CALIBRATE-THEN-ACT: COST-AWARE EXPLORATION IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents are deployed in environments where they must interact to acquire information. In these scenarios, the agent must reason about inherent cost-uncertainty tradeoffs in how to act, such as when to stop exploring and commit to an answer. For instance, on a programming task, an agent might run the code it generates, or it might generate tests for that code snippet; the cost of writing and running a test is nonzero, but typically lower than the cost of running buggy code. In this work, we show that we can induce LLM agents to explicitly reason about balancing these cost-uncertainty tradeoffs, then act more optimally in their environments. We formalize multiple tasks, including retrieval-augmented QA, a file reading coding task, and interactive software engineering, as sequential decision-making problems under uncertainty. Each problem has latent environment state that impacts the agent's performance. We introduce a framework called Calibrate-Then-Act (CTA), where we pass the agent an inferred prior about this environment state to enable it to act more optimally. This information qualitatively changes agent behavior, and adds environment sensitivity to the agent which is not learned via standard RL training. Our results on three tasks show that making cost-benefit tradeoffs explicit with CTA helps agents discover more optimal decision-making strategies.

1 INTRODUCTION

Large language model (LLM) agents are increasingly tasked with operating in environments where information is incomplete. Behaving rationally requires gaining information by exploring the environment. However, exploration comes with a cost: every additional step increases API costs, interaction latency, and user burden.

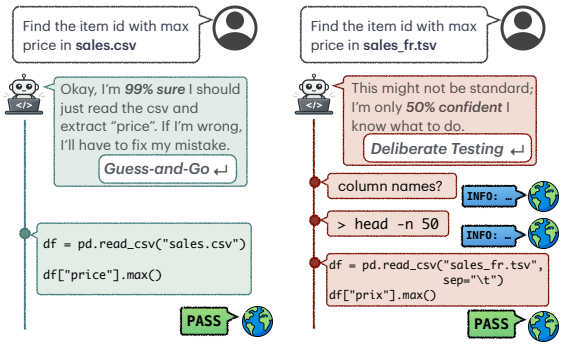


Figure 1: Given the same task, a coding agent may either verify assumptions via intermediate checks carefully (right) or attempt a direct solution as soon as possible (left). The optimal choice depends on uncertainty and specific cost constraints. Calibrate-Then-Act (CTA) materializes this information for better decision-making.

This exploration and its cost come in many forms. In software development and debugging, agents must decide whether to run targeted checks or perform full execution before committing to a solution (Zhou et al., 2025a). In machine learning experimentation, practitioners balance inexpensive proxy evaluations against costly full training runs under limited compute budgets (Ji & Carin, 2007; Hennig et al., 2024; Xu et al., 2025). In diagnosis and scientific discovery, additional tests or experiments reduce uncertainty but incur monetary, temporal, or safety costs (Kärkkäinen et al., 2019; Li & Oliva, 2025; Gupta et al., 2025). Online decision-making settings such as shopping (Yang et al., 2023; Wang et al., 2025d), recommendation (Herlihy et al., 2024), and tool-augmented question answering (Yao et al., 2023) exhibit the same structure, as agents weigh further information gathering against acting with partial information.

Agent policies for this exploration depend in a complex way on their prompt, their inputs, and their training data. However, these policies are frequently *static*. For instance, ChatGPT Deep Research

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

always asks a single round of clarifying questions before searching (Deng et al., 2025), and coding agents like SWE-agent (Yang et al., 2024) start by reading through an existing codebase. This contrasts with settings like Figure 1, where we see that a model can act more efficiently if it has confidence that it understands the problem setup and appropriately trades off exploration against action costs.

In this work, we ask: **how can we develop LLMs that explore in an optimal way under varying cost and uncertainty profiles?** Leveraging the strong reasoning capabilities of modern LLMs, we propose a framework called Calibrate-Then-Act (CTA), which decouples the calibration of uncertainty from the reasoning of action selection. The key insight is that by presenting priors explicitly to the model, we induce the model to reason about an underlying sequential decision-making problem abstractly and discover the optimal action.

We study three settings inspired by realistic problems: (1) knowledge-based QA with optional retrieval, where uncertainty is inferred from the model's own confidence; (2) coding tasks where priors regarding environmental structure (e.g., file schemas) are derived from cues learned through past experience; and (3) a realistic software engineering benchmark where the agent decides whether to ask the user a clarifying question, with priors estimated from the user's interaction history. We show that in these settings, LLM agents prompted or trained with RL fail to behave optimally. However, this is not due to an inability to perform the correct reasoning: we show with a synthetic "Pandora's Box" problem (Weitzman, 1979), and show that even a small thinking model (Qwen3-8B) can follow the optimal policy. The Calibrate-Then-Act method builds on this result, feeding the agent with relevant information about the environment, determined by either estimating model confidence or a trained predictor of environment state. A zero-shot prompted implementation of this method, CTA-PROMPTED, already enables dynamic decision-making. We show that training with RL further improves performance, and notably outperforms baseline RL training of the agent.

Our contributions are: (1) Framing environment exploration as a sequential decision-making problem, unified across four settings; (2) The Calibrate-Then-Act method, which induces LLMs to reason about the optimality of their actions and achieve better cost-performance tradeoffs than baselines.

2 BACKGROUND: AGENTIC EXPLORATION WITH COSTS

We formalize cost-aware environment exploration task as sequential decision-making problem. Our agent, which for the purposes of this work is an LLM, is given a query $\mathbf{x}$ and operates in some environment, which can be defined as a partially-observable Markov Decision Process $\mathcal{W} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, O, T, R, D_\theta)$, a tuple of states $\mathcal{S}$, actions $\mathcal{A}$, observations $\mathcal{O}$, observation function O, transition function T, reward function R, and parameterized discount function D_θ, which integrates the cost.

In the settings we consider, $\mathcal{A}$ and $\mathcal{O}$ are both string-valued spaces; LLMs produce string actions (code, API calls, etc.) and receive string-valued responses from the environment. The observation function O produces string realizations of the underlying environment; e.g., in Figure 1, the results of executing commands return string output in the terminal reflecting the underlying state of the environment.

The environment contains problem-critical unobserved features that will determine the agent's performance, e.g., details about the formatting of the unobserved file in Figure 1. These can be thought of as a subset of the information in $\mathcal{S}$. We represent these as a random variable Z taking values $\mathbf{z} \in \mathcal{Z}$.

The agent interacts with the environment over multiple timesteps before terminating. At each timestep t, the agent selects an action $a_t \in \mathcal{A}$ and receives an observation $o_t \in \mathcal{O}$; for simplicity, we assume that o_t encodes a_t. Based on this information, we can form an idealized posterior distribution $b_t(Z) = p(Z \mid \mathbf{x}, o_{0:t})$ which reflects remaining uncertainty over the latent variables. The action space generally consists of multiple *exploration* actions and a *commit* action, which terminates the episode by producing a final result. In Figure 1, exploration actions include querying aspects of the input data file. Other actions (not shown) include running the code or writing and running unit tests.

Each action incurs different costs depending on the setting, while the commit action corresponds to returning a final solution. These costs are reflected by a discount factor $D_\theta(a_{1:T}) \in [0, 1]$, which discounts the value of successful task completion based on the exploration actions taken prior to commitment.

Finally, the agent receives reward upon committing: $R = \mathbb{I}[\text{task completed at } a_t] \cdot D_\theta(a_{1:T})$,

Table 1: Unified formalization of cost-aware decision problems and their instantiations across tasks. We characterize each by latent variables z^*, a prior belief p over $\mathbf{z}$, an action space $\mathcal{A}$ consisting of exploration and commit actions, observations $\mathcal{O}$ revealed through exploration, costs θ associated with the exploration actions, and a final reward R that discounts task success by incurred costs.

Setting	$z^* \in \mathcal{Z}$	p	$\mathcal{A}$	$\mathcal{O}$	θ	R
UNIFIED (§2)	Latent env. state	Prior over relevant latent variables	Actions	Observations	Cost	Reward
PANDORA (§3.1)	$z^* \in \{z_1, \dots, z_K\}$ Prize-containing box	$\{p_k = \pi(z^* = z_k)\}$	Verify(k), Commit(k)	Whether z_k has prize	γ	$\gamma^T \mathbb{I}(z_k = z^*)$
QA (§3.2)	$(p_{\text{da}}, p_{\text{ret}})$ Answer probabilities with and without retrieval	$p_{\text{da}} = \delta(z = k_{\text{da}}(\mathbf{x}))$, $p_{\text{ret}} = \delta(z = k_{\text{ret}})$	Retrieve, Answer(a)	Retrieved context	γ	$\gamma^T \mathbb{I}(a = a^*)$
CODE (§3.3)	$\mathbf{z}^* = (z_d^*, z_q^*, z_s^*)$ $\in \mathcal{Z}_d \times \mathcal{Z}_q \times \mathcal{Z}_s$	$p(\mathbf{z} \mid n)$	`UNIT_TEST`(f), `CODE`(d, q, s), `ANSWER`(a)	Format value, stdout + stderr	(d_u, d_c)	$d_u^U d_c^C \mathbb{I}(a = a^*)$
SWE-CLARIFY (§3.4)	$(\mathbf{z}_u, p_{\text{act}})$ User interaction profile; success probability of acting directly	$\hat{p}_{\text{act}}$ from calibrated confidence; $p(\mathbf{z}_u \mid \mathbf{h})$ from interaction history	Act, Ask-Up(q), Ask-Mid(q)	User reply, repo + test feedback	γ	$\gamma^T \mathbb{I}$(tests pass)

Overall, the agent's objective is to maximize the expected discounted reward R by carefully selecting actions that adaptively balance exploration and commitment in response to uncertainty and cost constraints of the environment.

3 TASKS

We now connect our general task setting to four concrete problems we address in this work. The first, Pandora's Box (Weitzman, 1979), serves as a proof-of-concept instance of cost-aware exploration, where we demonstrate that even small LLMs have the capability to reason about uncertainty-cost tradeoffs in environments. We then describe three realistic problems where LLMs struggle to reason about costs out of the box, motivating the method in this paper. Table 1 summarizes how these problems map onto our general framework. Prompt templates for these tasks are provided in Appendix J.

3.1 PANDORA'S BOX: A MINIMAL SETTING FOR COST-AWARE DECISIONS

Formalization We consider a variant of the classic Pandora's Box Problem with discounted reward over time (Weitzman, 1979). The task involves a finite set of K boxes $\{z_1, z_2, \dots, z_K\}$, among which exactly one box z^* contains a prize of value 1. This box is unknown to the agent and is drawn from a prior distribution $p(z_k = z^*) = p_k, \quad \sum_{k=1}^{K} p_k = 1$. At each timestep t, the agent can either *verify* a box z_{k_t} of its choice to check if it contains the prize and discount their final reward by $\gamma \in [0, 1]$, or *commit* to a box given its current information and receive a reward $R = \gamma^t \cdot \mathbb{I}(z_{k_t} = z^*)$.

This task can be viewed as a toy version of real-world use cases for LLM agents. For instance, if trying to identify a bug in a piece of code, we can view committing as directly taking action to fix a bug and verifying as writing a unit test to check the correctness of the code.

Knowledge of the prior probabilities is necessary to behave optimally. The optimal policy is to commit if its posterior probability of a box is greater than γ, which means we have better expected value for guessing than for verifying. Otherwise, the optimal decision is to verify the highest-probability remaining box. The full algorithm is shown in Algorithm 1 and proof is provided in Appendix H.

3.2 TASK QA: KNOWLEDGE QA WITH OPTIONAL RETRIEVAL

We study a knowledge question answering setting in which an LLM can optionally acquire external information at a cost (Eisenstein et al., 2025). Given a factual query, the model must decide whether to rely on its parametric knowledge or defer commitment and retrieve additional evidence, trading off potential accuracy gains against latency and API costs.

Formalization Given a question $\mathbf{x}$, a discount factor $\gamma \in [0, 1]$, and access to a retriever, the agent chooses between actions $\mathcal{A} = \{\text{retrieve}, \text{answer}\}$. A retrieve action queries a retrieval system with the input $\mathbf{x}$, while an answer action invokes the LLM to answer the question given the context so far, producing an answer a. The model receives reward $R = \gamma^t \cdot \mathbb{I}(a = a^*)$, where a^* is the ground-truth answer to $\mathbf{x}$. This task only features up to one round of retrieval, so t is either 0 or 1.

Latent structure and prior There are two relevant latent variables for this problem. First, we define $p_{\text{da}} = p(a = a^* \mid \mathbf{x})$ that the model will correctly answer the question if asked directly, without retrieval. We model p_{da} with a distribution $p(p_{\text{da}} \mid \mathbf{x}) = \delta(z = k(\mathbf{x}))$ for an estimate $k_{\text{da}}(\mathbf{x})$ that the LLM returns the correct answer, where δ is the Dirac delta function. This distribution represents that we have a point estimate of the model's probability to answer correctly.

Second, we define the probability of answering post-retrieval to be $p_{\text{ret}} = p(a' = a^* \mid \mathbf{x}, \mathbf{c})$, where $\mathbf{c}$ is the retrieved context. We similarly represent this as a delta function $p(p_{\text{ret}} \mid \mathbf{x}) = \delta(z = k_{\text{ret}})$

Under this abstraction, the oracle policy retrieves whenever the expected discounted accuracy after retrieval exceeds that of direct answering: $p_{\text{ret}} \cdot \gamma \geq p_{\text{da}}$.

3.3 Task FileReading: Coding with Selective Testing

Following Figure 1, we design a coding setting to encapsulate the idea of a model choosing to either run code directly or test it. Concretely, the model needs to write code to load a data file with correct formatting options and return the answer to a task query (e.g., identifying the `user_id` associated with the maximum `score`). The true file schema is not specified in the input; instead, the model may infer likely formats from filename cues and past experience. Unlike the QA setting, these priors are not known to the model a priori, but must be learned from training on this task. Despite the simplicity of this setting, we show that LLM agents fail to learn the optimal behavior via RL.

Formalization Given a query $\mathbf{x}$ that contains a task specification and a CSV filename n, the agent needs to write code to load the file correctly and compute an answer a. Specifically, each file is associated with latent formatting attributes $\mathbf{z} = (z_d, z_q, z_s) \in \mathcal{Z}$, where $z_d \in \{\texttt{,}, \texttt{;}, \texttt{\textbackslash t}\}$ denotes the delimiter, $z_q \in \{\texttt{'}, \texttt{"}\}$ the quote character, and $z_s \in \{0, 1\}$ the number of skipped header rows. Without making a correct inference about $\mathbf{z}$, the task is not solvable.

At each timestep t, the agent selects one action $a_t \in \mathcal{A}$ from three types: `UNIT_TEST`(f), `CODE`(d, q, s), or `ANSWER`. A `UNIT_TEST` probes a chosen formatting attribute f and reveals its true value. A `CODE` action executes code written under the agent's current belief (d, q, s) and returns feedback via `stdout` and `stderr`, which may contain the answer or signals useful for debugging and refinement. An `ANSWER` action commits to a final answer a' and terminates the episode. The agent may interleave `UNIT_TEST` and `CODE` actions in any order, and may perform multiple `CODE` actions to refine its solution based on previous execution feedback.

Each `UNIT_TEST` and `CODE` action incurs multiplicative discounts d_u and d_c, respectively. Upon committing at time T by `ANSWER`, the agent receives reward $R = d_u^U \cdot d_c^C \cdot \mathbb{I}(a' = a^*)$, where a^* denotes the ground-truth answer.

Prior The prior distribution $p(\mathbf{z} \mid n)$ over formatting attributes may be inferred from conventions or past experience, or provided explicitly by a format predictor.

3.4 Task SWE-Clarify: Software Engineering with Optional User Clarification

Finally, SWE-Clarify asks whether a coding agent should implement a vague request or consult a human user. The agent chooses $a \in \{\text{act}, \text{ask-up}, \text{ask-mid}\}$ and receives $R = \gamma^t \cdot \mathbb{I}[\text{tests pass}]$, where $t = 1$ iff it asks. CTA estimates action success from model confidence and user history. This task is structurally analogous to task QA, with clarification replacing retrieval. Appendix F details Stateful SWE-bench (Zhou et al., 2025b).

4 Method: Calibrating Agent Environment Exploration

To behave Pareto-optimally in a sequential decision-making problem, an agent must jointly compare the cost of additional exploration against the expected value of additional information to decide whether to continue exploring or to commit based on its current partial information. The value of additional information depends on reasoning over current beliefs about the underlying world state via the prior $p(\mathbf{z} \mid \mathbf{x})$ and posterior b_t.

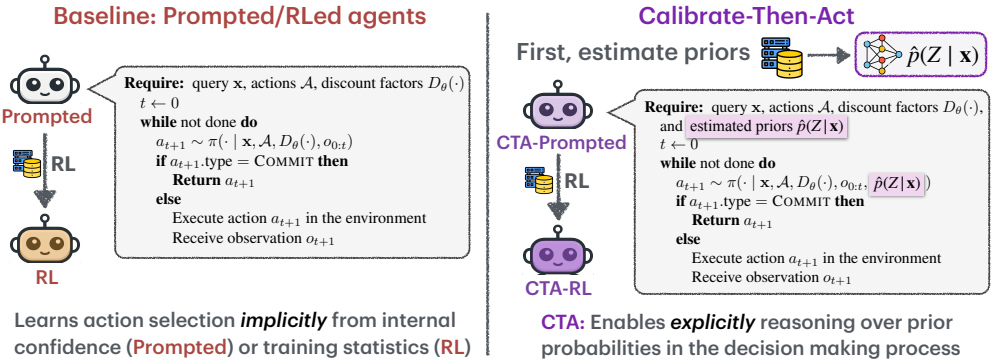


Figure 2: Standard agentic decision loop (left) and proposed method CTA (right). In CTA, we learn a prior estimator from training data and condition the agent on estimated $\hat{p}$ at inference and/or training time, inducing more optimal decision making through explicit reasoning over prior probabilities.

We define our LLM agent as $\pi(a_t \mid \mathbf{x}, \mathcal{A}, D_\theta(\cdot), o_{0:t})$, placing a distribution over the next action in a given state. Figure 2 (left side) shows the basic form of this agent. π can be implemented either via a prompted LLM or through a model trained with reinforcement learning. However, in practice, it is difficult for π to learn to do the right reasoning in the environments we evaluate on.

Our key methodological contribution is to *explicitly* provide estimates of the prior, denoted as $\hat{p}(Z \mid \mathbf{x})$. Figure 2 (right side) illustrates the role of a prior estimate in the agentic decision loop. We instantiate two versions of this approach:

- **CTA-Prompted:** In this setting, we use the same agent π as the baseline, but with a modified prompt to include the prior information $\hat{p}(Z \mid \mathbf{x})$ together with $\mathbf{x}$. Examples of the prompt are given in Appendix J.
- **CTA-RL:** In this setting, we use the prompt of CTA-Prompted and perform additional training via RL. We fine-tune the model end-to-end using GRPO (Shao et al., 2024) with the discounted reward objective from the task at hand. We evaluate it on new examples with the same prompt used at training time.

Both settings rely on the ability of a thinking LLM to determine how to use this information. In Section 6.1, we show that the Pandora's Box problem is solvable by thinking models given the appropriate priors. In Section 6.2 and Figure 3, we then show that thinking models also perform substantially better on our tasks of interest.

Having accurate prior estimates is important for CTA to do well. We now describe how to estimate the priors described in Table 1. For Pandora's Box, the priors are available explicitly, but for our other problems, estimation is needed. These problems illustrate three different ways of estimating priors: from model confidence, from training data, and from user interaction history. However, a commonality is that all three estimation techniques draw inferences about the problem that are not easily learned from end-to-end RL, thereby motivating the need for CTA.

4.1 Estimating Priors from Model Confidence (QA)

In QA, the true $k_{\text{da}}(\mathbf{x})$, or the probability that the model can answer $\mathbf{x}$ correctly without retrieval, is not directly observable. This problem is one of confidence estimation. There are several ways to obtain estimates of model confidence, including inspecting logits, probe-based methods, and verbalized confidence. In this work, we use verbalized confidence (Mohri & Hashimoto, 2024), which is a general method that can apply to both blackbox and glassbox model usage. Given a question $\mathbf{x}$, we prompt the model to produce a discrete verbalized confidence value $p_v(\mathbf{x})$, and apply an isotonic regression model (Zadrozny & Elkan, 2002) ISO trained on the validation set to obtain a calibrated estimate $\hat{k}_{\text{da}}(\mathbf{x}) = \text{ISO}\big(p_v(\mathbf{x})\big)$. The expected calibration error (ECE) reduces from 0.618 to 0.029 on PopQA (Mallen et al., 2023) (see Section 5) with this regression. This drop is reflective of initial poor calibration (Guo et al., 2017; Xiong et al., 2024; Shen et al., 2024; Wang et al., 2025b) and demonstrates that rescaling can help (Desai & Durrett, 2020). This kind of recalibration is very hard for the agent to learn with approaches like end-to-end RL: if RLed for a long time on a QA dataset, a model is more likely to memorize specific facts than it is to systematically recalibrate its beliefs.

4.2 Estimating Priors from Training Data (FileReading)

Coding agents bring to bear many kinds of priors when completing tasks. Our FileReading task studies inferences about file formats as a controlled representative problem. In our setting, we assume that the agent is trained on our task, and therefore both baseline RL and the CTA methods are exposed to data showing the mapping of filenames to format information.

For prior estimation, we train a filename-to-format predictor, denoted as $\mathcal{M}_{\text{BERT}}$, to estimate the distribution $p(\mathbf{z} \mid n)$ from the filename. The predictor is based on a lightweight BERT-tiny encoder (4.4M parameters) (Bhargava et al., 2021; Turc et al., 2019). Given a filename n, the model encodes the tokenized string and uses the `[CLS]` representation to produce three independent categorical distributions via linear heads: delimiter, quote character, and skiprows. The model is trained with a summed cross-entropy objective across the three heads for one epoch on the training split. On the validation split, $\mathcal{M}_{\text{BERT}}$ achieves an average classification accuracy of 67% across the three formatting attributes. This accuracy is expected: the ground-truth format values are sampled from a soft filename-dependent prior whose most likely option has probability about 67% on average. Thus, the task is intentionally stochastic rather than deterministically predictable from the filename. Appendix B shows that CTA performs worse as this predictor gets worse, but in a smooth fashion, with noisy predictors still giving benefit.

After training, $\mathcal{M}_{\text{BERT}}$ outputs marginal probabilities $\{p(z_d \mid n), p(z_q \mid n), p(z_s \mid n)\}$, which are provided to the agent associated with each task during RL training or test time, thereby decoupling uncertainty estimation from action selection.

4.3 Estimating Priors from Confidence and Interaction History (SWE-Clarify)

SWE-Clarify requires priors over both task success and user behavior. We estimate $\hat{p}_{\text{act}}$ by asking the agent how likely a direct implementation is to pass the tests, then recalibrating this confidence on validation data. A cheap LLM infers user-profile probabilities from the interaction history, while $\hat{p}_{\text{up}}$ and $\hat{p}_{\text{mid}}$ are validation success rates for asking upfront or mid-work. All agents see the same history; only CTA receives these estimates explicitly. See details in Appendix F.

5 Experimental Setup

Datasets For QA, we evaluate on PopQA (Mallen et al., 2023), a QA benchmark that covers long-tail factual knowledge and benefits from retrieval. We sample 1,000 questions for evaluation and build the retriever based on **Contriever** (Izacard et al., 2022; Bajaj et al., 2016). For each question, we sample a discount factor $\gamma \sim \mathcal{U}[0.1, 0.65]$ to study model behavior across varying retrieval costs.

Our coding problem involves a CSV-based question-answering dataset called FileReading, where filename cues provide informative signals about file formats and correct parsing requires executing code with appropriate format values. At test time, the true file format is hidden and only the filename is provided as part of the task query $\mathbf{x}$. FileReading contains 2,000 tasks, split into 1,400 training, 300 validation, and 300 test examples. For each task, we randomly sample a unit-test discount d_u from $[0.5, 1]$ and duplicate the instance across four code discount settings $d_c = d_u^{\rho}$, with $\rho \in \{0.5, 1.0, 2.0, 4.0\}$, varying the relative cost of code execution while holding the task fixed. Details of the dataset construction are provided in Appendix G.

For SWE-Clarify, we use Stateful SWE-bench (Zhou et al., 2025b), derived from SWE-bench Verified (Jimenez et al., 2024) rewritten as vague instructions and paired with user personas. We evaluate 156 test instances across 13 user profiles and fit estimators on 65 validation instances. Each instance is scored at six γ values and graded with hidden tests. (Details in Appendix F.)

Metrics We evaluate the model performance across three sets of metrics. (1) Exploration statistics: retrieval rate Retrieve$\%$ for QA, number of unit-test U and code calls C for FileReading, and ask rate Ask$\%$ for SWE-Clarify. (2) Task accuracy or success, binary metrics evaluating final task correctness. (3) Reward, which discounts correctness with exploration costs.

Models and Baselines We use Qwen3-8B (Qwen Team, 2025) for QA and FileReading, comparing Prompted, Prompted-NonThink, and RL against CTA-Prompted and CTA-RL. For SWE-Clarify, Claude Haiku 4.5 (Anthropic, 2025) is the coding agent, Claude Sonnet 4.5 simulates the user, and we compare Prompted with CTA-Prompted.

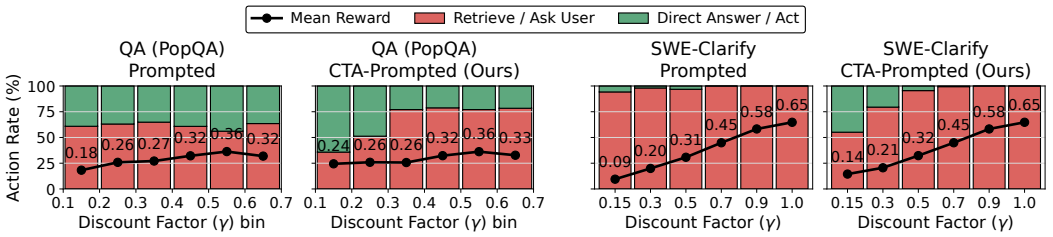


Figure 3: Model decisions across discount factors γ on QA (left) and SWE-CLARIFY (right). Red bars show exploration (retrieval or asking the user), and green bars show committing without exploration. The black line reports mean discounted reward; point labels give reward before rescaling to the shared 0–100 axis. CTA-PROMPTED adapts its exploration rate to the cost and shows higher rewards in more costly settings.

- PROMPTED: We prompt the base model directly with task description and query $\mathbf{x}$.
- PROMPTED-NONTHINK : Similar to PROMPTED, but with thinking mode disabled by prepending `<think></think>` tags without thinking content. Unless specified with -NONTHINK, we enable thinking mode by default for all other settings.
- RL: We fine-tune the model end-to-end using GRPO (Shao et al., 2024) with discounted reward objective, and evaluate it by prompting with the task description and query $\mathbf{x}$.

To expose both RL settings to diverse cost trade-offs in FILEREADING, we duplicate each training instance across multiple relative cost values $\rho \in \{0.5, 1.0, 2.0, 4.0\}$, yielding a $4\times$ larger RL training set than that used for training $\mathcal{M}_{\text{BERT}}$. More training and evaluation details are in Appendix I and F.

6 RESULTS

6.1 LLMS CAN SOLVE PANDORA'S BOX PROBLEMS

We first evaluate whether LLM agents can solve the proof-of-concept Pandora's Box task introduced in Section 3.1. We instantiate the agent using the framework in Figure 2, with prompts in Figure 9, and use Qwen3-8B as the base model. We evaluate on 100 examples with $K = 3$, discount factors sampled from $[0, 0.1, \ldots, 1.0]$, and priors sampled from a symmetric Dirichlet distribution with concentration parameter $\alpha = 0.5$.

Table 2: Performance comparison on the Pandora's Box task. A thinking model given access to priors can solve the problem near-optimally.

Method	Optimal Match Rate (%)	Avg. Reward
Oracle policy	100.0	0.649
PROMPTED-NONTHINK	11.0	0.441
PROMPTED	23.0	0.476
CTA-PROMPTED-NONTHINK	20.0	0.436
CTA-PROMPTED	**94.0**	**0.625**

Table 2 reports the performance with two metrics: average reward and optimal policy match rate, measuring whether the model's interaction trace aligns with the oracle strategy described in Appendix Algorithm 1. Without explicit priors or when the thinking mode is disabled, the agent exhibits low optimal match rates, indicating failure to recover the optimal decision rule. In contrast, CTA-PROMPTED achieves 94.0% optimal match rate and significantly higher reward, indicating that the model is capable of reasoning about optimal exploration-commitment tradeoffs and adapting to different discount factors and prior distributions when the environment constraints are made explicit. Appendix D compares interaction traces across settings, showing how explicit reasoning in thinking traces is used. **Crucially, the presence of explicit prior information triggers the model to reason about cost-accuracy tradeoffs, from which it can make optimal decisions.**

6.2 CALIBRATE-THEN-ACT ENABLES MODELS TO MAKE CALIBRATED DECISIONS

Baseline LLM agents struggle to make calibrated decisions, even after RL. Across both QA and FILEREADING, we find that without explicit priors, LLMs do not naturally form calibrated decision rules from task descriptions alone.

For **QA**, Figure 3 visualizes the retrieval behavior across retrieval discount factors γ. Each stacked bar shows the model's decisions within a discount-factor bin: red denotes retrieval or ask and green denotes direct actions. The black line reports mean discounted reward. PROMPTED (Figure 3, left) adapts only weakly to retrieval costs. Although PROMPTED sometimes reasons qualitatively about its parametric knowledge, it rarely quantifies these as certainty values for an expected-value comparison. Similarly, it sometimes mentions costs, but does not consistently

Table 3: Performance on QA. Our principal metric is reward. One-turn baselines use fixed strategies, while multi-turn agents adaptively decide when to retrieve. CTA-PROMPTED achieves the highest reward.

Method	Retrieve %	Acc.	Reward
Static baselines			
NEVER RETRIEVE	0.0	0.226	0.226
ALWAYS RETRIEVE	100.0	0.578	0.213
Interactive agents			
PROMPTED	61.4	0.501	0.283
CTA-PROMPTED (Ours)	65.3	0.512	**0.293**

Table 4: Performance on SWE-CLARIFY, averaged over all γ. Compared to other baselines, CTA achieves the highest reward while asks fewer clarification questions with adaptive behavior.

Method	Ask %	Success	Reward
Static baselines			
NEVER ASK	0.0	0.154	0.154
ALWAYS ASK UPFRONT	100.0	0.647	0.383
Interactive agents			
PROMPTED	98.2	0.629	0.380
CTA-PROMPTED (Ours)	88.2	0.596	**0.392**

tie the final decision to the costs. As a result, its decisions often fall back to a recall-based heuristic: answer if a plausible candidate comes to mind, retrieve otherwise, rather than systematically adapting to γ and confidence. This suggests that, without explicit prior estimates, the models struggle to assess their own parametric knowledge and make consistent retrieval decisions. We observe the same failure on SWE-CLARIFY: PROMPTED asks the user on 94–100% of instances at every γ. At $\gamma = 0.15$, this cost-insensitive policy earns 0.095 reward, below the 0.154 reward from never asking.

On **FILEREADING**, Figure 4 shows a similar analysis. The x-axis $\rho = \log d_c / \log d_u$ reflects relative cost: as ρ increases, code attempts become relatively more costly, and a rational agent will tend to test more before coding. Each stacked bar represents the tasks at a given ρ, and each color represents the proportion of action trace pattern in the collection. The reward is labeled above each bar, and the percentage of "guess-and-go" x% (attempting code without any preceding unit tests) are also labeled.

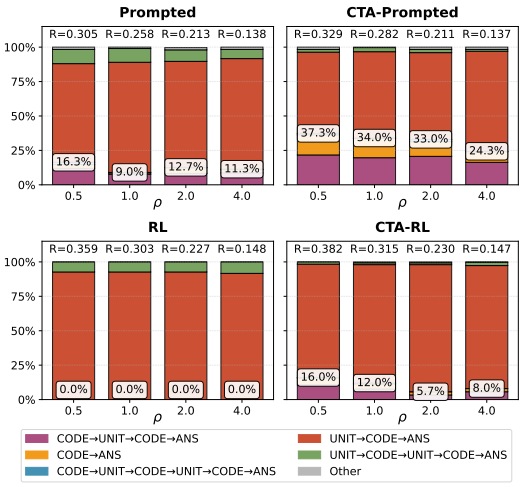


Figure 4: Action distribution for agents on FILEREADING across relative cost parameters ρ. Each stacked bar shows the proportion of decision traces corresponding to different action patterns, with the reward R labeled above. Annotated percentages indicate the fraction of tasks where the agent attempts code execution before any unit tests.

Both PROMPTED and RL collapse to a static test-first policy rather than adapting across cost regimes, with near 0% "code-first" traces (attempts to execute code before any unit tests). RL without prior cannot internalize the structure of the training data and instead defaults to a suboptimal exploration policy. Together, these results show that LLMs struggle to make calibrated, cost-aware decisions with implicit priors.

CTA pushes the model towards adaptive, optimal decision making. Tables 3, 4 show that CTA-PROMPTED outperforms both static baselines and PROMPTED in QA and SWE-CLARIFY, via adaptive actions conditioned on costs and uncertainty. Figure 3 further visualizes such adaptivity.

Table 5: Performance on FILEREADING, averaged across relative unit-test and code-execution cost ratios $\rho \in \{0.5, 1.0, 2.0, 4.0\}$. We report the average number of turns, unit-test calls (U), code executions (C), accuracy, and discounted reward.

Method	**# Turns**	**U**	**C**	**Acc.**	**Reward**
Without Training					
PROMPTED	3.62	2.67	1.42	0.958	0.229
CTA-PROMPTED (Ours)	3.47	2.51	1.41	0.945	0.240
With RL Training					
RL	3.51	2.13	1.39	0.997	0.259
CTA-RL (Ours)	3.46	1.98	1.46	0.991	**0.268**

Table 5 shows results on FILEREADING averaged across cost settings. Note that both RL and CTA-RL have access to the same set of training data which encodes correlations between filenames and file formats implicitly; CTA-PROMPTED uses more information than PROMPTED by virtue of the trained calibrator model. The agent achieves a reward of 0.259 by training end-to-end (RL) with the discounted reward as objective, while conditioning the training on explicit estimated priors (CTA-RL) further improves by 3.5% with an overall

reward of 0.268. This shows that incorporating estimated priors helps the model generalize better on unseen test data than end-to-end RL.

Figure 4 shows that while end-to-end RL improves reward over the non-training PROMPTED, its decision-making behavior collapses to a static policy which always triggers unit tests before the first code attempt (0% "guess-and-go"). In contrast, CTA-PROMPTED, which conditions the agentic decision-making on estimated priors, already exhibits adaptive behavior in response to costs by acting more conservatively with higher ρ. After training, this adaptivity remains pronounced in CTA-RL.

Figure 5 visualizes the Δ Reward against the naive baseline "3*Tests→Code" for each method. While RL only shows advantage with large ρ, and static policy such as Code first only performs well with small ρ, our CTA-RL stays at the Pareto frontier across ρ values (significance tests in Appendix C). This suggests that conditioning training with explicit priors effectively reinforces the adaptive decision reasoning.

Even with imperfect prior estimates, CTA-RL consistently performs better than RL baseline across cost regimes. Appendix B further analyzes sensitivity to prior estimator quality. Appendix K provides representative traces comparing RL and CTA-RL, illustrating how CTA induces more cost-aware decisions.

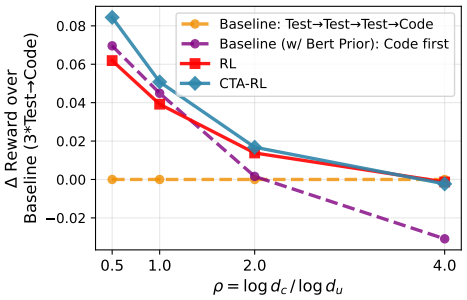


Figure 5: Average reward under varying costs for our approaches. Static strategies (test-first or code-first) achieve high reward only in limited regimes, whereas CTA-RL with estimated priors is consistently on the Pareto frontier.

7 RELATED WORK

Decision making under incomplete information LLMs are increasingly being applied to tasks with incomplete information, arising from underspecified user queries (Cole et al., 2023; Zhang et al., 2025; Zhang & Choi, 2025; Li et al., 2025; Shaikh et al., 2025), ambiguity (Min et al., 2020; Choi et al., 2025; Deng et al., 2025), and partially observed environments (Wong et al., 2023; Lin et al., 2024; Dwaracherla et al., 2024; Chen et al., 2025a; Grand et al., 2025). To resolve uncertainty, models often need to ask clarifying questions (Rao & Daumé III, 2018; Handa et al., 2024; Lalai et al., 2025), query the environments (Charikar et al., 2002; Nadimpalli et al., 2025; Monea et al., 2024), or engage in collaboration (Wu et al., 2025; Chen et al., 2025b). While prior work has typically focused on training or prompting strategies, we show that we can shape model behavior with explicit priors.

Agents in cost-aware deployment LLM-based agents are increasingly deployed in real-world settings that require multi-step reasoning and tool use, including interactive coding (Tang et al., 2024; Zhou et al., 2025a; Wang et al., 2025c; Jain et al., 2025), planning (Zhou et al., 2024; Liu et al., 2025a), question answering (Yao et al., 2023; Eisenstein et al., 2025), and scientific research (Schwettmann et al., 2023; GX-Chen et al., 2025; Khan et al., 2025; Abaskohi et al., 2026; Agarwal et al., 2025). While tool use expands agents' capabilities, environment interaction often incurs latency (Guan et al., 2025), resources (Damani et al., 2024; Liu et al., 2025b), and overhead that can negatively affect user experience (Elfleet & Chollet, 2024; Herlihy et al., 2024). In response, several lines of work have emerged to study agent behavior under explicit cost constraints. Liu et al. (2025a) introduce a cost-centric benchmark for evaluating agents' tool-planning abilities. Wang et al. (2025a); Gul et al. (2025); Wang et al. (2025b); Lin et al. (2025) study how models can reduce unnecessary actions while maintaining answer quality. Berant et al. (2025) train steerable clarification policies that adapt to cost coefficients. An underexplored direction is the joint treatment of uncertainty and costs, which together determine Pareto-optimality. We propose a unified framework for interactive agentic tasks and show that calibrated priors are key to inducing appropriate decision-making.

8 CONCLUSION

This paper presents a method for having LLMs balance uncertainty-cost tradeoffs in their environment interaction. By presenting an LLM with priors over unobserved features of the environment, the LLM can successfully reason about Pareto-optimal behavior and navigate action costs effectively. This work illustrates new ways of inducing agents to think optimally, and suggests that meta-level information (priors about capabilities) may have a role to play in shaping agent policies.

AI USE STATEMENT

We evaluated generative AI as decision-making agents in our experiments and as the simulated user in SWE-CLARIFY. An AI assistant also assist in implementing part of the experiments following our setup, and in visualization of experimental results. Generative AI was not used to formulate research ideas or hypotheses, create datasets, or develop mathematical claims or proofs. The authors reviewed all AI-assisted analyses and take responsibility for the final content of this work.

REFERENCES

Amirhossein Abaskohi, Tianyi Chen, Miguel Muñoz-Mármol, Curtis Fox, Amrutha Varshini Ramesh, Étienne Marcotte, Xing Han Lù, Nicolas Chapados, Spandana Gella, Christopher Pal, Alexandre Drouin, and Issam H. Laradji. DRBench: A realistic benchmark for enterprise deep research. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IGYQ4c92e2.

Dhruv Agarwal, Bodhisattwa Prasad Majumder, Reece Adamson, Megha Chakravorty, Satvika Reddy Gavireddy, Aditya Parashar, Harshit Surana, Bhavana Dalvi Mishra, Andrew McCallum, Ashish Sabharwal, et al. Open-ended Scientific Discovery via Bayesian Surprise. *arXiv preprint arXiv:2507.00310*, 2025.

Anthropic. Claude Haiku 4.5 and Claude Sonnet 4.5. Model cards, https://www.anthropic.com/claude, 2025.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. MS MARCO: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

Jonathan Berant, Maximillian Chen, Adam Fisch, Reza Aghajani, Fantine Huot, Mirella Lapata, and Jacob Eisenstein. Learning steerable clarification policies with collaborative self-play. *arXiv preprint arXiv:2512.04068*, 2025.

Prajjwal Bhargava, Aleksandr Drozd, and Anna Rogers. Generalization in NLI: Ways (Not) To Go Beyond Simple Heuristics, 2021.

Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. *Journal of Computer and System Sciences*, 64 (4):785–819, 2002. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.2002.1828. URL https://www.sciencedirect.com/science/article/pii/S0022000002918283.

Sanxing Chen, Xiaoyin Chen, Yukun Huang, Roy Xie, and Bhuwan Dhingra. When greedy wins: Emergent exploitation bias in meta-bandit llm training. *ArXiv*, abs/2509.24923, 2025a. URL https://api.semanticscholar.org/CorpusID:281674231.

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Optima: Optimizing effectiveness and efficiency for LLM-based multi-agent system. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11534–11557, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.601. URL https://aclanthology.org/2025.findings-acl.601/.

Jio Choi, Mohit Bansal, and Elias Stengel-Eskin. Language models identify ambiguities and exploit loopholes. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 32991–33006, 2025.

Jeremy R. Cole, Michael JQ Zhang, Daniel Gillick, Julian Martin Eisenschlos, Bhuwan Dhingra, and Jacob Eisenstein. Selectively answering ambiguous questions. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=x2W2dKdNI8.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of lm computation. *ArXiv*, abs/2410.04707, 2024. URL https://api.semanticscholar.org/CorpusID:273186996.

Mingyi Deng, Lijun Huang, Yani Fan, Jiayi Zhang, Fashen Ren, Jinyi Bai, Fuzhen Yang, Dayi Miao, Zhaoyang Yu, Yifan Wu, Yanfei Zhang, Fengwei Teng, Yingjia Wan, Song Hu, Yude Li, Xin Jin, Conghao Hu, Haoyu Li, Qirui Fu, Tai Zhong, Xinyu Wang, Xiangru Tang, Nan Tang, Chenglin Wu, and Yuyu Luo. InteractComp: Evaluating Search Agents With Ambiguous Queries. *ArXiv*, abs/2510.24668, 2025. URL https://api.semanticscholar.org/CorpusID:282401680.

Shrey Desai and Greg Durrett. Calibration of pre-trained transformers. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 295–302, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.21. URL https://aclanthology.org/2020.emnlp-main.21/.

Vikranth Dwaracherla, Seyed Mohammad Asghari, Botao Hao, and Benjamin Van Roy. Efficient exploration for LLMs. *arXiv preprint arXiv:2402.00396*, 2024.

Jacob Eisenstein, Reza Aghajani, Adam Fisch, Dheeru Dua, Fantine Huot, Mirella Lapata, Vicky Zayats, and Jonathan Berant. Don't lie to your friends: Learning what you know from collaborative self-play. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=2vDJiGUfhV.

Morad Elfleet and Mathieu Chollet. Investigating the Impact of Multimodal Feedback on User-Perceived Latency and Immersion with LLM-Powered Embodied Conversational Agents in Virtual Reality. In *IVA*, pp. 12:1–12:9, 2024. URL https://doi.org/10.1145/3652988.3673965.

Gabriel Grand, Valerio Pepe, Jacob Andreas, and Joshua B Tenenbaum. Shoot First, Ask Questions Later? Building Rational Agents that Explore and Act Like People. *arXiv preprint arXiv:2510.20886*, 2025.

Yilin Guan, Qingfeng Lan, Sun Fei, Dujian Ding, Devang Acharya, Chi Wang, William Yang Wang, and Wenyue Hua. Dynamic speculative agent planning. *arXiv preprint arXiv:2509.01920*, 2025.

Mustafa Omer Gul, Claire Cardie, and Tanya Goyal. Pay-per-search models are abstention models. *arXiv preprint arXiv:2510.01152*, 2025.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pp. 1321–1330. PMLR, 2017.

Rushil Gupta, Jason Hartford, and Bang Liu. LLMs for experiment design in scientific domains: Are we there yet? In *ICML 2025 Generative AI and Biology (GenBio) Workshop*, 2025. URL https://openreview.net/forum?id=dIEeOwrmOe.

Anthony GX-Chen, Dongyan Lin, Mandana Samiei, Doina Precup, Blake A. Richards, Rob Fergus, and Kenneth Marino. Language agents mirror human causal reasoning biases. how can we help them think like scientists? *ArXiv*, abs/2505.09614, 2025. URL https://api.semanticscholar.org/CorpusID:278602122.

Kunal Handa, Yarin Gal, Ellie Pavlick, Noah Goodman, Jacob Andreas, Alex Tamkin, and Belinda Z Li. Bayesian preference elicitation with language models. *arXiv preprint arXiv:2403.05534*, 2024.

Leona Hennig, Tanja Tornede, and Marius Lindauer. Towards leveraging AutoML for sustainable deep learning: A multi-objective HPO approach on deep shift neural networks. *arXiv preprint arXiv:2404.01965*, 2024.

Christine Herlihy, Jennifer Neville, Tobias Schnabel, and Adith Swaminathan. On Overcoming Miscalibrated Conversational Priors in LLM-based ChatBots. In *Uncertainty in Artificial Intelligence*, pp. 1599–1620. PMLR, 2024.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=jKN1pXi7b0.

Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-turn code generation through single-step rewards. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=aJeLhLcsh0.

Shihao Ji and Lawrence Carin. Cost-sensitive feature acquisition and classification. *Pattern Recognition*, 40(5):1474–1485, 2007.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Kimmo Kärkkäinen, Mohammad Kachuee, Orpaz Goldstein, and Majid Sarrafzadeh. Cost-sensitive feature-value acquisition using feature relevance. *arXiv preprint arXiv:1912.08281*, 2019.

Zaid Khan, Archiki Prasad, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. One life to learn: Inferring symbolic world models for stochastic environments from unguided exploration. *ArXiv*, abs/2510.12088, 2025. URL https://api.semanticscholar.org/CorpusID:282064346.

Harsh Nishant Lalai, Raj Sanjay Shah, Jiaxin Pei, Sashank Varma, Yi-Chia Wang, and Ali Emami. The world according to LLMs: How geographic origin influences LLMs' entity deduction capabilities. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=hJtvCfDfs1.

Belinda Z. Li, Been Kim, and Zi Wang. Questbench: Can LLMs ask the right question to acquire information in reasoning tasks? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=gpwA9aZLTZ.

Yang Li and Junier Oliva. Towards cost sensitive decision making. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Emtiyaz Khan (eds.), *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 3601–3609. PMLR, 03–05 May 2025. URL https://proceedings.mlr.press/v258/li25h.html.

Jessy Lin, Nicholas Tomlin, Jacob Andreas, and Jason Eisner. Decision-oriented dialogue for human-AI collaboration. *Transactions of the Association for Computational Linguistics*, 12:892–911, 2024. doi: 10.1162/tacl_a_00679. URL https://aclanthology.org/2024.tacl-1.50/.

Tzu-Han Lin, Wei-Lin Chen, Chen-An Li, Hung yi Lee, Yun-Nung Chen, and Yu Meng. AdaSearch: Balancing Parametric Knowledge and Search in Large Language Models via Reinforcement Learning. *ArXiv*, abs/2512.16883, 2025. URL https://api.semanticscholar.org/CorpusID:283933928.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R Fung. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents. *arXiv preprint arXiv:2511.02734*, 2025a.

Tengxiao Liu, Zifeng Wang, Jin Miao, I Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, et al. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006*, 2025b.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.546. URL https://aclanthology.org/2023.acl-long.546/.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. AmbigQA: Answering ambiguous open-domain questions. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5783–5797, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.466. URL https://aclanthology.org/2020.emnlp-main.466/.

Christopher Mohri and Tatsunori Hashimoto. Language models with conformal factuality guarantees. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 36029–36047, 2024.

Giovanni Monea, Antoine Bosselut, Kianté Brantley, and Yoav Artzi. LLMs Are In-Context Bandit Reinforcement Learners. *arXiv preprint arXiv:2410.05362*, 2024.

Shivam Nadimpalli, Mingda Qiao, and Ronitt Rubinfeld. No price tags? no problem: Query strategies for unpriced information. *ArXiv*, abs/2511.06170, 2025. URL https://api.semanticscholar.org/CorpusID:282911938.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Sudha Rao and Hal Daumé III. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2737–2746, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1255. URL https://aclanthology.org/P18-1255/.

Sarah Schwettmann, Tamar Shaham, Joanna Materzynska, Neil Chowdhury, Shuang Li, Jacob Andreas, David Bau, and Antonio Torralba. FIND: A Function Description Benchmark for Evaluating Interpretability Methods. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 75688–75715. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/ef0164c1112f56246224af540857348f-Paper-Datasets_and_Benchmarks.pdf.

Omar Shaikh, Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. Navigating rifts in human-LLM grounding: Study and benchmark. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20832–20847, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1016. URL https://aclanthology.org/2025.acl-long.1016/.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yuanhao Shen, Xiaodan Zhu, and Lei Chen. SMARTCAL: An approach to self-aware tool-use evaluation and calibration. *arXiv preprint arXiv:2412.12151*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Hao Tang, Keya Hu, Jin Peng Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. Code Repair with LLMs gives an Exploration-Exploitation Tradeoff. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 117954–117996. Curran Associates, Inc., 2024. doi: 10.52202/079017-3746. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d5c56ec4f69c9a473089b16000d3f8cd-Paper-Conference.pdf.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962, 2019. URL http://arxiv.org/abs/1908.08962.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting Less is Reasoning More! Teaching Model to Act Efficiently. *arXiv preprint arXiv:2504.14870*, 2025a.

Hongru Wang, Boyang Xue, Baohang Zhou, Tianhua Zhang, Cunxiang Wang, Huimin Wang, Guanhua Chen, and Kam-Fai Wong. Self-DC: When to Reason and When to Act? Self Divide-and-Conquer for Compositional Unknown Questions. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6510–6525, 2025b.

Yunkun Wang, Yue Zhang, Zhen Qin, Chen Zhi, Binhua Li, Fei Huang, Yongbin Li, and Shuiguang Deng. ExploraCoder: Advancing code generation for multiple unseen APIs via planning and chained exploration. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18124–18145, Vienna, Austria, July 2025c. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.887. URL https://aclanthology.org/2025.acl-long.887/.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Monica Lam, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Fei-Fei Li, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning. *ArXiv*, abs/2504.20073, 2025d. URL https://api.semanticscholar.org/CorpusID:278170861.

Martin L. Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–654, 1979. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1910412.

Li Siang Wong, Gabriel Grand, Alexander K. Lew, Noah D. Goodman, Vikash K. Mansinghka, Jacob Andreas, and Joshua B. Tenenbaum. From word models to world models: Translating from natural language to the probabilistic language of thought. *ArXiv*, abs/2306.12672, 2023. URL https://api.semanticscholar.org/CorpusID:259224900.

Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. CollabLLM: From passive responders to active collaborators. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=DmH4HHVb3y.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, YIFEI LI, Jie Fu, Junxian He, and Bryan Hooi. Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=gjeQKFxFpZ.

Yuebin Xu, Zhiyi Chen, and Zeyi Wen. EcoTune: Token-efficient multi-fidelity hyperparameter optimization for large language model inference. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7735–7745, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.394. URL https://aclanthology.org/2025.emnlp-main.394/.

Hui Yang, Sifu Yue, and Yunzhong He. Auto-GPT for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*, 2023.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International*

Conference on Learning Representations, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 694–699, 2002.

Michael JQ Zhang and Eunsol Choi. Clarify when necessary: Resolving ambiguity through interaction with LMs. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5526–5543, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.306. URL https://aclanthology.org/2025.findings-naacl.306/.

Michael JQ Zhang, W. Bradley Knox, and Eunsol Choi. Modeling future conversation turns to teach LLMs to ask clarifying questions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=cwuSAR7EKd.

Yaowei Zheng, Junting Lu, Shenzhi Wang, Zhangchi Feng, Dongdong Kuang, and Yuwen Xiong. Easyr1: An efficient, scalable, multi-modality rl training framework, 2025. https://github.com/hiyouga/EasyR1.

Lingfeng Zhou, Junhao Shi, Jin Gao, and Dequan Wang. Credit-budgeted ICPC-style coding: When LLM agents must pay for every decision. In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*, 2025a. URL https://openreview.net/forum?id=JEdeMSKvbT.

Xuhui Zhou, Valerie Chen, Zora Zhiruo Wang, Graham Neubig, Maarten Sap, and Xingyao Wang. ToM-SWE: User mental modeling for software engineering agents. *arXiv preprint arXiv:2510.21903*, 2025b.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=b6rA0kAHT1.

Table 6: Sensitivity to prior quality in the FILEREADING task. Performance degrades smoothly as priors become less accurate, without a sharp failure transition.

Prior	Prior Acc	Avg Reward	Task Acc	#Code	$R_{\rho=0.5}$	$R_{\rho=1.0}$	$R_{\rho=2.0}$	$R_{\rho=4.0}$
BERT	0.67	0.269	99.1%	1.46	0.382	0.315	0.230	0.147
Noisy	0.52	0.251	96.3%	1.71	0.365	0.295	0.206	0.136
Uniform	0.46	0.251	96.8%	1.53	0.349	0.300	0.219	0.134
Adversarial	0.33	0.203	94.4%	2.26	0.299	0.240	0.173	0.099

A LIMITATIONS

Our experiments focus on controlled QA and FILEREADING settings that isolate cost-aware exploration, but they are simpler than open-ended agent deployments with richer action spaces, longer horizons, and noisier observations. We use these settings to enable reliable evaluation of decision quality, while extending CTA to more complex environments remains a promising direction for future work.

CTA also relies on the availability of useful prior estimates. Our sensitivity analysis shows that CTA is robust to moderate estimation error, but performance can degrade when priors are highly miscalibrated or confidently wrong. We demonstrate several ways to obtain priors without ground-truth access, including from model confidence and historical task data, but designing reliable prior estimators for broader domains remains future work.

Finally, CTA does not guarantee optimal behavior in arbitrary environments. The model must still correctly interpret the provided priors and reason about their implications for action selection. In more complex scenarios, optimal action selection may require stronger reasoning capabilities of the base model to account for posterior updates, multi-step value of information, and dependencies among multiple latent variables.

B SENSITIVITY TO PRIOR QUALITY

We evaluate how performance varies with prior quality in the FILEREADING task by systematically degrading the prior estimator used by CTA-RL. In addition to the learned BERT-based prior, we construct three alternative priors with progressively worse calibration:

- **Noisy**: Derived from the BERT prior by randomly selecting 40% of predictions and swapping the probabilities assigned to the correct and incorrect options.
- **Uniform**: A uniform distribution over all candidate options, representing the absence of informative prior knowledge.
- **Adversarial**: Constructed by swapping the probabilities assigned to correct and incorrect options and inflating the incorrect option to 95% confidence.

Table 6 reports performance across different prior qualities. As prior quality decreases, both reward and task accuracy degrade gradually. Importantly, the degradation is smooth rather than exhibiting a sharp failure transition, indicating robustness to imperfect priors. This suggests that approximate or imperfectly estimated priors are sufficient to yield meaningful gains (even the BERT prior achieves only 67% accuracy), indicating that our approach is inherently error-tolerant and applicable in complex real-world settings where accurate priors are difficult to obtain due to noisy or high-dimensional features. At the same time, the adversarial setting highlights that being confidently wrong is substantially more harmful than being uncertain, leading to the largest drop in performance.

C SIGNIFICANCE TESTING FOR PARETO-OPTIMALITY IN FILEREADING

We evaluate whether the gains of CTA-RL are statistically reliable using paired bootstrap tests on per-task rewards. For each cost setting $\rho \in \{0.5, 1.0, 2.0, 4.0\}$, we pair the test examples across methods by task ID. We compute per-task reward differences between CTA-RL and each baseline,

Table 7: Paired bootstrap tests for CTA-RL on FILEREADING. Δ denotes the mean per-task reward gain of CTA-RL over each baseline. Tests use $n = 300$ paired tasks per ρ and $B = 10{,}000$ bootstrap resamples.

ρ	vs. Test first (B1)	vs. Code first (B2)	vs. RL
0.5	$+0.0845^{**}$	$+0.0149^{=}$	$+0.0224^{**}$
1.0	$+0.0508^{**}$	$+0.0059^{=}$	$+0.0117^{*}$
2.0	$+0.0169^{**}$	$+0.0153^{=}$	$+0.0030^{=}$
4.0	$-0.0021^{=}$	$+0.0287^{**}$	$-0.0010^{=}$

$^{*}p < 0.01$, $^{**}p < 0.001$, and $^{=}$ denotes no significant difference.

and use $10{,}000$ bootstrap resamples to test whether the mean difference is significantly different from zero. Table 7 reports the observed mean reward gain Δ.

Overall, CTA-RL achieves significant gains over different baselines in several cost regimes and is never statistically worse than any baseline. These results support the conclusion that CTA-RL remains on or near the Pareto frontier across cost regimes.

CTA-RL significantly outperforms the no-prior test-first baseline at $\rho \in \{0.5, 1.0, 2.0\}$ and is statistically indistinguishable from it at $\rho = 4.0$. Against the prior-aware code-first baseline, CTA-RL is statistically indistinguishable at lower ρ, where code-first already benefits from prior information, but significantly outperforms it at $\rho = 4.0$, where code execution becomes relatively expensive. Compared to the no-prior RL agent, CTA-RL achieves significant gains at $\rho = 0.5$ and $\rho = 1.0$ and is statistically indistinguishable at higher ρ. Overall, these results support the conclusion that CTA-RL remains on or near the Pareto frontier across cost regimes.

D QUALITATIVE TRACE ANALYSIS OF PANDORA'S BOX PROBLEM

We present representative interaction traces from three settings: CTA-PROMPTED-NONTHINK, PROMPTED, and CTA-PROMPTED. In CTA-PROMPTED-NONTHINK (with thinking mode disabled; Figure 6), the model does not appear to compare the expected value of additional information against the exploration cost. As a result, it tends to verify all options before committing, regardless of the prior distribution, leading to unnecessary exploration. In PROMPTED (Figure 7), the model does not have access to the prior probabilities. Lacking calibrated uncertainty information, it effectively operates under an implicit uniform prior and consequently follows a suboptimal verification strategy.

In contrast, in CTA-PROMPTED (Figure 8), the model is provided with prior probabilities and has thinking mode enabled. In this setting, it explicitly reasons about the trade-off between expected reward and exploration cost by comparing the value of immediate commitment with the discounted value of further verification. The resulting behavior aligns with the oracle policy.

These qualitative examples illustrate how explicit prior information induces the correct reasoning of the model over value of additional information against action cost and making the optimal decision accordingly.

E EXPERIMENT DETAILS FOR QA

For QA, we evaluate on the POPQA dataset. We build the retriever based on Contriever (Izacard et al., 2022; Bajaj et al., 2016). The retriever quality p_{ret}, defined as the probability that the model can answer a question correctly when conditioned on the document retrieved from the retriever, is estimated on a validation set and provided to the agent as a part of the input $\mathbf{x}$ at inference time. Note that p_{ret} depends on the retriever and the agent being used, but not on individual questions.

Case Study: Pandora's Box Problem, CTA-PROMPTED-NONTHINK

Problem Parameters
- Bag Labels: A, B, C
- Prior Probabilities: A: 0.04, B: 0.68, C: 0.28
- Discount Factor r: 0.2

Interaction Trace
Turn 0
User: {Instruction instantiated with the problem parameters}
Assistant: VERIFY A
Turn 1
User: The verification result is: NO, A is incorrect. Given this, please provide your next action.
Assistant: VERIFY B
Turn 2
User: The verification result is: NO, B is incorrect. Given this, please provide your next action.
Assistant: GUESS C

Figure 6: Example interaction trace on a 3-bag Pandora's Box instance with priors (0.04, 0.68, 0.28) and discount factor $\gamma = 0.2$ with thinking mode **disabled**. In this setting, the model explores all bags before committing and follows a suboptimal verification order, rather than prioritizing the highest-probability option.

Case Study: Pandora's Box Problem, PROMPTED

Problem Parameters
- Bag Labels: A, B, C
- Prior Probabilities: A: 0.04, B: 0.68, C: 0.28
- Discount Factor r: 0.2

Interaction Trace

Turn 0

User: {Instruction instantiated with the problem parameters, **but the prior probabilities are not provided to the model**}

Assistant: <think> Okay, let's see. I need to figure out the best action here. So, there are three bags: A, B, and C. Only one of them has the prize worth 1. The discount factor is 0.2. At each step, I can either verify a bag to get information (which takes a timestep) or guess a bag to end the game and get the reward, which is discounted by r^t where t is the timestep.

Starting at t=0. If I guess now, the reward would be $1 * 0.2^0 = 1$. But if I guess wrong, I get nothing. Since there's a $1/3$ chance of being correct, the expected reward for guessing at t=0 is $(1/3) * 1 + (2/3) * 0 = 1/3 \approx 0.333$.

Alternatively, if I verify a bag. Let's say I verify bag A. It takes one timestep (t=0 to t=1). If the verification says "YES", then I know A has the prize. Then, I can guess A at t=1, getting $1 * 0.2^1 = 0.2$. If the verification says "NO", then the prize is in B or C. Then, I have two bags left. But I need to think about the next steps.

Wait, but verifying takes a timestep. So, if I verify at t=0, then the next step is t=1. Let's think about the expected value of verifying. Suppose I verify A at t=0. Then, the result is YES with 1/3, NO with 2/3.

If YES: Then at t=1, I can guess A and get 0.2. If NO: Then the prize is in B or C. Now, at t=1, I have two bags left. If I guess at t=1, the expected reward would be $(1/2) * 0.2^1 + (1/2) * 0 = 0.1$. Alternatively, I can verify another bag. Let's say I verify B at t=1. Then, if it's YES, I can guess B at t=2 (reward $0.2^2 = 0.04$), else verify C at t=2, etc. But this might take longer.

Alternatively, maybe it's better to guess at t=1 if I have two bags left. The expected value would be 0.1 as above. But if I verify, maybe I can get more information. Let me calculate the expected value of verifying A at t=0.

So, if I verify A at t=0, then at t=1, if it's YES (prob 1/3), I get 0.2. If NO (prob 2/3), then I have two bags left. Then, the expected value from that point is the maximum between guessing or verifying again. Let's assume that the optimal strategy is to guess when there are two bags left. Then, the expected value would be 0.1 as before. So the total expected value of verifying A at t=0 is $(1/3) * 0.2 + (2/3) * 0.1 = 0.2/3 + 0.2/3 = 0.4/3 \approx 0.133$. Which is less than the 0.333 from guessing at t=0. So, it seems better to guess at t=0.

Wait, but maybe I'm missing something. Because if I verify A at t=0, and it's NO, then at t=1, I can verify another bag. Let's say I verify B at t=1. Then, if it's YES (prob 1/2), then I can guess B at t=2 (reward $0.2^2 = 0.04$). If NO (prob 1/2), then the prize is in C, so I can guess C at t=2, getting 0.04. So the expected value here would be 0.04. So, the total expected value if I verify A at t=0, then B at t=1, would be $(1/3) * 0.2 + (2/3) * [(1/2) * 0.04 + (1/2) * 0.04] = (1/3) * 0.2 + (2/3) * 0.04 = 0.0667 + 0.0267 \approx 0.0933$. Which is worse than the 0.333 from guessing at t=0. So, in that case, guessing at t=0 is better.

Alternatively, maybe there's a better strategy. Let me think again. Suppose I verify one bag, and if it's no, then verify another. But with discounting, the later rewards are smaller. So, perhaps the optimal strategy is to guess immediately. Because the expected reward from guessing at t=0 is 1/3, which is higher than the expected value of verifying and then proceeding. So, maybe the best action is to GUESS at t=0.

But wait, what if I verify one bag and then guess? Let's say I verify A at t=0. If it's YES, then I can guess at t=1 (0.2). If it's NO, then I have two bags left. Then, if I guess at t=1, the expected reward is 0.1. So the total expected value is $(1/3) * 0.2 + (2/3) * 0.1 = 0.2/3 + 0.2/3 = 0.4/3 \approx 0.133$, which is less than 0.333. So, indeed, guessing at t=0 is better.

Alternatively, if I verify two bags, but that would take more steps. For example, verify A at t=0, then B at t=1. If both are NO, then the prize is in C. Then at t=2, guess C and get $0.2^2 = 0.04$. So the expected value would be (probability that A is NO and B is NO) * 0.04. The probability that A is NO is $2/3$, then B is NO is $1/2$, so $2/3 * 1/2 = 1/3$. So expected value is $1/3 * 0.04 = 0.0133$. Which is worse than guessing at t=0. So, that's even worse.

So, the conclusion is that guessing at t=0 gives a higher expected value than any strategy involving verification. Therefore, the optimal action is to GUESS one of the bags. But since all bags are symmetric, it doesn't matter which one. So, the answer should be GUESS A, B, or C. But the user probably expects me to choose one. Let me check if there's any other consideration. For example, maybe the discount factor is 0.2, which is quite low. So, waiting to get more information might not be worth it. So, the best action is to GUESS now. </think>

GUESS A

Figure 7: Example interaction trace on a 3-bag Pandora's Box instance with priors (0.04, 0.68, 0.28) and discount factor $\gamma = 0.2$, where the model is not given access to the prior probabilities. In this setting, the model implicitly treats the bags as equally likely and follows a suboptimal strategy that deviates from the optimal policy.

Case Study: Pandora's Box Problem, CTA-PROMPTED

Problem Parameters
- Bag Labels: A, B, C
- Prior Probabilities: A: 0.04, B: 0.68, C: 0.28
- Discount Factor r: 0.2

Interaction Trace
Turn 0
User: {Instruction instantiated with the problem parameters}
Assistant:
<think> Okay, let's see. I need to figure out whether to verify or guess in this scenario. The problem has three bags: A, B, and C with prior probabilities 0.04, 0.68, and 0.28 respectively. The discount factor is 0.2.

First, I should remember that each VERIFY action costs a timestep, and GUESS ends the game immediately. The reward for a correct guess is discounted by r^t, where t is the timestep. So, if I guess at $t = 0$, the discount is 1 (since $0.2^0 = 1$). But if I verify, I spend a timestep and then maybe have more information to make a better decision.

The key here is to balance between getting more information (which might lead to a higher reward) and the risk of waiting too long, which could reduce the discounted reward.

Let me think about the expected value of guessing immediately versus verifying.

If I guess at $t = 0$, the expected reward would be the sum of the probabilities of each bag multiplied by the discount factor. Wait, no. Wait, the reward is 1 if the guessed bag has the prize, 0 otherwise. So the expected reward for guessing at t=0 would be the probability of the chosen bag having the prize. But since I can choose any bag to guess, I should pick the one with the highest prior probability.

Looking at the priors: B is 0.68, C is 0.28, A is 0.04. So if I guess B, the expected reward is 0.68 * 1 (since if it's correct, I get 1, else 0). But since the discount is $0.2^0 = 1$, the expected reward is 0.68.

Alternatively, if I verify a bag, I can get more information. Let's say I verify B first. If I verify B, then I spend a timestep (t=1). Then, depending on the result, I can decide whether to guess or verify again.

...

But the problem is that the discount factor is 0.2. So, each time you wait, the reward is discounted. So, the longer you wait, the less valuable the reward becomes.

So, the question is whether the information gained from verifying is worth the cost of waiting.

Let me think about the expected value of verifying versus guessing.

If I guess at t=0, the expected reward is the maximum prior probability, which is 0.68.

If I verify a bag, say B, then I spend one time step (t=1). Let's calculate the expected reward if I verify B.

After verifying B, there are two possibilities:

1. The result is YES (probability 0.68). Then, I know B has the prize. So, I can guess B at t=1, which would give me a reward of $1 * 0.2^1 = 0.2$.

2. The result is NO (probability $1 - 0.68 = 0.32$). Then, B does not have the prize. Now, I have to decide whether to guess or verify another bag.

But if I get NO on B, then the remaining bags are A and C. Their probabilities would be updated based on the fact that B is not the correct one. Wait, but the problem says that the bags are independent. Wait, actually, the problem states that the prior probabilities are for each bag. So, if B is not the correct one, then the prize is in A or C. However, the original priors are A:0.04, B:0.68, C:0.28. But if we know that B is not the correct one, then the total probability of the prize being in A or C is $0.04 + 0.28 = 0.32$. So, the posterior probabilities would be A: $0.04/0.32 = 0.125$, and C: $0.28/0.32 = 0.875$.

So, in this case, if we get a NO on B, then the optimal choice would be to guess C, which has a 0.875 chance of being correct. Then, the reward would be $0.875 * 0.2^1$ (since t=1).

But wait, after verifying B at t=0, we spend a timestep (t=1), and then we can guess. So, if we get NO on B, then we can guess C at t=1, which gives us $0.875 * 0.2 = 0.175$.

Alternatively, if we get YES on B, then we can guess B at t=1, which gives $1 * 0.2 = 0.2$.

So, the expected reward of verifying B is:

$0.68 * 0.2 + 0.32 * 0.175 = 0.136 + 0.056 = 0.192$.

Compare that to the expected reward of guessing immediately (0.68).

So, 0.68 is higher than 0.192. Therefore, verifying B is worse than guessing immediately.

</think>
Action: GUESS B

Figure 8: Example model reasoning trace on a 3-bag Pandora's Box instance with priors (0.04, 0.68, 0.28) and discount factor $\gamma = 0.2$. The model explicitly compares the expected value of immediate guessing versus verification and then chooses to guess B immediately, which is the optimal strategy in this case. Key reasoning steps, including the explicit comparison between action value and exploration cost, are highlighted in yellow. This example illustrates that when priors are provided explicitly, the model can reason about uncertainty and exploration cost to select an optimal strategy.

F EXPERIMENT DETAILS FOR SWE-CLARIFY

Formalization Given a vague repository instruction $\mathbf{x}$, a user interaction history $\mathbf{h}$, and a discount factor $\gamma \in [0, 1]$, the agent chooses from $\mathcal{A} = \{\text{act}, \text{ask-up}, \text{ask-mid}\}$. The agent may implement directly, send one clarification message of at most three questions before working, or explore the repository before sending the message. Each branch then runs to completion and produces a patch. The reward is $R = \gamma^t \mathbb{I}[\text{tests pass}]$, where $t = 0$ for direct action and $t = 1$ for either clarification action.

Latent structure and prior The probability that a branch succeeds depends on both task resolvability and the user's latent interaction profile. We denote the direct-action success probability by p_{act} and the success rates after upfront and mid-work clarification by p_{up} and p_{mid}. The user profile includes question timing $z_{\text{tim}} \in \{\text{upfront}, \text{ongoing}\}$ and response style; upfront-type users refuse mid-work questions, while short-response users generally answer only the first question. Given $q_{\text{up}} = p(z_{\text{tim}} = \text{upfront} \mid \mathbf{h})$, the corresponding expected values are

$$V(\text{act}) = p_{\text{act}}, \qquad V(\text{ask-up}) = \gamma p_{\text{up}},$$
$$V(\text{ask-mid}) = \gamma[q_{\text{up}} p_{\text{mid,ref}} + (1 - q_{\text{up}}) p_{\text{mid,ans}}] .$$

Prior estimation We elicit a verbalized direct-action confidence from Claude Haiku 4.5 using the vague instruction, repository name, and user context, without repository access. Platt scaling fitted on the 65 validation instances gives $\hat{p}_{\text{act}}$ and reduces ECE from 0.077 to 0.049 (AUROC 0.62). The user context contains the benchmark's overall user-model summary and three task-relevant excerpts retrieved from the 20-session history. A separate LLM call over this context estimates profile probabilities, including $\hat{p}(z_{\text{tim}} = \text{upfront} \mid \mathbf{h})$ and the probability of short responses. Validation branch outcomes provide $\hat{p}_{\text{up}} = 0.62$, $\hat{p}_{\text{mid,ans}} = 0.43$, and $\hat{p}_{\text{mid,ref}} = 0.24$. These estimators never observe the original issue, gold patch, hidden tests, or true profile identity.

Decision inputs Both PROMPTED and CTA-PROMPTED receive the same instruction, user context, action set, reward definition, γ, environment rules, and validation branch-success rates. CTA-PROMPTED additionally receives the per-instance $\hat{p}_{\text{act}}$ and user-profile probabilities in a prior block. Thus, the two methods differ only in whether these per-instance priors are made explicit.

Benchmark and splits Stateful SWE-bench (Zhou et al., 2025b) rewrites SWE-bench Verified issues (Jimenez et al., 2024) as vague instructions and pairs them with user profiles and 20-session interaction histories. We use the 13 of 15 profiles with complete histories, with 156 test instances (12 per profile), 65 validation instances, and 12 pilot instances used only for debugging. We exclude instances with unbuildable Docker images or estimated runtimes above four hours.

Agent and user Claude Haiku 4.5 runs a CodeAct-style loop for at most 60 steps in the official per-instance Docker image, with networking disabled. Success requires all FAIL_TO_PASS and PASS_TO_PASS tests to pass. Claude Sonnet 4.5 simulates the user from the profile and original issue; upfront-type users refuse questions asked mid-work. All methods use the same policy to generate the clarifying question.

Evaluation protocol For each instance, we run and cache three branches: act directly, ask upfront, and ask mid-work. A decision method selects one cached branch for each $(\text{instance}, \gamma)$ pair, giving paired comparisons without rerunning the coding agent.

Decision policies We also report NEVER ASK, ALWAYS ASK (upfront), a cost-blind CONFIDENCE GATE tuned on validation ($\theta = 0.33$), and an EV CONTROLLER that selects the highest of the three expected values above using CTA's priors. The TOM CONSULTATION baseline replaces CTA's prior block with the released ToM-SWE user analysis (Zhou et al., 2025b). The ORACLE selects the best cached branch for each instance and γ.

Results We use a paired bootstrap over matched evaluation rows. CTA-PROMPTED improves over PROMPTED by 0.012 reward overall ($p = 0.007$) and by 0.050 at $\gamma = 0.15$ ($p = 0.009$). Their rewards are equal for $\gamma \geq 0.7$, when both ask almost always. The EV CONTROLLER scores 0.006 higher than CTA, but the difference is not significant ($p = 0.09$).

Table 8: Reward and ask rate on 156 SWE-CLARIFY test instances for each γ.

		Reward					
Method	Mean	γ=0.15	0.3	0.5	0.7	0.9	1.0
NEVER ASK	0.154	0.154	0.154	0.154	0.154	0.154	0.154
ALWAYS ASK	0.383	0.097	0.194	0.324	0.453	0.583	0.647
CONFIDENCE GATE	0.383	0.112	0.204	0.327	0.450	0.573	0.635
EV CONTROLLER	0.398	0.159	0.219	0.327	0.453	0.583	0.647
PROMPTED	0.380	0.095	0.199	0.308	0.449	0.583	0.647
TOM CONSULTATION	0.381	0.106	0.191	0.321	0.449	0.571	0.647
CTA-PROMPTED	0.392	0.145	0.206	0.324	0.449	0.583	0.647
ORACLE	0.484	0.237	0.321	0.433	0.544	0.656	0.712
		Ask rate					
Method	Mean	γ=0.15	0.3	0.5	0.7	0.9	1.0
CONFIDENCE GATE	0.95	0.95	0.95	0.95	0.95	0.95	0.95
EV CONTROLLER	0.80	0.22	0.65	0.95	1.00	1.00	1.00
PROMPTED	0.98	0.94	0.98	0.97	1.00	1.00	1.00
TOM CONSULTATION	0.98	0.97	0.97	0.97	0.99	0.99	0.99
CTA-PROMPTED	0.88	0.55	0.79	0.96	0.99	1.00	1.00
ORACLE	0.56	0.56	0.56	0.56	0.56	0.56	0.56

Prior ablation Mean reward decreases from 0.392 with calibrated priors to 0.388 with uniform priors, 0.309 with 40% of the prior mass flipped, and 0.167 with adversarial priors. The noisy and adversarial variants both underperform PROMPTED (0.380); uniform priors slightly outperform it ($p = 0.04$).

ToM-SWE baseline TOM CONSULTATION obtains 0.381 reward, compared with 0.380 for PROMPTED ($p = 0.83$), and asks on 97–99% of instances across γ. It therefore does not improve cost adaptation.

G DATASET CONSTRUCTION DETAILS FOR TASK: CODING WITH SELECTIVE TESTING

This appendix describes the oracle filename-to-format model and the procedure used to generate the coding task dataset FILEREADING.

Each task consists of a structured filename, a CSV file generated under a latent formatting configuration, and an associated query. Filenames combine a small set of indicative tokens (e.g., `_eu`, `_tab`, `_sas`, `_cn`) with additional irrelevant strings, inducing a prior over possible parsing configurations.

To capture scenarios in which filename cues provide informative signals about file formats, we construct a synthetic dataset of CSV-based question-answering tasks with structured filenames. Each filename is generated by combining a small set of indicative tokens (e.g., `_eu`, `_tab`, `_sas`, `_cn`) with additional irrelevant strings.

We define an oracle filename-to-format model

$$\mathcal{M}_{\text{oracle}} : n \;\mapsto\; p(\mathbf{z} \mid n),$$

which maps a filename n to a prior distribution over formatting configurations $\mathbf{z} = (z_d, z_q, z_s)$. For example, filename tokens like `_tsv` substantially increase the probability of a tab delimiter relative to a default comma delimiter.

We then sample the true formatting configuration $\mathbf{z}^* \sim p(\mathbf{z} \mid n)$ and generate the corresponding CSV content and task query.

The dataset is constructed so that the correct answer is obtainable only when the file is parsed with the correct configuration; incorrect formatting assumptions lead to parsing failures or misaligned

columns. Filenames are represented by four binary features, yielding $2^4 = 16$ distinct filename feature configurations and corresponding prior distributions over formatting attributes. We generate 2,000 task instances, each consisting of a filename, a CSV file generated under a sampled formatting configuration, and an associated query. The dataset is split into 1,400 training examples, 300 validation examples, and 300 test examples. Details of the oracle filename-to-format model, feature templates.

Latent Formatting Variables Each task instance is associated with a latent formatting configuration

$$\mathbf{z} = (z_d, z_q, z_s),$$

where $z_d \in \{$ `,` , `;` , `\t` $\}$ denotes the delimiter, $z_q \in \{$ `"` , `'` $\}$ the quote character, and $z_s \in \{0, 1\}$ the number of header rows to skip. The correct answer can be obtained if and only if the file is parsed using the fully correct configuration $\mathbf{z}^*$.

Filename Features We extract four binary features from each filename n, each indicating the presence or absence of a specific substring: `has_eu`, `has_tsv`, `has_sas`, and `has_cn`. Each feature is either on or off, resulting in $2^4 = 16$ possible filename feature configurations. Each configuration corresponds to a distinct prior distribution over formatting attributes induced by the oracle model.

Oracle Filename-to-Format Model We define an oracle filename-to-format model

$$\mathcal{M}_{\text{oracle}} : n \mapsto p(\mathbf{z} \mid n),$$

which maps a filename n to a prior distribution over formatting configurations $\mathbf{z} = (z_d, z_q, z_s)$. For each of the 16 possible filename feature configurations, the model induces a corresponding prior over formatting attributes. The prior factorizes as

$$p(\mathbf{z} \mid n) = [p(z_d \mid n), p(z_q \mid n), p(z_s \mid n)],$$

where each factor is parameterized as a log-linear model over the filename features.

Sampling and File Generation For each filename n, we sample a formatting configuration $\mathbf{z} \sim p(\mathbf{z} \mid n)$ from the oracle model $\mathcal{M}_{\text{oracle}}$. We then generate a CSV file whose content conforms to $\mathbf{z}$. The data are constructed such that incorrect parsing—due to an incorrect delimiter, quote character, or number of skipped rows—either produces malformed outputs or prevents access to the correct answer.

Task Instances and Splits Each task instance consists of a filename n, a generated CSV file, and a query requiring the agent to compute an answer from the file. We generate 2,000 task instances in total, split into 1,400 training examples, 300 validation examples, and 300 test examples.

H Oracle Strategy for Pandora's Box Problem

Algorithm 1 presents the optimal policy for the Pandora's Box problem. In this section, we prove its optimality.

We begin by characterizing the structure of an optimal policy. At any state, let S denote the remaining set of boxes and let $q_i(S)$ denote the posterior probability that box $i \in S$ contains the prize. An optimal policy only needs to consider the box with maximum posterior probability at each step.

First, if the agent commits, the expected reward equals the posterior success probability of the chosen box. Hence, committing to any box other than a maximum-posterior box is suboptimal. Second, verification is beneficial only insofar as it increases the probability of early termination before further discounting. Verifying a higher-posterior box increases the chance of immediate success and therefore weakly dominates verifying a lower-posterior box. Consequently, it suffices to consider the box $i^\star \in \arg\max_{i \in S} q_i(S)$ at each decision step.

It remains to determine whether the agent should commit to $i^\star$ immediately or verify it first. Let $q = q_{i^\star}(S) = \max_{i \in S} q_i(S)$ be the posterior probability of the most likely box $i^\star$.

Algorithm 1 Oracle policy for Pandora's box with n boxes

```
function SOLVE(S)
    if |S| = 1 then
        return (1, COMMIT(only element of S))          ▷ Base case: Final box selection
    end if

    W ← Σ_{j∈S} p_j
    i* ← arg max_{i∈S} p_i                             ▷ Select candidate with highest success posterior
    q ← p_{i*} / W

    V_guess ← q                                        ▷ Expected value if committing to box i* now
    (V_fail, _) ← SOLVE(S \ {i*})                      ▷ Recurse to find value if box i* is empty
    V_verify ← γ(q + (1 − q) · V_fail)                 ▷ The value of verifying i* first

    if V_guess ≥ V_verify then
        return (V_guess, COMMIT(i*))
    else
        return (V_verify, VERIFY(i*))
    end if
end function
```

If the agent commits immediately, the expected value is

$$V_{\text{guess}}(S) = q.$$

If it verifies $i^\star$, then with probability q verification succeeds and yields reward 1, and with probability $1 - q$ it fails and the problem reduces to the smaller set $S \setminus \{i^\star\}$. Since verification incurs one multiplicative discount factor γ, the expected value of verifying is

$$V_{\text{verify}}(S) = \gamma\Big(q + (1 - q)\, V(S \setminus \{i^\star\})\Big).$$

Therefore, optimality implies the Bellman recursion

$$V(S) = \max\big\{V_{\text{guess}}(S),\ V_{\text{verify}}(S)\big\},$$

with base case $V(\{i\}) = 1$. This recursion is exactly implemented by Algorithm 1.

I GRPO Training Details

We use the GRPO implementation of the veRL library (Sheng et al., 2024; Zheng et al., 2025) for the RL training. The model is trained with a group size of $n = 8$ and a KL coefficient $\beta = 0.01$. During rollout, tokens are sampled with a temperature of 1.0. We use a learning rate of 1×10^{-6} and train with a global batch size of 16. We use a micro-batch size of 2 for model updates and a micro-batch size of 4 for experience. We conduct GRPO training on Qwen3-8B using a single node with 8 NVIDIA A100 GPUs.

J Prompt Templates

J.1 Prompts for Pandora

Prompt templates used for Pandora are shown in Figure 9.

J.2 Prompts for QA

Prompts used in QA are provided in Figure 10.

Prompts: Pandora's Box

SYSTEM

```
You are a rational agent tasked with solving sequential
decision-making problems under uncertainty.  You are given a set
of options (bags) with prior probabilities of containing a prize
with value 1.  You can either VERIFY an option to get information
(YES/NO) or GUESS an option to end the game and collect the
reward.
- Each VERIFY action consumes one timestep.
- The reward for a correct GUESS is discounted by a factor r^t, where
t is the timestep when you GUESS.
- You must balance information gathering (VERIFY) with timely
exploitation (GUESS) to maximize expected discounted reward.
Always respond with exactly one action token per step, using the
format:
VERIFY <Option> or GUESS <Option>.
```

Instruction

```
--- NEW GAME ---
TIMESTEP: t=0

PROBLEM PARAMETERS:
- Bag Labels: {labels_str}
- Prior Probabilities: {priors_str}
  //only provided to the model in CTA- settings
- Discount Factor r: {r}

Choose your action.
```

Figure 9: Prompt template for Pandora's Box setting.

Prompts: QA with Optional Retriever

SYSTEM
You are a rational agent tasked with answering factual questions under uncertainty. At each step, you can either directly answer the question or retrieve additional context before answering.
Available actions:
- RETRIEVE: request a related context to consult before deciding your final answer. This consumes one timestep.
- ANSWER: <your short factual answer>, provide your final answer and end the interaction.

Each question comes with your estimated probabilities of answering correctly:
- p_no_context: probability you can answer correctly without retrieving.
- p_with_context: probability you can answer correctly after retrieval.

Your goal is to maximize expected discounted reward:
Reward = $r^t *$ correctness, where t is the timestep when you issue ANSWER and correctness $\in \{0,1\}$.

Retrieval information:
If you choose RETRIEVE, your expected answer accuracy after retrieval is p_with_context. If you choose not to retrieve, you must rely only on your current knowledge.
Be deliberate — retrieving may improve accuracy but reduces reward due to time discounting. Balance speed and correctness carefully.
Always respond with exactly one action token per step, using the format:
RETRIEVE or ANSWER: <short factual answer>.

Instruction - turn 1

```
-- NEW QUESTION --
TIMESTEP: t=0
Question:  question
Parameters:
- Discount factor (r):  {r}
- Success probability with retrieval (p_with_context):
{p_with_context}

Choose your action:
RETRIEVE or ANSWER: <short factual answer>.
```

Instruction - turn 2

```
TIMESTEP: t=1
You have retrieved the following context:
{context}

Question:  {question}
Now decide whether to answer:
Respond with:
ANSWER: <short factual answer>
```

Figure 10: Prompt templates for QA.

System Prompt: FILEREADING

You are an agent tasked with writing code to fulfill an instruction about a CSV file (e.g., answering a question using its contents). Your goal is to produce a correct answer while efficiently using available resources, as measured by discounted reward.

The exact CSV formatting may not be fully known. In practice, you can either proceed using reasonable default assumptions about the format, or run unit tests to verify specific formatting details you are unsure about before committing to a final answer.

Allowed actions (choose exactly ONE per turn):

1) UNIT_TESTS
Run unit tests to debug CSV formatting assumptions. Unit test outputs are perfectly reliable.
Available unit tests:

- `test_delimiter(path)` → {',', ';', '\t'}
- `test_quotechar(path)` → {'"', " ' "}
- `test_skiprows(path)` → {0, 1}

Format (NO code fences):
`UNIT_TESTS: test_delimiter("file.csv"), test_quotechar("file.csv")`

You may include multiple unit tests in a single UNIT_TESTS action. Each individual unit test counts toward the total number of unit tests used.

2) CODE
Write Python code toward solving the task using your current assumptions about the CSV format.

- Enclose code in ```` ```python ... ``` ````
- You may import pandas as pd and read the file with:
  `pd.read_csv(filepath, delimiter=..., quotechar=..., skiprows=...)`
- Do NOT print the entire CSV.
- If your code computes the final result, print it to stdout so it can be read from the output.

After submission, the code will be executed and its stdout and stderr will be returned. You may use this feedback to extract the answer, debug, run additional unit tests, refine, or write additional CODE.

3) ANSWER
Provide the final answer to the task and end the conversation.
Format exactly: `ANSWER: <your_answer>`
The conversation ends immediately after you provide ANSWER.

Reward:

- Let U be the total number of unit tests used.
- Let C be the total number of CODE actions taken.
- Final reward = correctness $\times (d_{\text{unit}})^U \times (d_{\text{code}})^C$.
- Discount factors represent cost multiplicatively.
- A smaller discount factor means a MORE expensive action.
- If $d_{\text{code}} = d_{\text{unit}}^k$, one CODE attempt costs about as much as k UNIT_TESTS.

Figure 11: System prompt for FILEREADING.

J.3 PROMPTS FOR FILEREADING

Prompts used in the FILEREADING setting are provided in Figures 11, 12, 13, and 14.

J.4 PROMPTS FOR SWE-CLARIFY

The decision prompts used for SWE-CLARIFY are shown in Figures 15 and 16. The population success rates are provided to both methods; the bracketed prior block is provided only to CTA-PROMPTED.

System Prompt: FILEREADING (continuation)

General guidance:

- Start from reasonable default beliefs about the CSV format based on common conventions or provided likelihoods.
- Both UNIT_TESTS and CODE are costly actions; neither should be treated as free.
- Use UNIT_TESTS to reduce uncertainty when the expected benefit outweighs their cost.
- Use CODE to make progress toward solving the task, but recognize that failed or repeated CODE attempts are also costly.
- Decide when it is better to verify assumptions with UNIT_TESTS versus attempting CODE earlier, taking into account your confidence and the relative cost of these actions.
- Decide rationally how much debugging and iteration is worthwhile before committing to a final ANSWER.

Figure 12: Continuation of the system prompt for FILEREADING.

Instruction Prompt Template for FILEREADING (without estimated prior)

You are given a CSV file `{csv_name}`.

Your task: `{task_description}`

Additional context:

- No format likelihoods are provided.
- Make reasonable default assumptions about the CSV format based on common conventions, unless you choose to verify them with unit tests.

Reward parameters:

- Unit test discount d_{unit}: `{d_unit}`
- Code iteration discount d_{code}: `{d_code}`

Constraints:

- You should never print all rows of the CSV or you will get zero reward.
- You may use UNIT_TESTS, CODE, or ANSWER as described in the system instructions in any order; only the final ANSWER ends the conversation.
- Incorrect intermediate CODE does not end the episode; only the final ANSWER determines correctness.

Figure 13: Instruction prompt template specifying the CSV task, reward parameters, and constraints provided to the agent.

Instruction Prompt Template for FILEREADING (with CTA)

You are given a CSV file `{csv_name}`.

Your task: `{task_description}`

Additional context:

- Estimated format likelihoods are provided below.
- These likelihoods reflect how likely each formatting option is in practice and can be used as default assumptions.

Format likelihoods:
`{prior}`

Reward parameters:

- Unit test discount d_{unit}: `{d_unit}`
- Code iteration discount d_{code}: `{d_code}`

Constraints:

- You should never print all rows of the CSV or you will get zero reward.
- You may use UNIT_TESTS, CODE, or ANSWER as described in the system instructions.

Figure 14: Instruction prompt template with estimated CSV format likelihoods, enabling the agent to use probabilistic defaults when trading off unit tests, code execution, and early commitment.

System Prompt: SWE-CLARIFY Decision

You are a rational software engineering agent deciding how to start a task under uncertainty. The user's instruction may be vague, and the user has latent interaction preferences that affect whether questions are answered.

Available actions (choose exactly one):

- `ACT`: implement directly without asking anything.
- `ASK_UPFRONT`: send one clarification message of at most three questions before starting work, then implement.
- `ASK_MIDWORK`: explore the code first, send one clarification message during the work, then finish implementing.

Environment dynamics:

- Users who prefer upfront clarification do not answer questions asked mid-work.
- Users who prefer short responses generally answer only the first question, in under 15 words.

Objective: maximize

$$R = \gamma^t \times \text{success},$$

where $t = 0$ for `ACT`, $t = 1$ for either asking action, and success indicates that the final patch passes the hidden tests.

Success rates measured on validation tasks:

- `ACT`: depends on how resolvable the instruction is from context.
- `ASK_UPFRONT`: `{p_up}`.
- `ASK_MIDWORK`: `{p_mid_ans}` if answered and `{p_mid_ref}` if refused.

Respond with exactly one action token at the end: `ACT`, `ASK_UPFRONT`, or `ASK_MIDWORK`.

Figure 15: Shared decision prompt for SWE-CLARIFY.

Instruction Prompt Template for SWE-CLARIFY

`-- NEW TASK --`
Project: `{repository}`
The user's instruction:
`{vague_instruction}`

User context from previous sessions:
`{user_model_summary}`
`{retrieved_history_excerpts}`

Parameters:

- Discount factor γ: `{gamma}`

CTA-Prompted prior block (omitted for Prompted):

- Probability that direct implementation succeeds: `{p_direct}`
- Probability that the user answers only upfront questions: `{p_upfront}`
- Probability that the user gives short answers and answers only the first question: `{p_short}`

Choose your action:
`ACT`, `ASK_UPFRONT`, or `ASK_MIDWORK`.

Figure 16: Task prompt for SWE-CLARIFY. PROMPTED and CTA-PROMPTED receive identical prompts except for the displayed prior block.

Case study: Example thinking trace of RL model (without estimated priors) ($\rho = 4.0$)

Content:

<think>
Okay, let's tackle this problem. The user has a CSV file named 'race_tsv_sas.tsv' and they want the minimum salary value, excluding any None entries. The file's name ends with .tsv, which usually stands for Tab-Separated Values. So, the delimiter is likely a tab character. But I should confirm that.
First, I need to figure out the CSV format. Since it's a TSV, the delimiter is probably a tab. But maybe the file uses a different delimiter. To be safe, I should run a unit test to check the delimiter. The `test_delimiter` function can help here. If the delimiter is a tab, then the test should return '\t'.
Next, the quote character. In TSVs, sometimes fields are enclosed in quotes, but it's not always the case. The default quotechar for pandas is double quotes, but maybe this file uses single quotes. Testing with `test_quotechar` would help.
Also, sometimes CSV files have header rows that need to be skipped. The `test_skiprows` function can check if there's a need to skip rows. But since the file is a TSV, maybe the first row is the header. However, without knowing the exact structure, it's safer to assume that the first row is the header and set `skiprows=0`. But maybe the file has some leading rows to skip.
But since the user mentioned it's a TSV, I can start by assuming the delimiter is a tab. Let me run the `test_delimiter` first. If the delimiter is a tab, then I can proceed. If not, I'll adjust accordingly.
Once the delimiter is confirmed, I can use pandas to read the CSV. Then, check the 'salary' column, filter out None values, and find the minimum.
But to avoid errors, maybe I should first run the unit tests to confirm the delimiter and quotechar. Let's start with that.
</think>

Action:

```
UNIT_TESTS: test_delimiter("race_tsv_sas.tsv"),
test_quotechar("race_tsv_sas.tsv")
```

Figure 17: Example reasoning trace of an RL-trained model without explicit prior conditioning in the CSV exploration task. Despite operating under the same high relative code cost setting ($\rho = 4.0$), the model defaults to verification-first behavior based on surface cues (e.g., file extension) and does not explicitly reason about uncertainty or cost trade-offs, illustrating a lack of adaptive decision-making compared to the CTA-RL model.

K Case Study: Cost-Aware Decision Traces in FileReading with CTA-RL and RL

Figures 17 and 18 compare representative traces under a high relative code cost setting ($\rho = 4.0$). The RL model (trained without conditioning on explicit priors) tends to default to running unit tests before attempting any code, and does not explicitly reason about the relative costs of UNIT_TESTS versus FileReading when choosing its next action. In contrast, the CTA-RL model exhibits the intended cost-aware behavior: it reasons about both (i) uncertainty over the CSV format and (ii) the relative cost of unit tests and code execution, and uses these factors to decide whether verification is worth performing before committing to a code attempt.

Case study: Example thinking trace of CTA-RL model ($\rho = 4.0$)

Content:

<think>
Okay, let's tackle this problem. The task is to compute the average of the 'salary' column from the CSV file 'must_eu.csv', excluding any None values. The challenge is figuring out the correct CSV format parameters to read the file properly.
First, I need to determine the delimiter, quotechar, and skiprows parameters. The format likelihoods are given, so I should start with the most probable options. The delimiter is most likely to be ';' with a probability of ∼0.85. The quotechar is probably '"' with 0.589, and skiprows is 0 with 0.607. But I'm not 100% sure, so maybe I should run some unit tests to confirm.
But wait, the reward parameters are important. Each unit test costs 0.77 per use, and code attempts cost 0.3515 per use. Since $d_{\text{code}} = d_{\text{unit}}^4$, one code attempt is equivalent to 4 unit tests. So, if I'm uncertain, it's cheaper to run unit tests first. However, if I can make an educated guess, maybe I can proceed with code.
Let me think: the delimiter is most likely ';', so maybe I can try reading the file with that. But if the actual delimiter is different, the code will fail. Alternatively, running a unit test for the delimiter would be better. Let's check the delimiter first.
So, I'll run UNIT_TESTS to check the delimiter. The `test_delimiter` function will tell me if it's comma, semicolon, or tab. Since the likelihood is high for semicolon, but I need to confirm. Let's do that.
Once I know the delimiter, I can check the quotechar. The quotechar is either '"' or ''', with higher likelihood for '"' (0.589). But again, maybe a unit test is better. However, if I proceed with code, maybe I can handle the possible quotechar. But if the code fails, I'll have to adjust.
Alternatively, maybe I can proceed with the most likely parameters and see. Let's try that. Let's assume delimiter is ';', quotechar is '"', and skiprows 0. Then, read the CSV with these parameters, compute the average of salary, skipping None values.
But if the actual delimiter is different, the code will fail. So, perhaps it's better to run unit tests first. Let's start with the delimiter.
So, the first action is to run UNIT_TESTS to determine the delimiter and quotechar. Let's do that.
</think>

Action:

```
UNIT_TESTS: test_delimiter("must_eu.csv"),
            test_quotechar("must_eu.csv")
```

Figure 18: Example reasoning trace of the CTA-RL model on the FILEREADING task ($\rho = 4.0$), illustrating cost-aware trade-offs between unit tests and code execution under a high relative code cost setting, while jointly reasoning about format uncertainty.

L CASE STUDY: COST-AWARE CLARIFICATION IN SWE-CLARIFY

Figures 19 and 20 compare the methods on the same Django test instance at $\gamma = 0.15$. Both branches pass the hidden tests and produce the same patch, isolating the effect of the ask-versus-act decision.

Case study: PROMPTED asks upfront ($\gamma = 0.15$)

Instruction:
"Sitemap generation crashes when it's empty; seems tied to how we handle last-updated info."

Reasoning excerpt:
`<think>`
The issue is somewhat vague and could stem from several places in Django's sitemap framework. To investigate efficiently, I need the exact error, the relevant context, and a minimal reproduction. Given the task's ambiguity, a targeted clarification before working will likely yield better results.
`</think>`

Action:
`ASK_UPFRONT`

Exchange excerpt:
Agent: *"What's the current crash behavior—exception type, stack trace, or error message?"*
User: *"ValueError on empty sitemap with callable lastmod—max() gets empty sequence."*

Outcome: hidden tests pass; reward $= 0.15$.

Figure 19: PROMPTED recognizes the ambiguity but asks without comparing the benefit of clarification against its cost.

Case study: CTA-PROMPTED acts directly ($\gamma = 0.15$)

Instruction:
"Sitemap generation crashes when it's empty; seems tied to how we handle last-updated info."

Prior block:
`p_direct = 0.28; p_upfront = 0.15; p_short = 0.35`

Reasoning excerpt:
`<think>`
`ACT`: 0.28; `ASK_UPFRONT`: $0.15 \times 0.62 = 0.093$; `ASK_MIDWORK`: 0.060.
The aggressive discount means that clarification's higher success rate is heavily penalized. `ACT` maximizes expected reward despite its lower success probability.
`</think>`

Action:
`ACT`

Outcome: hidden tests pass; reward $= 1.0$.

Figure 20: CTA-PROMPTED explicitly compares the expected values and avoids an expensive clarification. Both methods produce the same fix, changing `except TypeError` to `except (TypeError, ValueError)`.