always be accepted. OP-CRITICAL adds the prefix-context filter $\Psi(\tau, t') = \mathbb{1}\{r(\tau) = 0\}$ in our experiments, so teacher budget is spent only on tasks the rollout policy ρ fails to solve on its own.

3 Experiments

We test the cost-and-performance framing of §2 on three distinct agentic environments. On HotpotQA [Yang et al., 2018] (search-based question answering) we compare all four recipe types in Table 1, including varying K for OP-SHORT and n for PURE-BC. On ALFWorld [Shridhar et al., 2021] (embodied household planning in text environments) we compare OP-SHORT at $K = 3$ to pure BC at four training-instance budgets, with in-distribution and out-of-distribution evaluation, an ablation on the ratio of on-policy to expert-prefix data. On Terminal-Bench-Dev [OpenThoughts-Agent Team, 2025] (solving code problems in an interactive shell) we sweep pure BC across five fractions of the BC corpus and run OP-SHORT at two continuation budgets across three prefix policies, and compare with the OpenThoughts-Agent baseline. For the three OP types in all our experiments, ρ is π_1 from Eq. 14, and the final augmented SFT set is $\mathcal{D}_{\text{BC}} \cup \mathcal{D}_{\text{OP}}$ (the same $\mathcal{D}_{\text{BC}}$ used to obtain π_1).

3.1 Search Agent: HotpotQA

HotpotQA [Yang et al., 2018] was originally designed to test a model's ability to answer a question by synthesizing evidence from *preset* Wikipedia snippets. We use the Search-R1 + VeRL adaptation [Jin et al., 2025, Sheng et al., 2025], which *removes* the preset snippets and turns each question (i.e. task) into a sequential interaction with a searchable database: the agent interleaves thinking with search queries to a 2018 snapshot of the English Wikipedia, receives retrieved passages as observations, and terminates by submitting an answer in no more than $T_{\max} = 5$ turns. Since this database is static, the environment state s_t is time-invariant; the rollout's meaningful evolution is the LLM-readable context h_t as tool calls and observations accumulate. The student is initialized from `Qwen2.5-3B-Instruct` (π_0), the teacher is `Qwen3-235B-A22B-Instruct` (π_e). pass@1 is normalized exact match against the official references, following Search-R1's GitHub implementation.

Figure 2 fixes $|\mathcal{Q}_{\text{train}}| = 1000$ medium-level HotpotQA questions and varies only the SFT trace construction recipe. PURE-BC scales by increasing n, while OP-SHORT scales by increasing the teacher turns limit K. For all OP variants, $\rho = \pi_1$, corresponding to Figure 2's leftmost purple PURE-BC point. We roll out ρ for $n = 3$ times per question, sample $\mathcal{T}_{\text{switch}}$ uniformly over each completed trajectory, and train on π_e's continuation from the resulting student-induced context. This

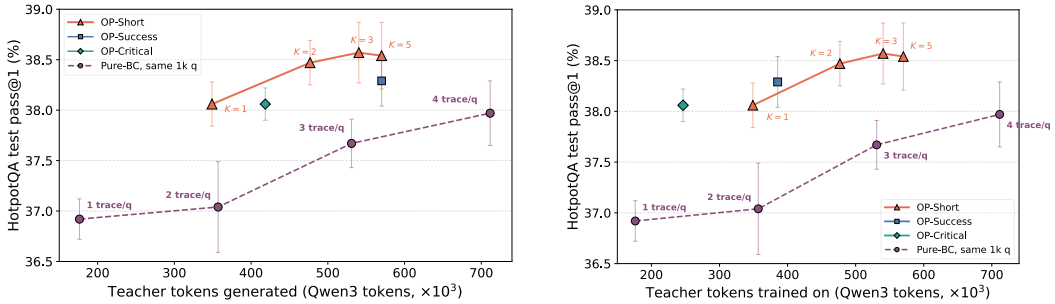

Figure 2: HotpotQA SFT test accuracies for the recipe instances in Table 1. All variants use the same 1000 medium-level training questions $\mathcal{Q}_{\text{train}}$; PURE-BC scales by sampling more i.i.d. teacher traces per question ($n\uparrow$). Checkpoints are selected on a held-out dev set and then evaluated on the HotpotQA official test set (7.4k hard-level questions). **Left**: pass@1 versus teacher-inference cost C_{i}, measured in Qwen3 teacher output tokens generated during data construction. **Right**: the same pass@1 values versus retained training cost C_{tr}, measured in teacher tokens trained on per epoch. Error bars are one standard error estimated from 8 i.i.d. pass@1 evaluations on the full test set.

directly instantiates the trajectory-uniform distribution $d_{\mu}^{\pi_1;\mathbf{traj}}$ from Eq. 8. All variants use the same retriever, system prompt, and a temperature of 1 for data generation and evaluation.

To avoid selecting noisy test-set wins, the learning rate and final SFT checkpoint are chosen using held-out HotpotQA dev runs before final test evaluation; Figure 6 reports the training hyperparameters and checkpoint selection process.

On-policy supervisions improve SFT sample efficiency. On both cost axes in Figure 2, teacher completions from π_1-induced prefixes make better use of the fixed question set than additional PURE-BC demonstrations. PURE-BC improves as more complete teacher traces are added but enters a region of diminishing returns, whereas the OP recipes achieve higher pass@1 at comparable supervision budgets, suggesting that once naive scaling of expert demonstrations saturates, redirecting teacher supervision toward student-induced contexts offers a higher marginal return per token spent.

Scaling teacher continuation length has non-monotone returns. The OP-SHORT K-sweep shows that longer teacher continuations are not automatically better, even before cost normalization. Increasing K helps up to $K = 3$, but $K = 5$ is worse despite spending more teacher tokens. This non-monotonicity likely reflects two factors: the main learning is the bounded repair at the student-induced state, beyond which extra turns produce completions unlikely for a student; and in HotpotQA, trajectories that exhaust all $T_{\max}$ might correlate with more ill-posed questions, biasing long continuations to be less instructional.

Filtering trades teacher inference for fewer training tokens. The filtered recipes, OP-SUCCESS and OP-CRITICAL, can look attractive on retained training cost C_{tr} because rejected continuations are not trained on. They are less favorable on teacher-inference cost C_{i}, where rejected teacher tokens are still charged. The gap between the two panels therefore clarifies the tradeoff: filtering is most useful when the SFT training compute is scarce, not when teacher inference is the binding resource.

3.2 Embodied Agent: ALFWorld

We use `Qwen3-0.6B` as the student and `Qwen3-8B` (after RL) as the teacher. Following the official splits, we train on `train` ($\sim$3500 tasks) and evaluate on `valid_seen`. We compare the pure BC against OP-SHORT with $K \in \{1, 5, \infty\}$, OP-SUCCESS, and OP-CRITICAL, while sweeping the unique training tasks count $|\mathcal{Q}_{\text{train}}| \in \{64, 128, 256\}$. We also fine-tune the student using a subsequent RL stage, initialized from the $|\mathcal{Q}_{\text{train}}|$=128 pure BC and OP-SHORT checkpoints and uses the remaining `train` task. SFT uses learning rate 5×10^{-5}, batch size 16, and 8 epochs; GRPO uses learning rate 1×10^{-6}, batch size 16, group size 8, and 4 epochs. The SFT stage result is shown in Figure 3. OP-SHORT is consistently above the Pure BC curve at similar teacher-supervision budgets, showing that on-policy data augmentation enjoys better sample efficiency.

Fewer teacher steps could be more cost-effective and sample-efficient than full teacher completions. We conduct ablation experiments on the number of teacher continuation steps. Figure 3

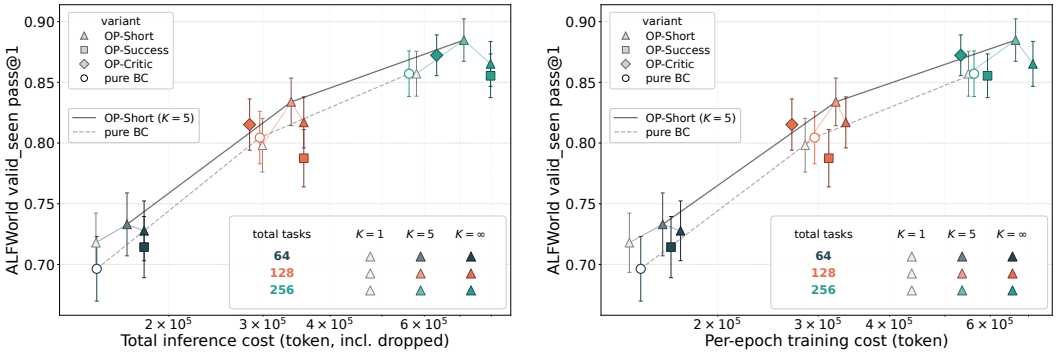

Figure 3: ALFWorld test accuracy on `valid_seen` pass@1. **Left:** The left panel counts total teacher labeling cost, including discarded. **Right:** the right panel counts teacher tokens retained for SFT. Note: for the case of 64 tasks 64, the traces are all filtered out under OP-CRITICAL, and thus there is no corresponding point.

shows that longer teacher continuations do not monotonically improve ALFWorld performance. For all of task budgets $|\mathcal{Q}_{\text{train}}| \in \{64, 128, 256\}$, the best short-continuation runs occur at $K = 5$ and outperform the corresponding $K = \infty$ runs despite spending fewer teacher tokens. Filtering shows the same cost tradeoff: once the discarded teacher continuations are counted as labeling cost (See Figure 3 left), the filtered variants do not necessarily deliver a consistent pass@1 gain matching their extra teacher inference tokens spend. Therefore, after a few corrective teacher actions, additional budget is often better spent on covering more student states than on extending more continuation.

Ablation on optimal mixing ratio. Given a fixed teacher supervision budget, we also study the optimal mixture of the BC data (induce the on-policy student state distribution as well as part of the final training data) and on-policy data. Figure 4 sweeps the ratio between expert-prefix data and on-policy-prefix data under three total teacher-supervision budgets. Across three fully swept budgets 147K, 296K, and 562K, allocating a nonzero fraction to on-policy prefixes (i.e., OP-SHORT) is better than pure expert-prefix supervision (Pure BC), but using only on-policy data is also suboptimal. The intuition is that it is better for the student to start approaching the desired state distribution before collecting on-policy data. The best observed on-policy ratios are about 90%, 50%, and 90% for the three token budgets, respectively, indicating that the optimal mixture can depend on the total supervision budget. This pattern suggests that on-policy data should be treated as targeted augmentation rather than a replacement for expert demonstrations: expert-prefix data provides stable coverage of canonical traces, while on-policy-prefix data corrects the student on states induced by its own errors.

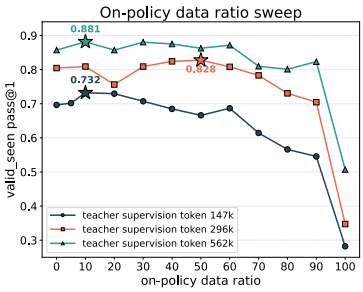

Figure 4: ALFWorld optimal on-policy data ratio r. ratio r is the percentage of tasks used for on-policy data generation. $r = 0$ degenerates to BC, and $r = 1$ allocates all the tasks to use student prefix.

3.3 Coding Agent: Terminal-Bench-Dev

We use `Qwen3-8B` as the student and `GLM-4.6` as the teacher, following the OpenThoughts-Agent project [OpenThoughts-Agent Team, 2025]. The BC corpus $\mathcal{D}_{\text{BC}}$ is OpenThoughts-Agent-v1-SFT [OpenThoughts-Agent Team, 2025], which contains 15,209 teacher traces on `nl2bash` [Lin et al., 2018] and `inferredbugs` [Jin et al., 2023]. We evaluate our SFT data curation strategies on Terminal-Bench-Dev (70 questions total) for direct comparability to the OT-Agent baselines.

We shuffle the trajectories in $\mathcal{D}_{\text{BC}}$, and then our pure BC baselines are trained on the first $\{10\%, 25\%, 50\%, 75\%, 100\%\}$ portion of the shuffled $\mathcal{D}_{\text{BC}}$ (1,520 to 15,209 traces) such that each larger SFT train set fully contains the preceding ones. OP-SHORT is run at two continuation budgets, $K = 3$ and $K = \infty$ with no outcome filter. The mean score we report in Figure 5 is the average of

the official OT-Agent codebase's verifier scores over $70 \times 3 = 210$ trials, i.e. the 70-question TB-dev set was attempted three times independently to capture response variance.

OP-SHORT matches the post-RL OpenThoughts-Agent baseline at one-tenth the BC budget. At a matched fraction of the corpus size, OP-SHORT improves substantially over pure BC: finetuning with 10% of $\mathcal{D}_{\mathrm{BC}}$, pure BC reaches 11.4% TB-dev, while OP-SHORT at $K = 3$ reaches 16.0% and 19.0%, surpassing the OpenThoughts-Agent post-RL baseline of 17.3% that uses the full $15{,}209$ BC traces followed by 720 GRPO tasks. On this benchmark on-policy data construction, applied at the SFT stage and without any RL, recovers the gain that the OpenThoughts-Agent recipe extracts from a combination of full-corpus SFT and subsequent RL.

Hyperparameters and more ablation experiments, including pure On-Policy cases and the scenario where tasks in BC and On-Policy data are different, are shown in Table 4 and 5 in Appendix F. For OP-SHORT we additionally vary the prefix policy across three checkpoints, the untrained student π_0 and the BC checkpoints trained on 10% and 100% of $\mathcal{D}_{\mathrm{BC}}$, holding the questions and the number of traces in $\mathcal{D}_{\mathrm{OP}}$ identical across prefix choices. Our observations still hold under different circumstances.

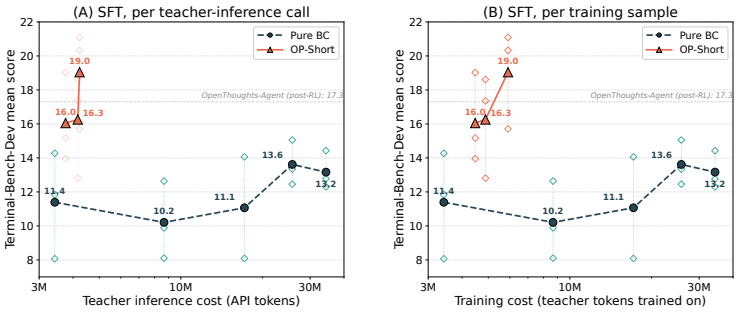

Figure 5: Terminal-Bench-Dev results at the same prefix model. (A, B) TB-dev mean score across three attempts, plotted against (A) teacher-inference cost (in teacher model tokens) and (B) training cost (in teacher model tokens). OP-Short points from bottom to top corresponds to: 329 BC traces + 329 OP traces, $K = 3$ (329 questions); 329 BC traces + 329 OP traces, $K = \infty$ (329 questions); 720 BC traces + 720 OP traces, $K = 3$ (720 questions). Diamond markers indicate the per-run mean score for each SFT checkpoint (three independent TB-dev evals).

4 Discussion

The main practical lesson is not that "on-policy" data should replace pure behavioral cloning. Rather, it is useful as targeted augmentation once BC has produced a student whose mistakes define informative contexts. In this view, the key knob is how much teacher-descendant trajectory to buy at each learner-induced context state. Our results suggest that bounded continuations often provide enough local recovery signal, while spending the same budget on longer completions, additional full demonstrations, or success filtering can reduce coverage of distinct student states. The continuation budget K should therefore be treated as a task-dependent breadth–depth parameter, not a universal constant. Filtering should likewise be chosen according to the binding cost axis: it can reduce retained SFT tokens, but rejected continuations still consume teacher inference.

Limitations. We study one-round augmentation on three verifiable agentic benchmarks, with the most complete variant comparison on HotpotQA. The conclusions may change for long-horizon tasks where recovery requires extended planning, for settings without automatic reward verification, or under iterative data collection. We also do not provide a theory predicting the optimal K or the observed non-monotone returns to continuation length.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.13649.

Anthropic. Demystifying evals for ai agents. https://web.archive.org/web/20260406090051/https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents, January 2026. Archived by the Internet Archive Wayback Machine on 2026-04-06. Original article published 2026-01-09.

Massimo Caccia, Megh Thakkar, Léo Boisvert, Thibault Le Sellier de Chezelles, Alexandre Piché, Nicolas Chapados, Alexandre Drouin, Maxime Gasse, and Alexandre Lacoste. Fine-tuning web agents: It works, but it's trickier than you think. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024. URL https://openreview.net/forum?id=SkwtxEkst2.

Yihan Chen, Benfeng Xu, Xiaorui Wang, Yongdong Zhang, and Zhendong Mao. Training LLM-based agents with synthetic self-reflected trajectories and partial masking. *arXiv preprint arXiv:2505.20023*, 2025a. URL https://arxiv.org/abs/2505.20023.

Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. ATLaS: Agent tuning via learning critical steps. *arXiv preprint arXiv:2503.02197*, 2025b. URL https://arxiv.org/abs/2503.02197.

Sanjiban Choudhury and Paloma Sodhi. Better than your teacher: LLM agents that learn from privileged AI feedback. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=st7XqFgbAH.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: On-policy distillation of large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.08543.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025.

Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms. In *2023 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)*. ACM, 2023.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

Li-Cheng Lan, Andrew Bai, Minhao Cheng, Ruochen Wang, Cho-Jui Hsieh, and Tianyi Zhou. Exploring expert failures improves LLM agent tuning. *arXiv preprint arXiv:2504.13145*, 2025. URL https://arxiv.org/abs/2504.13145.

Niklas Lauffer, Xiang Deng, Srivatsa Kundurthy, Brad Kenstler, and Jeff Da. Imitation learning for multi-turn LM agents via on-policy expert corrections. *arXiv preprint arXiv:2512.14895*, 2025. URL https://arxiv.org/abs/2512.14895. To appear in the Proceedings of the 43rd International Conference on Machine Learning (ICML 2026).

Changhao Li, Rushi Qiang, Jiawei Huang, Chenxiao Gao, Chao Zhang, Niao He, and Bo Dai. Revisiting dagger in the era of llm-agents. *arXiv preprint arXiv:2605.12913*, 2026.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. *CoRR*, abs/1802.08979, 2018. URL http://arxiv.org/abs/1802.08979.

Yuanjie Lyu, Chengyu Wang, Jun Huang, and Tong Xu. From correction to mastery: Reinforced distillation of large language model agents. *arXiv preprint arXiv:2509.14257*, 2025. URL https://arxiv.org/abs/2509.14257.

OpenAI. Introducing apis for gpt-3.5 turbo and whisper. https://openai.com/index/introducing-chatgpt-and-whisper-apis/, 2023. Accessed: 2026-06-16.

OpenThoughts-Agent Team. OpenThoughts-Agent. https://www.open-thoughts.ai/blog/agent, December 2025.

Chongli Qin and Jost Tobias Springenberg. Supervised fine tuning on curated data is reinforcement learning (and can be improved). *arXiv preprint arXiv:2507.12856*, 2025. URL https://arxiv.org/abs/2507.12856.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010. URL https://proceedings.mlr.press/v9/ross10a.html.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, 2011. URL https://proceedings.mlr.press/v15/ross11a.html.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03768.

Richard D Smallwood and Edward J Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. STeCa: Step-level trajectory calibration for LLM agent learning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11597–11614, 2025. doi: 10.18653/v1/2025.findings-acl.604. URL https://aclanthology.org/2025.findings-acl.604/.

Jing Xiong, Hui Shen, Shansan Gong, Yuxin Cheng, Jianghan Shen, Chaofan Tao, Haochen Tan, Haoli Bai, Lifeng Shang, and Ngai Wong. OVD: On-policy verbal distillation. *arXiv preprint arXiv:2601.21968*, 2026. URL https://arxiv.org/abs/2601.21968.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Tianzhu Ye, Li Dong, Zewen Chi, Xun Wu, Shaohan Huang, and Furu Wei. Black-box on-policy distillation of large language models. *arXiv preprint arXiv:2511.10643*, 2025. URL https://arxiv.org/abs/2511.10643.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026. URL https://arxiv.org/abs/2602.12275.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-R: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025. URL https://arxiv.org/abs/2501.11425.

Anhao Zhao, Haoran Xin, Yingqi Fan, Junlong Tong, Wenjie Li, and Xiaoyu Shen. Decoupling KL and trajectories: A unified perspective for SFT, DAgger, offline RL, and OPD in LLM distillation. *arXiv preprint arXiv:2605.16826*, 2026a.

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. *arXiv preprint arXiv:2601.18734*, 2026b. URL https://arxiv.org/abs/2601.18734.

A Related Work

DAgger and on-policy SFT. Our work builds on interactive imitation learning, especially DAgger [Ross and Bagnell, 2010, Ross et al., 2011, Ross and Bagnell, 2014, Kelly et al., 2019], which reduces compounding error by collecting expert labels at learner-induced states. Caccia et al. [2024] show that, for LLM agents, better prediction on expert trajectories need not translate to better downstream performance once the student acts on its own induced contexts, and mention DAgger as a potential remedy. Lauffer et al. [2025] are the closest prior work to our intervention-style setup: they roll in with the student to a random switch step, let the expert continue to termination, mask the student prefix during training, and filter by verifiable success and student repetition. This motivates our filtered baselines, whereas our main OP-SHORT method keeps every sampled continuation but bounds it to a few teacher turns. Other variants use privileged train-time teacher state [Choudhury and Sodhi, 2025], teacher-identified first-error corrections [Lyu et al., 2025], self-reflected teacher-guided corrections [Chen et al., 2025a], or turn-level teacher-student mixing for long-horizon software agents [Li et al., 2026]. Related methods also address trajectory-level error accumulation through step-level comparison [Wang et al., 2025], teacher-free first-error revision [Yuan et al., 2025], or off-policy selection of critical trajectory segments [Chen et al., 2025b, Lan et al., 2025].

On-policy distillation. A related line studies on-policy distillation (OPD) for autoregressive language models, training on student-sampled sequences while matching teacher behavior via generalized knowledge distillation [Agarwal et al., 2024], reverse-KL objectives [Gu et al., 2024], context internalization [Ye et al., 2026], privileged self-distillation [Zhao et al., 2026b], verbal-score supervision [Xiong et al., 2026], or black-box discriminator-based distillation [Ye et al., 2025]. Concurrently, Zhao et al. [2026a] decouple prefix source from KL direction, identifying student-prefix forward-KL as DAgger-style on-policy SFT and student-prefix reverse-KL as OPD. Our setting focuses on explicit teacher completions at executable, learner-visited agent states; this is natural for multi-turn tool-use, supports black-box teachers, and lets us measure the budget in generated and retained teacher tokens rather than in token-level distribution queries.

Rejection-sampling SFT and the SFT-as-RL connection. A complementary line analyzes outcome-filtered SFT in its own right. Qin and Springenberg [2025] show that SFT on success-filtered trajectories optimizes a lower bound on the sparse-reward RL objective and propose an importance-weighted variant that tightens this bound. This perspective varies which generated trajectories are retained; we additionally vary the state distribution by querying the teacher at learner-induced prefix states. Accordingly, we treat filtered SFT as one data-construction choice within a broader budget-allocation problem, rather than only as a stand-alone alternative to RL.

Positioning relative to prior work. Recent agent post-training recipes often bundle several choices: prefix policy, switch-time sampling, continuation length, masking, filtering, and sometimes a subsequent RL stage. We instead make these choices explicit and compare them under matched supervision resources, separating teacher-inference cost from retained SFT training cost. This isolates the question central to our experiments: once a reasonable behavioral-cloning student is available, whether additional teacher budget is better spent on more complete expert demonstrations, longer or filtered teacher continuations, or short unfiltered corrections spread over more learner-induced contexts.

B Mathematical Details

B.1 Multi-turn teacher continuations and the SFT objective

§2.2 defines the population SFT objective at the single-next-turn level. We now spell out how the same objective can be revised to model SFT on multi-turn continuations for §2.4 recipes with $K > 1$.

One notational distinction is useful before defining the multi-turn objective. In the single-next-turn objective of Eq. 10, a sample from d only needs to specify an LLM-readable context h, since the teacher action is used as a supervised label and need not be executed. For a multi-turn teacher continuation, however, the teacher's first action must be executed to produce the next observation, so the sampled object must also contain a replayable environment state. We therefore let $\bar{d}$ denote a distribution over *executable partial trajectories*, which include replicable environment snapshots,

whose context pushforward recovers the main-text context distribution d:

$$H_{\#}\bar{d} = d, \qquad \tau_{t-1} \sim \bar{d}, \qquad h_{t-1} = H(\tau_{t-1}). \tag{15}$$

In our digital agent environments, such as HotpotQA, ALFWorld, Terminal-Bench, and SWE-Bench-style settings, this replayable prefix is typically stored or reconstructed losslessly from the task seed, scaffold state, and action history; this is much easier to re-simulate than in physical environments, where replaying a trajectory may introduce non-trivial sensor and environment variation.

Consider a partial trajectory τ_{t-1} with LLM-readable context $h_{t-1} = H(\tau_{t-1})$. A teacher continuation from this context consists of at most K subsequent teacher turns and environment observations,

$$y_{\geq t} = (a^{\star}_{t}, o_t, \ldots, a^{\star}_{t+L-1}, o_{t+L-1}), \qquad L \leq K, \tag{16}$$

where L is a random variable that is completely determined by $y_{\geq t}$'s realization and may be much smaller than K if the episode terminates early. We write

$$y_{\geq t} \sim C_e^K(\cdot \mid \tau_{t-1}) \tag{17}$$

for the distribution over such continuations obtained by repeatedly sampling teacher actions and stepping the environment:

$$a^{\star}_{t+j} \sim \pi_e(\cdot \mid h_{t+j-1}), \qquad s_{t+j} \sim \mathrm{P}_s(\cdot \mid s_{t+j-1}, a^{\star}_{t+j}), \qquad o_{t+j} \sim \mathrm{P}_o(\cdot \mid s_{t+j}), \tag{18}$$

for $j = 0, \ldots, L-1$, and the context extended as

$$h_{t+j} = H(\tau_{t+j}) = h_{t+j-1} + (a^{\star}_{t+j}, o_{t+j}). \tag{19}$$

Thus C_e^K is the closed-loop continuation operator induced by the teacher policy, the environment dynamics, the observation kernel, and the stopping rule. A completion filter Φ re-weights this continuation law by rejecting some continuations. We denote the accepted traces' law by $C_e^{K,\Phi}$.

From the student policy's perspective, the likelihood of a continuation $y_{\geq t}$ factors sequentially as

$$p_\theta(y_{\geq t} \mid \tau_{t-1}) = \prod_{j=0}^{L-1} \pi_\theta(a^{\star}_{t+j} \mid h_{t+j-1}) \cdot \mathrm{P}_s(s_{t+j} \mid s_{t+j-1}, a^{\star}_{t+j}) \cdot \mathrm{P}_o(o_{t+j} \mid s_{t+j}). \tag{20}$$

The environment transition and observation terms do not depend on θ. Therefore, up to an additive constant independent of θ, the negative log-likelihood of an accepted continuation is

$$\ell_{\mathrm{sft}}(\theta; \tau_{t-1}, y_{\geq t}) = \sum_{j=0}^{L-1} -\log \pi_\theta(a^{\star}_{t+j} \mid h_{t+j-1}). \tag{21}$$

Equivalently, the multi-turn population SFT objective induced by a replayable prefix distribution $\bar{d}$ and an accepted teacher-continuation law is

$$\widetilde{\mathcal{L}}_{\mathrm{sft}}^{K,\Phi}(\theta; \bar{d}) = \mathbb{E}_{\tau_{t-1} \sim \bar{d}} \left[\mathbb{E}_{y_{\geq t} \sim C_e^{K,\Phi}(\cdot \mid \tau_{t-1})} \left[\sum_{j=0}^{L(y_{\geq t})-1} -\log \pi_\theta(a^{\star}_{t+j} \mid h_{t+j-1}) \right] \right]. \tag{22}$$

This expression should be read at the turn level: each $a^{\star}_{t+j}$ is a complete assistant turn, whose cross entropy is implemented as a sum over its per-token log probabilities. The summation over j does not assume that the teacher turns within $y_{\geq t}$ are independent. Rather, it is the sequential factorization of the likelihood under teacher forcing, where each later context is obtained by appending the earlier teacher action and environment observation.

For $K = 1$, only the pushforward context distribution $d = H_{\#}\bar{d}$ matters and this reduces to the next-turn objective in Eq. 10. For $K > 1$, the first loss term supervises the student at the sampled context h_{t-1}, while later terms supervise the student at teacher-descendant contexts induced by the same continuation. Thus increasing K trades broader coverage of sampled contexts for deeper teacher recovery from each sampled context.

B.2 $\mathrm{Unif}\{1, \ldots, T_{max}\}$ switching with invalid-switch rejection induces pooled-turn sampling

This appendix justifies the pooled-turn statement for the proposal-budget convention used in Algorithm 1. Consider one switch proposal. Sample a task $q \sim \mu$, roll out the context-generating policy to obtain $\tau \sim \mathbb{P}_q^\rho$ with stopping time $T(\tau) \leq T_{\max}$, and independently sample $U \sim \mathrm{Unif}\{1, \ldots, T_{\max}\}$. Accept the proposal iff $U \leq T(\tau)$; otherwise no prefix context exists at that turn and the proposal is discarded. For any measurable set of contexts $B \subseteq \mathcal{H}$,

$$\mathbb{P}\{H_{U-1}(\tau) \in B \mid U \leq T(\tau)\} = \frac{\mathbb{E}_{q\sim\mu,\, \tau\sim\mathbb{P}_q^\rho}\left[\sum_{t=1}^{T(\tau)} \frac{1}{T_{\max}} \mathbf{1}\{H_{t-1}(\tau) \in B\}\right]}{\mathbb{E}_{q\sim\mu,\, \tau\sim\mathbb{P}_q^\rho}\left[\sum_{t=1}^{T(\tau)} \frac{1}{T_{\max}}\right]} \tag{23}$$

$$= \frac{\mathbb{E}_{\tau\sim\mathbb{P}_\mu^\rho}\left[\sum_{t=1}^{T(\tau)} \mathbf{1}\{H_{t-1}(\tau) \in B\}\right]}{\mathbb{E}_{\tau\sim\mathbb{P}_\mu^\rho}[T(\tau)]} = d_\mu^{\rho;\mathrm{pool}}(B). \tag{24}$$

Thus sampling a global-horizon switch and rejecting switches that occur after the realized trajectory terminates induces the same accepted prefix-context distribution as the pooled-turn convention of Eq. 8. With an empirical task set and n proposal attempts per task, the same calculation applies with μ equal to the uniform empirical distribution over $\mathcal{Q}_{\mathrm{train}}$. Tasks whose realized rollouts are shorter simply produce fewer accepted proposals in expectation, since $\mathbb{P}(U \leq T(\tau) \mid \tau) = T(\tau)/T_{\max}$.

This equivalence isolates invalid-switch rejection. The prefix-context filter Ψ and post-continuation filter Φ in the recipe intentionally reweight the accepted distribution. For example, let $p_\Phi(\tau, t)$ be the probability that a teacher continuation from prefix context $H_{t-1}(\tau)$ passes the post-continuation filter. After applying Ψ and Φ, the accepted prefix-context distribution becomes

$$\frac{\mathbb{E}_{\tau\sim\mathbb{P}_\mu^\rho}\left[\sum_{t=1}^{T(\tau)} \Psi(\tau, t)\, p_\Phi(\tau, t)\, \mathbf{1}\{H_{t-1}(\tau) \in B\}\right]}{\mathbb{E}_{\tau\sim\mathbb{P}_\mu^\rho}\left[\sum_{t=1}^{T(\tau)} \Psi(\tau, t)\, p_\Phi(\tau, t)\right]}. \tag{25}$$

Invalid-switch rejection alone gives $d_\mu^{\rho;\mathrm{pool}}$; additional filters produce the corresponding filtered pooled-turn distribution.

As an aside, the above proposal-budget convention is different from topping up every task to the same number of accepted switches. If one resamples until each task has n accepted prefix contexts, invalid-switch rejection instead induces the equal-task average of per-task pooled distributions,

$$\frac{1}{|\mathcal{Q}_{\mathrm{train}}|} \sum_{q\in\mathcal{Q}_{\mathrm{train}}} \frac{\mathbb{E}_{\tau\sim\mathbb{P}_q^\rho}\left[\sum_{t=1}^{T(\tau)} \mathbf{1}\{H_{t-1}(\tau) \in B\}\right]}{\mathbb{E}_{\tau\sim\mathbb{P}_q^\rho}[T(\tau)]}, \tag{26}$$

which coincides with the global pooled-turn law only when expected horizons are constant across tasks or when the task distribution is reweighted accordingly. More generally, if $U \sim \nu$ for a fixed distribution on $\{1, \ldots, T_{\max}\}$, the accepted prefix-context distribution is the corresponding ν-weighted turn distribution; the uniform case above is exactly pooled-turn sampling.

C Compute Resources

All HotpotQA SFT and GRPO experiments were run on a single H100 node with 8 GPU cards and 64 CPUs; the Wikipedia document retriever tool was co-located with the `verl` trainer setup. The Qwen3-235B teacher model was queried via API calls through OpenRouter. All ALFWorld SFT and GRPO experiments were run on a two H100 GPU cards and 64 CPUs. Most of Terminal-Bench-Dev SFT tasks were run on a single A100 node with 4 GPUs and 64 CPUs. Some tasks were run on a single H100 node with 4 GPUs and 64 CPUs. The GLM 4.6 teacher model was queried via API calls through OpenRouter. Virtual sandboxes were accessed through Daytona.

D HotpotQA

D.1 Training Setup

All HotpotQA experiments use **Qwen2.5-3B-Instruct** as the student model, trained on a single node with 8× H100 GPUs via the `verl` framework [Sheng et al., 2025] with FSDP2 sharding in BF16

mixed precision. Gradient checkpointing is enabled throughout. Full-parameter finetuning is used. Table 2 lists the SFT hyperparameters. Each iteration fine-tunes the latest checkpoint for 3 epochs over the accumulated correction dataset, and an author manually reviews the validation loss to select the checkpoint with the lowest loss.

Table 2: HotpotQA SFT hyperparameters.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	3×10^{-5}
LR scheduler	Cosine
Warmup ratio	0.1
Weight decay	0.01
Optimizer betas	$(0.9, 0.95)$
Train batch size	64
Micro batch size / GPU	2
Max sequence length	4096
Number of epochs	3
Precision	BF16

D.2 HotpotQA Model Checkpoint Selection

In this section, we show how we selection SFT model checkpoints from the results in Figure 6. We compare pass@1 performance on 3k medium problems in many different cases to get the model checkpoint to use for later experiments.

E ALFWorld

E.1 Training Setup

ALFWorld experiments are also built upon `VeRL` repo. We detail the experiment configs for SFT in Table 3.

Table 3: ALFWorld SFT hyperparameters.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	5×10^{-5}
LR scheduler	Cosine
Warmup ratio	0.1
Weight decay	0.01
Optimizer betas	$(0.9, 0.95)$
Train batch size	16
Micro batch size / GPU	2
Max sequence length	16384
Number of epochs	8
Precision	BF16

F Terminal-Bench-Dev

This section describes the SFT training details in Section 3.3, see Table 4. The hyperparameters work for all the SFT training process we mentioned, including the pure BC SFT process, as well as OP and BC+OP experiments. This training pipeline is inherited from OpenThoughts-Agent Team [OpenThoughts-Agent Team, 2025] with light modifications to make fair comparison.

[illegible]

All OP runs: same 1k BC traces + OP augmentation · Qwen2.5-3B → Qwen3-235B · LR 3e-5 · values in % · 95% CI under values

STRATEGY	K	EP	EP 1	EP 2	EP 3	EP 4	EP 5	EP 6	EP 7	PEAK
OP-Short	1	3	49.2 47.4–51.0	**51.7**★ 49.9–53.5	51.5 49.7–53.3	—	—	—	—	51.7
		5	48.0 46.2–49.8	51.3 49.5–53.1	53.2 51.4–55.0	**53.6**★ 51.8–55.4	53.2 51.4–55.0	—	—	53.6
		7	49.4 47.6–51.2	50.6 48.9–52.4	52.8 51.0–54.6	51.9 50.1–53.8	**53.5**★ 51.7–55.3	52.9 51.1–54.7	52.9 51.1–54.8	53.5
	2	3	49.3 47.5–51.1	51.9 50.1–53.7	**52.1**★ 50.3–53.9	—	—	—	—	52.1
		5	48.8 47.0–50.6	52.4 50.6–54.2	53.2 51.4–55.0	53.1 51.3–54.9	**53.3**★ 51.5–55.1	—	—	53.3
		7	49.7 48.0–51.6	51.2 49.4–53.0	52.5 50.7–54.3	53.4 51.6–55.2	**53.7**★ 51.9–55.5	52.9 51.1–54.8	53.5 51.7–55.3	53.7
	3	3	51.0 49.2–52.8	51.2 49.4–53.0	**52.7**★ 50.9–54.5	—	—	—	—	52.7
		5	49.8 48.0–51.6	52.0 50.2–53.8	**54.1**★ 52.3–55.9	53.4 51.6–55.2	53.6 51.8–55.4	—	—	54.1
	5	3	49.3 47.5–51.1	53.1 51.3–54.9	**53.4**★ 51.6–55.2	—	—	—	—	53.4
		5	47.7 45.9–49.5	52.2 50.4–54.0	52.6 50.8–54.4	**54.1**★ 52.3–55.9	53.8 52.0–55.6	—	—	54.1
		7	49.2 47.4–51.0	52.5 50.7–54.3	53.1 51.4–54.9	53.9 52.1–55.7	53.9 52.1–55.7	54.1 52.3–55.9	**54.3**★ 52.5–56.1	54.3
OP-Success	1	3	47.5 45.7–49.3	50.7 48.9–52.5	**52.0**★ 50.2–53.8	—	—	—	—	52.0
		5	47.9 46.1–49.7	50.7 48.9–52.5	51.9 50.1–53.7	53.0 51.2–54.8	**53.5**★ 51.7–55.3	—	—	53.5
	2	3	48.7 46.9–50.5	50.9 49.1–52.7	**52.0**★ 50.2–53.8	—	—	—	—	52.0
		5	48.4 46.6–50.2	50.0 48.2–51.8	53.4 51.6–55.2	52.0 50.2–53.8	**54.4**★ 52.6–56.2	—	—	54.4
		7	49.1 47.4–51.0	50.1 48.3–51.9	**53.2**★ 51.4–55.0	52.6 50.8–54.4	52.3 50.5–54.1	53.0 51.2–54.8	53.2 51.5–55.0	53.2
	5	3	50.4 48.6–52.2	51.1 49.3–52.9	**53.1**★ 51.3–54.9	—	—	—	—	53.1
		5	48.8 47.0–50.6	51.7 49.9–53.5	54.0 52.2–55.8	53.4 51.6–55.2	**54.2**★ 52.4–56.0	—	—	54.2
		7	49.1 47.3–50.9	*FAIL*	53.0 51.2–54.8	53.2 51.4–55.0	53.3 51.5–55.1	53.0 51.2–54.8	**53.6**★ 51.8–55.4	53.6
OP-Critical	1	3	45.4 43.6–47.2	50.5 48.7–52.3	**51.8**★ 50.0–53.6	—	—	—	—	51.8
		5	45.9 44.1–47.7	50.7 48.9–52.5	**52.6**★ 50.8–54.4	52.1 50.3–53.9	52.0 50.2–53.8	—	—	52.6
	2	3	46.6 44.8–48.4	**51.7**★ 49.9–53.5	51.1 49.3–52.9	—	—	—	—	51.7
		5	47.5 45.7–49.3	50.2 48.4–52.0	52.7 50.9–54.5	52.8 51.0–54.6	**53.6**★ 51.8–55.4	—	—	53.6
	3	3	47.4 45.6–49.2	51.5 49.7–53.3	**52.0**★ 50.2–53.8	—	—	—	—	52.0
		5	47.5 45.7–49.3	49.5 47.7–51.3	52.8 51.0–54.6	52.7 50.9–54.5	**53.3**★ 51.5–55.1	—	—	53.3
	5	3	46.3 44.5–48.1	**52.1**★ 50.3–53.9	51.5 49.7–53.3	—	—	—	—	52.1
		5	46.4 44.6–48.2	51.8 50.0–53.6	52.3 50.5–54.1	**53.0**★ 51.2–54.8	52.7 50.9–54.5	—	—	53.0
Pure BC no OP aug.	1k q×1 K=∞	5	44.9 43.1–46.7	49.1 47.3–50.9	51.2 49.4–53.0	**51.5**★ 49.7–53.3	**51.5**★ 49.7–53.3	—	—	51.5
		7	45.4 43.6–47.2	49.3 47.5–51.1	51.7 50.0–53.6	51.1 49.3–52.9	51.7 49.8–53.5	**52.1**★ 50.3–53.9	51.8 50.0–53.7	52.1
	1k q×2 K=∞	5	48.6 46.8–50.4	50.0 48.2–51.8	**52.2**★ 50.4–54.0	51.2 49.4–53.0	51.0 49.2–52.8	—	—	52.2
	1k q×4 K=∞	5	49.6 47.8–51.4	51.6 49.8–53.4	**53.4**★ 51.6–55.2	52.9 51.1–54.7	52.3 50.5–54.1	—	—	53.4
	2k q×1 K=∞	5	43.6 41.8–45.4	50.4 48.6–52.2	49.3 47.5–51.1	49.8 48.0–51.6	**50.7**★ 48.9–52.5	—	—	50.7
		7	43.8 42.1–45.6	50.0 48.2–51.8	50.2 48.4–52.1	50.5 48.7–52.3	51.4 49.6–53.2	50.7 48.9–52.5	**51.5**★ 49.7–53.4	51.5
	4k q×1 K=∞	5	47.4 45.6–49.2	51.6 49.8–53.4	53.7 51.9–55.5	53.2 51.4–55.0	**54.0**★ 52.2–55.8	—	—	54.0

≤48 55 pass@1 % (clipped below 48)

"Ep" = total epochs in cosine LR schedule · "Ep N" column = checkpoint saved after epoch N · ★ best epoch per run (bold + inset border) · FAIL = eval crashed, pending rerun

Notes: OP-Success K=2 ep7 — ep3 & ep7 tied at 53.2; ep3 marked best (first occurrence). OP-Success K=5 ep7 — ep2 eval failed. Pure BC 1k q×1 ep5 — ep4 & ep5 tied at 51.5.

Figure 6: HotpotQA checkpoint selection results.

Table 4: Terminal-Bench-Dev SFT training hyperparameters.

Hyperparameter	**Value**
Learning rate	4×10^{-5}
Train batch size per device	1
Gradient Accumulation	4
Seed	42
Distributed type	Multi-GPU
Number of devices	4
Total train batch size	16
Optimizer	AdamW Torch Fused
Optimizer betas	$(0.9, 0.98)$
Optimizer epsilon	1×10^{-8}
Learning rate scheduler	Cosine
Warmup ratio	0.1
Number of epochs	7.0

F.1 Terminal-Bench-Dev extra experiments

Table 5 shows all TB-dev experiment results in different settings. First two pure OP results validates our conclusion that the optimal continuation length should be set with the task horizon and is not a fixed number. The BC+OP with different tasks experiment is more alike to OEC [Lauffer et al., 2025]. The result show that this data mixture also works on Qwen3-8B, different from OEC which SFT on the prefix model directly. And our RL result shows training and do RL directly on student model can outperform reported RL result from OpenThoughts-Agent team. We would like to mention that our method is not exclusive to OEC. Instead, one can regard our method as a better initialization point than pure BC in OEC.

Table 5: Comparison of TB-dev evaluation scores for baseline models and Qwen3-8B under different SFT and RL settings. (numbers with * are from Openthoughts, others are from our implementation. Score is averaged across three attempts)

Base Model	**Teacher Model**	**Type**	**Prefix Training Data**	**BC Data**	**On-policy Data**	**RL**	**TB-dev Score** $(\%, \uparrow)$
Qwen3-8B	GLM-4.6	Pure BC	10%	10%	0	0	11.9
			25%	25%		0	10.7
			50%	50%		0	11.6
			75%	75%		0	14.2
			100%	100%		0	13.8
			100%	100%		720 tasks	16.7
			100%	100%		720 tasks	17.3*
	GLM-4.6	Pure OP	10%	0%	10% $(K = 3)$	0	14.7
		Pure OP	10%	0%	10% $(K = \infty)$	0	16.7
		BC+OP (different tasks)	10%	15%	10% $(K = 3)$	0	15.1
		BC+OP (different tasks)	10%	15%	10% $(K = 3)$	720 tasks	18.3
		OP-SHORT	10%	2%	2% $(K = 3)$	0	16.1
		OP-SHORT	10%	2%	2% $(K = \infty)$	0	18.0
		OP-SHORT	10%	5%	5% $(K = 3)$	0	19.1
Qwen3-8B	Baselines						5.7*
Qwen3-235B-A22B							26.9*
GLM-4.6							29.9
GPT-5 Nano							37.4*

G Anatomy of an LLM agent's rollout

This appendix illustrates how a ReAct-style rollout decomposes into actions, observations, and tokens, instantiating the formalism in §2.1. Figure 7 shows a single rollout on one of our HotpotQA training questions: a system prompt and question followed by alternating agent turns (thinking and tool-call segments) and environment observations (tool responses), with token IDs annotated so the next-token-prediction target is explicit. Figure 8 re-draws the same trajectory as a linearized sequential decision process: the policy π_θ emits each action a_t autoregressively conditioned on the growing history $h_{t-1} = (q, a_1, o_1, \ldots, a_{t-1}, o_{t-1})$, and the dashed box marks the next-step prediction target supervised at training time.

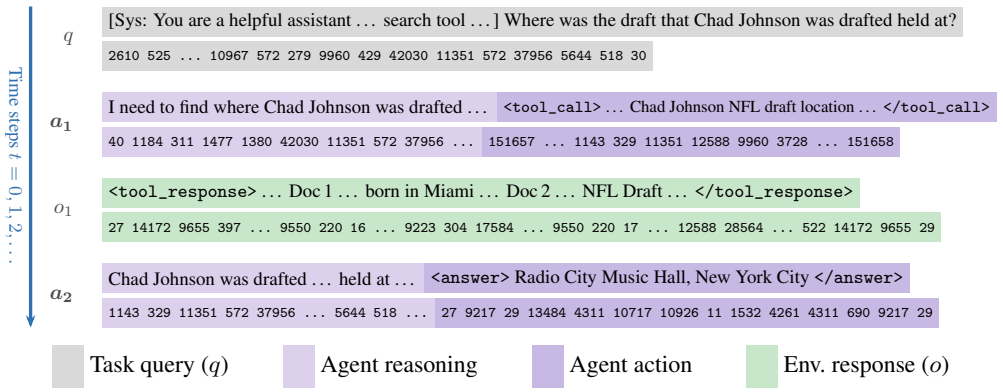

Figure 7: A real ReAct-style rollout instantiating the sequential decision process in actual model output and token IDs. For each span, the top row shows abbreviated text and the bottom row shows corresponding token IDs (under a Qwen 2.5 tokenizer). The agent emits a reasoning trace and tool call (together as a_1), receives a search result (o_1), then boxes a final answer in a_2. XML tags such as `<tool_call>`/`</tool_call>` delimit span boundaries in the raw sequence.

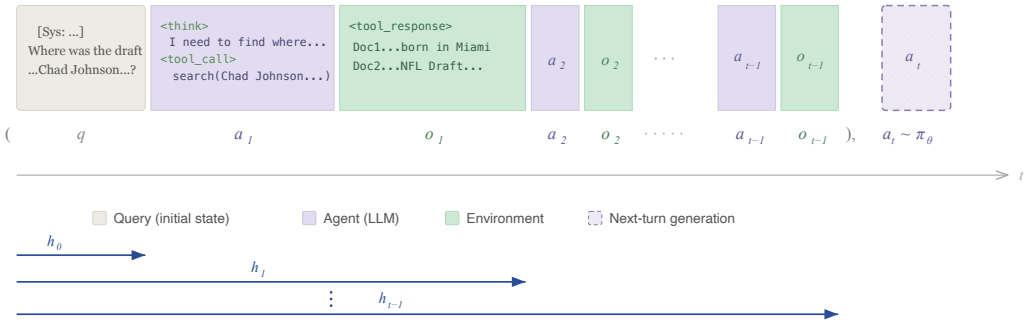

Figure 8: Linearized schematic of the Figure 7 rollout trajectory as a sequential decision process. The agent's next assistant turn generation, a_t, is sampled token by token, autoregressively from the conditional policy distribution $\pi_\theta(\cdot|h_{t-1})$.

H Example On-Policy Agent Trajectory

We attach one example of on-policy agent trajectory which is used in Terminal-Bench-Dev task for OP-Short. The prefix model is $\text{SFT}(\pi_0, 10\%\mathcal{D}_{\text{BC}})$, student model is Qwen3-8B and teacher model is GLM 4.6. Note we sometimes also call prefix model "student model" in certain context, people should be clear about the usage. We choose this trajectory because this task cannot be marked complete either in $\mathcal{D}_{\text{BC}}$ or when prefix model is Qwen3-8B as well.

Step 1: Full User Prompt and Task Description

```
You are an AI assistant tasked with solving command-line tasks in a Linux environment. You will be
      given a task description and the output from previously executed commands. Your goal is to
      solve the task by providing batches of shell commands.

Format your response as JSON with the following structure:

{
  "analysis": "Analyze the current state based on the terminal output provided. What do you see?
      What has been accomplished? What still needs to be done?",
  "plan": "Describe your plan for the next steps. What commands will you run and why? Be specific
      about what you expect each command to accomplish.",
  "commands": [
    {
      "keystrokes": "ls -la\n",
      "duration": 0.1
    },
    {
      "keystrokes": "cd project\n",
      "duration": 0.1
    }
  ],
  "task_complete": true
}

Required fields:
- "analysis": Your analysis of the current situation
- "plan": Your plan for the next steps
- "commands": Array of command objects to execute

Optional fields:
- "task_complete": Boolean indicating if the task is complete (defaults to false if not present)

Command object structure:
- "keystrokes": String containing the exact keystrokes to send to the terminal (required)
- "duration": Number of seconds to wait for the command to complete before the next command will be
      executed (defaults to 1.0 if not present)

IMPORTANT: The text inside "keystrokes" will be used completely verbatim as keystrokes. Write
      commands exactly as you want them sent to the terminal:
- You must end every command with a newline (\n) or it will not execute.
- For special key sequences, use tmux-style escape sequences:
  - C-c for Ctrl+C
  - C-d for Ctrl+D

The "duration" attribute specifies the number of seconds to wait for the command to complete (
      default: 1.0) before the next command will be executed. On immediate tasks (e.g., cd, ls, echo,
       cat) set a duration of 0.1 seconds. On commands (e.g., gcc, find, rustc) set a duration of 1.0
       seconds. On slow commands (e.g., make, python3 [long running script], wget [file]) set an
      appropriate duration as you determine necessary.

It is better to set a smaller duration than a longer duration. It is always possible to wait again
      if the prior output has not finished, by running {"keystrokes": "", "duration": 10.0} on
      subsequent requests to wait longer. Never wait longer than 60 seconds; prefer to poll to see
      intermediate result status.

Important notes:
- Each command's keystrokes are sent exactly as written to the terminal
- Do not include extra whitespace before or after the keystrokes unless it's part of the intended
      command
- Extra text before or after the JSON will generate warnings but be tolerated
- The JSON must be valid - use proper escaping for quotes and special characters within strings
- Commands array can be empty if you want to wait without taking action
```

Task Description:
InferredBugs Task - Csharp

Project Information

Project: IdentityModel.AspNetCore
Bug ID: 1
Language: csharp

Bug Information

File-Level Changes:

Before (Buggy File):
```csharp
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;

namespace Tests.Infrastructure
{
    class NetworkHandler : HttpMessageHandler
    {
        public Uri Address { get; set; }

        public HttpContent Content { get; set; }

        public IDictionary<string, object> Properties { get; set; }

        protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
      CancellationToken cancellationToken)
        {
            Address = request.RequestUri;
            Content = request.Content;
            Properties = request.Properties;

            return Task.FromResult(new HttpResponseMessage(System.Net.HttpStatusCode.NotFound));
        }
    }
}
```

Bug Type:
DOTNET_RESOURCE_LEAK

Qualifier:
Leaked resource (output of System.Net.Http.HttpResponseMessage::.ctor() at Line 23) of type System.
      Net.Http.HttpResponseMessage in method "Task`1<!0> NetworkHandler.SendAsync(HttpRequestMessage,
      CancellationToken)".

Severity:
ERROR

Procedure:
Task`1<!0> NetworkHandler.SendAsync(HttpRequestMessage,CancellationToken)

File:
test/Tests/Infrastructure/NetworkHandler.cs

Key:
NetworkHandler.cs|SendAsync|DOTNET_RESOURCE_LEAK

Hash:
e571a69e7e29eb0a1446ed12ba4f3815

Bug Type Hum:
Dotnet Resource Leak

Task

Fix the bug in the code.

<!-- OT_ROLLOUT_TAG: rollout=003 task_id=inferredbugs-0634__rollout003 -->

```
Current terminal state:
Current Terminal Screen:
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 2: Prefix Model Agent

```
{
  "analysis": "I need to analyze the current terminal state and understand the task. The task is to
     fix a C# resource leak bug in the IdentityModel.AspNetCore project. The bug is in the
     NetworkHandler.cs file where an HttpResponseMessage is created but not properly disposed,
     causing a resource leak. The current terminal shows I'm in the /app directory as root. I need
     to explore the project structure to locate the file and understand the codebase.",
  "plan": "First, I'll explore the current directory structure to understand the project layout and
     locate the NetworkHandler.cs file mentioned in the bug report. Then I'll examine the current
     implementation to understand the resource leak issue and implement a fix. I will execute one
     command first, inspect the result, and continue step by step.",
  "commands": [
    {
      "keystrokes": "ls -la\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```

Environment after Step 2

```
New Terminal Output:
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app# ls -la
total 0
drwxr-xr-x 2 root root   6 Apr 17 19:53 .
drwxr-xr-x 1 root root 102 Apr 26 04:56 ..
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 3: Prefix Model Agent

```
{
  "analysis": "The /app directory is empty, which means the project files are not yet present. I
     need to locate or create the project structure. Based on the bug report, the file should be at
     'test/Tests/Infrastructure/NetworkHandler.cs'. Let me check if there are any files in the root
     directory or if I need to create the project structure.",
  "plan": "I'll check the root directory to see if there are any project files or if I need to
     create the entire project structure. The bug report indicates the file should be in test/Tests/
     Infrastructure/NetworkHandler.cs, so I may need to create this structure. I will execute one
     command first, inspect the result, and continue step by step.",
  "commands": [
    {
      "keystrokes": "cd /\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```

Environment after Step 3

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app# cd /
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/#
```

Step 4: Teacher Model Agent


```
{
  "analysis": "I'm now in the root directory. I need to check if there are any project files or if
      the project structure needs to be created. Let me explore the directory structure to see what's
       available.",
  "plan": "I'll list the contents of the root directory to see if there are any existing project
      files or directories that might contain the IdentityModel.AspNetCore project. If not, I'll need
       to create the project structure based on the bug report information.",
  "commands": [
    {
      "keystrokes": "ls -la\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```


Environment after Step 4

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/# ls -la
total 0
drwxr-xr-x     1 root   root     102 Apr 26 04:56 .
drwxr-xr-x     1 root   root     102 Apr 26 04:56 ..
-rwxr-xr-x     1 root   root       0 Apr 26 04:50 .dockerenv
drwxr-xr-x     2 root   root       6 Apr 17 19:53 app
lrwxrwxrwx     1 root   root       7 Apr 22  2024 bin -> usr/bin
drwxr-xr-x     2 root   root      10 Apr 22  2024 boot
drwxr-xr-x    14 root   root    4420 Apr 26 04:50 dev
drwxr-xr-x     1 root   root      51 Apr 26 04:56 etc
drwxr-xr-x     3 root   root      28 Apr 10 02:29 home
lrwxrwxrwx     1 root   root       7 Apr 22  2024 lib -> usr/lib
lrwxrwxrwx     1 root   root       9 Apr 22  2024 lib64 -> usr/lib64
drwxr-xr-x     4 root   root      47 Apr 26 04:50 logs
drwxr-xr-x     2 root   root      10 Apr 10 02:20 media
drwxr-xr-x     2 root   root      10 Apr 10 02:20 mnt
drwxr-xr-x     2 root   root      10 Apr 10 02:20 opt
dr-xr-xr-x 15473 root   root       0 Apr 26 04:50 proc
drwx------     1 root   root      30 Apr 26 04:50 root
drwxr-xr-x     4 root   root      45 Apr 10 02:29 run
lrwxrwxrwx     1 root   root       8 Apr 22  2024 sbin -> usr/sbin
drwxr-xr-x     2 root   root      10 Apr 10 02:20 srv
dr-xr-xr-x    13 nobody nogroup    0 Jan 24 15:54 sys
drwxrwxrwt     1 root   root      74 Apr 26 04:56 tmp
drwxr-xr-x     1 root   root      89 Apr 10 02:20 usr
drwxr-xr-x     1 root   root      57 Apr 10 02:29 var
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/#
```

Step 5: Teacher Model Agent

```
{
  "analysis": "I can see the root directory structure. The /app directory exists but was empty.
     Since the bug report specifies a file at 'test/Tests/Infrastructure/NetworkHandler.cs', I need
     to create the project structure. Let me go back to /app and create the necessary directory
     structure and the file with the buggy code.",
  "plan": "I'll return to the /app directory and create the required directory structure (test/Tests
     /Infrastructure/) and the NetworkHandler.cs file with the buggy code as specified in the bug
     report. Then I'll fix the resource leak issue.",
  "commands": [
    {
      "keystrokes": "cd /app\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```

Environment after Step 5

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/# cd /app
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 6: Teacher Model Agent

```
{
  "analysis": "I'm back in the /app directory. I need to create the directory structure and the
     NetworkHandler.cs file with the buggy code as specified in the bug report, then fix the
     resource leak issue.",
  "plan": "I'll create the directory structure test/Tests/Infrastructure/ and then create the
     NetworkHandler.cs file with the buggy code provided in the bug report.",
  "commands": [
    {
      "keystrokes": "mkdir -p test/Tests/Infrastructure\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```

Environment after Step 6

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app# mkdir -p test/Tests/Infrastructure
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 7: Teacher Model Agent


```
{
  "analysis": "The directory structure has been created successfully. Now I need to create the
     NetworkHandler.cs file with the buggy code as specified in the bug report.",
  "plan": "I'll create the NetworkHandler.cs file in the test/Tests/Infrastructure/ directory with
     the exact buggy code provided in the bug report. Then I'll analyze and fix the resource leak
     issue.",
  "commands": [
    {
      "keystrokes": "cat > test/Tests/Infrastructure/NetworkHandler.cs << 'EOF'\nusing System;\
     nusing System.Collections.Generic;\nusing System.Net.Http;\nusing System.Threading;\nusing
     System.Threading.Tasks;\n\nnamespace Tests.Infrastructure\n{\n    class NetworkHandler :
     HttpMessageHandler\n    {\n        public Uri Address { get; set; }\n\n        public
     HttpContent Content { get; set; }\n\n        public IDictionary<string, object> Properties {
     get; set; }\n\n        protected override Task<HttpResponseMessage> SendAsync(
     HttpRequestMessage request, CancellationToken cancellationToken)\n        {\n
     Address = request.RequestUri;\n            Content = request.Content;\n            Properties =
      request.Properties;\n\n            return Task.FromResult(new HttpResponseMessage(System.Net.
     HttpStatusCode.NotFound));\n        }\n    }\n}\nEOF\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```


Environment after Step 7

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app# cat > test/Tests/Infrastructure/NetworkHandler.cs <<
      'EOF'
> using System;
> using System.Collections.Generic;
> using System.Net.Http;
> using System.Threading;
> using System.Threading.Tasks;
>
> namespace Tests.Infrastructure
> {
>     class NetworkHandler : HttpMessageHandler
>     {
>         public Uri Address { get; set; }
>
>         public HttpContent Content { get; set; }
>
>         public IDictionary<string, object> Properties { get; set; }
>
>         protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
     CancellationToken cancellationToken)
>         {
>             Address = request.RequestUri;
>             Content = request.Content;
>             Properties = request.Properties;
>
>             return Task.FromResult(new HttpResponseMessage(System.Net.HttpStatusCode.NotFound));
>         }
>     }
> }
> EOF
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 8: Teacher Model Agent

```
{
  "analysis": "The NetworkHandler.cs file has been created successfully with the buggy code. Now I
     need to analyze the resource leak issue. The bug report indicates that the HttpResponseMessage
     created on line 23 is not being properly disposed, causing a resource leak. In C#,
     HttpResponseMessage implements IDisposable and should be disposed when no longer needed.",
  "plan": "I'll examine the current file content to confirm it was created correctly, then fix the
     resource leak by ensuring the HttpResponseMessage is properly disposed. The typical fix would
     be to use a using statement or ensure proper disposal of the HttpResponseMessage.",
  "commands": [
    {
      "keystrokes": "cat test/Tests/Infrastructure/NetworkHandler.cs\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```

Environment after Step 8

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app# cat test/Tests/Infrastructure/NetworkHandler.cs
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;

namespace Tests.Infrastructure
{
    class NetworkHandler : HttpMessageHandler
    {
        public Uri Address { get; set; }

        public HttpContent Content { get; set; }

        public IDictionary<string, object> Properties { get; set; }

        protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
     CancellationToken cancellationToken)
        {
            Address = request.RequestUri;
            Content = request.Content;
            Properties = request.Properties;

            return Task.FromResult(new HttpResponseMessage(System.Net.HttpStatusCode.NotFound));
        }
    }
}
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 9: Teacher Model Agent

```
{
  "analysis": "Perfect! The file has been created with the exact buggy code. I can see the resource
     leak issue on line 23 where `new HttpResponseMessage(System.Net.HttpStatusCode.NotFound)` is
     created and returned via Task.FromResult, but there's no mechanism to ensure it gets disposed
     properly. HttpResponseMessage implements IDisposable and needs to be disposed when the caller
     is done with it.",
  "plan": "I'll fix the resource leak by implementing a proper disposal mechanism. The best approach
      is to modify the NetworkHandler class to implement IDisposable and keep track of the created
     HttpResponseMessage so it can be disposed properly. This ensures the resource doesn't leak.",
```

```
  "commands": [
    {
      "keystrokes": "cat > test/Tests/Infrastructure/NetworkHandler.cs << 'EOF'\nusing System;\
     nusing System.Collections.Generic;\nusing System.Net.Http;\nusing System.Threading;\nusing
     System.Threading.Tasks;\n\nnamespace Tests.Infrastructure\n{\n    class NetworkHandler :
     HttpMessageHandler, IDisposable\n    {\n        public Uri Address { get; set; }\n\n
     public HttpContent Content { get; set; }\n\n        public IDictionary<string, object>
     Properties { get; set; }\n\n        private HttpResponseMessage _responseMessage;\n
     private bool _disposed = false;\n\n        protected override Task<HttpResponseMessage>
     SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)\n        {\n
         Address = request.RequestUri;\n            Content = request.Content;\n
     Properties = request.Properties;\n\n            // Dispose previous response if exists\n
          _responseMessage?.Dispose();\n            \n            _responseMessage = new
     HttpResponseMessage(System.Net.HttpStatusCode.NotFound);\n            return Task.FromResult(
     _responseMessage);\n        }\n\n        public void Dispose()\n        {\n            Dispose(
     true);\n            GC.SuppressFinalize(this);\n        }\n\n        protected virtual void
     Dispose(bool disposing)\n        {\n            if (!_disposed)\n            {\n
      if (disposing)\n                {\n                _responseMessage?.Dispose();\n
            }\n                _disposed = true;\n            }\n        }\n    }\n}\nEOF\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```


Environment after Step 9

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app# cat > test/Tests/Infrastructure/NetworkHandler.cs <<
      'EOF'
> using System;
> using System.Collections.Generic;
> using System.Net.Http;
> using System.Threading;
> using System.Threading.Tasks;
>
> namespace Tests.Infrastructure
> {
>     class NetworkHandler : HttpMessageHandler, IDisposable
>     {
>         public Uri Address { get; set; }
>
>         public HttpContent Content { get; set; }
>
>         public IDictionary<string, object> Properties { get; set; }
>
>         private HttpResponseMessage _responseMessage;
>         private bool _disposed = false;
>
>         protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
     CancellationToken cancellationToken)
>         {
>             Address = request.RequestUri;
>             Content = request.Content;
>             Properties = request.Properties;
>
>             // Dispose previous response if exists
>             _responseMessage?.Dispose();
>
>             _responseMessage = new HttpResponseMessage(System.Net.HttpStatusCode.NotFound);
>             return Task.FromResult(_responseMessage);
>         }
>
>         public void Dispose()
>         {
>             Dispose(true);
>             GC.SuppressFinalize(this);
>         }
>
```

```
>         protected virtual void Dispose(bool disposing)
>         {
>             if (!_disposed)
>             {
>                 if (disposing)
>                 {
>                     _responseMessage?.Dispose();
>                 }
>                 _disposed = true;
>             }
>         }
>     }
> }
> EOF
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 10: Teacher Model Agent

```
{
  "analysis": "The resource leak fix has been implemented. I've modified the NetworkHandler class to
      implement IDisposable and added proper disposal mechanisms for the HttpResponseMessage. The
     fix includes: 1) Making NetworkHandler implement IDisposable, 2) Adding a private field to
     track the HttpResponseMessage, 3) Disposing previous responses before creating new ones, 4)
     Implementing the full Dispose pattern with proper cleanup. Let me verify the fix is correctly
     applied.",
  "plan": "I'll verify that the fix has been correctly applied by viewing the updated file content
     to ensure all the resource leak fixes are in place.",
  "commands": [
    {
      "keystrokes": "cat test/Tests/Infrastructure/NetworkHandler.cs\n",
      "duration": 1.0
    }
  ],
  "task_complete": false
}
```

Environment after Step 10

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app# cat test/Tests/Infrastructure/NetworkHandler.cs
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;

namespace Tests.Infrastructure
{
    class NetworkHandler : HttpMessageHandler, IDisposable
    {
        public Uri Address { get; set; }

        public HttpContent Content { get; set; }

        public IDictionary<string, object> Properties { get; set; }

        private HttpResponseMessage _responseMessage;
        private bool _disposed = false;

        protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
     CancellationToken cancellationToken)
        {
            Address = request.RequestUri;
```

```
            Content = request.Content;
            Properties = request.Properties;

            // Dispose previous response if exists
            _responseMessage?.Dispose();

            _responseMessage = new HttpResponseMessage(System.Net.HttpStatusCode.NotFound);
            return Task.FromResult(_responseMessage);
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    _responseMessage?.Dispose();
                }
                _disposed = true;
            }
        }
    }
}
root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Step 11: Teacher Model Agent


```
{
  "analysis": "Perfect! The resource leak fix has been successfully implemented. The NetworkHandler
     class now properly handles the HttpResponseMessage resource leak by: 1) Implementing
     IDisposable interface, 2) Tracking the HttpResponseMessage instance in a private field, 3)
     Disposing previous responses before creating new ones to prevent accumulation, 4) Implementing
     the full Dispose pattern with proper cleanup. The bug has been fixed according to the
     DOTNET_RESOURCE_LEAK issue described in the bug report.",
  "plan": "The task is complete. The resource leak bug in NetworkHandler.cs has been fixed by
     implementing proper disposal of the HttpResponseMessage objects. The fix addresses the specific
      issue mentioned in the bug report where the HttpResponseMessage created on line 23 was leaking
      resources.",
  "commands": [],
  "task_complete": true
}
```


Environment after Step 11

```
Current terminal state:
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#

Are you sure you want to mark the task as complete? This will trigger your solution to be graded and
      you won't be able to make any further corrections. If so, include "task_complete": true in
     your JSON response again.
```

Step 12: Teacher Model Agent

```
{
  "analysis": "The task has been completed successfully. I have created the project structure,
     implemented the buggy NetworkHandler.cs file as specified in the bug report, and then fixed the
      DOTNET_RESOURCE_LEAK issue. The fix includes implementing IDisposable on the NetworkHandler
     class, adding proper resource tracking, and ensuring HttpResponseMessage objects are disposed
     correctly. The specific resource leak mentioned in the bug report (line 23 HttpResponseMessage
     creation) has been resolved with proper disposal mechanisms.",
  "plan": "The task is complete. The resource leak bug has been fixed according to the
     specifications provided in the bug report.",
  "commands": [],
  "task_complete": true
}
```

Environment after Step 12

```
New Terminal Output:

root@ad793e58-22a6-42c6-a49f-72280f1dabab:/app#
```

Figure 9: Example trajectory from an InferredBugs command-line task. Purple denotes the full user prompt and task description, blue denotes terminal/environment observations, orange denotes student model actions up to Step 3, and green denotes teacher model actions from Step 4 onward.