BETTER THAN YOUR TEACHER: LLM AGENTS THAT LEARN FROM PRIVILEGED AI FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

While large language models (LLMs) show impressive decision-making abilities, current methods lack a mechanism for automatic self-improvement from errors during task execution. We propose LEAP, an iterative fine-tuning framework that continually improves LLM agents using feedback from AI expert teachers. Our key insight is to equip the expert teachers with a *privileged state* —information available during training but hidden at test time. This allows even weak experts to provide precise guidance, significantly improving the student agent's performance without access to privileged information at test time. We evaluate LEAP on diverse decision-making benchmarks, including text-based games (ALFWorld), web navigation (WebShop), and interactive coding (Intercode Bash). Our experiments show that LEAP (1) outperforms behavior cloning and ReAct baselines (2) enables weak student models (e.g., Llama3-8B) to exceed performance of strong teacher models (GPT-4o), and (3) allows weak models to self-improve using privileged versions of themselves. We also provide a theoretical analysis showing that LEAP 's success hinges on balancing privileged information with the student's realizability, which we empirically validate. Our code is provided as part of supplementary material.

1 INTRODUCTION

Large language models (LLM) show impressive decision-making abilities across domains (Yao et al., 2022b; Jimenez et al., 2023; Sodhi et al., 2024). However, they struggle to learn and recover from errors encountered at test time (Madaan et al., 2024; Shinn et al., 2023). Prompting alone is insufficient, as it requires manually specifying exceptions and examples (Khot et al., 2022), which is difficult to scale and leads to intractably long context lengths (Liu et al., 2024).

We address the problem of fine-tuning LLM agents to improve their decision-making using online interaction data. Imitation learning offers a sample-efficient way to train such models, where a teacher corrects the student agent's actions (Ross et al., 2011). However, finding an expert capable of both demonstrating the task and correcting mistakes from any state the student visits is extremely challenging (Chen et al., 2020). The expert must not only know how to complete the task but also how to recover from the student's errors, which is often more complex than simply performing the original task (Walsman et al., 2022). For example, in a task like "Heat the mug and put it in the cabinet," if the student places the wrong object in the microwave, the expert must first backtrack and remove the incorrect object before proceeding with the task.

Our key insight is to equip the expert teacher with access to a privileged state — information useful for solving the task but only available at train time (Vapnik et al., 2015). This includes hidden states of a partially observable environment or intermediate subgoals. For a task "Heat the mug and put it in the cabinet", the privileged state reveals the mug's location and steps for heating it (Fig. 1). With this knowledge, the expert first clears the microwave and then teaches the student effective search strategies, like checking common mug locations. Over time, this enables the student to learn general strategies for solving the task and recovering from errors, ultimately outperforming the expert, even without access to the privileged state at test time.

We propose **L**earning from **E**xperts with **A**ccess to **P**rivilege (LEAP), an iterative learning algorithm that fine-tunes LLM agents using privileged AI expert feedback. As shown in Fig. 1, LEAP begins

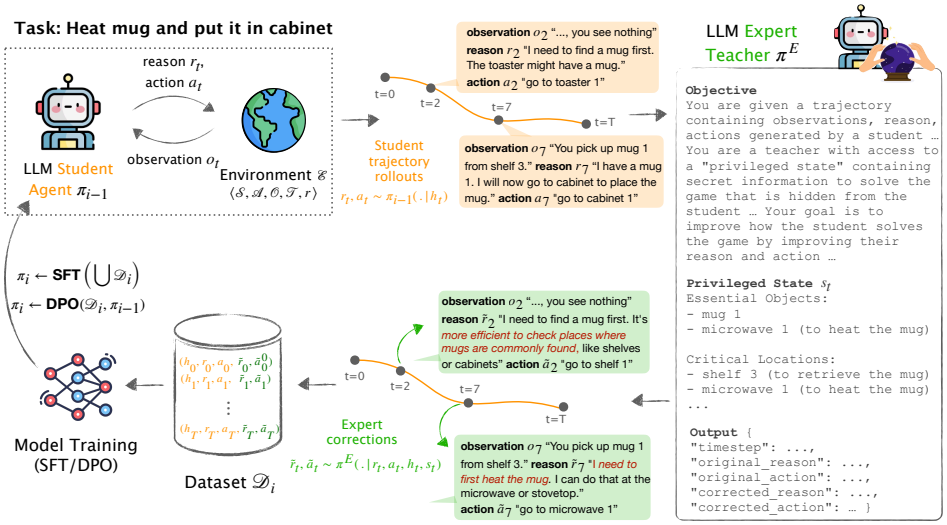


Figure 1: **LEAP overview**. LLM student agent interacts with the environment, generating a reason-action trajectory (in orange) based on its policy π_{i-1}. An expert teacher, with privileged state available only during training, evaluates and corrects the trajectory (in green). These corrections update the learner's policy to π_i through SFT/DPO training. Updated policy π_i is then rolled out at test time without access to privileged state.

with a student agent trained on demonstration data (π_0). During interactions, the student agent generates trajectories, and the privileged expert (π^E) identifies key time steps where corrections are needed. The expert provides improved reason and action labels based on the privileged state, which are then assembled into a dataset used to update the student model ($\pi_i \leftarrow \pi_{i-1}$). This process is repeated iteratively, progressively improving the student agent's performance.

Interestingly, we identify a key *trade-off between leveraging privileged information and the student agent's realizability*. While providing unrestricted privileged feedback allows for highly accurate corrections, these are often too complex for the student to act on effectively. Conversely, limiting feedback to what the student can easily process makes it realizable but often suboptimal. Our theoretical and empirical analyses show that balancing these extremes is key, which we achieve by introducing a constrained privileged expert that provides optimal yet realizable feedback.

Our key contributions are:

1. A novel iterative learning framework `LEAP` that fine-tunes LLM agents using privileged expert feedback, balancing privileged information and realizability.
2. Theoretical and empirical analysis of the optimal trade-off between privileged information and agent realizability.
3. Experimental validation on diverse interactive decision-making benchmarks: AlfWorld (Shridhar et al., 2020b), WebShop (Yao et al., 2022a), and InterCode (Yang et al., 2024).
   (a) `LEAP` consistently outperforms behavior cloning agents trained on `gpt-4o` or human demonstrations, showing significant improvements on ALFWorld (65.7% $\rightarrow$ 91.8%), WebShop (29.4 $\rightarrow$ 61.8), and InterCode Bash (60.3% $\rightarrow$ 71.8%).
   (b) `LEAP` enables weaker models (e.g. `Llama3-8B`) to surpass stronger models (`gpt-4o`), with improvements on ALFWorld (91.8% vs. 65.7%), WebShop (61.8 vs. 58.4), and InterCode Bash (71.8% vs. 71.3%).
   (c) Ablations that show LLM agents can self-improve using their privileged versions as teachers (65.7% $\rightarrow$ 82.1%), explore the tradeoff between privileged information and realizability, and compare supervised finetuning with preference optimization.

2 Preliminaries

Problem Formulation: Decision-making LLMs. We frame decision-making as a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998) because real-world environments often contain hidden or incomplete information. The POMDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, r \rangle$, where $\mathcal{S}$ represents the underlying state space of the environment, $\mathcal{A}$ the action space, $\mathcal{O}$ the observation space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ the transition dynamics, and $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ the reward function. The

true state $s_t \in \mathcal{S}$ is not fully observable; instead, the agent receives partial observations $o_t \in \mathcal{O}$ at each timestep. Hence it must rely on the history $h_t \in \mathcal{H}$ of observation-actions to choose actions.

LLM-based agents are policies $\pi \in \Pi$, which predict both a *reason* ρ_t and an action a_t based on the agent's history h_t. The reason ρ_t is expressed in natural language to justify action a_t. The history $h_t = o_1, a_1, \ldots, o_t$ is the sequence of past observations and actions. The policy is expressed as:

$$(\rho_t, a_t) = \pi(h_t) \tag{1}$$

While reinforcement learning (RL) for POMDPs is already challenging due to the large history space $\mathcal{H}$ (Papadimitriou & Tsitsiklis, 1987), this is further exacerbated by the addition of reason space, which is infinite and unconstrained. This makes RL impractical to run for LLM agents.

Interactive Imitation Learning with DAGGER. Imitation learning (IL) offers a promising alternative to RL, by allowing the agent to mimic an expert teacher policy π^E bypassing the need for complex exploration. The gold-standard for imitation learning are interactive methods like DAGGER (Ross et al., 2011) that query the teacher π^E for feedback on states h_t visited by the student. At each DAGGER iteration i, the student policy π_i is rolled out to collect histories h_t, and the teacher provides corrective reasons and actions (ρ_t^*, a_t^*) on those histories h_t. The new student π_{i+1} then minimizes the cumulative loss $\ell(.)$ on all the data up until the current iteration i:

$$\pi_{i+1} = \min_{\pi} \sum_{j=0}^{i} \mathbb{E}_{h_t \sim \pi_j, (\rho_t^*, a_t^*) \sim \pi^E(h_t)} \left[\ell((\rho_t^*, a_t^*), \pi(h_t)) \right], \tag{2}$$

DAGGER guarantees that as iterations $N \to \infty$, it will find the best policy in policy class Π.

3 APPROACH

We present **L**earning from **E**xperts with **A**ccess to **P**rivilege (LEAP), a framework for iteratively training an LLM agent through online AI feedback from a privileged expert (Fig. 1). Central to our approach is the concept of a *privileged expert teacher*, which leverages access to privileged state information to guide the student's recovery from errors. Section 3.1 defines the privileged teacher, Section 3.2 details LEAP, and Section 3.3 presents the theoretical analysis of LEAP.

3.1 PRIVILEGED EXPERT TEACHER

An expert teacher is a policy that can be queried with student history h_t of observations and actions, along with the student's predicted reason action (ρ_t, a_t) to generate a corrected reason action $(\tilde{\rho}_t, \tilde{a}_t) \sim \pi^E(\cdot|\rho_t, a_t, h_t)$. For POMDPs, computing such corrections is intractable and strictly more challenging than just solving the original task, often resulting in suboptimal feedback.

We define a *privileged expert teacher* as a teacher that has access to privileged state information, s_t, which fully describes the environment at a given time. This access transforms the teacher's decision-making into a fully observable Markov Decision Process (MDP), making it significantly more tractable. We formally define the privileged expert teacher as:

$$(\tilde{\rho}_t, \tilde{a}_t) \sim \pi^E(\cdot|\rho_t, a_t, h_t, s_t) \tag{3}$$

The privileged state s_t can either be explicitly provided or is implicit in privileged text information provided to the expert. For instance, in Alfworld (Shridhar et al., 2020a), it includes the exact location of objects and the goal necessary for the agent to navigate effectively. In WebShop (Yao et al., 2022a), it involves the specific name of the target product and its attributes, aiding precise search queries or recommendations. Similarly, in InterCode (Yang et al., 2024), the privileged state consists of the ground truth code and any unit tests the code must pass, providing a clear benchmark for the agent to generate or correct code. Note that the privileged information is available only at train time.

A crucial requirement for the privileged expert teacher is that *it must not reveal privileged information in its feedback to the learner.* The prompt instructions specify that the teacher should provide general reason and action that avoid the direct use of privileged information. Using such information directly might solve tasks quickly but makes the corrected actions $(\tilde{\rho}_t, \tilde{a}_t)$ unrealizable for the learner, who does not have access to the privileged state s_t, leading to a significant imitation gap and poor learner performance. In Section 3.3, we characterize the trade-off between using privileged information and student realizability. We show in Appendix. B that a constrained privileged expert that effectively

Algorithm 1 LEAP: Iterative Learning with Privileged Expert Teacher

Input: Privileged expert teacher π^E, Environment $\mathcal{E}$
Output: Learned policy π
Collect an initial set of demonstrations from expert teacher $\mathcal{D} \leftarrow \{(h_t^*, \rho_t^*, a_t^*)\}$
Train BC π_0 on $\mathcal{D}$ using supervised fine-tuning (SFT)
for iteration $i = 1, 2, \ldots, N$ **do**
  Roll out policy π_{i-1} in environment $\mathcal{E}$ to collect new data: $\mathcal{D}_i \leftarrow \{(h_t, \rho_t, a_t)\}$
  Compute privileged state s_t for every datapoint in $\mathcal{D}_i$
  Invoke privileged expert to generate corrected reason and action: $\tilde{\rho}_t, \tilde{a}_t \sim \pi^E(.|\rho_t, a_t, h_t, s_t)$
  Augment dataset with corrected reason and action: $\mathcal{D}_i \leftarrow \{(h_t, \rho_t, a_t, \tilde{\rho}_t, \tilde{a}_t)\}$
  Train policy π_i using a no-regret update: $\pi_i \leftarrow \text{SFT}\left(\bigcup \mathcal{D}_i\right)$ or $\pi_i \leftarrow \text{DPO}(\mathcal{D}_i, \pi_{i-1})$
return Best $\pi \in \{\pi_1, \ldots, \pi_N\}$ on validation dataset

utilizes privileged information while remaining realizable for the student has the optimal performance. We approximate such an expert by designing prompts for the expert to instruct it to use privileged information, without revealing it. See Appendix D for prompts.

3.2 The LEAP Algorithm

The LEAP algorithm iteratively trains the student policy π through online interactions with the environment $\mathcal{E}$ and corrections from a privileged expert teacher π^E. The process begins by collecting an initial set of demonstrations of history, reason, and action from the teacher $\mathcal{D} \leftarrow \{h_t^*, \rho_t^*, a_t^*\}$, and training a BC policy π_0 on this data using supervised fine-tuning (SFT).

At each iteration, the current learner policy π_i is rolled out in the environment to generate new data $\mathcal{D}_i \leftarrow (h_t, \rho_t, a_t)$. For each timestep in $\mathcal{D}_i$, LEAP computes the privileged state s_t by leveraging information available only during training. The privileged teacher is then invoked on trajectories where the agent fails, generating corrected reasoning and actions, $\tilde{\rho}_t, \tilde{a}_t \sim \pi^E(.|\rho_t, a_t, h_t, s_t)$. These corrections are used to augment the dataset $\mathcal{D}_i$. The policy π_i is subsequently updated using this augmented dataset through a no-regret learning method. This iterative process continues, refining the policy over multiple iterations until the best-performing policy is selected based on validation.

The update step in LEAP should be designed to ensure no-regret learning, a property that guarantees the policy's cumulative performance will converge to that of the best possible policy in hindsight. The most common update is to aggregate data $\bigcup \mathcal{D}_i$, and train π_i using supervised fine-tuning (SFT) on the aggregated dataset, as done in DAGGER (Ross et al., 2011). This is equivalent to Follow the Regularized Leader (FTRL), a no-regret update step. An alternate update is to treat the problem as a preference optimization problem where the corrected reason action $(\tilde{\rho}_t, \tilde{a}_t)$ is preferred over the student reason action (ρ_t, a_t). Preference optimizers like Direct Policy Optimization (DPO) (Rafailov et al., 2024) or Kahneman-Tversky Optimization (KTO) (Ethayarajh et al., 2024) optimize a preference loss while regularizing to the base policy, in this case the previous policy π_{i-1}. This is equivalent to an Online Mirror Descent (OMD) update, which is also a no-regret update. We use SFT by default but compare it against preference optimization as an ablation in the results.

3.3 Analysis

We briefly touch on the theoretical guarantees of LEAP, and refer the reader to Appendix. B for a full exposition. We define the realizability of an expert policy and show how it affects LEAP performance.

Definition 3.1 (Average Realizability Gap)**.** The average realizability gap $epsilon(\pi^E, T)$ between the privileged expert π^E and the best policy $\pi^\star$ over time horizon T is:

$$\epsilon(\pi^E, T) := \sup_{\pi} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{s_t, h_t \sim d_t^\pi} ||\pi^E(a_t|s_t) - \pi^\star(a_t|h_t))||_1 \tag{4}$$

where $||.||_1$ is the L1 distance, d_t^π is the induced distribution over state and history of *any policy* π.

Theorem 3.2 (LEAP with Privileged Expert)**.** *Running N iterations of LEAP with privileged expert π^E yields at least one policy π such that the time-average performance $\frac{1}{T}J(\pi)$ is bounded as:*

$$\frac{1}{T}J(\pi) \geq \frac{1}{T}J(\pi^E) - H(\pi^E)\left(\epsilon(\pi^E, T) + \gamma(N)\right) \tag{5}$$

where π^E is the privileged expert, $H(\pi^E)$ is the recoverability coefficient of π^E, $\epsilon(\pi^E, T)$ is the average realizability gap, and $\gamma(N)$ is the average regret of the DAGGER *update.*

The equation shows a trade-off between performance of the privileged expert (first term) and the realizability of the privileged expert (second term), which we show how to optimize in Appendix. B.

4 EXPERIMENTS

4.1 OVERVIEW OF RESULTS

We evaluate LEAP across 3 decision-making domains: ALFWorld, a text-based game, WebShop, a web navigation environment, and Intercode, an interactive coding environment. Below is a summary:

1. **LEAP improves upon base behavior cloning (BC) policy:** LEAP significantly improves the base BC policy π_0 over successive fine-tuning iterations, showing gains on AlfWorld (65.7% $\rightarrow$ 91.8%), WebShop (29.4 $\rightarrow$ 61.8), and InterCode (60.3% $\rightarrow$ 71.8%). See Tables 1, 2, Fig. 3.
2. **LEAP enables weaker student models to surpass stronger teacher models:** Over successive iterations, student models (e.g., Llama3-8B) outperform their stronger teachers (e.g., GPT-4o), despite starting with lower or comparable performance. For example, on ALFWorld, the student model (π_3 : 91.8%) surpasses the teacher (π^E : 65.7%), with similar trends on WebShop (π_2 : 61.8 vs. π^E : 58.4) and InterCode (π_3 : 71.8% vs. π^E : 71.3%). See Tables 1, 2, Fig. 3.
3. **LEAP requires a balance between using privileged information and generating realizable corrections:** Experts using minimal privileged information (π_1^E : 73.1%) generate more realizable actions, while those with excessive privileged feedback (π_5^E : 5.22%) produce unrealizable corrections. π_3^E strikes a perfect balance achieving optimal performance (π_3^E : 91%). See Fig. 5.
4. **LEAP enables LLM agents to self-improve by using their own privileged versions as experts:** LLM agents can self-improve by using their privileged versions as teachers. For example, in ALFWorld, Llama3-8B bootstraps itself through iterations, improving from 65.7% to 82.1% and finally to 91.8%. See Fig. 6.
5. **LEAP can be trained using both SFT or preference optimization:** Both SFT and preference optimization improve over the base BC policy (π_0), with SFT delivering stronger gains (68.6% $\rightarrow$ 96.4%) compared to DPO (68.6% $\rightarrow$ 74.3%) and KTO (68.6% $\rightarrow$ 87.1%). See Table 3.

4.2 DOMAIN 1: TEXT-BASED GAMES

Setup. ALFWorld (Shridhar et al., 2020b) is a text-based game where each task has a high-level objective, e.g. "heat mug and put it in cabinet" which the agent must complete by navigating and interacting with a virtual household through text commands, e.g. go to shelf 1, pick up mug 2, etc. To solve it, the agent must plan subgoals, track progress, and search for objects efficiently (e.g. mugs are more likely to be on shelves or cabinets). Each task has 30 timesteps. There are 6 categories of tasks, training dataset of 3257 games and two evaluation datasets: 139 in-distribution games and 134 out-of-distribution games. We compare against a prior work BUTLER (Shridhar et al., 2020b) and ReAct (Yao et al., 2022b); the original ReAct paper used text-davinci-002 with task-dependent in-context examples. We add ReAct with stronger models such as gpt-4o[1], claude[2], and gemini[3], and a general instruction prompt. For LEAP, we LoRA (Hu et al., 2021) fine-tune Llama3-8B (Dubey et al., 2024) model using gpt-4o as the privileged expert teacher for 4 iterations. The privileged information for each game contains the essential objects to solve the task, object locations, and the optimal actions. See Appendix. C.2, D.1 for full details and prompts.

Results. Table 1 shows that LEAP significantly outperforms all baselines, with the best policy achieving 91.8% success rate on out-of-distribution tasks. Iteration 1 of LEAP has the biggest performance gain (65.7% $\rightarrow$ 91.0%), **leading to a student policy π_1 that surpasses the teacher gpt-4o** with higher success rate (91.8% $>$ 65.7%) and lower actions (11.3 $<$ 20.2). Note that performance improvements are not monotonic over iterations of LEAP, (65.7% $\rightarrow$ 91.0% $\rightarrow$ 83.6% $\rightarrow$ 91.8%), however, all policies are better than the BC policy π_0. The regression from π_1 to π_2 are mainly from HEAT and LOOK tasks, which gets corrected immediately in the next iteration π_3. Finally. we note that LEAP policies generalize well from in-distribution to out-of-distribution tasks with only a slight dip in performance for the final policy π_3 (94.2% $\rightarrow$ 91.8%).

[1] https://platform.openai.com/docs/models [2] https://docs.anthropic.com/en/docs/about-claude/models
[3] https://ai.google.dev/gemini-api/docs/models/gemini

Method	All tasks		PICK tasks	CLEAN tasks	HEAT tasks	COOL tasks	LOOK tasks	PICK 2 tasks
	%suc↑	#act↓	%suc↑	%suc↑	%suc↑	%suc↑	%suc↑	%suc↑
BUTLER [1]	35.0	-	50.0	74.0	83.0	91.0	39.0	65.0
ReAct few-shot [2]	57.0	-	65.0	39.0	83.0	76.0	55.0	24.0
Autogen gpt-3.5 [3]	77.0	-	-	-	-	-	-	-
ExpeL gpt-3.5 [4]	59.0	-	-	-	-	-	-	-
Reflexion gpt-3 [5]	88.0	-	75.0	90.3	91.3	90.5	88.9	94.1
AdaPlanner gpt-3 [6]	91.7	-	100.0	96.7	95.6	100.0	100.0	47.0
ReAct gpt-4o	65.7	20.2	91.7	35.5	56.5	52.4	100.0	76.5
ReAct gpt-4o-mini	29.9	25.5	33.3	25.8	17.4	14.3	66.7	29.4
ReAct claude-3.5-sonnet	76.1	19.0	95.8	61.3	60.9	81.0	88.9	76.5
ReAct claude-3.5-haiku	16.4	27.2	33.3	9.7	8.7	9.5	38.9	0.0
ReAct gemini-1.5-flash	19.4	26.3	41.7	12.9	13.0	19.0	16.7	11.8
LEAP Llama3-8B	0.7	29.8	0.0	0.0	0.0	4.8	0.0	0.0
LEAP Llama3-8B π_0	65.7	18.6	66.7	74.2	73.9	66.7	66.0	35.3
LEAP Llama3-8B π_1	91.0	11.9	83.3	90.3	91.3	95.2	94.4	94.1
LEAP Llama3-8B π_2	83.6	13.1	87.5	90.3	73.9	95.2	66.7	82.4
LEAP Llama3-8B π_3	91.8	11.3	87.5	93.5	91.3	90.5	94.4	94.1

Method	All tasks		PICK tasks	CLEAN tasks	HEAT tasks	COOL tasks	LOOK tasks	PICK 2 tasks
	%suc↑	#act↓	%suc↑	%suc↑	%suc↑	%suc↑	%suc↑	%suc↑
BUTLER [1]	40.0	-	69.0	67.0	88.0	76.0	69.0	54.0
ReAct gpt-4o	54.3	22.9	91.4	33.3	31.2	12.0	84.6	66.7
ReAct gpt-4o-mini	40.0	26.1	11.1	12.5	4.0	53.8	20.8	20.8
LEAP Llama3-8B	2.9	29.5	8.6	0.0	0.0	0.0	7.7	0.0
LEAP Llama3-8B π_0	68.6	16.9	88.6	51.9	56.2	80.0	76.9	50.0
LEAP Llama3-8B π_1	96.4	10.5	100.0	100.0	100.0	92.0	92.3	91.7
LEAP Llama3-8B π_2	90.0	11.4	94.3	92.6	87.5	88.0	92.3	83.3
LEAP Llama3-8B π_3	94.2	10.3	100.0	96.3	100.0	87.5	100.0	83.3

Table 1: **AlfWorld Evaluation. (top)** 136 out-of-distribution games and **(bottom)** 140 in-distribution (max 30 actions). Baseline comparisons include [1] BUTLER (Shridhar et al., 2020a), [2] ReAct few-shot (Yao et al., 2022b), [3] Autogen (Shridhar et al., 2020a), [4] ExpeL (Zhao et al., 2024). Note [5] Reflexion (Shinn et al., 2023) and [6] AdaPlanner (Sun et al., 2024) make multiple attempts on the same test task, while we do not. We also add our own REACT instruction prompt with different models. LEAP with an 8B model across iterations (π_1, π_2, π_3) outperforms the stronger teacher ReAct gpt-4o.

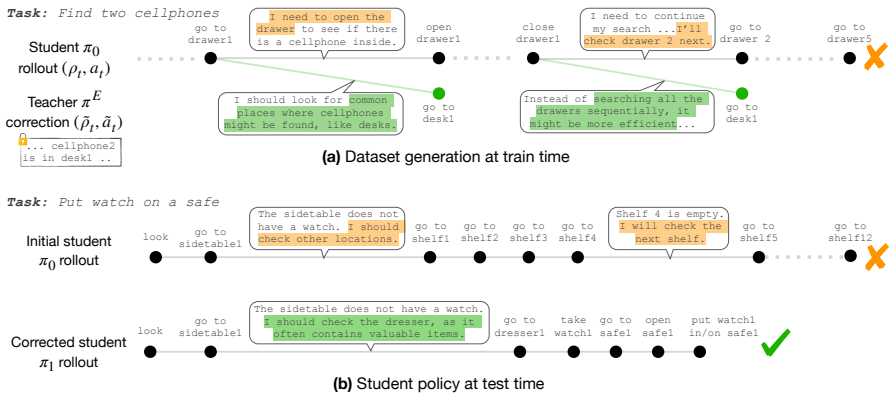
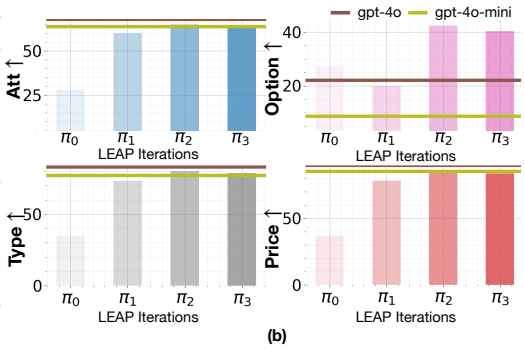


Figure 2: **ALFWorld Training and Testing for LEAP. (a) Training:** Student policy π_0 rolled out on training task to generate reason and actions, e.g. it fails to cell phone because it inefficiently searches all drawers. Expert teacher π^E uses privileged information (cellphone in desk1) to generate general corrected reason actions that don't reveal privileged information (cellphones commonly found in desks, more efficient to search desks than drawers). **(b) Testing:** π_0 fails to find a watch as it inefficently explores shelves one by one. π_1 learns a more efficient exploration policy, prioritizing areas like sidetables and dressers, solving the task quickly.

Fig. 2 (a) shows an example of LEAP training and testing. A typical failure mode of π_0 is that it searches for items inefficiently, e.g. looking at shelves or drawers or cabinets one by one until time runs out. The expert teacher leverages the privileged state of where objects are to show the student how to search efficiently without revealing the object location, e.g. it's more efficient to search desks where cellphones are likely to be than drawers sequentially one after another. Training on this correction data, π_1 *generalizes this reasoning to new test-time tasks*, e.g. putting a watch in a safe. Fig. 2 (b) shows that while π_0 inefficiently searches shelves, π_1 searches likely locations for the watch, first the sidetable and then the dresser to solve the task. The main performance gains of LEAP over gpt-4o come from such efficient exploration. gpt-4o searches likely locations, however, its prior is from internet data and isn't necessarily aligned with the statistics of the ALFWorld environment. In contrast, LEAP leverages privileged feedback to learn this. See Appendix C for more results.

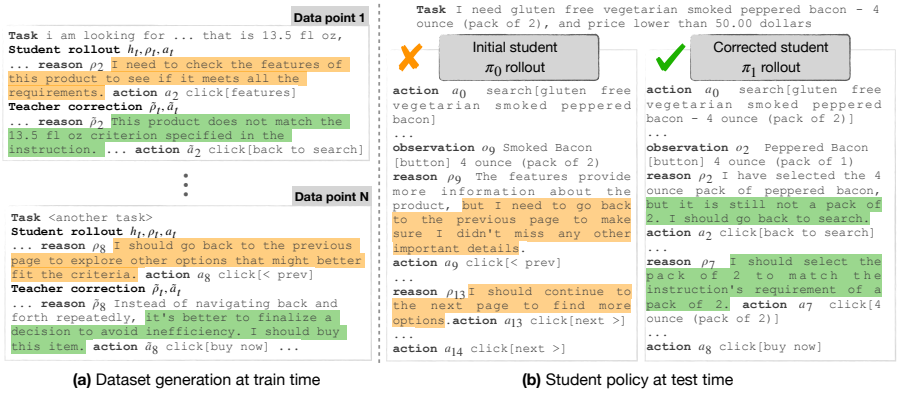
Method	score↑	#act↓
IL [1]	59.9	-
ReAct gpt-4o	58.4	8.0
ReAct gpt-4o-mini	51.0	11.3
ReAct claude-3-sonnet	44.4	11.4
ReAct gemini-flash	46.3	8.6
LEAP Llama3-8b	1.2	29.9
LEAP Llama3-8b π_0	29.4	21.1
LEAP Llama3-8b π_1	51.8	16.7
LEAP Llama3-8b π_2	61.8	11.6
LEAP Llama3-8b π_3	60.7	11.9

(a)



(b)

Figure 3: **WebShop Evaluation. (a)** Overall score↑ and #act↓ on 500 test tasks (max 30 actions). **(b)** Performance of LEAP over iterations on 4 different score components. Baseline comparisons include [1] IL (Yao et al., 2022a) and our ReAct instruction prompt with different models. LEAP with an 8B model across iterations (π_2, π_3) outperforms the stronger teacher ReAct gpt-4o.



(a) Dataset generation at train time (b) Student policy at test time

Figure 4: **WebShop Training and Testing for LEAP. (a) Training:** Teacher policy generates corrections on student rollout to backtrack when the product does not fit criteria or to commit to a product when it does. **(b) Evaluation:** π_0 fails to solve the task since it continues to search page after page even after discovering a good product. π_1 learns when to backtrack and when to commit to a product to solve the task in time.

4.3 Domain 2: Web Agents

Setup. WebShop (Yao et al., 2022a) is an online shopping environment where each task is a shopping request from a user ("I need a gluten-free bacon, 4 oz pack of 2, priced under $50") that the agent has to solve through web interactions (search["gluten-free bacon"] or click options). Performance is evaluated using an overall score with 4 components on whether: attributes match (Att), options match (Opt), product types match (Type) and price matches (Price). Note that the score is different from success rate, and assigns partial credit for matching 1 or more components. Each task has 30 timesteps. The training dataset has 12086 tasks, the test dataset has 500 tasks. We compare against ReAct (Yao et al., 2022b) with different base models like GPT-4o and GPT-4o-mini. We also compare against IL baseline in WebShop (Yao et al., 2022a), which trains two models, one for search (from a list of keywords) and one for clicking, thus simplifying the problem. We LoRA fine-tune Llama3-8B model using gpt-4o as the privileged expert teacher for 4 iterations. The privileged information contains product attributes, product option, price constraint, and an example product that would satisfy all criteria. See Appendix. C.3, D.2 for full details and prompts.

Results. Fig. 3 (a) shows that LEAP outperforms all baselines, with the best policy (π_2) achieving 61.4 score. Iteration 1 has the biggest performance gain ($29.4 \rightarrow 51.8$) followed by iteration 2 ($51.8 \rightarrow 61.8$). This **leads to a student policy π_2 that surpasses the teacher GPT-4o** with a higher score ($61.8 > 58.4$). Similar to ALFWorld, while performance improvements are not monotonic, both π_2, π_3 remain above the teacher gpt-4o performance. Fig. 3 (b) shows how LEAP improves on all 4 components of the score. The biggest improvements between LEAP and gpt-4o is in Opt component ($42.4 > 22.1$), indicating gpt-4o fails to select a product with the right option often.

Fig.4 (a) shows LEAP's correction of π_0 and π_1 using training data. The teacher leverages privileged information (e.g., product attributes or options) to guide the student in recognizing when to backtrack

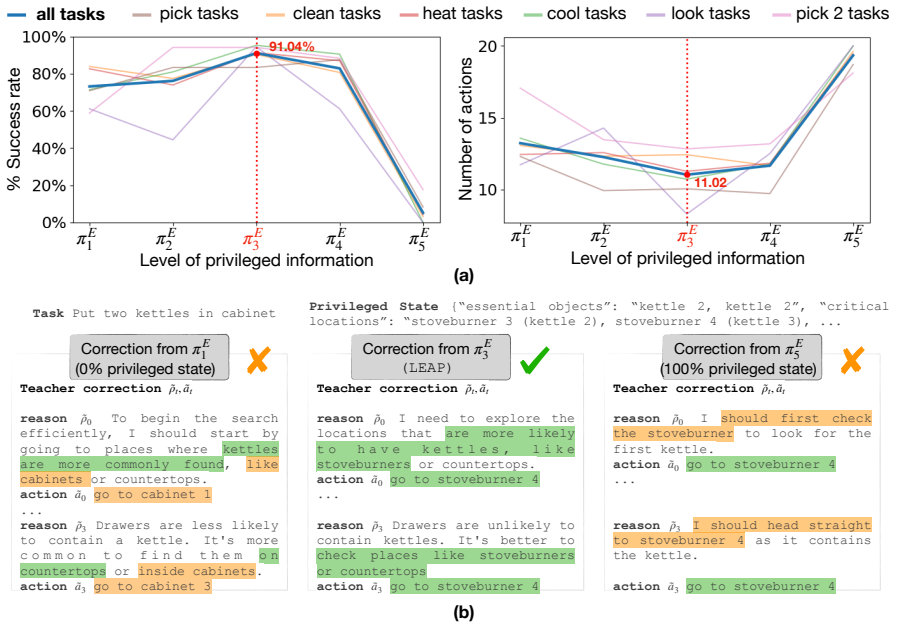


Figure 5: **Privileged Information vs Realizability.** **(a)** Performance of 5 policies trained with experts with varying levels of privileged information on ALFWorld, peaking for expert π_3^E. **(b)** Examples of corrections from experts $\pi_1^E, \pi_3^E, \pi_5^E$. π_1^E generates realizable reason action but predicts wrong action. π_5^E predicts correction action, but produces unrealizable reason action that contains privileged information. π_3^E strikes a perfect balance.

on mismatched items and when to commit to a suitable product. Training on this correction data produces policy π_2, which balances backtracking and committing effectively. Fig.4 (b) highlights π_0's failure to commit, endlessly cycling through pages, whereas π_2 successfully backtracks and selects a suitable item to complete the task.

4.4 Domain 3: Interactive Coding

Setup. Intercode Bash (Yang et al., 2024) is an interactive coding environment, where the agent receives a task that involves operations on a filesystem, e.g. "Find all text files and write their names to a single file", and must interact with Bash to solve it and submit a final command. Intercode uses NL2Bash (Lin et al., 2018) to create 4 datasets. We use the first 2 for training, and the next 2 for testing. We limit time horizon to 10. We compare against `ReAct` (Yao et al., 2022b) with `gpt-4o` and `gpt-4o-mini`. We also compare against the top 3 entries on the Intercode leaderboard, which are the `TryAgain` baselines from (Yang et al., 2024) with `gpt-4`, `gpt-3.5` and CodeLlama-34B-INST (Roziere et al., 2023) respectively. We LoRA fine-tune both Llama-3.1-8B model and Llama-3.1-70B model, each for 2 iterations, using `gpt-4o` as the privileged expert teacher. The privileged information is the ground truth Bash command.

Method	%succ↑	#act↓
gpt-4 [1]	48.5⋆	-
gpt-3.5 [1]	46.5⋆	-
CodeLlama-34B-INST [1]	36.0⋆	-
ReAct gpt-4o	71.3	3.8
ReAct gpt-4o-mini	43.4	6.0
LEAP Llama-3.1-8b π_0	41.4	6.6
LEAP Llama-3.1-8b π_1	49.9	5.5
LEAP Llama-3.1-70b π_0	60.3	4.8
LEAP Llama-3.1-70b π_1	71.8	5.1

Table 2: **Intercode Bash Evaluation.** Overall success rate and actions on test data (max 10 actions). Baselines from [1] (Yang et al., 2024) evaluated on ⋆(train + test) data.

Results. Table 2 shows that `LEAP` trained with 70B policy outperforms all baselines, with policy π_1 achieving 71.8% success rate to match the `gpt-4o` expert teacher (71.8% > 71.3%). We find that on this coding benchmark, the `gpt-4o` expert is a very strong baseline. `LEAP` trained with 8B models improve over the BC policy, but falls short of the `gpt-4o` expert teacher. These results show that if the expert teacher is already a strong baseline, even after leveraging privileged information, it might be hard for the student to outperform the teacher.

4.5 What is the trade-off between Privileged Information and Realizability?

We study the tradeoff between how much privileged state the expert uses vs the realizability of their corrections on ALFworld. We create 5 different privileged experts $\pi_E^1, \ldots, \pi_E^5$ by varing the

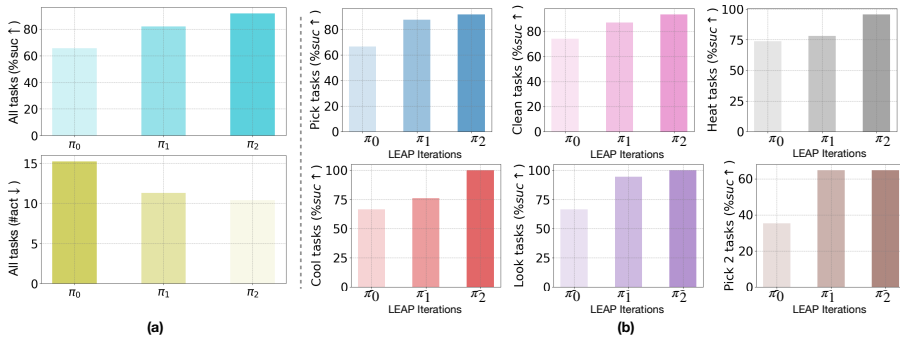


Figure 6: **Self Correction.** LEAP self-improves a Llama3-8B model that is used as both student and teacher. **(a)** Overall success rate and num actions. **(b)** Category-wise breakdown of success rate.

prompt to instruct them to use increasing amounts of privileged state, e.g. for π_E^2 it says "Use this information sparingly for your correction," but for π_E^4 it says "feel free to include information from the privileged state". We run 1 iteration of LEAP to generate updated $\pi^1, \ldots, \pi^5$. Fig. 5 (a) shows that success rate initially rises from π^1 to π^3, but then falls off sharply till π^5. Num actions has a similar trend. This validates the hypothesis that there is indeed an optimal tradeoff. Interestingly, π_5 is much worse than π_1, showing that it's more important to be realizable than to provide optimal yet unrealizable corrections. Fig. 5 (b) shows the corrections from $\pi_1^E, \pi_3^E, \pi_5^E$. π_1^E generates reasoning that is perfectly realizable to the student, but since it has no privileged state the actions are not optimal. In contrast, π_5^E predicts the correct action, but blatantly reveals the privileged information in the reason, being unrealizable to the student. π_3^E strikes a perfect balance where it offers the correct action, but offers a reason that is very much realizable, i.e. kettles are likely to be in stoveburners.

4.6 Can LEAP be used to self-correct a student?

We test the hypothesis that LEAP should enable a model to self-improve by using its privileged version as the expert. On ALFWorld, we fine-tune Llama3-8B initializing with the BC policy π_0 and running 2 iterations of self-improvement. Fig. 6 shows that LEAP is able to significantly self-improve policies over iterations ($65.7\% \rightarrow 82.1\% \rightarrow 91.8\%$), with the final policy **matching the best policy when using GPT-4o as a teacher**. We see uniform improvements across all categories, with some being improved more from $\pi_0 \rightarrow \pi_1$ and others from $\pi_1 \rightarrow \pi_2$. We conclude that for some environments, like ALFworld, the privileged state does the heavy lifting, i.e. performance improvements happen at a similar rate to using a strong teacher vs using the model itself as a teacher. We note that this need not hold for all environments, e.g. for some a strong teacher is required to extract the optimal reason and action from the privileged state.

4.7 How does SFT compare to Preference Optimization?

We study the effect of different update methods for LEAP: SFT vs Preference Optimization. Our initial hypothesis is that KL regularized preference optimization should also result in performance improvement without needing to aggregate datasets. We run 1 iteration of DPO with different regularization β values. Table. 3 shows that while DPO improves upon the base π_0 policy, the improvement is far smaller than SFT for both $\beta = 0.1, 0.01$. One explanation is that SFT is far more aggressive in using the correction data compared to preference optimization, i.e., SFT deems the corrected reason action to be better than any alternative. To test this, we used a different preference optimizer KTO, that operates with unpaired preference data and set weight $\lambda_D = 1.0, \lambda_D = 0.0$, effectively emulating SFT-like behavior. KTO with these settings has a significantly high performance of 88.1% close to SFT (91.0%).

	Method	%succ↑	#act↓
in-dist.	LEAP π_0	68.6	16.9
	LEAP DPO (β : 0.1) π_1	74.3	15.7
	LEAP DPO (β : 0.01) π_1	74.3	15.6
	LEAP KTO (λ_U : 0.0) π_1	87.1	11.6
	LEAP SFT π_1	96.4	10.5
out-dist.	LEAP π_0	65.7	18.6
	LEAP DPO (β : 0.1) π_1	71.6	17.9
	LEAP DPO (β : 0.01) π_1	70.9	17.7
	LEAP KTO (λ_U : 0.0) π_1	88.1	11.7
	LEAP SFT π_1	91.0	11.9

Table 3: **SFT vs Preference:** LEAP with SFT vs DPO vs KTO on ALFWorld.

5 Related Work

Imitation Learning and Privileged Information. A powerful paradigm in machine learning is leveraging privileged information (Vapnik et al., 2015)—data available only during training and

inaccessible at test time. This concept has been transformative in robotics, where a simulated expert policy with full state supervises a learner that only perceives sensor observations, such as navigation (Zhang et al., 2016; Uppal et al., 2024), self-driving (Chen et al., 2020), manipulation (Chen et al., 2023a; Hu et al., 2024), and legged locomotion (Lee et al., 2020; Kumar et al., 2020).

A fundamental challenge in utilizing privileged information is the realizability gap—where actions suggested by an expert may be infeasible for the learner to predict. This gap often leads to spurious correlations, manifesting as the "latching effect" where learners repetitively predict the same action, commonly observed in autonomous driving (Muller et al., 2005; Kuefler et al., 2017; Bansal et al., 2018; Codevilla et al., 2019) and language models (Bengio et al., 2015; Holtzman et al., 2019; Wang & Sennrich, 2020; Ortega et al., 2021). While off-policy methods (Wen et al., 2020; 2022) have been proposed, (Swamy et al., 2022) show that online interaction with the environment is necessary.

Interactive imitation learning methods such as DAGGER (Ross et al., 2011), where experts provide corrective actions during the learner's rollouts, both theoretically and empirically lead to very effective policies in this regime (Choudhury et al., 2018). These methods operate under the assumption of asymptotic realizability (Swamy et al., 2022), where the learner's policy can eventually match the expert's actions as the episode progresses. However, when this assumption fails—often due to the partial observability of the task or model capacity—suboptimal performance ensues, particularly for LLM agents, as demonstrated by our experiments (Sec. 4.5). While recent approaches resort to exploration or RL (Walsman et al., 2022; Tennenholtz et al., 2021; Nguyen et al., 2022; Weihs et al., 2021), these are intractable to extend to LLM agents. Our work provides a practical recipe for LLM agents that optimally balances the trade-off between using privileged information and maintaining realizability, and further shows how such an expert can be used to enable policy self-improvement.

Learning from AI Feedback. A scalable method for aligning language model using AI feedback (Lee et al., 2023). Recent works have explored different types of feedback, from self-generated corrections to those provided by external AI models or environments (Pan et al., 2023). One body of work focuses on self-correction, where LLMs refine their responses based on feedback from their own outputs (Bai et al., 2022; Madaan et al., 2024). This can occur either at train-time, where LLMs are fine-tuned to generate corrected responses (Bai et al., 2022; Ganguli et al., 2023), or at test-time, where the model improves its response interactively (Madaan et al., 2024). However, while simple in principle, Huang et al. (2024) shows that without external sources of feedback, LLMs often struggle to accurately judge the correctness of their reasoning, limiting the effectiveness of purely self-generated corrections.

Another cluster of work explores feedback from external sources. Some approaches use feedback from strong models like GPT-4 acting as judges (Zheng et al., 2023) or critics trained on high-quality curated data (Wang et al., 2023; Paul et al., 2023). Similarly, An et al. (2024) and Guo et al. (2024) use GPT-4 to provide corrections and preference feedback. Welleck et al. (2023); Qu et al. (2024) trains LLMs to self-correct from errors. Others leverage environmental feedback or external tools to guide improvements. Chen et al. (2023b); Olausson et al. (2023) utilize execution feedback to improve code generation, and (Gou et al., 2023) employs external tools. Notably, Reflexion (Shinn et al., 2023) leverages feedback from external environments to generate self-reflections for agents, although it assumes multiple attempts in the same environment, which our approach does not. Our work introduces the use of privileged information in decision-making tasks for LLM agents, a technique not yet fully explored in the literature. This also enables LLM agents to self-improve using privileged information, setting it apart from existing self-improvement methods that rely on iterative solution generation and evaluation (Zelikman et al., 2022; Pang et al., 2024; Hosseini et al., 2024).

6 LIMITATIONS

We proposed LEAP, an iterative fine-tuning framework that improves LLM agent performance using privileged AI feedback, enabling weaker models to surpass stronger expert teachers and allowing agents to self-improve. However, LEAP has notable limitations. First, generating interaction rollouts is time-consuming, particularly in complex environments requiring multi-step reasoning. Each rollout requires simulating the environment and generating complete trajectories, that becomes computationally expensive across multiple fine-tuning iterations. Second, corrective feedback from the expert often results in lengthy output tokens. As trajectory lengths grow, balancing the effectiveness of feedback with its verbosity becomes challenging. Exploring methods to distill and summarize expert feedback at critical points is an interesting direction for future work.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our findings, we have made several efforts, as outlined below. Detailed information and resources are provided in the main paper, appendix, and supplementary materials:

1. **Open source code.** We open source the `LEAP` code that includes the Python package implementing `LEAP`, along with all prompts, configuration files, and training scripts necessary to reproduce our results.
2. **Experimental details.** We provide detailed descriptions of our experiments in the main paper (Section 4), with additional details on the setup, hyperparameters, and prompts found in Appendix C, D. We use open-source datasets and models in our experiments, with citations and links provided in the main paper.
3. **Analysis.** The key results of our theoretical analysis are presented in the main paper (Section 3.3), with full derivations, proofs, and assumptions detailed in Appendix B.

REFERENCES

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. Learning from mistakes makes LLM better reasoner. *arXiv preprint arXiv:2310.20689*, 2024.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pp. 66–75. PMLR, 2020.

Tao Chen, Megha Tippur, Siyang Wu, Vikash Kumar, Edward Adelson, and Pulkit Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84): eadc9244, 2023a.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023b.

Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Data-driven planning via imitation learning. *The International Journal of Robotics Research*, 37(13-14):1632–1672, 2018.

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9329–9338, 2019.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

Deep Ganguli, Amanda Askell, Nicholas Schiefer, Thomas I Liao, Kamilė Lukošiūtė, Anna Chen, Anna Goldie, Azalia Mirhoseini, Catherine Olsson, Danny Hernandez, et al. The capacity for moral self-correction in large language models. *arXiv preprint arXiv:2302.07459*, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, et al. Direct language model alignment from online ai feedback. *arXiv preprint arXiv:2402.04792*, 2024.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*, 2024.

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

Edward S Hu, James Springer, Oleh Rybkin, and Dinesh Jayaraman. Privileged sensing scaffolds reinforcement learning. *arXiv preprint arXiv:2405.14853*, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 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.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 267–274, 2002.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *2017 IEEE intelligent vehicles symposium (IV)*, pp. 204–211. IEEE, 2017.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

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. *arXiv preprint arXiv:1802.08979*, 2018.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann Cun. Off-road obstacle avoidance through end-to-end learning. *Advances in neural information processing systems*, 18, 2005.

Hai Nguyen, Andrea Baisero, Dian Wang, Christopher Amato, and Robert Platt. Leveraging fully observable policies for learning under partial observability. *arXiv preprint arXiv:2211.01991*, 2022.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Demystifying gpt self-repair for code generation. *arXiv preprint arXiv:2306.09896*, 2023.

Pedro A Ortega, Markus Kunesch, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Joel Veness, Jonas Buchli, Jonas Degrave, Bilal Piot, Julien Perolat, et al. Shaking the foundations: delusions in sequence models for interaction and control. *arXiv preprint arXiv:2110.10819*, 2021.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*, 2023.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733*, 2024.

Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*, 2023.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *arXiv preprint arXiv:2407.18219*, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Stéphane Ross, Geoffrey Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Artificial Intelligence and Statistics (AISTATS)*, 2011.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning.(2023). *arXiv preprint cs.AI/2303.11366*, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition*, 2020a.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020b.

Paloma Sodhi, SRK Branavan, Yoav Artzi, and Ryan McDonald. Step: Stacked llm policies for web actions. In *Conference on Language Modeling (COLM)*, 2024.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Gokul Swamy, Sanjiban Choudhury, J Bagnell, and Steven Z Wu. Sequence model imitation learning with unobserved contexts. *Advances in Neural Information Processing Systems*, 35:17665–17676, 2022.

Guy Tennenholtz, Assaf Hallak, Gal Dalal, Shie Mannor, Gal Chechik, and Uri Shalit. On covariate shift of latent confounders in imitation and reinforcement learning. *arXiv preprint arXiv:2110.06539*, 2021.

Shagun Uppal, Ananye Agarwal, Haoyu Xiong, Kenneth Shaw, and Deepak Pathak. Spin: Simultaneous perception interaction and navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18133–18142, 2024.

Vladimir Vapnik, Rauf Izmailov, et al. Learning using privileged information: similarity control and knowledge transfer. *J. Mach. Learn. Res.*, 16(1):2023–2049, 2015.

Aaron Walsman, Muru Zhang, Sanjiban Choudhury, Dieter Fox, and Ali Farhadi. Impossibly good experts and how to follow them. In *The Eleventh International Conference on Learning Representations*, 2022.

Chaojun Wang and Rico Sennrich. On exposure bias, hallucination and domain shift in neural machine translation. *arXiv preprint arXiv:2005.03642*, 2020.

Tianlu Wang, Ping Yu, Xiaoqing Ellen Tan, Sean O'Brien, Ramakanth Pasunuru, Jane Dwivedi-Yu, Olga Golovneva, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. Shepherd: A critic for language model generation. *arXiv preprint arXiv:2308.04592*, 2023.

Luca Weihs, Unnat Jain, Iou-Jen Liu, Jordi Salvador, Svetlana Lazebnik, Aniruddha Kembhavi, and Alex Schwing. Bridging the imitation gap by adaptive insubordination. *Advances in Neural Information Processing Systems*, 34:19134–19146, 2021.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*, 2023.

Chuan Wen, Jierui Lin, Trevor Darrell, Dinesh Jayaraman, and Yang Gao. Fighting copycat agents in behavioral cloning from observation histories. *Advances in Neural Information Processing Systems*, 33:2564–2575, 2020.

Chuan Wen, Jianing Qian, Jierui Lin, Jiaye Teng, Dinesh Jayaraman, and Yang Gao. Fighting fire with fire: avoiding dnn shortcuts through priming. In *International Conference on Machine Learning*, pp. 23723–23750. PMLR, 2022.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022a.

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*, 2022b.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 528–535. IEEE, 2016.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Appendix

Table of Contents

A Broader Impacts **16**

B Analysis **16**
B.1 Trade-off between realizability and privileged information 16
B.2 Practical approximation for constrained privileged expert 20

C Experimental details **20**
C.1 Hyper-parameters 20
C.2 ALFWorld 22
C.3 WebShop 22
C.4 Intercode 23

D Prompts **23**
D.1 AlfWorld 23
D.2 WebShop 27
D.3 InterCode 31

A BROADER IMPACTS

Technological Impact. By iteratively fine-tuning LLMs through privileged expert feedback, LEAP can significantly enhance AI performance across diverse applications, such as virtual assistants and dialogue agents handling complex decision-making tasks. This approach allows LLMs to autonomously refine their decision-making, reducing the need for human intervention, streamlining automation, and improving efficiency in both consumer and enterprise environments.

Additionally, LEAP 's ability to enable weaker models to outperform stronger ones democratizes AI access, making state-of-the-art models more accessible to organizations with limited computational resources. This also opens new possibilities for improving model generalization in settings where perfect information is unavailable. However, the reliance on privileged feedback during training underscores the importance of further research into model transparency and interpretability. As these models grow more autonomous, ensuring their decision-making remains understandable to users will be critical for building trust and fostering widespread adoption.

Societal Impact. Equipping LLMs with iterative self-improvement capabilities unlocks a range of societal benefits. In sectors such as education, customer service, and software development, these AI systems can assist professionals in decision-making, automate routine tasks, and enhance overall productivity. By continuously improving their performance, these models can reduce the cognitive load on workers, allowing them to focus on more complex, creative, and value-added activities.

Despite these advantages, there are ethical concerns that must be addressed. As LLMs become capable of self-improvement, ensuring their trustworthiness and accountability becomes critical. Without proper safeguards, the autonomous refinement of AI could lead to unpredictable behaviors, making it harder to ensure alignment with human values. Furthermore, there is a risk that self-improving AIs could be misused for malicious purposes, such as automating harmful tasks or spreading disinformation. To mitigate these risks, strict ethical guidelines and robust oversight is necessary to ensure deployment of such systems serves societal good and prevents misuse.

B ANALYSIS

B.1 TRADE-OFF BETWEEN REALIZABILITY AND PRIVILEGED INFORMATION

We now analyze the performance of a policy imitating an expert with access to privileged information. The performance depends both on the performance of the expert policy and the realizability gap between the learner and the expert. We then derive a constrained privileged expert to better trade off these two terms and provide a practical approximation of this constrained expert. For notational simplicity, we assume that a_t represents both reason and action. We follow the derivation in (Swamy et al., 2022) but simplify for our setting.

Let $\pi \in \Pi$ be the student policy that we are training. We assume the class of policies Π is rich enough to contain all history-based policies. Let $J(\pi)$ be the performance, i.e. the cumulative reward of a learner policy that selects actions based on the history $a_t = \pi(h_t)$. Let $J(\pi_E)$ be the performance of the expert policy that selects actions based on the privileged state $a_t = \pi^E(s_t)$. Let $\tau \sim \pi$ be a *privileged rollout* which is the sequence of observation, actions, and privileged state upon executing the policy. We represent the rollout as $\tau = \{(h_t, s_t, a_t)\}$. Note that such a privileged rollout is only accessible at train time.

We begin by defining the average imitation gap between an expert policy π^E and learner π.

Definition B.1 (Average Imitaiton Gap)**.** The average imitation gap of a learner π over a horizon T:

$$\text{AIG}(\pi, T) := \frac{1}{T}\left(J(\pi^E) - J(\pi)\right) = \mathbb{E}_{\tau\sim\pi^E}\left[\frac{1}{T}r(s_t, a_t)\right] - \mathbb{E}_{\tau\sim\pi}\left[\frac{1}{T}r(s_t, a_t)\right] \tag{6}$$

We next define the realizability gap between the privileged expert and the best policy in policy class as the L1 distance (or equivalently Total-Variation (TV) distance) between the two policies on the distribution of states and histories induced by *any policy*.

Definition B.2 (Average Realizability Gap)**.** The average realizability gap over time between the privileged expert π^E and the best policy in the policy class $\pi^\star$ is:

$$\epsilon(\pi^E, T) := \sup_{\pi} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{s_t, h_t \sim d_t^\pi} ||\pi^E(a_t|s_t) - \pi^\star(a_t|h_t))||_1 \tag{7}$$

We next define the recoverability of the expert. Let $A^{\pi^E}(s,a) = Q^{\pi^E}(s,a) - V^{\pi^E}(s)$ be the advantage of a state action (s,a) with respect to the expert policy π^E. A high negative advantage means that the mistaken action at a state results in a high value difference from the action the expert would have taken. Recoverability is the worst case advantage.

Definition B.3 (Recoverability Coefficient of the Privileged Expert)**.** The recoverability of the expert policy π^E is the max advantage of the expert policy

$$H(\pi^E) := \max_{s,a} ||A^{\pi^E}(s,a)|| \tag{8}$$

We are now ready to state the main theorem of LEAP.

Theorem B.4 (LEAP with Privileged Expert)**.** *Running N iterations of LEAP with privileged expert π^E yields at least one policy π such that*

$$\frac{1}{T} J(\pi) \geq \frac{1}{T} J(\pi^E) - H(\pi^E) \left(\epsilon(\pi^E, T) + \gamma(N) \right) \tag{9}$$

where π^E is the privileged expert, $H(\pi^E)$ is the recoverability coefficient of π^E, $\epsilon(\pi^E, T)$ is the average realizability gap, and $\gamma(N)$ is the average regret of the DAGGER *update.*

The first term is the performance of the privileged expert and the second term is the average realizability of the privileged expert. The more privileged information the expert policy π^E leverages the better $J(\pi^E)$ and the lower recoverability coefficient $H(\pi^E)$. However, the realizability gap $\epsilon(\pi^E, T)$ grows larger.

Proof. The proof relies on two important results.

The first is the Performance Difference Lemma (PDL) (Kakade & Langford, 2002) which states that the performance difference between any two policies can be expressed as the sum of advantages.

$$J(\pi) - J(\pi') = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^\pi} \left[\sum_{a_t} A^{\pi'}(s_t, a_t) \pi(a_t|s_t) \right] \tag{10}$$

where $s_t \sim d_t^\pi$ is the induced state distribution by π, and $A^{\pi'}(s_t, a_t) = Q^{\pi'}(s_t, a_t) - V^{\pi'}(s_t)$ is the advantage w.r.t. π'.

To adapt the equality to our setting where two policies operate on different input spaces, we note that the equality to different input spaces, e.g. state s_t or history h_t or joint state-history s_t, h_t. We express in terms of the joint state-history s_t, h_t space and write the inequality as:

$$\begin{aligned}
J(\pi) - J(\pi^E) &= \sum_{t=1}^{T} \mathbb{E}_{s_t, h_t \sim d_t^\pi} \left[\sum_{a_t} A^{\pi^E}(s_t, h_t, a_t) \pi(a|s_t, h_t) \right] && \text{(From (10))} \\
&= \sum_{t=1}^{T} \mathbb{E}_{s_t, h_t \sim d_t^\pi} \left[\sum_{a_t} A^{\pi^E}(s_t, h_t, a_t) \pi(a|h_t) \right] && (\pi \text{ depends only on } h_t) \\
&= \sum_{t=1}^{T} \mathbb{E}_{s_t, h_t \sim d_t^\pi} \left[\sum_{a_t} A^{\pi^E}(s_t, a_t) \pi(a|h_t) \right] && (\pi^E \text{ depends only on } s_t) \\
&= \sum_{t=1}^{T} \mathbb{E}_{s_t, h_t \sim d_t^\pi} \left[\sum_{a_t} A^{\pi^E}(s_t, a_t) \left(\pi(a|h_t) - \pi^E(a_t|s_t) \right) \right] && \text{as } \left(\sum_{a_t} A^{\pi^E}(s_t, a_t) \pi^E(a_t|s_t) = 0 \right)
\end{aligned} \tag{11}$$

Rearranging, we get the following form of PDL which we use in the main proof:

$$J(\pi^E) - J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t,h_t\sim d_t^\pi}\left[\sum_{a_t} A^{\pi^E}(s_t,a_t)\left(\pi^E(a_t|s_t) - \pi(a_t|h_t)\right)\right] \tag{12}$$

The second result we use us from interactive imitation learning DAGGER (Ross et al., 2011) that reduces imitation learning to no-regret online learning. DAGGER shows that with π^E as the expert teacher guarantees that after N iterations, it will find at least one policy

$$\mathbb{E}_{s,h\sim d^\pi}||\pi(.|h) - \pi^E(.|s)||_1 \leq \mathbb{E}_{s,h\sim d^\pi}||\pi^\star(.|h) - \pi^E(.|s)||_1 + \gamma(N) \tag{13}$$

where $\gamma(N)$ is the average regret, and d^π is the time average distribution of states and history induced by policy π.

We now prove the main result

$$\begin{aligned}
\text{AIG}(\pi,T) &= \frac{1}{T}\left(J(\pi^E) - J(\pi)\right) \\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d_t^\pi}\left[\sum_{a_t} A^{\pi^E}(s_t,a_t)\left(\pi^E(a|s_t) - \pi(a|h_t)\right)\right] && \text{(From (12))} \\
&\leq ||A^{\pi^E}(.,.)||_\infty \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d_t^\pi}||\pi(.|h_t) - \pi^E(.|s_t)||_1 && \text{(Holder's Inequality)} \\
&\leq H(\pi^E)\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d_t^\pi}||\pi(.|h_t) - \pi^E(.|s_t)||_1 \\
&\leq H(\pi^E)\left(\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d_t^\pi}||\pi^\star(.|h_t) - \pi^E(.|s_t)||_1 + \gamma(N)\right) && \text{From (13)} \\
&\leq H(\pi^E)\left(\epsilon(\pi^E,T) + \gamma(N)\right)
\end{aligned}$$

Rearranging, we get

$$\frac{1}{T}J(\pi) \geq \frac{1}{T}J(\pi^E) - H(\pi^E)\left(\epsilon(\pi^E,T) + \gamma(N)\right)$$

□

To optimize the performance bound, we introduce a constrained privileged expert that provides actions similar to the privileged expert but is constrained to be close to realizable.

Definition B.5 (Constrained Privileged Expert)**.** Given a state s_t and history h_t, a constrained privileged expert minimizes KL distance to the privileged expert while being in a δ KL ball of the non-privileged expert.

$$\pi_\delta^E(.|s_t,h_t) := \arg\min_{\tilde{\pi}} \text{KL}\left(\tilde{\pi}(.|s_t,h_t)||\pi^E(.|s_t)\right) \quad \text{s.t.} \quad \text{KL}\left(\tilde{\pi}(.|s_t,h_t)||\pi^E(.|h_t)\right) \leq \delta \tag{14}$$

We use the fact that we have a rich policy class Π that includes the non-privileged expert $\pi^E(.|h)$, to show that the constrained privileged expert bounds the average realizability gap

$$\begin{aligned}
\epsilon(\pi_\delta^E,T) &= \sup_\pi \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d_t^\pi}||\pi_\delta^E(a_t|s_t) - \pi^\star(a_t|h_t))||_1 \\
&\leq \sup_\pi \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d_t^\pi}\sqrt{2\text{KL}(\pi_\delta^E(a_t|s_t)||\pi^\star(a_t|h_t))} \\
&\leq \sup_\pi \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d_t^\pi}\sqrt{2\text{KL}(\pi_\delta^E(a_t|s_t)||\pi^E(a_t|h_t))} && \text{since } \pi^E(.|h_t) \in \Pi \\
&\leq \sqrt{2\delta} && \text{from (14)}
\end{aligned} \tag{15}$$

Theorem B.6 (LEAP with Constrained Privileged Expert)**.** *Running N iterations of LEAP with constrained privileged expert π^E_δ yields at least one policy π such that*

$$\frac{1}{T}J(\pi) \geq \frac{1}{T}J(\pi^E) - \left(\frac{1}{T}\underbrace{\left(J(\pi^E) - J(\pi^E_\delta)\right)}_{\text{gap from privileged expert}} + H(\pi^E_\delta)\left(\underbrace{\sqrt{2\delta}}_{\text{realizability}} + \gamma(N)\right)\right) \tag{16}$$

where π^E is the privileged expert, $H(\pi^E_\delta)$ is the recoverability coefficient of π^E_δ, δ is the KL constraint, $\gamma(N)$ is the average regret of the DAGGER *update.*

The first term is the performance gap between the privileged and the constrained privileged expert. The second term is the realizability of the constrained privileged expert. As $\delta \to 0$, the realizability gap goes down while the performance gap and recoverability coefficient go up. There is a critical value of δ that achieves an optimal tradeoff.

Proof. The proof is similar to Theorem B.4 but with the modification that π^E depends on both state and history.

The PDL in this case can be expressed as

$$J(\pi^E) - J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t,h_t\sim d^\pi_t}\left[\sum_{a_t} A^{\pi^E}(s_t,h_t,a_t)\left(\pi^E(a_t|s_t,h_t) - \pi(a_t|h_t)\right)\right] \tag{17}$$

DAGGER (Ross et al., 2011) with π^E_δ as the expert teacher guarantees that after N iterations, it will find at least one policy

$$\mathbb{E}_{s,h\sim d^\pi}||\pi(.|h) - \pi^E_\delta(.|s,h)||_1 \leq \mathbb{E}_{s,h\sim d^\pi}||\pi^\star(.|h) - \pi^E_\delta(.|s,h)||_1 + \gamma(N) \tag{18}$$

where $\gamma(N)$ is the average regret.

$$\begin{aligned}
&\text{AIG}(\pi,T) = \frac{1}{T}\left(J(\pi^E) - J(\pi)\right)\\
&= \frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + \frac{1}{T}\left(J(\pi^E_\delta) - J(\pi)\right)\\
&= \frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d^\pi_t}\left[\sum_{a_t} A^{\pi^E_\delta}(s_t,h_t,a_t)\left(\pi^E_\delta(a_t|s_t,h_t) - \pi(a_t|h_t)\right)\right] && \text{(From (17))}\\
&\leq \frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + ||A^{\pi^E_\delta}(.,.,.)||_\infty \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d^\pi_t}||\pi(.|h_t) - \pi^E_\delta(.|s_t,h_t)||_1 && \text{(Holder's Inequality)}\\
&\leq \frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + H(\pi^E_\delta)\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d^\pi_t}||\pi(.|h_t) - \pi^E_\delta(.|s_t,h_t)||_1\\
&\leq \frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + H(\pi^E_\delta)\left(\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{s_t,h_t\sim d^\pi_t}||\pi^\star(.|h_t) - \pi^E_\delta(.|s_t,h_t)||_1 + \gamma(N)\right) && \text{From (18)}\\
&\leq \frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + H(\pi^E_\delta)\left(\epsilon(\pi^E_\delta,T) + \gamma(N)\right)\\
&\leq \frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + H(\pi^E_\delta)\left(\sqrt{2\delta} + \gamma(N)\right) && \text{From (15)}
\end{aligned}$$

Rearranging, we get

$$\frac{1}{T}J(\pi) \geq \frac{1}{T}J(\pi^E) - \left(\frac{1}{T}\left(J(\pi^E) - J(\pi^E_\delta)\right) + H(\pi^E_\delta)\left(\sqrt{2\delta} + \gamma(N)\right)\right)$$

□

B.2 PRACTICAL APPROXIMATION FOR CONSTRAINED PRIVILEGED EXPERT

Writing out the Lagrangian of the constrained privileged expert (Def. B.5) and solving for the primal we get:

$$\pi^E(\rho, a|s, h) = \frac{1}{Z(s,h)} \pi^E(\rho, a_t|s_t) \left(\pi^E(\rho, a_t|h_t)\right)^\lambda \tag{19}$$

where λ is the Lagrange multiplier.

For a fixed multiplier λ, this closed-form expression suggests a simple penalty-method approximation to constrained privileged expert:

1. Sample N responses from both $\pi^E(\tilde{\rho}, \tilde{a}|\rho, a, h, s)$ and $\pi^E(\tilde{\rho}, \tilde{a}|\rho, a, h)$ to create $\mathcal{R} = \{\tilde{\rho}^i, \tilde{a}^i\}_{i=1}^N$

2. Reweigh the responses in $\mathcal{R}$ according to

$$\log \pi^E(\tilde{\rho}, \tilde{a}|\rho, a, h, s) + \lambda \log \pi^E(\tilde{\rho}, \tilde{a}|\rho, a, h) \tag{20}$$

3. Sample from the reweighed distribution.

Since the majority of our experiments involve closed-weights models (like `gpt-4o`), we are unable to access the logits of the expert. Instead, we design `LEAP` prompts to approximate the constrained privileged expert by providing instructions to not reveal the privileged information in the reasoning, and provide general reason and actions that should be predictable given only history h_t.

C EXPERIMENTAL DETAILS

C.1 HYPER-PARAMETERS

C.1.1 TRAINING PARAMETERS

Tables 4 and 5 contain hyperparameters for SFT training and DPO/KTO training using LoRA for the different datasets. All training runs were on machines with either 2 or 4 RTX A6000 GPUs, each with 48 GB of memory per GPU.

Dataset	**AlfWorld**	**WebShop**	**InterCode**
Model	Llama3-8B	Llama3-8B	Llama3-8B / Llama3-70B
Batch size	64	16	16
Max seq length	8000	6000	6000
Max epochs	1	1	5
Learning rate	3e-5	3e-5	3e-5
LoRA α	64	64	64
LoRA r	128	128	128
LoRA dropout	0.05	0.05	0.05
Optimizer	AdamW	AdamW	AdamW
LR scheduler	cosine	cosine	cosine

Table 4: Hyperparameters for SFT training using LoRA

Method	DPO	KTO
Model	Llama3-8B	Llama3-8B
Batch size	32	64
Max seq length	8000	8000
Max prompt length	6000	6000
Max epochs	1	1
Regularization β	0.01	0.01
Learning rate	5e-7	5e-7
LoRA α	64	64
LoRA r	128	128
LoRA dropout	0.05	0.05
Optimizer	AdamW	AdamW
LR scheduler	cosine	cosine

Table 5: Hyperparameters for preference training (DPO, KTO) using LoRA

C.1.2 EVALUATION PARAMETERS

For inference using `Llama3-8b` and `Llama3-70b`, we use a temperature setting of 0.3 and a maximum token length of 256. The following code snippet shows the `.generate()` function used:

```
outputs = model.generate(
    tokenized_inputs["input_ids"],
    attention_mask=tokenized_inputs["attention_mask"],
    max_new_tokens=256,
    eos_token_id=[
        tokenizer.eos_token_id,
        tokenizer.convert_tokens_to_ids("<|eot_id|>"),
    ],
    temperature=0.3,
    pad_token_id=self.tokenizer.eos_token_id
)
```

Listing 1: Generation function with hyperparameters for Llama3 models

For all OpenAI model calls, including tasks such as generating expert teacher corrections, labeling reasoning for actions, or running prompt baselines like ReAct, we use the API call with the following hyperparameters. See Appendix D for the detailed prompts.

```
#  model: gpt-4o
chat_completion = client.chat.completions.create(
    messages=messages,
    model="gpt-4o",
    temperature=0.3,
    top_p=1,
    n=1
)
response = chat_completion.choices[0].message.content

#  model: gpt-4o-mini
chat_completion = client.chat.completions.create(
    messages=messages,
    model="gpt-4o-mini",
    temperature=0.3,
    top_p=1,
    n=1
)
response = chat_completion.choices[0].message.content
```

Listing 2: Hyperparameters for OpenAI GPT-4o and GPT-4o-mini models

C.2 ALFWORLD

ALFWorld (Shridhar et al., 2020b) is a text-based game built on the ALFRED embodied benchmark (Shridhar et al., 2020a). Each task has a high-level objective, e.g. "heat mug and put it in cabinet" which the agent must complete by navigating and interacting with a virtual household through text commands, e.g. go to shelf 1, pick up mug 2, etc. It is a POMDP where the hidden state is the location of all objects, and the observations reveal only the objects the agent can see. To solve it, the agent must plan subgoals, track progress, and explore efficiently till time runs out. An interesting aspect of ALFWorld is the requirement to infer the likely locations of household objects (e.g. mugs are more likely to be on shelves or cabinets), which the agent can learn from interaction with the environment. There are 6 categories of tasks and two datasets: 139 in-distribution games and 134 out-of-distribution games. Time horizon is limited to 30.

We compare against `ReAct` (Yao et al., 2022b) with different base models and prompts. The original `ReAct` paper (Yao et al., 2022b) used text-davinci-002 with task-dependent in-context examples. We run `ReAct` wtug `gpt-4o` and `gpt-4o-mini` and an instruction prompt that is shared by all our methods. We also compare against BUTLER (Shridhar et al., 2020b), a seq2seq BERT model trained using DAGGER.[4]

We LoRA fine-tune Llama-3-8B model using `gpt-4o` as the expert teacher for 4 iterations. We use observation-action demonstrations on the training dataset of 3257 gamesShridhar et al. (2020b), which we annotate with `gpt-4o` to generate reasons and use this to train the BC policy π_0. We then roll out a policy and generate corrections from the privileged expert on unsuccessful rollouts. The correction data is then aggregated with the existing dataset and used to train a policy using SFT.

The privileged state in ALFworld consists of

1. Task: What is the overall task
2. Essential Objects: Enumerate all objects that are necessary to solve the game.
3. Critical Locations: List the important locations involved in the game solution.
4. Optimal Action Sequence: Suggest an optimal sequence of actions that could theoretically minimize the steps needed to complete the game, based on the log provided. Be sure to copy the action verbatim.

C.3 WEBSHOP

WebShop (Yao et al., 2022a) is an online shopping environment consisting of 1.18 million real-world products and 12,586 human-provided instructions. Each task is a shopping request from a user ("I need a gluten-free bacon, 4 oz pack of 2, priced under $50") that the agent has to solve through a set of interactions (search["gluten-free bacon"] or selecting options such as "4 oz pack of 2"). The metrics are the overall score, and 4 components of the score: whether the attributes match (ATT), whether the options match (OPT), whether the product types match (TYPE) and whether the price matches (PRICE). We evaluate on 500 test instructions, and limit time horizon to 30.

We compare against `ReAct` (Yao et al., 2022b) with different base models like GPT-4o and GPT-4o-mini. We use the same instruction prompt as in `LEAP`. We also compare against IL baseline in WebShop (Yao et al., 2022a) trained using BC. These baselines train two different selection models, one for selecting search queries from a list of options and one for clicking from a list of options. This constraint search space simplifies the problem.

We LoRA fine-tune Llama-3-8B model using GPT4-o as the expert teacher for 4 iterations. We initialize π_0 using observation-action demonstrations from Yao et al. (2022a) which we annotate with GPT4-o to generate reasons. We roll-out policies on the training dataset of 12086 tasks, and generate corrections from the privileged expert on unsuccessful rollouts. For every game in the training dataset, we construct privileged information that contains: the attributes the product must satisfy, the price constraint, the option that has to be selected, and an example product that would satisfy all criteria. The correction data is then aggregated with the existing dataset and used to train a policy using SFT.

The privileged state in ALFworld consists of

[4] Since we look at the single trial setting, it rules out approaches like Reflexion (Shinn et al., 2023) where agents require multiple trials on the same game.

1. Task: What is the user instruction
2. Attributes: What are the attributes the product must have
3. Price: What price constraints must the product satisfy.
4. Example Product: Name of a product that satisfies all criteria.

C.4 INTERCODE

Intercode Yang et al. (2024) is an interactive coding environment, where an agent interacts with a compiler/interpreter to execute its code and submit further refinements. We use the Bash environment, where a typical task is "Find all text files and write their names to a single file.", and the agent issues commands like `ls`, `find .` to solve the task. Intercode builds on the NL2Bash (Lin et al., 2018) to create 4 interactive datasets. We use the first 2 for training, and the next 2 for testing. We limit the time horizon to 10. We compare against `ReAct` Yao et al. (2022b) with `gpt-4o` and `gpt-4o-mini`. We also compare against the top 3 entries on the Intercode leaderboard, which are the `TryAgain` baselines from Yang et al. (2024) with `gpt-4`, `gpt-3.5` and CodeLlama-34B-INST respectively. Note the numbers are directly from the leaderboard which evaluates on all (train+test) data and, while our numbers are only on test.

D PROMPTS

We define the following three prompts across all datasets:

1. **Expert corrections template:** This prompt is used by the LLM expert teacher to provide corrections on the student agent's trajectories. These corrections offer improved reasoning and actions that balance the trade-off between utilizing the privileged state (accessible only to the expert during training) and ensuring the corrections are realizable by the student agent.
2. **Student agent template:** This prompt is used during test time by the Llama3 student agent models to generate reason and action trajectories that achieve the task objective. It is also used for prompting baselines, such as GPT-4 or ReAct, which generate reasoning and action trajectories to achieve the task objectives.
3. **Autolabel reasons template:** For datasets that contain human demonstrations with action labels (like AlfWorld, WebShop), this prompt is used to generate reasoning annotations for actions that the human took. Generating reasons is necessary as the student agent needs to be trained to produce both reason and actions in its output trajectories.

For prompts used in different ablation studies, please refer to the accompanying code.

D.1 ALFWORLD

D.1.1 EXPERT CORRECTIONS TEMPLATE

```
{% if system %}
You are given a trajectory containing observations, reason and actions
    generated by a student agent solving a text world game. You are a
    teacher who has access to a "privileged state" that contains secret
     information sufficient to solve the game but is hidden from the
    student. Your goal is to improve how the student solves the game by
     improving their reason and action at every timestep.

### Input
You will be provided with a JSON file that logs the student agent's
    observation, candidate_actions, reason and action at every timestep
     while playing a text-based game. The student fails to solve the
    task within the time horizon of the game.

The structure is an array of objects, containing the following at each
    timestep:
- timestep: Index of the current timestep
- observation: The observation provided to the student agent
```

```
- candidate_actions: The set of allowed actions
- reason: The reason generated by the student agent to justify their
    action
- action: The action taken by the student agent

You will also be provided the privileged state that contains hidden
    information that specifies how to solve the task.

### Task
* Analyze the student trajectory and summarize the mistakes it is
    making when trying to solve the game
* Refer to your privileged state to know how the game can be solved
* Generate IMPROVED reason and action for the student at every timestep
     to guide them towards the goal
* Base your improved reasons solely on the student's historical
    observations and actions up to each timestep
* Do NOT include any information from your privileged state in the
    improved reasons, as the student does not have access to those
* Offer GENERAL principles or hints in your improved reasons that
    explain why the student should prefer your suggested action over
    their original action. This would help the student generalize
    better.
* When generating improved reason, action at timestep t, assume that
    the student has followed their original trajectory up until
    timestep t. Copy over the original observation at timestep t from
    the student trajectory.
* When generating improved action at timestep t, make sure the action
    is available in the candidate_actions at timestep t. Don't select
    an action that is not available.

Important: Provide GENERAL principles or hints in your improved reasons
     that explain why the student should prefer your suggested action
    over their original action.
* If you know from your privileged state that an object is in a
    different location from where the student is exploring, use common
    sense rationale to guide the student
* Do not directly instruct the student to go to the desired location,
    as they do not have access to the privileged information you
    possess
* For example, if the student is exploring the wrong area, instead of
    stating the object is in a different location, suggest a general
    principle like, "It might be useful to explore areas because ... ",
     ".. this item is often found in such places", etc
* By following these steps, you help the student understand the logic
    behind the actions without revealing privileged information
* Also note that you cannot carry more than one item at a time

### Output
The output is a JSON containing a summary and a trajectory with the
    same length as the input student trajectory as follows:
```json
{
    "summary": your summary of the mistakes the student is making,
    "trajectory": [
    {
        "timestep": Index of the current timestep,
        "original_observation": The original observation made by
    student agent at timestep t (copy as is from student trajectory),
        "original_reason": The original reason generated by the student
     at timestep t (copy as is from student trajectory),
        "original_action": The original action the student took at
    timestep t (copy as is from student trajectory),
        "corrected_reason": The corrected reason that the student
    should generate,
```
```

```
        "corrected_action": The corrected action that the student
    should take (chosen from list of candidate_actions at timestep t)
    },
    {
        ...
    }
    ...
    ]
}
```
{% endif %}
{% if not system %}
The student trajectory is below:
{{student_trajectory}}

The privileged state for the task is below:
{{privileged_state}}

Provide the ### Output in the JSON format specified above.
{% endif %}
"""

D.1.2 STUDENT AGENT TEMPLATE

```
{% if mode == 'input' %}
You are an intelligent assistant named ALFRED in a text-based
    interactive game called TextWorld. Your objective is to complete
    the given tasks by reasoning through the information provided and
    taking appropriate actions.

Your task is the following:
{{task}}

Below is the history of previous observations and actions:
{{ observation_action_history }}


Given the history of previous observation and action above, a reminder
    that your task is:
{{task}}

You are given as input the current observation and the list of possible
     candidate_actions:
{
    "observation": {{observation}},
    "candidate_actions": {{candidate_actions}}
}


Your goal is to generate the action to take at this time step (chosen
    from candidate_actions) along with the reason for taking the action
    .

Please follow these general instructions:
* You MUST choose action from the list of candidate_actions.
* If "observation": "Nothing happens.", it is because you chose an
    invalid action not from the list of candidate_actions in the
    previous timestep.
* Oftentimes the task requires you interact with objects not present in
     your observation. You must search the environment to locate the
    objective.
```

```
* Consult the history of previous observations and actions to see what
    actions you have tried already so as to not repeat your actions.
* Do NOT repeat the same action as the last action in your
    observation_action_history. It's going to yield the same result.
* Make sure action is VERBATIM copied from the list of
    candidate_actions.

You need to generate a response in the following format. Please issue
    only a single action at a time.
REASON:
Rationale for what action to take next based on the task and previous
    history. In your reason, consult candidate_actions to precisely
    state VERBATIM which action you will do.
ACTION:
The action to be taken, chosen ONLY from candidate_actions
{% elif mode == 'output' %}
REASON:
{{ reason }}
ACTION:
{{ action }}
{% endif %}
```

D.1.3 AUTOLABEL REASONS TEMPLATE

```
{% if system %}
You are given a JSON file containing observations and actions generated
     by an expert agent solving text world games. Your goal is to
    generate the reasoning process of the agent to justify each of
    their actions.

### Input
The input is a JSON array of objects, each containing:
- `observation`: The state observed by the agent at a specific timestep
    .
- `action`: The action taken by the agent in response to the
    observation.

Example input format:
```json
[
    {
        "observation": "You see a locked door.",
        "action": "Use key on door."
    },
    ...
]
```

Output
The output should be a JSON array of objects, each containing:
- `observation`: The state observed by the agent at a specific timestep
    .
- `reason`: The reasoning behind the action taken by the agent.
- `action`: The action taken by the agent in response to the
    observation.

Example output format:
```json
[
    {
        "observation": "You see a locked door.",
```
```

```
        "reason": "The door is locked and I have a key that might open
    it.",
        "action": "Use key on door."
    },
    ...
]
```

Requirements
1. **Causal Constraints**: Ensure the reasoning satisfies causal
    constraints. Use only the observations up to the current timestep
    to justify the action. Do not use future information.
2. **Brevity and Relevance**: The reasoning should be brief and
    relevant, using the observations up until that timestep to justify
    the action.
3. **Sequence Consistency**: Ensure the output maintains the same
    sequence of observations and actions as in the input.

{% endif %}

{% if not system %}
The input JSON file is below:
{{input}}

Provide the output in the format specified above.
{% endif %}

D.2 WebShop

D.2.1 Expert corrections template

{% if system %}
You are given a trajectory containing observations, reason and actions
    generated by a student agent solving a shopping task by searching
    for items in a website and clicking on links. You are a teacher who
     has access to a "privileged state" that contains secret
    information sufficient to solve the task but is hidden from the
    student. Your goal is to improve how the student solves the task by
     improving their reason and action at every timestep.

Input
You will be provided with a JSON file that logs the student agent's
    observation, candidate_actions, reason and action at every timestep
     while solving a shopping task. The student fails to solve the task
     within the time horizon of the task.

The structure is an array of objects, containing the following at each
    timestep:
- timestep: Index of the current timestep
- observation: The observation provided to the student agent
- candidate_actions: The set of allowed actions
- reason: The reason generated by the student agent to justify their
    action
- action: The action taken by the student agent

You will also be provided the privileged state that contains hidden
    information that specifies how to solve the task.

Task
* Analyze the student trajectory and summarize the main mistakes it is
    making when trying to solve the task
* Refer to your privileged state to know how the task can be solved

```
* Correct a select set of timesteps where you know the student was
    definitely doing a wrong action
* Generate IMPROVED reason and action for the student at every timestep
     to guide them towards the goal
* Base your improved reasons solely on the student's historical
    observations and actions up to each timestep
* Do NOT include any information from your privileged state in the
    improved reasons, as the student does not have access to those
* Offer GENERAL principles or hints in your improved reasons that
    explain why the student should prefer your suggested action over
    their original action. This would help the student generalize
    better.
* When generating improved reason, action at timestep t, assume that
    the student has followed their original trajectory up until
    timestep t. Copy over the original observation at timestep t from
    the student trajectory.
* When generating improved action at timestep t, make sure the action
    is available in the candidate_actions at timestep t. Don't select
    an action that is not available.  If candidate_actions has only
    click actions, you can only choose a click action from the list. If
     candidate_actions has only search[<search query>], you must search
     by generating the search keywords on your own.

Important:
(1) Provide GENERAL principles or hints in your improved reasons that
    explain why the student should prefer your suggested action over
    their original action.
* If you know from your privileged state that the desired product is
    different from the one the student is considering, use common sense
     rationale to guide the student
* Do not directly instruct the student to search the ground truth
    product, as they do not have access to the privileged information
    you possess
* By following these steps, you help the student understand the logic
    behind the actions without revealing privileged information

(2) Put corrections sparingly.
* Be strategic about which timesteps you want to correct.
* If you think an action is good enough or reasonable, don't bother
    correcting. Just copy over the original reason and action.

(3) Correct indecisive behavior of the student agent
* Often times, you will see the student agent not finishing the task
    and instead continually browsing through items.
* Provide corrections at key moments to help it resolve indecisiveness.
* It's better to purchase a suboptimal item than not finish the task
    within the time horizon.
* The student gets partial points for matching a subset of the criteria
     in the instruction. They get 0 points for not finishing the task
    in the time horizon (max timesteps in student trajectory)

(4) Considerations for search.
* Specifying prices in search does not work. Other specifications are
    fine.
* Only correct egregious failures in search queries, e.g. searching for
     a specification not mentioned in the instruction. Otherwise search
     usually does not require any correction.

(5) Considerations for click.
* When suggesting corrected_action, make sure this action exists in the
     list of candidate_actions at that timestep.

### Output
The output is a JSON containing a summary and a trajectory with the
    same length as the input student trajectory as follows:
```

```
```json
{
    "summary": your summary of the mistakes the student is making,
    "trajectory": [
    {
        "timestep": Index of the current timestep,
        "is_corrected": True/False depending on whether this timestep
    is corrected or not
        "corrected_reason": The corrected reason that the student
    should generate. If is_corrected=False, copy over original reason.
    '
        "corrected_action": The corrected action that the student
    should take (chosen from list of candidate_actions at timestep t).
    If is_corrected=False, copy over original action. If click action,
    make sure corrected_action belongs to list of candidate_actions at
    the corresponding timestep in input student_trajectory.
    },
    {
        ...
    }
    ...
    ]
}
```
{% endif %}
{% if not system %}
The student trajectory is below:
{{student_trajectory}}

The privileged state for the task is below:
{{privileged_state}}

Provide the ### Output in the JSON format specified above.
{% endif %}
```

D.2.2 STUDENT AGENT TEMPLATE

```
{% if mode == 'input' %}
You are an intelligent shopping agent. Your objective is to solve
    shopping tasks by searching for items in a website, clicking on
    links until you solve the task.

Below is the history of previous observations and actions:
{{ observation_action_history }}

You are given as input the current observation that contains
    instructions for the task and contents of the current webpage. You
    are also given a list of possible candidate_actions:
{
    "observation": {{observation}},
    "candidate_actions": {{candidate_actions}}
}

Your goal is to generate the action to take at this time step along
    with the reason for taking the action.

Please follow these general instructions:
* You MUST choose action from the list of candidate_actions. If
    candidate_actions has only click actions, you can only choose a
    click action from the list. If candidate_actions has only search[<
    search query>], you must search by generating the search keywords
    on your own.
```

```
* When choosing a search action, you MUST follow the format search[<
    search query>]
* Choose search keywords that are informative but not too specific or
    too broad. Examples of search:
search[6 foot coaxial cable, pack of 3]
search[satin brass frosted hallway light fixture]
* When choosing click action, make sure action is VERBATIM copied from
    the list of candidate_actions. Examples of click:
click[satin brass | frosted]
click[buy now]
* Often times the task requires you to find objects not in the current
    webpage. You must search the webpages to locate the object and
    verify it indeed matches the specifications in the instructions. If
     not, go back and refine your search.
* Consult the history of previous observations and actions to see what
    actions you have tried already so as to not repeat your actions.
* Do NOT repeat the same action as the last action in your
    observation_action_history. It's going to yield the same result.
* When choosing click action, make sure action is VERBATIM copied from
    the list of candidate_actions.
* Important: You must try to finish the task and click on buy now as
    quickly as possible. It's better to purchase a suboptimal item than
     not finish the task within the time horizon. You get partial
    points for matching a subset of the criteria in the instruction.
    You get 0 points for not finishing the task in the time horizon (
    typically 20 timesteps).

You need to generate a response in the following format. Please issue
    only a single action at a time.
REASON:
Rationale for what action to take next based on the task and previous
    history. In your reason, consult candidate_actions to precisely
    state VERBATIM which action you will do.
ACTION:
The action to be taken, chosen ONLY from candidate_actions
{% elif mode == 'output' %}
REASON:
{{ reason }}
ACTION:
{{ action }}
{% endif %}
```

D.2.3 AUTOLABEL REASONS TEMPLATE

```
{% if system %}
You are given a trajectory containing observations and actions
    generated by an agent solving a web shopping task. Your goal is to
    generate the reasoning process that the agent must have been
    thinking at each timestep to justify each of their actions.

### Input
You will be provided with a JSON file that logs the agent's observation
     and action at every timestep while solving the web shop task.

The structure is an array of objects, containing the following at each
    timestep:
- timestep: Index of the current timestep
- observation: The instruction the agent has to solve and the current
    webpage observed by the agent at a specific timestep.
- action: The action taken by the agent.
```

```
### Task
* Analyze the agent trajectory
* Generate reason for the agent at every timestep. The reason should be
     in first person, e.g. "I should .."
* Base your generated reason solely on the historical observations and
    actions up to each timestep
* Offer GENERAL principles or hints in your reasons that explain why
    the agent took the action
* For example, if the agent is backtracking by clicking on previous,
    explain why that is the case
* Do NOT include any information from future observations as the agent
    does not know the future at that time
* Ensure the output maintains the same sequence of actions as in the
    input.
* In your REASON, be sure to provide detailed justification for the
    action. If candidate_actions has only click actions, specify which
    option from the list are you choosing as your action.
* Often times, the task requires you to select both a product and
    specific options for that product. Be sure to justify why you are
    selecting a particular option for a product by referring to the
    task in your observation.
* Often times, the agent settles for an option even though it may not
    be exactly optimal. This is because the agent has a fixed time
    horizon within which they must click buy now. Ensure that your
    reason rationalizes why it's good to finish the task by clicking on
     buy now.

### Output
The output is a trajectory represented as a JSON array of objects with
    the same length as the input trajectory as follows:
```json
[
    {
        "timestep": Index of the current timestep,
        "reason": A reason to justify the action given observation and
    action up until the current timestep
        "action": The action taken by the agent (copy as is from input
    trajectory)
    },
    ...
]
{% endif %}
{% if not system %}
The input trajectory is below:
{{input}}

Provide the ### Output in the JSON format specified above.
{% endif %}
```

D.3 InterCode

D.3.1 Expert corrections template

```
{% if system %}
You are given a trajectory containing observations, reason and actions
    generated by a student agent interacting with a MySQL Database
    system using sql queries to answer a question.  You are a teacher
    who has access to a "privileged state" that contains secret
    information sufficient to solve the task but is hidden from the
    student. Your goal is to improve how the student solves the task by
     improving their reason and action at every timestep.
```
```

```
### Input
You will be provided with a JSON file that logs the student agent's
    observation, reason and action at every timestep while solving a
    shopping task. The student fails to solve the task within the time
    horizon of the task.

The structure is an array of objects, containing the following at each
    timestep:
- timestep: Index of the current timestep
- observation: The observation provided to the student agent
- reason: The reason generated by the student agent to justify their
    action
- action: The action taken by the student agent

You will also be provided the privileged state. The privileged state
    contains an example of an optimal SQL command to solve the task and
     the expected observation that solves the task. Note that it is the
     observation that the student gets evaluated on. Multiple possible
    actions can lead to that observation.

### Task
* Analyze the student trajectory and summarize the main mistakes the
    student is making when trying to solve the task.
* Refer to your privileged state to understand how the task can be
    solved optimally.
* Correct a select set of timesteps where the student's action is
    clearly wrong.
* Generate **IMPROVED REASONS** and **ACTIONS** for the student at each
     timestep to guide them towards the goal. The improved reason
    should be framed as if it is the student's new reasoning, e.g., "I
    should ...".
* The improved reason and action are intended to replace the student's
    original reason and action at that timestep. Do NOT refer to the
    original incorrect reason and action when generating improved
    reason and action.
* Base your improved reasons solely on the student's historical
    observations and actions up to that timestep.
* Do NOT include any information from your privileged state in the
    improved reasons, as the student does not have access to that
    information.
* Offer general principles or hints in your improved reasons that
    explain why the student should prefer your suggested action over
    their original action. This would help the student generalize
    better.
* When generating an improved reason and action at timestep `t`, assume
     the student followed their original trajectory up until timestep `
    t`.

Important:
(1) Provide **GENERAL principles or hints** in your improved reasons.
* If you know from your privileged state that a particular set of
    tables should be queried to solve the task, suggest reasons and
    actions that guide the student to first discover those tables.
* Do not directly instruct the student to output the privileged action,
     as they do not have access to privileged information.
* By following these steps, you help the student understand the logic
    behind the actions without revealing privileged information.

(2) Apply corrections sparingly.
* Be strategic about which timesteps you choose to correct.


### Output
The output is a JSON containing a summary and a trajectory with the
    same length as the input student trajectory as follows:
```

```
```json
{
    "summary": your summary of the mistakes the student is making,
    "trajectory": [
    {
        "timestep": Index of the current timestep,
        "original_reason": The original reason generated by the student
     at timestep t (copy as is from student trajectory),
        "original_action": The original action the student took at
    timestep t (copy as is from student trajectory),
        "is_corrected": True/False depending on whether this timestep
    is corrected or not,
        "corrected_reason": The corrected reason that the student
    should generate at timestep t. If is_corrected=False, leave blank ,
        "corrected_action": The corrected action that the student
    should take at timestep t. If is_corrected=False, leave blank
    },
    {
        ...
    }
    ...
    ]
}
```
{% endif %}
{% if not system %}
The student trajectory is below:
{{student_trajectory}}

The privileged state for the task is below:
{{privileged_state}}

Provide the ### Output in the JSON format specified above.
{% endif %}
```

D.3.2 STUDENT AGENT TEMPLATE

```
{% if system %}
You are given a trajectory containing observations, reason and actions
    generated by a student agent interacting with a MySQL Database
    system using sql queries to answer a question.  You are a teacher
    who has access to a "privileged state" that contains secret
    information sufficient to solve the task but is hidden from the
    student. Your goal is to improve how the student solves the task by
     improving their reason and action at every timestep.

### Input
You will be provided with a JSON file that logs the student agent's
    observation, reason and action at every timestep while solving a
    shopping task. The student fails to solve the task within the time
    horizon of the task.

The structure is an array of objects, containing the following at each
    timestep:
- timestep: Index of the current timestep
- observation: The observation provided to the student agent
- reason: The reason generated by the student agent to justify their
    action
- action: The action taken by the student agent

You will also be provided the privileged state. The privileged state
    contains an example of an optimal SQL command to solve the task and
```

```
     the expected observation that solves the task. Note that it is the
     observation that the student gets evaluated on. Multiple possible
    actions can lead to that observation.

### Task
* Analyze the student trajectory and summarize the main mistakes the
    student is making when trying to solve the task.
* Refer to your privileged state to understand how the task can be
    solved optimally.
* Correct a select set of timesteps where the student's action is
    clearly wrong.
* Generate **IMPROVED REASONS** and **ACTIONS** for the student at each
     timestep to guide them towards the goal. The improved reason
    should be framed as if it is the student's new reasoning, e.g., "I
    should ...".
* The improved reason and action are intended to replace the student's
    original reason and action at that timestep. Do NOT refer to the
    original incorrect reason and action when generating improved
    reason and action.
* Base your improved reasons solely on the student's historical
    observations and actions up to that timestep.
* Do NOT include any information from your privileged state in the
    improved reasons, as the student does not have access to that
    information.
* Offer general principles or hints in your improved reasons that
    explain why the student should prefer your suggested action over
    their original action. This would help the student generalize
    better.
* When generating an improved reason and action at timestep `t`, assume
     the student followed their original trajectory up until timestep `
    t`.

Important:
(1) Provide **GENERAL principles or hints** in your improved reasons.
* If you know from your privileged state that a particular set of
    tables should be queried to solve the task, suggest reasons and
    actions that guide the student to first discover those tables.
* Do not directly instruct the student to output the privileged action,
     as they do not have access to privileged information.
* By following these steps, you help the student understand the logic
    behind the actions without revealing privileged information.

(2) Apply corrections sparingly.
* Be strategic about which timesteps you choose to correct.


### Output
The output is a JSON containing a summary and a trajectory with the
    same length as the input student trajectory as follows:
```json
{
    "summary": your summary of the mistakes the student is making,
    "trajectory": [
    {
        "timestep": Index of the current timestep,
        "original_reason": The original reason generated by the student
     at timestep t (copy as is from student trajectory),
        "original_action": The original action the student took at
    timestep t (copy as is from student trajectory),
        "is_corrected": True/False depending on whether this timestep
    is corrected or not,
        "corrected_reason": The corrected reason that the student
    should generate at timestep t. If is_corrected=False, leave blank ,
        "corrected_action": The corrected action that the student
    should take at timestep t. If is_corrected=False, leave blank
```
```

```
    },
    {
        ...
    }
    ...
    ]
}
```
{% endif %}
{% if not system %}
The student trajectory is below:
{{student_trajectory}}

The privileged state for the task is below:
{{privileged_state}}

Provide the ### Output in the JSON format specified above.
{% endif %}