Extended Abstract

Learning to Explore Through Information-Directed Bayesian Optimal Experimental Design

Lucia Zheng (zlucia@stanford.edu)

Motivation Frontier LLMs struggle on novel, open-ended discovery problems that are genuinely unsolved, have little relevant pretraining signal, and require strategic exploration in noisy information environments at test time. Solving them requires not just *task-level information* but also learning how to gather it, that is, which strategies are effective. We call this *design-level information* uncertainty. This meta-level learning is central to expert performance across domains, yet it remains underexplored in the design of foundation model agents. Standard approaches optimize directly for the task at hand, and these task-focused objectives may forgo actions that, while revealing little about the current task, may nonetheless teach an agent *how to investigate effectively*.

Method We introduce Information-Directed Bayesian Optimal Experimental Design (ID-BOED), a prompting-based action-selection principle for LLM agents. The agent maintains a *task* belief over the solution θ and a *design* belief over which information-gathering strategies are effective, both as flexible natural-language representations revised per step by an approximate Bayesian posterior update in language space. It selects actions by minimizing an information-directed sampling (IDS) ratio $\pi^{\text{ID-BOED}}(s_t) = \arg\min_a \Delta_\theta(a)^2 / I_a(A^*; y)$, trading off exploitation in the task-information space (the task-information regret $\Delta_\theta(a)$ in the numerator) against exploration of effective design strategies (the design expected information gain $I_a(A^*; y)$ about the optimal action in the denominator), the beyond-current-step information that myopic BOED ignores.

Implementation Action selection and belief updating are implemented entirely via structured LLM prompts. Expected information gain is computed in closed form when belief-MDP values are available (the Toy Bandit) and otherwise estimated by the LLM in language space. We instantiate all agents with Gemini 3.1 Flash and compare against No-Op (zero-shot), ReAct (reasoning/self-reflection), and BOED (myopic task-EIG maximization). We validate on two motivating experiments that isolate components (an analytically tractable BOED bandit and Wordle) and three open-ended discovery tasks: KernelBench, EsoLang-Bench, and SCOTUS Cert.

Results ID-BOED achieves the best performance on all three tasks. Its gains over No-Op / ReAct / BOED are significant on KernelBench ($+0.55\times$ / $+0.31\times$ / $+0.22\times$ speedup) and EsoLang-Bench ($+0.27$ / $+0.12$ / $+0.07$ test pass rate), and over No-Op / ReAct on SCOTUS Cert ($+0.37$ / $+0.24$ F1). On EsoLang-Bench's solvable tier, ID-BOED on Gemini 3.1 Flash reaches 22.0%, exceeding the benchmark's reported SOTA (GPT-5.4 xhigh, 16.9%) and roughly doubling its Gemini 3.1 Pro result (11.0%); it is the only method to solve 6 problems (4 Shakespeare, 2 Befunge-98) every baseline misses, and uniquely cracks KernelBench problem 49 (Mamba-2) at $6.11\times$. In the closed-form bandit, ID-BOED attains $4.5\times$–$7.4\times$ lower regret than BOED and Thompson Sampling.

Discussion The advantage compounds over the horizon: ID-BOED invests in design-level actions early then exploits, catching up and surpassing baselines on long-horizon tasks, mirroring the Toy Bandit. The open-ended tasks confirm the Toy Bandit finding that the advantage peaks under high design-prior uncertainty, with ID-BOED's edge over BOED concentrating on the most data-scarce, idiosyncratic EsoLang languages, Shakespeare ($+23.3$ pp test pass rate, p=0.002) and Befunge-98 ($+9.2$ pp, p=0.02). Even myopic BOED beats ReAct and No-Op, and baselines rarely seek design-level information unprompted even when it is available, underscoring the value of an explicit design belief.

Conclusion We highlight a key distinction between gaining information and learning *how* to gain information—a challenge we call strategic exploration. Verbalized Bayesian experimental design in open-ended LLM action spaces is itself novel, and ID-BOED adds a Pareto improvement on top, establishing it as a robust Pareto improvement over BOED across challenging open-ended domains. The method is a simple prompting technique with strong empirical results that, in addition, happens to rest on a principled theoretical foundation connecting Bayesian optimal experimental design and information-directed sampling.

Learning to Explore Through Information-Directed Bayesian Optimal Experimental Design

Lucia Zheng
Department of Computer Science
Stanford University
zlucia@stanford.edu

Abstract

Challenging open-ended domains require efficiently exploring the world to gather information about potential solutions. Humans confronted with novel problems allocate effort not only to learning the correct solution but also to learning effective strategies for obtaining it. While a good strategy aids exploration, uncertainty about the efficacy of competing strategies yields a second tier of exploration. Motivated by this, we formulate sequential information acquisition as planning in a Bayesian decision-making problem whose structure captures both uncertainty about task information and uncertainty about productive information-gathering strategies. Drawing on connections with Bayesian optimal experimental design (BOED) and information-directed sampling (IDS), we introduce Information-Directed Bayesian Optimal Experimental Design (ID-BOED), a prompting-based method for sampling actions based on these dual uncertainties. Through verbal articulations of knowledge and uncertainty, we operationalize both traditional BOED and ID-BOED for LLM agents and demonstrate improvements over baselines like ReAct. By optimizing between solution-level and strategy-level information gathering, ID-BOED achieves the best performance across three open-ended discovery tasks (GPU kernel optimization, esoteric-language programming, and legal outcome prediction), with significant gains over all baselines and a Pareto improvement over BOED.

1 Introduction

"In the fields of observation, chance favors only the prepared mind," Pasteur famously observed in his 1854 lecture at Lille, made more concrete in *The Art of Scientific Investigation* by Beveridge: "a mind prepared by the hypothesis of evolution would make many more significant observations on a field excursion than one not so prepared" [Beveridge, 1957]. Their insights reflect that scientific inquiry requires not just gathering information, but first learning *how* to gather it. Which experimental designs are promising? Which methods are reliable? Which lines of inquiry are productive? This meta-level learning is central to expert performance across domains. Superforecasters invest substantial effort in learning which signals matter and which sources are reliable before gathering task-specific evidence [Tetlock and Gardner, 2015]; and researchers consult meta-analyses that characterize the reliability and applicability of different methodological approaches.

Frontier LLMs struggle on novel, open-ended discovery problems that are genuinely unsolved, have little relevant pretraining signal, and require strategic exploration in noisy information environments at test time. Solving them requires not just *task-level information* but also learning how to gather it, that is, which strategies are effective. We call this *design-level information* uncertainty. Yet, while this methodological preparation is central to human expertise, it remains underexplored in the design of foundation model agents. Standard approaches optimize directly for the task at hand [Yao et al., 2023,

Shinn et al., 2023, Shao et al., 2024]. These task-focused objectives may forgo actions that, while revealing little about the current task, may nonetheless teach an agent *how to investigate effectively* (or *how to explore*), thereby improving the efficiency of all subsequent information gathering. As agents are increasingly used for open-ended real-world problem-solving, where there may be little in-domain training data, learning how to investigate at test time becomes an imperative skill.

So, when should an agent invest in learning how to investigate versus gathering task-specific information? To help answer this, we establish a connection between Bayesian optimal experimental design (BOED) [Chaloner and Verdinelli, 1995] and information-directed sampling (IDS) [Russo and Van Roy, 2014]. Specifically, we cast sequential information acquisition as a Bayesian belief MDP and consider the corresponding Bayesian reinforcement learning (RL) problem, which accounts for uncertainty over belief dynamics, what we call *design beliefs*, representing an agent's understanding of which investigative strategies are effective. In this formulation, BOED corresponds to optimizing immediate task-level information gain; by further leveraging IDS, we extend this and additionally value design-level information that shapes future learning.

Motivated by these insights, we make the following contributions:

- **Leveraging Bayesian RL, we construct a bridge between IDS and BOED that translates to efficient exploration of both design- and task-level beliefs, which we call Information-Directed Bayesian Optimal Experimental Design (ID-BOED).** This connection opens new avenues for theoretical work on learning to explore at test time.
- **We operationalize BOED and ID-BOED for LLM agent exploration within complex, open-ended domains.** These agents verbalize beliefs over task- and design-spaces, maintaining language-based "posteriors" [Arumugam and Griffiths, 2026, Lidayan et al., 2025] over long horizons. Our key idea is to recognize that such verbal articulations of knowledge and uncertainty can be maintained not only with respect to the underlying task (thereby facilitating traditional exploration) but also with respect to the experimental designs available to the agent, further enabling *strategic exploration*.
- **We demonstrate the efficacy of our proposed agents through challenging, real-world exploration problems: GPU kernel optimization, esoteric-language programming, and legal outcome prediction.** Our agents significantly outperform standard self-reflection methods, and ID-BOED in particular boasts a Pareto-improvement over BOED across all tasks. We also find that baseline agents almost never seek design-level information without explicit prompting to do so, even when documents exist that would reveal the exact experiments needed to identify a solution.

2 Related Work

Strategic Exploration of LLM Agents. LLM agents typically explore via prompted reasoning and self-reflection (ReAct [Yao et al., 2023]), with no explicit uncertainty tracking. While early LLM agents mostly relied on such heuristics for exploration [Yao et al., 2023, Shinn et al., 2023, Wang et al., 2023], recent research has shifted toward formalizing information acquisition through more rigorous search and learning objectives. Methods like Li et al. [2025], Putta et al. [2024], Zhou et al. [2024], Antoniades et al. [2025] incorporate Monte Carlo Tree Search and self-correction loops for more complex exploration. BED-LLM [Choudhury et al., 2026] applies Bayesian Optimal Experimental Design (BOED) to LLM agents, maximizing the expected information gain of queries to resolve task ambiguity, but only in simpler, more stylized settings (e.g., 20 Questions game, preference elicitation). Aside from these inference-time methods, Tajwar et al. [2025] train generally curious LLM agents by curating synthetic interaction data across tasks requiring diverse exploration strategies. Two gaps remain: first, BOED is *myopic*, ignoring the long-horizon expected information gain of an action beyond the current step in its action-selection rule; second, these methods are untested on natural-language tasks in *truly* open-ended settings: unsolved problems where even experts disagree on how to solve them.

Test-Time Adaptation and Rigorous Validation. A growing body of work focuses on adapting agents at test time to handle long-horizon open-ended tasks. TTT-discover [Yuksekgonul et al., 2026] applies test-time training on LLM agents to update their weights at inference time based on environment feedback. POPPER [Huang et al., 2025] proposes an agentic framework for sequential

falsification, designing experiments to stress-test hypotheses with strict Type-I error control. These works highlight the necessity of test-time adaptation and rigorous validation as the application of AI gradually shifts from static, fixed-answer tasks to complex, open-ended problems.

Exploration-Exploitation Tradeoff. The exploration-exploitation tradeoff is a fundamental challenge in RL and a central problem when designing agents that can succeed in complex environments. Thompson sampling [Russo et al., 2018], upper confidence bound (UCB) [Auer, 2002], information-directed sampling (IDS) [Russo and Van Roy, 2014, 2018], and posterior sampling for reinforcement learning (PSRL) [Osband and Van Roy, 2017] are standard methods for balancing the tradeoff. Arumugam and Griffiths [2026] implements PSRL in LLM agents, demonstrating that maintaining an explicit posterior belief and leveraging posterior sampling can achieve more efficient exploration. Our ID-BOED framework leverages these insights, applying the IDS principle to a Bayes-Adaptive MDP to explicitly trade off between task-specific exploitation and design-level exploration.

3 Method

3.1 The Bayesian Belief MDP

We treat experimental designs as available information-gathering actions in an agent's action space, and lean upon the standard formulation of Bayesian optimal experimental design (BOED) [Chaloner and Verdinelli, 1995, Rainforth et al., 2024], where a latent parameter of interest $\theta \in \Theta$ is accompanied by a space of possible experiment designs (actions); to any action a, a random observation y is obtained by performing the corresponding experiment in nature, $y \sim p(\cdot \mid \theta, a)$.

To formalize the problem of solving tasks via strategic exploration, we define a Bayesian belief MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, H \rangle$ that encapsulates the evolution of an ideal Bayesian learner's beliefs over time:

- $\mathcal{S} = \Delta(\Theta)$: belief states $s_t \equiv b_t(\cdot)$, posteriors over the latent task parameter θ.
- $\mathcal{A}$: task-specific information-gathering actions, including internal reasoning (THINK), search and document-reading actions, submitting a partial solution for verifiable feedback (SUBMIT), and providing a final response (PROVIDE_FINAL_RESPONSE); the full per-task action spaces are listed in Appendix C.
- $\mathcal{R}(s, a, s') = \mathbb{H}(s) - \mathbb{H}(s')$: entropy reduction, which telescopes to the cumulative information-gain objective of sequential BOED.
- $\mathcal{T}$: the Bayesian belief update $b_{t+1}(\theta) \propto p(y_t \mid \theta, a_t)\, b_t(\theta)$.

The value function induced by any policy in this MDP is a telescoping sum and simplifies to the cumulative information gain $\mathbb{E}[\mathbb{H}(s_1) - \mathbb{H}(s_{H+1})]$, the expected total reduction in entropy between the agent's initial prior and final posterior after H experiments; with $H = 1$ this recovers one-step BOED. Since the transition function $\mathcal{T}$ depends on the unknown θ, the induced Bayes-Adaptive MDP (BAMDP) makes uncertainty about the optimal action A^* explicit. We name the latent that captures this the *design* D: which strategies are effective. The solution to the BAMDP is the Bayes-optimal policy yielding the optimal trade-off between exploration and exploitation; intuitively, in the bandit ($H = 1$) case, an optimal action A^* maximizes one-step information gain as in traditional BOED, so exploitation is commensurate with optimizing information gained about the solution, while good exploration constitutes gaining information about A^*—that is, identifying those actions that reveal *how* to effectively gain information about θ.

3.2 Approximating the Optimal Policy via Information-Directed Sampling

The Bayes-optimal policy is intractable. As approaches based on Thompson Sampling [Thompson, 1933, Russo et al., 2018], while effective and computationally convenient, struggle to seek out information at the cost of incurring regret, our approach instead draws inspiration from information-directed sampling (IDS) [Russo and Van Roy, 2014, 2018], an algorithmic design principle that selects actions by minimizing an information ratio balancing exploration against exploitation. We approximate the Bayes-optimal policy per step by scoring actions by *expected information gain* (EIG): the EIG about a latent X from the observation Y that action a produces is the mutual information

$$I_a(X; Y) = \mathbb{E}_Y\big[D_{\mathrm{KL}}\big(P_{X|Y,a} \,\|\, P_X\big)\big],$$

computed in closed form when belief-MDP values are available (the Toy Bandit) and otherwise estimated by the LLM in language space. The two EIGs we score are the *task EIG* $I_a(\theta; y)$ and the *design EIG* $I_a(A^*; y)$. We define the *task-information regret* $\Delta_\theta(a) = \max_{a'} I_{a'}(\theta; y) - I_a(\theta; y)$, the gap between the most task-informative action and a. The baselines and our method then correspond to different action-selection rules:

- **No-Op**: zero-shot, providing the answer without taking any actions.
- **ReAct** [Yao et al., 2023]: reasoning and self-reflection over the history to select the next action.
- **BOED**: $\pi^{\text{BOED}}(s_t) = \arg\max_a I_a(\theta; y) = \arg\min_a \Delta_\theta(a)^2$, myopically maximizing task EIG.
- **ID-BOED (ours)**: $\pi^{\text{ID-BOED}}(s_t) = \arg\min_a \dfrac{\Delta_\theta(a)^2}{I_a(A^*; y)}$, the IDS ratio, trading off *exploitation* in the task-information space (numerator) against *exploration* of effective design strategies (denominator), the beyond-current-step information about which action is optimal that BOED ignores.

3.3 Operationalizing Design Beliefs in Language Space

The hypothesis space of possible task and design latents is generally too large and unstructured to enumerate explicitly. Drawing on theoretical work establishing a foundation for approximate Bayesian RL via lossily compressed belief states [Arumugam and Van Roy, 2022] (agents may realistically need only resolve uncertainty over a lower-resolution model of the world that retains the minimum details for approximately-optimal decision-making), and on Arumugam and Griffiths [2026], who show that natural language provides a practical way to represent such compressed beliefs, our agent holds two lossily compressed *natural-language* beliefs, each revised per step by an approximate Bayesian posterior update in language space:

- *Task belief* b_θ: a natural-language description of the agent's uncertainty over the task solution θ.
- *Design belief* b_D: a natural-language description of the agent's uncertainty over which information-gathering strategies are effective (the design latent D).

Algorithm 1 ID-BOED LLM Agent

Initialize task and design belief priors b_θ^0, b_D^0; history $\mathcal{H}_0 = \varnothing$
for $t = 0, \ldots, H-1$ **do**
  $a_t \leftarrow \textsc{IDActionSelect}(b_\theta^t, b_D^t, \mathcal{H}_t)$
  Agent executes a_t and observes $y_t \sim p(\cdot \mid \theta, a)$
  $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t \cup \{(a_t, y_t)\}$
  $b_\theta^{t+1} \leftarrow \textsc{BeliefUpdate}(b_\theta^t, y_t)$
  $b_D^{t+1} \leftarrow \textsc{BeliefUpdate}(b_D^t, y_t, b_\theta^{t+1})$
end for
Return: Final solution given b_θ^H, b_D^H, $\mathcal{H}_H$

Algorithm 1 summarizes the full agent; action selection and belief updating are implemented via structured LLM prompts. IDActionSelect selects the action that approximately minimizes the IDS ratio, with the model estimating in bits the task-information regret $\Delta_\theta(a)$ (from the task EIG $I_a(\theta; y)$) and the design EIG $I_a(A^*; y)$. Due to LLM context window constraints, we truncate history to the most recent steps ($\mathcal{H}_{t-3:t}$) in the prompt; the beliefs b_θ^t and b_D^t serve as compressed summaries that retain salient information from earlier steps. BeliefUpdate updates the task and design beliefs given observation y_t by instructing the model to perform an approximate Bayesian posterior update in language space. Our approach also naturally yields the traditional sequential BOED agent by dropping the design belief and tracking only the task belief.

4 Experimental Setup

We organize our evaluation as a progression. We begin with two motivating experiments that each isolate a single component of the agent: the Toy Bandit isolates the IDS action-selection rule from

LLM estimation, and Wordle isolates the verbalized design belief and its Bayesian update. We then evaluate the full LLM agent on three realistic, open-ended discovery tasks, where real-world experts may disagree on the best strategy, the solution space is large and open-ended, or both. We instantiate all agents using Gemini 3.1 Flash, and compare ID-BOED with three baselines: No-Op, ReAct [Yao et al., 2023], and BOED. BOED and ReAct operate within the same environment instance and have access to the same action space; the only differences are their prompts and the beliefs they maintain. We report 95% confidence intervals throughout, computed across the evaluation examples within a single run, and assess gains over baselines with a paired bootstrap test over those examples. To mitigate data leakage, for all SEARCH actions we apply a semantic filter using a separate LLM judge (Gemini-2.5-Flash) to all search results, where the judge determines whether each result reveals the ground-truth solution.

4.1 Toy BOED Bandit

The Toy Bandit, our first motivating example, isolates the IDS action-selection rule by making every information-theoretic quantity exactly computable. The latent parameter $\theta = (G, S)$ pairs a *group* $G \in \{1, \ldots, M\}$, identifying which category of solution is correct, with a within-group *side* S. **Specialist** actions reveal S; **probe** actions binary-search G, revealing S only with low probability when the probed group is correct (K denotes the total number of arms). Because the information gain is available in closed form, this setting isolates the action rule from LLM estimation. Myopic BOED never probes; ID-BOED probes, then exploits. Appendix A shows a worked example of the $M = 2$ case.

4.2 Wordle

Our second motivating example is Wordle. We use standard Wordle on historical answer words; the only action is GUESS_WORD. Here the action-selection rule *only follows the design belief* (with no closed-form or LLM EIG estimation) to isolate the verbalized design prior and its Bayesian updating. We initialize agents with good, bad (with and without belief updating), and uninformed design priors, and evaluate on 200 historical Wordle games. With no explicit strategy source, the agent must *implicitly infer* the design strategy (which letter-frequency strategies are productive) from the evidence in its own guesses.

4.3 KernelBench

KernelBench [Ouyang et al., 2025], the first of our three open-ended discovery tasks, measures an agent's ability to write fast GPU (CUDA) kernels for full ML architectures. An LLM is provided with a PyTorch implementation of a model or operator and is tasked with generating a custom CUDA kernel that is both correct and achieves speedup over the PyTorch implementation. We specifically test on the 50 Level 3 problems, which provide full ML architectures such as AlexNet and MiniGPT. Kernels are evaluated on NVIDIA L40S GPUs using Modal [Modal Labs, 2026]. Each kernel is first evaluated for correctness and, if correct, is then measured for its speedup compared with the PyTorch baseline. The agent is evaluated by the speedup of the kernel it writes.

4.4 EsoLang-Bench

EsoLang-Bench [Sharma and Chopra, 2026] requires the agent to program in esoteric languages, which have $340\times$–$60{,}000\times$ fewer public repositories than Python and so are largely unseen in pretraining (less scarce: Brainfuck, Befunge-98; more scarce: Whitespace, Unlambda, Shakespeare). Identical problems are solvable in Python, where frontier models drop from 100% to roughly 11% when moved to these languages, making this a genuinely open-ended, data-scarce setting. The agent can search for and read language documentation and submit partial programs for verifiable test feedback. We report the test pass rate, which gives partial credit for the number of the 6 unit tests provided for each problem that the submitted program passes.

4.5 SCOTUS Writ of Certiorari Grant Prediction

Predicting which petitions the Supreme Court will grant is a hard, open-ended forecasting problem on which even experts disagree about the relevant signals. We curate a SCOTUS Certiorari (SCOTUS

Cert) petition prediction dataset of 200 petitions for writ of certiorari[1] filed with the U.S. Supreme Court during the October 2023 and October 2024 Terms. Because grants are rare relative to denials, producing a more balanced dataset requires heavily downsampling denials, yielding 88 granted and 112 denied petitions. The agent predicts certiorari grant or deny from party filings (docket documents), relevant precedential cases, and academic research on cert prediction. We report F1.

5 Results

5.1 Quantitative Evaluation

We first report the two motivating experiments that isolate the agent's components, then turn to the three open-ended discovery tasks.

Toy Bandit. To show that the advantage of the IDS decision principle does not rely on an LLM's estimation of EIG values, we first evaluate on the Toy BOED bandit, where every information-theoretic quantity is computed in closed form. Here, ID-BOED attains $4.5\times$–$7.4\times$ lower regret than BOED and Thompson Sampling, with the advantage growing with the horizon and with design uncertainty (Figure 1). Myopic BOED and Thompson Sampling never probe and accumulate regret throughout, learning G only indirectly through rare specialist hits, while ID-BOED probes for only a few rounds before identifying the group and switching to exploitation, so that its cumulative Bayes regret plateaus. Because Thompson Sampling tracks BOED closely, the gain comes from IDS's valuation of design-level information rather than from generic exploration.

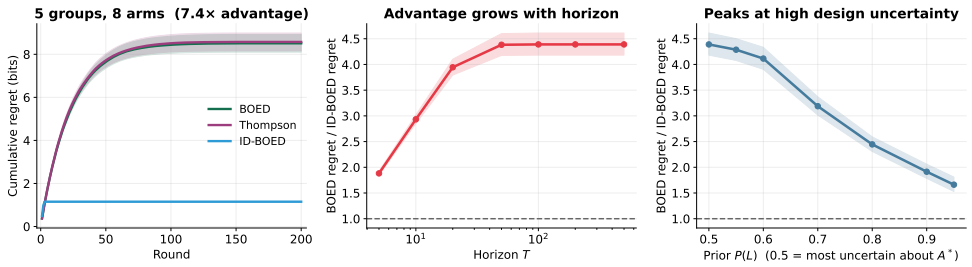


Figure 1: Toy Bandit. **Left:** cumulative Bayes regret (ID-BOED plateaus, baselines accumulate). **Middle:** ID-BOED's advantage over BOED grows with the horizon. **Right:** ID-BOED's advantage peaks at high design prior uncertainty (95% CI).

Wordle. We next ask whether the verbalized design belief and its Bayesian posterior updating are operationalizable on their own, without any LLM estimation of EIG values. In Wordle, the agent has no explicit strategy source and must implicitly infer the design strategy from the evidence in its own guesses. As Figure 2 shows, an agent that updates its design belief (green) adapts out of a bad design prior, recovering toward high-frequency guesses, while one with a frozen prior (red) does not, despite receiving the same feedback. Design information can thus be learned implicitly from experiments, not only from explicit design-information sources, which motivates the explicit, adaptive design belief at the core of ID-BOED.

We evaluate the full ID-BOED agent on three open-ended or unsolved problems, where experts may disagree on strategies and there is little pretraining data: KernelBench, EsoLang-Bench, and SCOTUS Cert.

ID-BOED achieves the best performance on all three tasks. Figure 3 reports task performance and Figure 4 the gains of ID-BOED over each baseline. ID-BOED's gains over No-Op / ReAct / BOED are significant on KernelBench ($+0.55\times$ / $+0.31\times$ / $+0.22\times$ speedup) and EsoLang-Bench ($+0.27$ / $+0.12$ / $+0.07$ test pass rate), and over No-Op / ReAct on SCOTUS Cert ($+0.37$ / $+0.24$ F1). On EsoLang-Bench's solvable tier, ID-BOED on Gemini 3.1 Flash reaches 22.0%, exceeding the benchmark's reported SOTA (GPT-5.4 xhigh, 16.9%) and roughly doubling its Gemini 3.1 Pro result

[1] A petition for writ of certiorari is a request to a higher court, like the U.S. Supreme Court, asking it to review a decision of a lower court. The Supreme Court is selective and only grants a small percentage of these petitions.

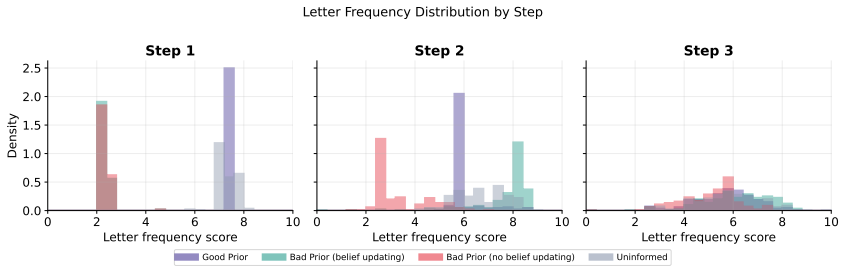
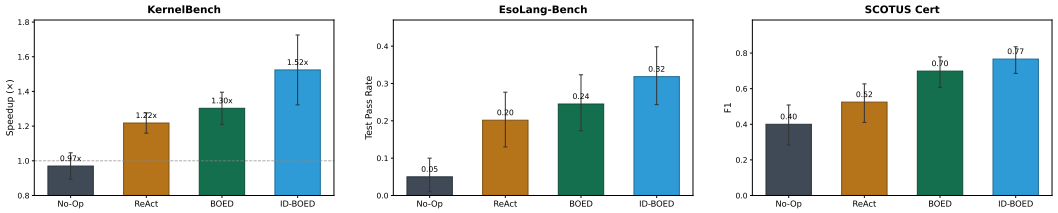


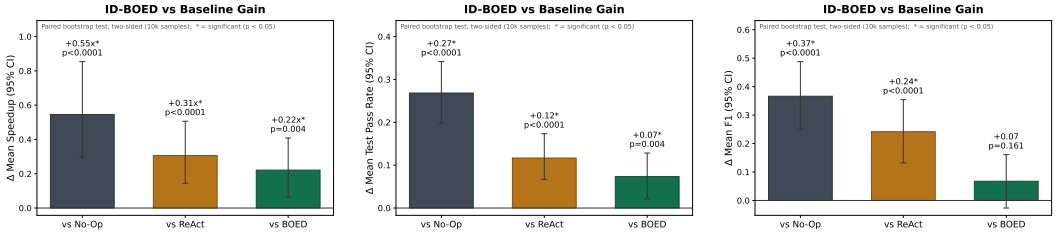
Figure 2: Guess letter-frequency by step: the updating bad prior (green) recovers toward high-frequency guesses, while the frozen prior (red) does not.

(11.0%); it is the only method to solve 6 problems (4 Shakespeare, 2 Befunge-98) that every baseline misses, and uniquely cracks KernelBench problem 49 (a Mamba-2 kernel) at $6.11\times$. Even myopic BOED beats ReAct and No-Op; operationalizing sequential BOED with fully verbalized beliefs and language-space posterior updating on realistic, open-ended discovery tasks is itself an extension of prior work, which has applied BOED to LLM agents only in more stylized settings [Choudhury et al., 2026]. Notably, baselines rarely seek design-level information unprompted even when it is available, underscoring the value of an explicit design belief.



(a) KernelBench (b) EsoLang-Bench (c) SCOTUS Cert

Figure 3: **ID-BOED achieves the best performance on all three tasks.** KernelBench speedup, EsoLang-Bench test pass rate, and SCOTUS Cert F1 (95% CI).

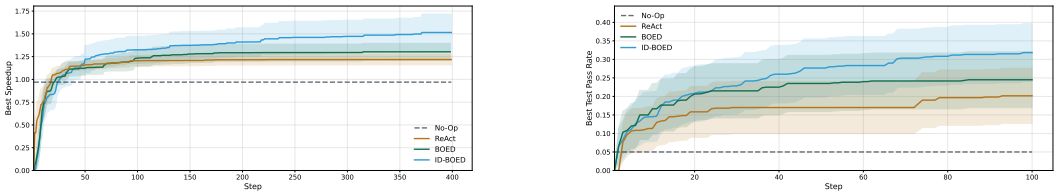


(a) KernelBench (b) EsoLang-Bench (c) SCOTUS Cert

Figure 4: Gains of ID-BOED over each baseline (paired bootstrap test, $*$ = significant at $p < 0.05$).

The advantage compounds over the horizon. On the realistic tasks, ID-BOED's gains compound over the horizon (Figure 5), supporting the Toy Bandit finding that ID-BOED's advantage over BOED grows with the horizon, where the agent has more steps to exploit early design information. ID-BOED invests in design-level actions early then exploits, catching up and surpassing baselines on long-horizon tasks, mirroring the Toy Bandit.

The advantage peaks under high design-prior uncertainty. The open-ended tasks confirm the Toy Bandit finding that the advantage peaks under high design-prior uncertainty, with ID-BOED's edge over BOED concentrating on the most data-scarce, idiosyncratic EsoLang languages, Shakespeare ($+23.3$ pp test pass rate, p=0.002) and Befunge-98 ($+9.2$ pp, p=0.02).



(a) KernelBench (b) EsoLang-Bench

Figure 5: Task performance vs. step on KernelBench and EsoLang-Bench. ID-BOED surpasses baselines as the horizon grows (95% CI).

5.2 Qualitative Analysis

Qualitative inspection illustrates how agents escape ineffective strategies as their design beliefs evolve through observations from taking actions in the environment. On KernelBench Problem 49 (Level 3), a Mamba-2 kernel that ID-BOED is the only method to solve, the agent initially follows an expert-suggested "Monolithic Fusion" strategy drawn from a meta-document, but after roughly 40 failed submissions its design belief shifts to a "White-Box decomposition" that isolates the compute-intensive intra-chunk state contractions in a CUDA kernel while keeping the inter-chunk recurrence orchestration in PyTorch, ultimately achieving a $6.11\times$ speedup (Appendix B). More generally, the outcomes of actions taken in an environment constitute indirect evidence about the strategies that generated them, from which an agent can draw inferences about effective design strategies even in the absence of information sources revealing explicit strategies, which is particularly valuable in open problem settings where effective strategies may not yet be known a priori.

Using an LLM judge (Gemini-2.5-Flash), we score each action's task and design focus (each in $[0, 1]$ and summing to one). Figure 6 plots the stepwise averages on KernelBench: ID-BOED selects more design-focused actions in the early steps, while BOED and ReAct shift to predominantly task-focused actions more quickly. This reflects the core insight of ID-BOED: by explicitly accounting for design-level uncertainty, the agent learns that early investment in understanding *how* to solve the task pays off later.

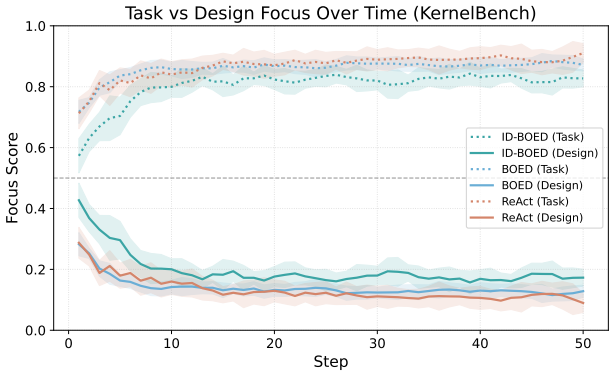


Figure 6: **Compared with the baselines, ID-BOED demonstrates greater early design-space exploration.** Stepwise average task- and design-focus scores on KernelBench. ID-BOED maintains higher design-focus scores in the early steps, while BOED and ReAct quickly converge to almost exclusively task-focused actions, reflecting ID-BOED's strategy of early exploration followed by later exploitation.

6 Discussion

ID-BOED's gains hinge on two ingredients that also bound the method. First, action selection requires good, calibrated LLM estimates of the relevant EIG values: quantifying task-information regret and design EIG in bits is difficult for a language model, and errors in those estimates could distort

ID-BOED action selection and degrade performance. Second, a core theoretical motivation for our work is the efficient exploration enabled by Bayesian updates of beliefs, but our implementation relies on natural language to approximate belief distributions, which is an imperfect proxy for true probabilistic representations. While LLMs can reason about uncertainty to some extent, they cannot precisely verbalize a posterior distribution in paragraph form, which hinders the implementation of rigorous Bayesian updates. Improving both the calibration of LLM EIG estimates and the faithful representation of beliefs in natural-language space remains open; future work could align natural-language outputs with probabilistic constraints, which would also open up sampling from posterior beliefs (a crucial component of algorithms such as PSRL [Arumugam and Griffiths, 2026]), and could move beyond inference-time prompting to explicitly train agents to internalize the ID-BOED objective.

7 Conclusion

In this work, we have highlighted a key distinction between gaining information and learning *how* to gain information. Whereas the former is a hallmark of the classic exploration problem studied in RL, the latter reflects a unique challenge, which we refer to as strategic exploration, critical to real-world decision-making agents that must learn to proficiently strategize in order to solve complex tasks. Using the design principle of IDS, we introduce an LLM agent capable of deftly navigating between classic and strategic exploration, resulting in significant performance gains across challenging open-ended domains including GPU kernel optimization, esoteric-language programming, and legal outcome prediction. Verbalized Bayesian experimental design in open-ended LLM action spaces is itself novel, and ID-BOED adds a Pareto improvement on top, establishing it as a robust Pareto improvement over BOED.

8 Team Contributions

- **Lucia Zheng:** All contributions made by LZ (one-person team).

Changes from Proposal The motivating Toy Bandit and Wordle settings were carried out as planned in the proposal, isolating, respectively, the IDS action-selection rule and the natural-language belief update. For the realistic tasks, we finalized our evaluation to three tasks: KernelBench, EsoLang-Bench [Sharma and Chopra, 2026], and SCOTUS Cert.

References

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing Software Agents with Monte Carlo Tree Search and Iterative Refinement, 2025. URL https://arxiv.org/abs/2410.20285.

Dilip Arumugam and Thomas L Griffiths. Toward efficient exploration by large language model agents. In *International Conference on Learning Representations (ICLR)*, 2026.

Dilip Arumugam and Benjamin Van Roy. Deciding what to model: Value-equivalent sampling for reinforcement learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 9024–9044. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/3b18d368150474ac6fc9bb665d3eb3da-Paper-Conference.pdf.

Peter Auer. Using Confidence Bounds for Exploitation-Exploration Trade-Offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

William Ian Beardmore Beveridge. *The Art of Scientific Investigation*. W. W. Norton, New York, 1957.

Kathryn Chaloner and Isabella Verdinelli. Bayesian Experimental Design: A Review. *Statistical Science*, 10:273–304, 1995.

Deepro Choudhury, Sinead Williamson, Adam Goliński, Ning Miao, Freddie Bickford Smith, Michael Kirchhof, Yizhe Zhang, and Tom Rainforth. Bed-llm: Intelligent information gathering with llms and bayesian experimental design. In *International Conference on Learning Representations (ICLR)*, 2026.

Kexin Huang, Ying Jin, Ryan Li, Michael Y. Li, Emmanuel Candès, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications, 2025. URL https://arxiv.org/abs/2502.09858.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models, 2025. URL https://arxiv.org/abs/2501.05366.

Aly Lidayan, Jakob Brandt Bjorner, Satvik Golechha, and Alane Suhr. ABBEL: LLM Agents Acting through Belief Bottlenecks Expressed in Language. In *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*, 2025.

Modal Labs. Modal, 2026. URL https://modal.com/. Accessed: 2026-01-27.

Ian Osband and Benjamin Van Roy. Why is Posterior Sampling Better than Optimism for Reinforcement Learning? In *International Conference on Machine Learning*, pages 2701–2710, 2017.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can LLMs Write Efficient GPU Kernels?, 2025. URL https://arxiv.org/abs/2502.10517.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents, 2024. URL https://arxiv.org/abs/2408.07199.

Tom Rainforth, Adam Foster, Desi R Ivanova, and Freddie Bickford Smith. Modern Bayesian experimental design. *Statistical Science*, 39(1):100–114, 2024.

Daniel Russo and Benjamin Van Roy. Learning to optimize via information-directed sampling. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Daniel Russo and Benjamin Van Roy. Learning to Optimize via Information-Directed Sampling. *Operations Research*, 66(1):230–252, 2018.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A Tutorial on Thompson Sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL https://arxiv.org/abs/2402.03300.

Aman Sharma and Paras Chopra. Esolang-bench: Evaluating genuine reasoning in large language models via esoteric programming languages. *arXiv preprint arXiv:2603.09678*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *Advances in Neural Information Processing Systems*, 36, 2023.

Fahim Tajwar, Yiding Jiang, Abitha Thankaraj, Sumaita Sadia Rahman, J Zico Kolter, Jeff Schneider, and Ruslan Salakhutdinov. Training a generally curious agent, 2025. URL https://arxiv.org/abs/2502.17543.

P. Tetlock and D. Gardner. *Superforecasting: The Art and Science of Prediction*. Random House, 2015.

William R Thompson. On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3/4):285–294, 1933.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

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

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2024. URL https://arxiv.org/abs/2310.04406.

A Toy BOED Bandit Worked Example of EIG Values

This appendix provides a closed-form worked example for the $M = 2$ bandit (Table 1). The expected information gain in θ decomposes by the chain rule as $I(\theta; Y) = I(G; Y) + I(S; Y \mid G)$. The specialist action earns a full bit about S but only 0.108 bits about the group G, whereas a probe action earns exactly 1 bit, all of it about G. Myopic BOED, which maximizes total information gain, therefore always selects a specialist and never probes. The probe nonetheless has the lower IDS ratio: although its task-information regret is larger ($\Delta = 0.207$ versus 0.099), it provides roughly $9\times$ more information about which action is optimal, so $\Gamma(a) = \Delta(a)^2 / I(G; Y \mid a)$ favors the probe (0.043 versus 0.091), and ID-BOED selects probe actions. Across problem scales, this preference yields the $4.5\times$–$7.4\times$ regret advantage reported in Figure 1, growing with both the horizon and the design-prior uncertainty.

Table 1: One-step quantities for the $M = 2$ bandit (bits). Myopic BOED ranks actions by total information gain and never probes; IDS ranks by Γ and probes.

Action	$I(\theta; Y)$ (reward)	$I(G; Y)$ (design)	$I(S; Y \mid G)$ (task)	regret Δ	ratio Γ
Specialist s_i	1.108	0.108	1.000	0.099	0.091
Probe p_j	1.000	1.000	0.000	0.207	**0.043**

B Case Study: Problem 49, Level 3, KernelBench

We present a case study on Problem 49, Level 3 of KernelBench. The task is to optimize the a Mamba-2 Structured State Space Duality (SSD) implementation. The reference implementation is:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from einops import rearrange

class Model(nn.Module):
    def __init__(self, batch_size, seq_length, n_heads, d_head,
        d_state, block_len=64):
        """
        Mamba Structured State Space model implementation for
            benchmarking.

        :param batch_size: Size of the batch
        :param seq_length: Length of the input sequence
        :param n_heads: Number of attention heads
        :param d_head: Dimension of each head
        :param d_state: Dimension of the state space
        :param block_len: Length of each block for chunked computation
```

```
        """
        super(Model, self).__init__()

        assert seq_length % block_len == 0, "Sequence length must be
            divisible by block length"

        self.batch_size = batch_size
        self.seq_length = seq_length
        self.n_heads = n_heads
        self.d_head = d_head
        self.d_state = d_state
        self.block_len = block_len

        # Initialize parameters
        self.A = nn.Parameter(torch.randn(batch_size, seq_length,
            n_heads))
        self.B = nn.Parameter(torch.randn(batch_size, seq_length,
            n_heads, d_state))
        self.C = nn.Parameter(torch.randn(batch_size, seq_length,
            n_heads, d_state))

    def segsum(self, x):
        """Naive segment sum calculation."""
        T = x.size(-1)
        x_cumsum = torch.cumsum(x, dim=-1)
        x_segsum = x_cumsum[..., :, None] - x_cumsum[..., None, :]
        mask = torch.tril(torch.ones(T, T, device=x.device, dtype=bool
            ), diagonal=0)
        x_segsum = x_segsum.masked_fill(~mask, -torch.inf)
        return x_segsum

    def forward(self, X, initial_states=None):
        """
        Forward pass implementing the SSD operation.

        :param X: Input tensor of shape (batch, length, n_heads,
            d_head)
        :param initial_states: Optional initial states
        :return: Output tensor Y and final state
        """
        # Rearrange into blocks/chunks
        X_blocks, A_blocks, B_blocks, C_blocks = [
            rearrange(x, "b (c l) ... -> b c l ...", l=self.block_len)
            for x in (X, self.A, self.B, self.C)
        ]

        A_blocks = rearrange(A_blocks, "b c l h -> b h c l")
        A_cumsum = torch.cumsum(A_blocks, dim=-1)

        # 1. Compute diagonal block outputs
        L = torch.exp(self.segsum(A_blocks))
        Y_diag = torch.einsum("bclhn,bcshn,bhcls,bcshp->bclhp",
                              C_blocks, B_blocks, L, X_blocks)

        # 2. Compute intra-chunk states
        decay_states = torch.exp((A_cumsum[:, :, :, -1:] - A_cumsum))
        states = torch.einsum("bclhn,bhcl,bclhp->bchpn",
                              B_blocks, decay_states, X_blocks)

        # 3. Compute inter-chunk recurrence
        if initial_states is None:
            initial_states = torch.zeros_like(states[:, :1])
        states = torch.cat([initial_states, states], dim=1)
```

```
        decay_chunk = torch.exp(self.segsum(F.pad(A_cumsum[:, :, :,
            -1], (1, 0))))
        new_states = torch.einsum("bhzc,bchpn->bzhpn", decay_chunk,
            states)
        return new_states[:, -1]

# Test parameters
batch_size = 2048
seq_length = 128
n_heads = 8
d_head = 64
d_state = 16
block_len = 64

def get_inputs():
    return [torch.rand(batch_size, seq_length, n_heads, d_head)]

def get_init_inputs():
    return [batch_size, seq_length, n_heads, d_head, d_state,
        block_len]
```

The ultimate working code of the ID-BOED agent is given as follows:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.cpp_extension import load_inline

# Optimized CUDA kernel for intra-chunk state computation using float4
    vectorization.
# This kernel handles the heavy O(B*C*L*H*P*N) contraction efficiently
    .
fused_state_source = """
#include <torch/extension.h>
#include <cuda_runtime.h>
#include <math.h>

__global__ void vectorized_state_kernel(
    const float4* __restrict__ X,          // [B, C, L, H, P/4]
    const float* __restrict__ A_cumsum,    // [B, H, C, L]
    const float4* __restrict__ B,          // [B, C, L, H, N/4]
    float4* __restrict__ out_states,       // [B, C, H, P, N/4]
    int B_sz, int C_sz, int L_sz, int H_sz, int P_vec, int N_vec)
{
    int b = blockIdx.x;
    int c = blockIdx.y;
    int h = blockIdx.z;

    if (b >= B_sz || c >= C_sz || h >= H_sz) return;

    __shared__ float s_A_cumsum[64];
    int tid = threadIdx.x;
    if (tid < L_sz) {
        s_A_cumsum[tid] = A_cumsum[((b * H_sz + h) * C_sz + c) * L_sz
            + tid];
    }
    __syncthreads();

    float last_A = s_A_cumsum[L_sz - 1];
    int p = tid / N_vec;
    int n_v = tid % N_vec;

    if (p < 64 && n_v < N_vec) {
        float4 acc = make_float4(0.0f, 0.0f, 0.0f, 0.0f);
        int p_vec_idx = p / 4;
```

```
        int p_comp = p % 4;

        #pragma unroll
        for (int l = 0; l < 64; ++l) {
            float decay = expf(last_A - s_A_cumsum[l]);
            float4 b_val = B[(((b * C_sz + c) * L_sz + l) * H_sz + h)
                * N_vec + n_v];
            float4 x_vec = X[(((b * C_sz + c) * L_sz + l) * H_sz + h)
                * 16 + p_vec_idx];

            float x_val;
            if (p_comp == 0) x_val = x_vec.x;
            else if (p_comp == 1) x_val = x_vec.y;
            else if (p_comp == 2) x_val = x_vec.z;
            else x_val = x_vec.w;

            acc.x += b_val.x * decay * x_val;
            acc.y += b_val.y * decay * x_val;
            acc.z += b_val.z * decay * x_val;
            acc.w += b_val.w * decay * x_val;
        }
        out_states[(((b * C_sz + c) * H_sz + h) * 64 + p) * N_vec +
            n_v] = acc;
    }
}

torch::Tensor vectorized_state_cuda(torch::Tensor X, torch::Tensor
    A_cumsum, torch::Tensor B) {
    int B_sz = X.size(0);
    int C_sz = X.size(1);
    int L_sz = X.size(2);
    int H_sz = X.size(3);
    int P_sz = X.size(4);
    int N_sz = B.size(4);
    auto out_states = torch::empty({B_sz, C_sz, H_sz, P_sz, N_sz}, X.
        options());
    dim3 blocks(B_sz, C_sz, H_sz);
    dim3 threads(256);
    vectorized_state_kernel<<<blocks, threads>>>(
        (const float4*)X.data_ptr<float>(),
        A_cumsum.data_ptr<float>(),
        (const float4*)B.data_ptr<float>(),
        (float4*)out_states.data_ptr<float>(),
        B_sz, C_sz, L_sz, H_sz, P_sz / 4, N_sz / 4);
    return out_states;
}
"""

cpp_source = "torch::Tensor vectorized_state_cuda(torch::Tensor X,
    torch::Tensor A_cumsum, torch::Tensor B);"

fused_lib = load_inline(
    name="vectorized_state_lib_final_v2",
    cpp_sources=cpp_source,
    cuda_sources=fused_state_source,
    functions=["vectorized_state_cuda"],
    verbose=False
)


class ModelNew(nn.Module):
    def __init__(self, batch_size, seq_length, n_heads, d_head,
        d_state, block_len=64):
        super(ModelNew, self).__init__()
        self.batch_size = batch_size
```

```
        self.seq_length = seq_length
        self.n_heads = n_heads
        self.d_head = d_head
        self.d_state = d_state
        self.block_len = block_len
        self.A = nn.Parameter(torch.randn(batch_size, seq_length,
            n_heads))
        self.B = nn.Parameter(torch.randn(batch_size, seq_length,
            n_heads, d_state))
        self.C = nn.Parameter(torch.randn(batch_size, seq_length,
            n_heads, d_state))

    def segsum(self, x):
        T = x.size(-1)
        x_cumsum = torch.cumsum(x, dim=-1)
        x_segsum = x_cumsum[..., :, None] - x_cumsum[..., None, :]
        mask = torch.tril(torch.ones(T, T, device=x.device, dtype=bool
            ), diagonal=0)
        x_segsum = x_segsum.masked_fill(~mask, -float('inf'))
        return x_segsum

    def forward(self, X, initial_states=None):
        # B=2048, L=128, H=8, P=64, N=16, block_len=64 -> 2 chunks
        X_blocks = X.view(self.batch_size, 2, 64, self.n_heads, 64).
            contiguous()
        B_blocks = self.B.view(self.batch_size, 2, 64, self.n_heads,
            16).contiguous()
        A_blocks_h = self.A.view(self.batch_size, 2, 64, self.n_heads)
            .permute(0, 3, 1, 2).contiguous()

        A_cumsum = torch.cumsum(A_blocks_h, dim=-1)

        # 1. Compute intra-chunk states using the vectorized CUDA
            kernel
        states = fused_lib.vectorized_state_cuda(X_blocks, A_cumsum,
            B_blocks)

        # 2. Robust inter-chunk recurrence using segsum and einsum for
             numerical stability
        if initial_states is None:
            initial_states = torch.zeros(
                (self.batch_size, 1, self.n_heads, 64, 16),
                device=X.device,
                dtype=X.dtype
            )
        else:
            initial_states = initial_states.view(self.batch_size, 1,
                self.n_heads, 64, 16)

        states = torch.cat([initial_states, states], dim=1)
        A_cumsum_last = A_cumsum[:, :, :, -1]
        decay_chunk = torch.exp(self.segsum(F.pad(A_cumsum_last, (1,
            0))))

        # Final state is the result of the recurrence across chunks
        new_states = torch.einsum("bhzc,bchpn->bzhpn", decay_chunk,
            states)
        return new_states[:, -1]
```

This problem is highly representative; it is both challenging and analytically interesting. The ID-BOED agent acquires a suboptimal design strategy in the very beginning, but later corrected itself by leveraging appropriate design-level belief updating and exploration, ultimately achieves a $6.11\times$ speedup. None of the other agents manage to even solve this problem. In the following, we analyze the trajectories of the ID-BOED agent and BOED agent on this problem, respectively, and discuss

why ID-BOED succeeds while BOED fails. Since the full trajectories are extremely long, we only analyze the critical points and major observations within them.

In the ID-BOED agent's trajectory, it first reads from a meta-document "Mamba2: The Hardware-Algorithm Co-Design That . . ." in step 4 and learns that

> "The Structured State Space Duality (SSD) algorithm divided sequences into chunks of size Q (typically 64–256 tokens). For each chunk, it performed four operations: Intra-chunk computation: Process tokens within each chunk using matrix multiplication. Chunk state computation: Calculate the final hidden state for each chunk. Inter-chunk communication: Pass states between chunks sequentially. Output combination: Merge intra-chunk and inter-chunk information."

After reading this document, the agents design belief explicitly incorporated this framework in the step 5:

> "A key design heuristic is the 'Chunking Strategy': by partitioning the sequence into blocks (e.g., 64 or 128), we can transform the global $O(L^2)$ dependency into a local $O(B^2)$ attention-like computation and a global $O(L/B)$ state recurrence... Experts prioritize 'Monolithic Fusion', where a single CUDA kernel handles the entire sequence."

In particular, the meta-document suggested "Monolithic Fusion" as the expert approach, which turns out to be incorrect or very prone to introduce errors for this problem. The ID-BOED agent then followed this suggestion. However, after around 40 steps of failed submissions, the agent's beliefs evolved in step 47:

> "The persistent numerical divergence of 10^{13} in the Mamba-2 SSD implementation indicates a structural misunderstanding of the recurrence logic rather than simple precision issues. My meta-level strategy has shifted from a 'Black-Box' approach (trying to implement the entire recurrence in a single CUDA kernel) to a 'White-Box' decomposition. In complex recurrent architectures, the most effective optimization strategy is to isolate the compute-intensive components—specifically the intra-chunk state contractions—while maintaining the high-level orchestration (like inter-chunk recurrence and padding logic) in PyTorch."

with a clear implementation strategy:

> "I will now implement a version that uses PyTorch for the inter-chunk recurrence and only uses CUDA for the intra-chunk state calculation to isolate the error."

This shift to the new "White-Box decomposition" strategy turns out to be the right move. In contrast, the BOED agent gets stuck within an infinite debugging cycle, not being able to find the bug with 321 steps and 146 submissions, which is significantly more than ID-BOED's 72 submissions across 115 steps.

C Implementation Details

All agents and all tasks are evaluated with Gemini 3.1 Flash; the LLM judge used for the semantic data-leakage filter and for scoring each action's task/design focus is Gemini-2.5-Flash. We use a different maximum number of steps per trajectory (episode) for each task: 400 for KernelBench, 100 for EsoLang-Bench, and 12 for SCOTUS Cert. The main reason is that these tasks are of different levels of difficulty and so reach a task-performance plateau after differing amounts of exploration, so we set each task's step budget to allow agents to reach this plateau.

The available actions vary by task:

- **Toy BOED Bandit**: SPECIALIST, PROBE.
- **Wordle**: GUESS_WORD.
- **KernelBench**: THINK, OPEN_WEB_SEARCH, READ, SUBMIT, PROVIDE_FINAL_RESPONSE.

- **EsoLang-Bench**: THINK, ESOLANG_WIKI_SEARCH, REQUEST_URL (request the URL payload from embedded links in the wiki pages), READ, SUBMIT, PROVIDE_FINAL_RESPONSE.
- **SCOTUS Cert**: THINK, COURTLISTENER_SEARCH (for checking case precedent cited in the party docket files), CLOSED_SEARCH (for party docket files), READ, PROVIDE_FINAL_RESPONSE.

Action selection and belief updating are implemented via structured LLM prompts for IDACTIONSELECT and BELIEFUPDATE. The prompts for ID-BOED, BOED, and ReAct follow the same structure described in Section 3, differing only in the beliefs maintained and the action-selection objective: ID-BOED maintains both a task and design belief and minimizes the IDS ratio, BOED maintains only a task belief and maximizes task EIG, and ReAct maintains no explicit belief and instead reasons and self-reflects over the history.