# Efficient Cost-Aware LLM Evaluation via Bayesian Bandit Gittins Indices

**Qian Xie** [1] **Yueli He** [2] **Nairen Cao** [2]

## Abstract

Selecting a high-performing LLM configuration increasingly requires comparing many candidate models, prompts, or decoding settings under limited evaluation budgets. Exhaustively evaluating every candidate on every benchmark example can be expensive, and each additional query incurs costs such as API pricing, token usage, latency, or human grading effort. We formulate this configuration-selection task as cost-aware Bayesian bandit evaluation and propose a Gittins index policy that treats each configuration's benchmark performance as latent, updates posterior uncertainty, allocates queries according to their value of information, and provides a stopping signal. The method combines sample efficiency with computational efficiency: after precomputing Gittins indices for the Gaussian posterior dynamics induced by batch-mean approximations, online allocation only requires table lookup and posterior updates, and it naturally handles varying evaluation costs. Across GSM8K, PIQA, and MMLU response matrices, Gittins shows two clear gains: on large-example benchmarks it outperforms configuration-level Bayesian optimization, and on large-candidate MMLU tasks it markedly outperforms cost-unaware frequentist bandit baselines. At its stopping point, the recommended arm often attains near-zero simple regret after using only about 1–10% of the exhaustive evaluation budget.

## 1. Introduction

Modern LLM evaluation increasingly compares many configurations, such as models, prompting techniques, temperatures, or decoding strategies, across benchmarks such as GSM8K, PIQA, and MMLU (Cobbe et al., 2021; Bisk et al., 2020; Hendrycks et al., 2020). Running every candidate on every example is simple but wasteful: weak candidates can be screened early, strong candidates merit more measurement, and systems can have very different inference costs. Recurring deployment decisions therefore need an adaptive rule for which candidate to test next and when enough evidence has been collected.

This perspective naturally suggests a bandit formulation. BanditEval considers UCB-E, a frequentist fixed-budget best-arm identification method, and UCB-E-LRF, a low-rank response-matrix variant (Zhou et al., 2025); PromptEval uses related racing ideas for multi-prompt evaluation (Polo et al., 2024). We also compare with configuration-level Bayesian optimization (Snoek et al., 2012), which evaluates each selected configuration on all benchmark examples. These methods turn benchmark data into an online allocation problem, but many LLM evaluations also contain useful prior information from historical results, model metadata, pilot runs, or domain expertise. A Bayesian formulation provides a principled way to encode such information and update it as new observations arrive.

Our experiments show two distinct regimes. Across many settings, the arm recommended at the Gittins stopping point attains near-zero simple regret using only about 1–10% of the exhaustive evaluation budget. On large-example benchmarks such as GSM8K and PIQA, configuration-level Bayesian optimization spends each query on an all-example candidate evaluation, while Gittins can distribute partial batches and reaches low regret much earlier. On MMLU, where each subject contains 1500 candidate model–template arms, Gittins is much stronger than cost-unaware frequentist bandit rules because it combines posterior uncertainty, prior information, and evaluation cost in one allocation index.

We introduce a Gittins index policy for cost-aware Bayesian bandit evaluation of LLMs. Each arm represents an LLM configuration, its latent mean is the benchmark score, and each pull queries the configuration on benchmark data at an arm-dependent cost. The final decision is the recommended arm, so performance is measured by recommendation quality versus cumulative evaluation cost, rather than cumulative reward during evaluation.

Under a Gaussian approximation to batch-mean benchmark observations, each arm's posterior mean evolves as a Gaus-

[1]Cornell University, Ithaca, New York, USA [2]New York University, Brooklyn, New York, USA. Correspondence to: Qian Xie, Nairen Cao <qx66@cornell.edu, nc1827@nyu.edu>.

*Accepted at ICML 2026 Workshop on Decision-Making from Offline Datasets to Online Adaptation: Black-Box Optimization to Reinforcement Learning.* Copyright 2026 by the author(s).

sian random walk with deterministic variance decay. The resulting Gittins index is an interpretable priority score: it is high when an arm currently looks good, remains uncertain enough to justify more measurement, and is cheap enough to evaluate.

Our contributions are:

- We formulate LLM evaluation from benchmark data as a cost-aware Bayesian bandit problem whose query costs can reflect API pricing, token usage, latency, or other evaluation costs.
- We derive a Gittins index policy under a Gaussian approximation to batch-mean observations, enabling offline index precomputation and online table lookup.
- We extend the policy to varying model costs through cost normalization, allowing adaptive allocation to trade off information value against evaluation expense.
- We compare against UCB-E, correlation-aware UCB-E-LRF, and configuration-level Bayesian optimization, finding gains over Bayesian optimization in large-example settings and over cost-unaware frequentist bandits in large-candidate settings, with the stopping-time recommendation often attaining near-zero simple regret using about 1–10% of the exhaustive evaluation budget.

### 1.1. Related Work

**Efficient LLM evaluation** Efficient LLM evaluation has recently been studied as adaptive sampling. Zhou et al. (2025) propose UCB-E-LRF, which uses low-rank response-matrix structure to share information across arms and examples. Shi et al. (2024) introduce TRIPLE, an elimination method that predicts final arm performance from partial evaluations. Their response-matrix setup is closest to our fixed-cost-budget reporting view: adaptive evaluation proceeds under unit costs and is judged as best-arm identification, especially the probability of selecting the best arm. We instead report simple regret and allow varying evaluation costs. Polo et al. (2024) propose PromptEval for efficient multi-prompt evaluation. Our work shares the goal of reducing benchmark evaluation cost, but differs in its Bayesian bandit treatment of uncertainty and use of Gittins indices.

**The Gittins Policy** The Gittins index is a classical design principle for sequential decision making under uncertainty (Gittins et al., 2011; Scully & Terenin, 2025; Xie et al., 2024). It converts certain multi-armed allocation problems into single-arm stopping problems, producing one index per arm. Our terminal-reward formulation differs from the traditional discounted-reward bandit objective: evaluation scores are measurements, while utility comes from the final recommendation after paying query costs. Up to an additive best-arm value, this trades simple regret against cumulative evaluation cost. The key technical convenience is that the Gaussian batch-mean approximation makes each arm's posterior mean follow a Gaussian random walk, enabling efficient precomputation of the index schedule.

## 2. Cost-Aware LLM Recommendation

### 2.1. Configurations, Examples, and Latent Performance

We consider an evaluation study with $K$ candidate LLM configurations, indexed by $k \in \{1, \dots, K\}$. A configuration may be a model, prompt, decoding setting, retrieval pipeline, or any deployable system whose benchmark performance is to be estimated.

A benchmark supplies a finite collection of examples. Let $Z_{k,j}$ denote the primitive score obtained by evaluating configuration $k$ on example $j$. We model these scores as samples from a configuration-specific distribution with latent mean $\theta_k$; in a finite response matrix, this approximates randomized sampling without replacement from the corresponding row. Adaptive evaluation reveals only selected scores. The score may be binary correctness, a normalized human or LLM-judge score, a task-level utility, or another scalar outcome.

The evaluator's goal is to recommend a configuration with large latent mean $\theta_k$ while spending as little evaluation cost as possible. Each allocation to configuration $k$ incurs a cost $c_k > 0$, after any rescaling of raw monetary or computational costs into benchmark-score units. In LLM evaluation, cost may reflect API pricing, token usage, latency, local compute, or human grading effort. Prior efficient-evaluation work is often cost-unaware in this sense: BanditEval (Zhou et al., 2025) considers UCB-E (Audibert & Bubeck, 2010), a frequentist fixed-budget allocation method, together with UCB-E-LRF, its low-rank response-matrix variant, and both treat evaluations as unit-cost allocations. TRIPLE (Shi et al., 2024) and PromptEval (Polo et al., 2024) similarly focus on $c_k \equiv 1$.

### 2.2. Adaptive Evaluation and Recommendation

At global time $t$, an adaptive evaluation policy $\pi$ can depend on the history

$$\mathcal{H}_t = \{(I_s, Y_s, c_{I_s}) : s \leq t\}.$$

Here $I_s \in \{1, \dots, K\}$ is the selected configuration, and $Y_s$ is the scalar observation from that allocation, either one score $Z_{I_s,j}$ or a batch average. We use policy broadly: $\pi$ selects the next configuration $I_t$, decides whether to stop at time $T$, and outputs a final recommendation $\hat{k}_T \in \{1, \dots, K\}$. In fixed-budget experiments, the stopping rule is replaced by the budget constraint, but the policy still

produces an anytime recommendation after each step.

When a finite per-configuration horizon is imposed, a configuration is completed once it reaches that horizon; stopping and recommendation then depend on the current configuration states and posterior beliefs.

### 2.3. Objectives

We consider two cost-aware formulations. The first is a fixed-cost-budget objective. Unlike reward-maximization problems, the observed scores are measurements of benchmark performance; payoff comes from the terminal recommendation. For budget $\mathcal{C}$, the policy minimizes simple regret subject to cumulative cost:

$$\min_{\pi} \ \mathbb{E}_{\pi}\left[\max_{k} \theta_k - \theta_{\hat{k}_T}\right] \quad \text{subject to} \quad \sum_{s=1}^{T} c_{I_s} \le \mathcal{C}.$$

Because $\theta_k$ is latent, experiments with completed response matrices report regret using the finite-matrix row mean

$$\theta_k^{\mathrm{mat}} = \frac{1}{N}\sum_{j=1}^{N} Z_{k,j} \tag{1}$$

as a plug-in proxy for $\theta_k$. This lets us compare policies using only entries revealed online while measuring regret against the best configuration in the completed matrix.

The second formulation lets the terminal time be adaptive rather than fixed in advance. For a stopping time $T$, the objective is

$$\mathbb{E}\left[\theta_{\hat{k}_T} - \sum_{t=1}^{T} c_{I_t}\right].$$

If raw costs $\tilde{c}_k$ are measured in dollars, seconds, or tokens, we write $c_k = \lambda \tilde{c}_k$ for a scale $\lambda$ that converts them into benchmark-score units. This objective trades recommendation quality against measurement cost.

## 3. Gittins Index Policy for Cost-Aware Bayesian Evaluation

Our goal is to adaptively estimate the mean benchmark performance of each candidate LLM configuration while accounting for varying evaluation costs. From this section onward, we use the standard Bayesian bandit term *arm* for a candidate configuration: arm $k$ is configuration $k$, $A_t$ denotes the arm selected at global allocation step $t$, one pull evaluates that configuration on one example or a batch of examples, and $\theta_k$ is its latent mean score. In our implementation, one allocation to an arm returns the average score over a batch of randomized benchmark examples, and we treat this batch average as a noisy measurement of the arm's latent performance.

The algorithm has two components. First, it maintains a Bayesian posterior distribution over each arm mean and updates that posterior after every observed batch average. Second, it uses a cost-aware Gittins index to decide which arm is worth measuring next. The index is high when an arm currently looks promising, remains uncertain enough that new evidence could change the final recommendation, or is inexpensive to evaluate. The policy stops when an arm attaining the largest Gittins index has reached the finite evaluation horizon, and then recommends a completed arm with the largest posterior mean. Figure 1 summarizes how posterior beliefs, Gittins indices, benchmark batches, and the response matrix interact over one iteration of this loop.

### 3.1. Bayesian Bandit Model

We use the Bayesian bandit model as a way to update beliefs about the latent benchmark mean $\theta_k$ of each arm. This differs from frequentist allocation rules such as UCB-E, which build confidence bounds from empirical estimates. Here the evaluator starts with a prior belief, observes benchmark evidence, and converts that evidence into a posterior belief about each $\theta_k$.

After $n$ local pulls of arm $k$, let $\mathcal{D}_{k,n}$ denote the observations collected from that arm. The posterior distribution given $\mathcal{D}_{k,n}$ summarizes the current belief about $\theta_k$. If evaluation stops, the posterior expected reward from recommending arm $k$ is its posterior mean $\mu_{k,n} = \mathbb{E}[\theta_k \mid \mathcal{D}_{k,n}]$, so the recommended arm is $\arg\max_k \mu_{k,n_k}$.

A pull may evaluate one example or a batch of examples. Following the notation in Section 2, let $Z_{k,j}$ be the example-level score for arm $k$ on example $j$. If the $i$th pull of arm $k$ evaluates batch $\mathcal{B}_{k,i}$ of size $B$, the scalar observation used for posterior updating is the batch mean

$$\overline{Y}_{k,i} = \frac{1}{B}\sum_{j \in \mathcal{B}_{k,i}} Z_{k,j}.$$

The batch mean has conditional mean $\theta_k$. Let $\tau_k^2$ denote its conditional variance, so that, by the central limit theorem, $\overline{Y}_{k,i} \mid \theta_k$ is approximately $\mathcal{N}(\theta_k, \tau_k^2)$ as a Gaussian approximation. In the common binary-accuracy case, $\tau_k^2 = \theta_k(1-\theta_k)/B$. For simplicity and efficient index computation, we use a fixed approximation $\tau^2$ to the batch-mean variance; Section 3.4 describes this choice.

We use the Gaussian prior-likelihood approximation

$$\theta_k \sim \mathcal{N}(\mu_0, v_0), \qquad \overline{Y}_{k,i} \mid \theta_k \sim \mathcal{N}(\theta_k, \tau^2),$$

where $\tau^2$ is the fixed observation-noise variance in the likelihood. To state the update compactly, fix a representative arm and suppress the arm index, writing $\theta$, $\mathcal{D}_n$, $\mu_n$, $v_n$, and $\overline{Y}_{n+1}$. The posterior after $n$ local pulls is

$$\theta \mid \mathcal{D}_n \sim \mathcal{N}(\mu_n, v_n),$$

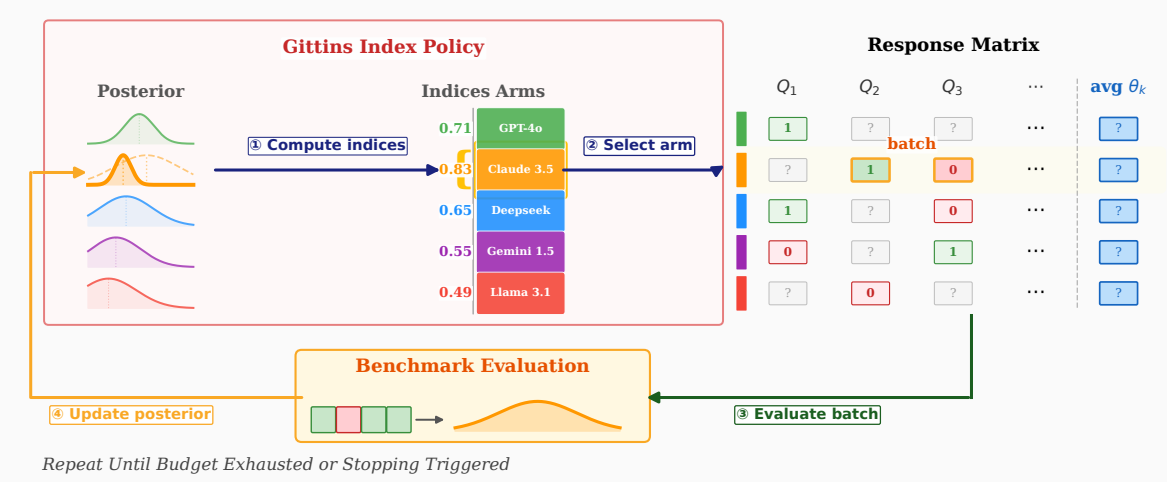



*Figure 1.* Overview of the proposed Gittins-index evaluation framework. The policy maintains posterior beliefs over candidate LLM configurations, computes Gittins indices to adaptively select the next arm batch to evaluate, observes benchmark responses from the response matrix, and updates the posterior iteratively until the evaluation budget is exhausted or a stopping condition is met.

with update

$$\begin{aligned} v_{n+1} &= \left(v_n^{-1} + \tau^{-2}\right)^{-1}, \\ \mu_{n+1} &= \mu_n + \frac{v_n}{v_n + \tau^2}(\overline{Y}_{n+1} - \mu_n). \end{aligned} \tag{2}$$

When Eq. (2) is applied to arm $k$ in Algorithm 1, the symbols are read as $\mu_{k,n}$, $v_{k,n}$, and $\overline{Y}_{k,n+1}$. Under fixed $\tau^2$, the variance schedule for this representative arm is deterministic. In particular, the one-step variance reduction is

$$v_n - v_{n+1} = \frac{v_n^2}{v_n + \tau^2}.$$

Thus arms share the same variance schedule at a given local pull count only when they also share the same initial variance $v_0$ and fixed noise $\tau^2$; with arm-specific prior variances or noise levels, the same update gives arm-specific deterministic schedules. The next step is to use these posterior updates not only to summarize current evidence, but also to decide which arm should be evaluated next.

### 3.2. Posterior Dynamics and the Gittins Policy

The posterior update also tells us what could happen to an arm's posterior mean if we evaluate it again. Let $S_n$ denote the posterior mean of a representative arm after $n$ local pulls, viewed as a random variable over future observations. Conditional on the current realized state $S_n = s$, selecting the arm gives

$$S_{n+1} \mid S_n = s \sim \mathcal{N}\left(s, \frac{v_n^2}{v_n + \tau^2}\right).$$

Fix a finite per-arm horizon $H$, measured in local pulls, so that an arm is completed after $H$ batch evaluations. The arm state is therefore $(S_n, n)$: the posterior mean follows a Gaussian random walk, while the posterior variance is determined by the local pull count. Selecting an arm changes only that arm's state and incurs its evaluation cost; an arm with $n = H$ is completed and becomes eligible for terminal recommendation. This separation lets us compare arms through single-arm index values.

To define the index, fix one arm with posterior-mean state $S_n = s$, query cost $c$, and remaining horizon $H - n$. Let $\alpha$ denote the outside terminal reward available from the other arms. Before the arm reaches $H$, the local actions are CONTINUE, which evaluates the arm once more, and STOP, which declines this unfinished arm in favor of the outside value $\alpha$. Suppressing the fixed cost $c$ in the value-function notation, the terminal value is

$$V_H(s; \alpha) = \max\{s, \alpha\},$$

and, for $n < H$,

$$\begin{aligned} Q_n^{\text{cont}}(s; \alpha) &= -c + \mathbb{E}[V_{n+1}(S_{n+1}; \alpha) \mid S_n = s], \\ Q_n^{\text{stop}}(s; \alpha) &= \alpha, \\ V_n(s; \alpha) &= \max\{Q_n^{\text{cont}}(s; \alpha), Q_n^{\text{stop}}(s; \alpha)\}. \end{aligned}$$

Here $Q_n^{\text{cont}}$ is the Q-value function for evaluating the arm one more time, while $Q_n^{\text{stop}}$ is the value of declining this unfinished arm in favor of the outside reward. For $n < H$ and cost $c$, the Gittins index is the smallest outside reward at which stopping is optimal in this local problem:

$$G_n^c(s) = \inf\{\alpha \in \mathbb{R} : Q_n^{\text{stop}}(s; \alpha) \geq Q_n^{\text{cont}}(s; \alpha)\}.$$

An arm receives a large index when its posterior mean is high, its uncertainty creates value of information, or its evaluation cost is low.

At global time $t$, let $n_k(t)$ be the number of batches observed from arm $k$. For an unfinished arm, set

$$G_{k,t} = G^{c_k}_{n_k(t)}(\mu_{k,n_k(t)}).$$

A completed arm has terminal index $G_{k,t} = \mu_{k,H}$. Under required completion, if an arm attaining $\max_k G_{k,t}$ is completed, the policy stops and recommends a completed arm with the highest posterior mean among completed arms; otherwise, it evaluates an unfinished arm with largest index. This is the stopping rule used for the Gittins stopping markers in our experiments.

Markov-chain selection applies because the Gaussian random walk is a Markov chain (Dumitriu et al., 2003; Scully & Terenin, 2025).

**Proposition 3.1** (Bayesian optimality under required completion)**.** *Consider the Gaussian Bayesian bandit model in Section 3.1 with independent arms, common prior $\mathcal{N}(\mu_0, v_0)$, fixed observation noise $\tau^2$, finite per-arm horizon $H$, and nonnegative effective costs of arm pulls $c_k$ for arms $k = 1, \ldots, K$. Let $\Pi^{\text{req}}_H$ be the class of policies that evaluate only unfinished arms and may terminate only by recommending an arm that has reached its finite horizon. For the required-completion objective*

$$\sup_{\pi \in \Pi^{\text{req}}_H} \mathbb{E}_\pi \left[ \theta_{\hat{k}_T} - \sum_{t=1}^{T} c_{A_t} \right],$$

*an optimal policy is to assign terminal index $G_{k,t} = \mu_{k,H}$ to completed arms, assign $G_{k,t} = G^{c_k}_{n_k(t)}(\mu_{k,n_k(t)})$ to unfinished arms, and stop when an arm attaining $\max_k G_{k,t}$ is completed. Otherwise, the policy evaluates an unfinished arm with largest Gittins index $G_{k,t}$. At stopping, an optimal recommendation is any completed arm with largest posterior mean among completed arms, equivalently any completed arm with largest terminal index.*

The proof is deferred to Appendix A.2.

### 3.3. Efficient Gittins Index Computation

We next describe how the index values used in the previous subsection are efficiently computed. The definition above suggests a dynamic program over both the posterior-mean state $s$ and the outside reward $\alpha$. Solving such a program at every online decision would be too expensive. We make the computation practical with three reductions. First, translational invariance converts the two-dimensional continuation value into a one-dimensional dynamic program over the advantage $s - \alpha$. Second, under fixed observation noise, the posterior-variance schedule is deterministic, so the resulting root schedule can be precomputed once for each cost and variance schedule. Third, the Gaussian expectations in the dynamic program are one-dimensional convolutions and can be evaluated efficiently offline.

**Lemma 3.2** (Translational invariance of the continuation value)**.** *Fix the cost $c$ and Gaussian random-walk transition. For any shift $\beta \in \mathbb{R}$ and any stage $n < H$,*

$$Q^{\text{cont}}_n(s + \beta; \alpha + \beta) = Q^{\text{cont}}_n(s; \alpha) + \beta.$$

**Corollary 3.3** (Centered continuation value)**.** *Let $x = s - \alpha$ and define $q_n(x) := Q^{\text{cont}}_n(x; 0)$. For any $n < H$,*

$$Q^{\text{cont}}_n(s; \alpha) = q_n(s - \alpha) + \alpha.$$

*Thus evaluating the unfinished arm is worthwhile exactly when $q_n(s - \alpha) > 0$.*

The proof of Lemma 3.2 and Corollary 3.3 is included in Appendix A.3.

By Corollary 3.3, only the posterior-mean advantage $x = s - \alpha$ matters. Let $W_n(x) := V_n(x; 0)$, then the centered recursion is

$$\begin{aligned} W_H(x) &= \max\{x, 0\}, \\ W_n(x) &= \max\{0, q_n(x)\}, \\ q_n(x) &= -c + \mathbb{E}[W_{n+1}(X_{n+1}) \mid X_n = x], \end{aligned} \tag{3}$$

where $X_{n+1} \mid X_n = x \sim \mathcal{N}(x, v_n^2/(v_n + \tau^2))$ and the second line applies for $n < H$. Let $r_n$ denote the root of the continuation value such that $q_n(r_n) = 0$. Then evaluating an unfinished arm is worthwhile exactly when $s - \alpha > r_n$. Equivalently, the index can be evaluated by

$$G^c_n(s) = s - r_n. \tag{4}$$

For each distinct tuple $(c, v_0, \tau^2, H)$, offline computation stores the root schedule

$$\{r_n(c, v_0, \tau^2, H)\}_{n=0}^{H-1}.$$

The fixed-noise approximation makes this lookup table reusable: arms with the same effective cost, prior variance, observation noise, and horizon share the same schedule. The conditional expectation in (3) is a Gaussian convolution on a one-dimensional grid, which we compute offline; our implementation accelerates this step with FFT-based convolution. Online allocation is then inexpensive: each decision requires only posterior updates, table lookups, and index comparisons. For a finite benchmark with $N$ examples, $H = N$ for single-example pulls and $H = \lceil N/B \rceil$ for batches of size $B$.

Additional FFT derivation details are deferred to the full version of the paper.

### 3.4. Design Choices for LLM Benchmarks

**Prior specification.** For accuracy-style benchmarks, the latent mean lies in $[0, 1]$, so the prior can be chosen directly

on the accuracy scale. A general choice is $\mathcal{N}(0.5, 0.04)$: its mean expresses no preference between low and high accuracy, and most prior mass lies in the plausible interval $[0.1, 0.9]$. When benchmark- or dataset-specific knowledge is available, we can instead use an informative common prior whose mean and variance are estimated from related tasks, historical evaluations, or pilot examples. Unless otherwise stated, all arms within a benchmark share the same prior so that the comparison focuses on adaptive allocation rather than arm-specific prior information.

**Observation noise.** For batched binary correctness observations, the empirical batch mean has variance $\theta_k(1-\theta_k)/B$. Batching also has a practical systems benefit: examples in a batch can often be evaluated in parallel, reducing wall-clock latency, a motivation also noted by Zhou et al. (2025). Since $\theta_k$ is unknown, we use the conservative fixed-noise approximation $\tau^2 = 1/(4B)$, the worst-case Bernoulli variance divided by the batch size. This choice is deliberately simple and can overestimate the noise for very easy or very hard tasks, but it has a computational advantage: the posterior variance schedule becomes deterministic, so the Gittins root schedule can be precomputed once and reused by all arms with the same cost, prior variance, noise level, and horizon. More adaptive alternatives include plug-in noise $\tau^2_{k,t} \approx \mu_{k,t}(1-\mu_{k,t})/B$ or estimates from pilot examples and historical response matrices; these require additional index computation because the variance schedule can depend on the arm's evolving state.

**Cost scaling.** Costs must be expressed in the same utility scale as benchmark performance. In API-based evaluation, raw cost may be computed from provider pricing, input-token counts, and output-token counts; for local models, latency or compute can be used as a proxy. If the scale $\lambda$ is small, the policy explores more aggressively; if it is large, the stopping rule becomes conservative. We therefore study cost-scaling factors in addition to raw inference-cost ratios.

**Recommendation and reporting.** The Gittins index is used for allocation and for the optional required-completion stopping rule. Under that stopping rule, the policy stops once an arm attaining the largest current Gittins index has reached horizon $H$. Its terminal output is restricted to completed arms: it chooses the completed arm with the largest posterior mean. If no completed arm has the largest index, the policy evaluates an unfinished arm with largest Gittins index. This matches the required-completion formulation in Proposition 3.1.

The simple-regret curves use a different anytime reporting convention. At every allocation step, including steps before any arm is completed, we report the arm with the largest current posterior mean among all arms,

$$\hat{k}_t \in \operatorname*{arg\,max}_{k=1,\dots,K} \mu_{k,n_k(t)}.$$

Thus an unfinished arm can be the recommendation used to compute the plotted simple regret. The completion restriction above applies only to the required-completion stopping rule and its terminal recommendation; in the experiments, the dashed Gittins stopping markers indicate where that rule would stop, while the regret curve itself is computed using the all-arm posterior-mean recommendation at each time.

Algorithm 1 summarizes the implementation. When early stopping is disabled, the algorithm ignores this stopping condition and continues until the budget is exhausted, unless all arms have reached the horizon.

**Algorithm 1** Gittins index evaluation for a Bayesian bandit

**Require:** Prior mean and variance $(\mu_0, v_0)$, raw costs $\{\tilde{c}_k\}_{k=1}^K$, cost scale $\lambda$.
**Require:** Observation noise $\tau^2$, batch size $B$, horizon $H$, and budget.
Set $c_k = \lambda \tilde{c}_k$ for all arms $k$.
Precompute $\{r_n(c, v_0, \tau^2, H)\}_{n=0}^{H-1}$ for each needed cost $c$.
Initialize posterior mean $\mu_{k,0} = \mu_0$, posterior variance $v_{k,0} = v_0$, and count $n_k(0) = 0$ for all $k$; set $t = 0$.
Set $\hat{k}_0 \in \arg\max_k \mu_{k,0}$ as the recommendation by posterior mean.
**while** budget remains **do**
  **if** $n_k(t) = H$ for all arms $k$ **then**
    **return** $\hat{k}_t$.
  **end if**
  For completed arms ($n_k(t) = H$), set terminal index $G_{k,t} = \mu_{k,H}$.
  For unfinished arms ($n_k(t) < H$), set continuation index $G_{k,t} = G^{c_k}_{n_k(t)}(\mu_{k,n_k(t)})$.
  **if** early stopping is enabled **then**
    **if** an arm in $\arg\max_k G_{k,t}$ is completed **then**
      **return** $\arg\max_{k:n_k(t)=H} \mu_{k,H}$.
    **end if**
  **end if**
  Select arm $A_t \in \arg\max_{k:n_k(t)<H} G_{k,t}$.
  Observe batch mean $\overline{Y}_{A_t,n_{A_t}(t)+1}$.
  Update $(\mu_{A_t,\cdot}, v_{A_t,\cdot})$ using Eq. (2).
  Set $n_{A_t}(t+1) = n_{A_t}(t) + 1$, set $n_j(t+1) = n_j(t)$ for all $j \neq A_t$, and set $t \leftarrow t+1$.
  Set $\hat{k}_t \in \arg\max_k \mu_{k,n_k(t)}$ as the recommendation by posterior mean.
**end while**
**return** $\hat{k}_t$.

## 4. Experiments

**Benchmarks.** We evaluate adaptive allocation policies using completed LLM response matrices built on three standard benchmarks: GSM8K for grade-school mathematical reasoning (Cobbe et al., 2021), PIQA for physical commonsense reasoning (Bisk et al., 2020), and MMLU for multitask knowledge and reasoning (Hendrycks et al., 2020). The benchmark examples define the evaluation tasks, while the observed LLM correctness results come from existing response-matrix datasets. GSM8K and PIQA use the various-LLM and sampling-configuration response matrices from BanditEval (Zhou et al., 2025): GSM8K contains 122 arms and 1000 examples, while PIQA contains 103 arms and 1000 examples. MMLU uses response matrices from DOVE (Habba et al., 2025) across 57 subject datasets. For each subject, we evaluate 15 models with 100 prompting templates; each arm is a model-template pair, giving 1500 arms per subject.

For MMLU, we further group subjects by their empirical mean arm quality. This gives a natural notion of subject difficulty: if the average arm accuracy is high, then the subject is relatively easy for the collection of evaluated models and prompting templates; if the average arm accuracy is low, then the subject is relatively hard. We use this criterion to divide MMLU subjects into easy, medium, and hard groups. More details and analysis of the datasets can be found in Appendix B.

**Baselines.** We compare two versions of the proposed Gittins index policy with UCB-E, a classical fixed-budget best-arm identification method (Audibert & Bubeck, 2010), UCB-E-LRF, the low-rank-factorization variant used for faster language model evaluation by Zhou et al. (2025), and configuration-level Bayesian optimization baselines using PBGI, LogEI, and its cost-aware variant, LogEIPC. PBGI is inherently cost-aware. We abbreviate UCB-E-LRF as LRF in figures and captions. Gittins-G denotes Gittins with the general prior, while Gittins-S selects a shared data-specific prior for each dataset or MMLU subject from four difficulty buckets: easy $\mathcal{N}(0.75, 0.01)$, medium $\mathcal{N}(0.6, 0.02)$, hard $\mathcal{N}(0.4, 0.02)$, and very hard $\mathcal{N}(0.2, 0.01)$. The exact assignments are listed in Table 5. UCB-E and UCB-E-LRF represent frequentist allocation rules that do not explicitly encode Bayesian priors or Gittins index stopping values. When a baseline includes a random initialization phase, the percentage refers to the sampling unit: for arm-level initialization, $5\%$ means sampling $5\%$ of arms uniformly at random and evaluating each sampled arm on all benchmark examples. For BanditEval's UCB-E-LRF warm-up, the default is entry-level and samples $T_0 = 0.05KN$ arm–example entries from a $K$-arm, $N$-example response matrix before active selection.

**Evaluation metric.** The primary metric is simple regret as a function of the evaluation resources used. After allocation step $t$, each policy reports a recommendation $\hat{k}_t$, the arm with the highest current estimate of benchmark performance. Given the completed response matrix, we compute simple regret $\mathrm{SR}_t = \max_k \theta_k^{\mathrm{mat}} - \theta_{\hat{k}_t}^{\mathrm{mat}}$ using the finite-matrix target in Eq. (1). We plot this simple regret against cumulative example evaluations $M_t = \sum_{s=1}^{t} b_s$ in the unit-cost setting, where $b_s$ is the number of examples evaluated at step $s$. In the cost-aware setting, recalling that $c_k$ denotes the per-example evaluation cost of arm $k$, the allocation at step $s$ costs $b_s c_{A_s}$; we therefore plot simple regret against cumulative evaluation cost $C_t = \sum_{s=1}^{t} b_s c_{A_s}$. For configuration-level Bayesian optimization baselines, one query evaluates a configuration on all benchmark examples and consumes its corresponding all-example evaluation cost.

**Experimental conditions.** For each benchmark, we consider both unit-cost and cost-aware settings. In the unit-cost setting, every batch has the same cost, isolating the effect of adaptive statistical allocation. In the cost-aware setting, batch costs reflect model-specific evaluation expense, so the policy must trade off information value against heterogeneous inference cost. Configurations that use the same base model share the same model-level token price, with task-specific input/output accounting described in Appendix C.

For the bandit methods, we use a fixed evaluation budget equal to 10% of the corresponding exhaustive evaluation budget: 10% of the response-matrix entries in the unit-cost setting and 10% of the total exhaustive evaluation cost in the cost-aware setting. UCB-E and UCB-E-LRF are run to this fixed budget, following the BanditEval protocol of Zhou et al. (2025); for the Gittins variants, we also report where the Gittins stopping rule from Section 3.4 would terminate. The Bayesian optimization baselines follow a configuration-level protocol, where evaluating one candidate on all benchmark examples reveals its aggregate score and consumes that configuration's all-example evaluation cost. They use a random-initialization phase capped at 5% of the configurations, followed by acquisition-driven selection under the same nominal 10% budget fraction; cost-aware Bayesian optimization runs stop before exceeding the 10% exhaustive-evaluation-cost budget. For the main GSM8K and PIQA plots, UCB-E and Gittins use batch size $B = 8$, while UCB-E-LRF uses $B = 32$. For the main MMLU aggregate plots, the small size bucket (at most 150 examples) uses $B = 2$ and the large size bucket (more than 400 examples) uses $B = 8$ for UCB-E and Gittins, while UCB-E-LRF uses $B = 32$. The medium-size bucket, full subject-level MMLU results, and batch-size ablations are included in the appendix.

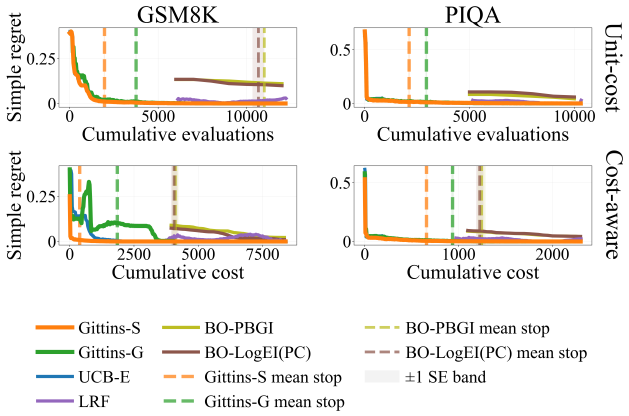


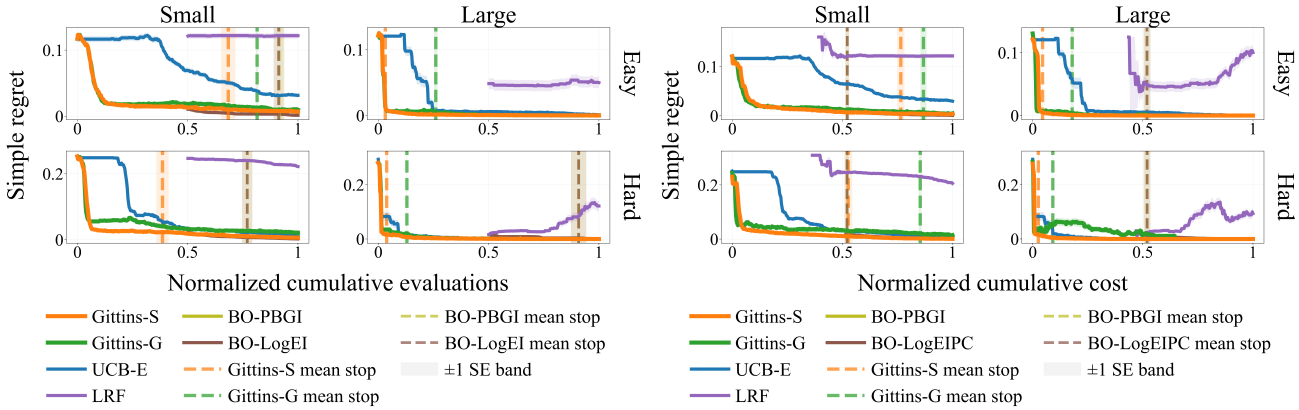

*Figure 2.* Main GSM8K and PIQA results. Simple regret is plotted against cumulative evaluations in the unit-cost setting and against cumulative cost in the cost-aware setting, using batch size $B = 8$ for UCB-E and Gittins and $B = 32$ for LRF. Each panel averages over 100 independent runs (5 ground-truth matrices $\times$ 20 randomized trials). Shaded bands denote $\pm 1$ standard error. Bayesian optimization curves are shown after the 5% configuration-level random-initialization phase. Dashed vertical lines show mean stopping locations when available: for Gittins, when the index-based stopping rule first triggers; for Bayesian optimization, the recorded Bayesian optimization stopping point. These stopping markers are distinct from the fixed 10% budget used for UCB-E and UCB-E-LRF. On these large-example benchmarks, Gittins is much more effective than configuration-level Bayesian optimization because it can allocate partial batches instead of full candidate evaluations.

**Reporting.** The main results report simple-regret curves for GSM8K and PIQA under unit-cost and cost-aware evaluation. Each curve compares the Gittins variants, UCB-E, UCB-E-LRF, and Bayesian optimization baselines on the same response matrices. Unless otherwise stated, shaded bands denote one standard error of the mean across repeated runs. Figure 2 presents these main curves. Figure 3 summarizes the corresponding MMLU trends. Dashed vertical markers indicate average stopping locations when available: for Gittins, these correspond to the Gittins stopping rule; for Bayesian optimization, they correspond to the Bayesian optimization budget or acquisition stopping point. These markers are distinct from the fixed 10% budget used for UCB-E and UCB-E-LRF. For the MMLU aggregate plots, we normalize the $x$-axis by each run's final cumulative evaluations in the unit-cost setting or final cumulative cost in the cost-aware setting, while keeping the $y$-axis as raw simple regret. Appendix D reports the MMLU unit-cost and cost-aware curves, common-prior, batch-size, and cost-scaling ablations, together with descriptive plots of the latent accuracy distribution for each MMLU prior bucket.

**Implementation details.** For the Gittins index policy, root schedules are precomputed for the common prior variance, noise, batch size, horizon, and cost scale used in each condition. During evaluation, the policy updates posterior means and variances using the Gaussian approximation in Section 3.1. Because online allocation only requires posterior updates and root-table lookup after this offline precomputation step, the total runtime of the Gittins variants remains close to UCB-E and much smaller than UCB-E-LRF; Figures 14 and 15 in Section D report runtime comparisons. The Bayesian optimization baselines operate on converted configuration-level inputs: after the random-initialization phase, they fit a mixed Gaussian-process model to the observed configurations and select the next configuration using the corresponding acquisition function. We implement these baselines in BoTorch using Gaussian processes with categorical kernels, encoding BanditEval configurations for GSM8K and PIQA by four categorical coordinates and DOVE configurations for MMLU by two categorical coordinates. For GSM8K and PIQA, we average over five response matrices, reflecting LLM output randomness, and run 20 randomized trials on each matrix. For MMLU, we run 20 trials for each subject and aggregate within each reported bucket.

**Main findings.** The results show that the Gittins-based policies reduce simple regret rapidly in the low-budget regime; in many settings, the arm recommended at the Gittins stopping point attains near-zero simple regret after only about 1–10% of the exhaustive evaluation budget. On GSM8K and PIQA, where the matrices have only 122 and 103 arms but 1000 examples per arm, the main separation is between Gittins and configuration-level Bayesian optimization. Bayesian optimization is less competitive because each configuration-level query consumes a full row of the response matrix; under the same 10% budget fraction, the 5% random-initialization phase already uses much of the Bayesian optimization budget, leaving relatively few acquisition-driven updates. Gittins can instead allocate partial batches across configurations and reach low regret early in both cost settings.

The MMLU aggregate results show the large-candidate regime. Each MMLU subject has 1500 model–template arms, so the policy must search over a much larger candidate set than in GSM8K or PIQA. Here the main comparison changes: Gittins, especially Gittins-S, is markedly stronger than the cost-unaware frequentist bandit baselines UCB-E and UCB-E-LRF, which spread a fixed budget across many arms without using priors or cost-normalized index values. Notably, UCB-E-LRF has substantially higher simple regret than standard UCB-E across the MMLU aggregates. This agrees with the broader empirical finding of **?** that UCB-E-LRF usually underperforms UCB-E. Bayesian optimization can remain competitive on small and easy tasks, where all-example configuration evaluations are relatively cheap, but the Gittins variants typically reach low regret earlier across the reported size–difficulty buckets.




*(a)* Unit-cost setting. *(b)* Cost-aware setting.

*Figure 3.* Aggregate MMLU results by subject size and difficulty. Simple regret is plotted against cumulative evaluations in a and cumulative cost in b; the $x$-axis is rescaled to $[0, 1]$ by each run's final cumulative evaluations or cumulative cost, while the $y$-axis shows raw simple regret. For UCB-E and Gittins, the small size bucket (at most 150 examples) uses batch size $B = 2$ and the large size bucket (more than 400 examples) uses $B = 8$; LRF uses batch size $B = 32$. Each panel averages over all subjects in the size–difficulty bucket and 20 independent runs per subject. Shaded bands denote $\pm 1$ standard error. Bayesian optimization curves are shown after the 5% configuration-level random-initialization phase. Dashed vertical lines show mean stopping locations when available: for Gittins, when the index-based stopping rule first triggers; for Bayesian optimization, the recorded Bayesian optimization stopping point. On these 1500-arm tasks, Gittins-G and Gittins-S usually reach low regret earlier than UCB-E and LRF, especially in cost-aware and harder buckets.

The Gittins stopping locations usually appear after the Gittins regret curves have flattened, suggesting that the rule stops once a completed arm is also the most valuable arm under the index comparison. These stopping points are generally earlier than the Bayesian optimization budget stop and earlier than the 10% fixed-budget endpoint used for UCB-E and UCB-E-LRF in BanditEval (Zhou et al., 2025). On larger response matrices, the stopping point can occur around 1% of the full response-matrix budget, giving a large reduction in evaluation effort while preserving the low-regret recommendation reached by the curve.

The comparison between Gittins-G and Gittins-S highlights the role of prior calibration. Gittins-S tends to benefit from data-specific priors, especially on harder MMLU groups where the prior mean is better aligned with the latent accuracy range and early allocation decisions are more consequential. At the same time, Gittins-G remains competitive in several easier or smaller settings, suggesting that a general prior can be sufficient when many arms have similar high accuracies or when regret becomes small quickly. Overall, the results suggest that data-specific priors are useful, but their mean and variance should be calibrated carefully rather than assumed to be uniformly beneficial across all task difficulties.

## 5. Conclusion

We formulated LLM evaluation as cost-aware Bayesian bandit evaluation and implemented it with a practical Gittins index policy. The empirical takeaway is simple: Gittins outperforms configuration-level Bayesian optimization on large-example benchmarks such as GSM8K, PIQA, and AlpacaEval, and markedly outperforms cost-unaware frequentist bandit methods when the candidate set is large, as in MMLU. After index precomputation, the policy is lightweight, handles heterogeneous evaluation costs, and provides a stopping signal once regret has flattened.

**Future work.** Future work includes broader comparisons with methods such as PromptEval (Polo et al., 2024), TRIPLE (Shi et al., 2024), and SySRs (**?**), more principled prior calibration, incorporating correlations across arms, including benchmarks with continuous scores such as AlpacaEval (Li et al., 2023), and adaptive alternatives to the fixed-noise approximation.

## Acknowledgments

QX thanks Theodore Brown, Ziv Scully, and Alexander Terenin for a prior collaboration from which the translational-invariance reduction and FFT-based approach to efficient Gittins-index computation originated. QX also thanks Ziv Scully for helpful discussions on the optional-completion setting, Kyuseong Choi for introducing QX to the area of efficient LLM evaluation and sharing related work, Tianyi Peng for inviting QX to a related project and discussions connecting LLM evaluation with multi-armed bandits, and Jin Peng Zhou and Ruihan Wu for sharing the BanditEval data and answering questions about the datasets.

## References


Audibert, J.-Y. and Bubeck, S. Best arm identification in multi-armed bandits. In *COLT-23th Conference on learning theory-2010*, pp. 13–p, 2010.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dumitriu, I., Tetali, P., and Winkler, P. On playing golf with two balls. *SIAM Journal on Discrete Mathematics*, 16(4): 604–615, 2003.

Gittins, J., Glazebrook, K., and Weber, R. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

Habba, E., Arviv, O., Itzhak, I., Perlitz, Y., Bandel, E., Choshen, L., Shmueli-Scheuer, M., and Stanovsky, G. DOVE: A large-scale multi-dimensional predictions dataset towards meaningful LLM evaluation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11744–11763. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.611. URL https://aclanthology.org/2025.findings-acl.611/.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Li, X., Zhang, T., Dubois, Y., Taori, R., Gulrajani, I., Guestrin, C., Liang, P., and Hashimoto, T. B. Alpacaeval: An automatic evaluator of instruction-following models, 2023.

Polo, F. M., Xu, R., Weber, L., Silva, M., Bhardwaj, O., Choshen, L., de Oliveira, A. F., Sun, Y., and Yurochkin, M. Efficient multi-prompt evaluation of llms. *Advances in Neural Information Processing Systems*, 37:22483–22512, 2024.

Scully, Z. and Terenin, A. The gittins index: A design principle for decision making under uncertainty. In *Tutorials in Operations Research: Advances in Analytics and Operations Research: Improving Decisions to Secure the Future*, pp. 28–70. INFORMS, 2025.

Shi, C., Yang, K., Chen, Z., Li, J., Yang, J., and Shen, C. Efficient prompt optimization through the lens of best arm identification. *Advances in Neural Information Processing Systems*, 37:99646–99685, 2024.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 2012.

Xie, Q., Astudillo, R., Frazier, P. I., Scully, Z., and Terenin, A. Cost-aware bayesian optimization via the pandora's box gittins index. *Advances in Neural Information Processing Systems*, 37:115523–115562, 2024.

Zhou, J. P., Belardi, C. K., Wu, R., Zhang, T., Gomes, C. P., Sun, W., and Weinberger, K. Q. On speeding up language model evaluation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3cvwO5DBZn.

# A. Theoretical Details

## A.1. Gaussian Random-Walk Details

We derive the posterior update and the induced random-walk transition used in Section 3. Fix an arm and suppress the arm index. Let $\overline{Y}_{n+1}$ denote the scalar observation from the next pull, which may be either a single-example score or a batch mean. Suppose that after $n$ local pulls the current posterior distribution of the arm mean is

$$\theta \mid \mathcal{D}_n \sim \mathcal{N}(\mu_n, v_n),$$

and that the next observation follows the fixed-noise Gaussian likelihood

$$\overline{Y}_{n+1} \mid \theta \sim \mathcal{N}(\theta, \tau^2).$$

By Bayes' rule, the posterior density after observing $\overline{Y}_{n+1}$ is proportional to the likelihood times the current posterior:

$$p(\theta \mid \mathcal{D}_{n+1}) \propto p(\overline{Y}_{n+1} \mid \theta)\, p(\theta \mid \mathcal{D}_n).$$

Substituting the Gaussian likelihood and the current Gaussian posterior gives

$$p(\theta \mid \mathcal{D}_{n+1}) \propto \exp\left(-\frac{1}{2}\left[\frac{(\theta-\mu_n)^2}{v_n} + \frac{(\overline{Y}_{n+1}-\theta)^2}{\tau^2}\right]\right).$$

Expanding the terms that depend on $\theta$,

$$\frac{(\theta-\mu_n)^2}{v_n} + \frac{(\overline{Y}_{n+1}-\theta)^2}{\tau^2} = \left(v_n^{-1}+\tau^{-2}\right)\theta^2 - 2\left(\frac{\mu_n}{v_n} + \frac{\overline{Y}_{n+1}}{\tau^2}\right)\theta + \text{const.}$$

Thus the exponent is quadratic in $\theta$, so the posterior is Gaussian. Matching the quadratic and linear coefficients with the Gaussian form

$$\exp\left(-\frac{(\theta-\mu_{n+1})^2}{2v_{n+1}}\right)$$

gives

$$v_{n+1}^{-1} = v_n^{-1} + \tau^{-2}, \qquad \frac{\mu_{n+1}}{v_{n+1}} = \frac{\mu_n}{v_n} + \frac{\overline{Y}_{n+1}}{\tau^2}.$$

Therefore,

$$v_{n+1} = \left(v_n^{-1}+\tau^{-2}\right)^{-1}, \qquad \mu_{n+1} = v_{n+1}\left(\frac{\mu_n}{v_n} + \frac{\overline{Y}_{n+1}}{\tau^2}\right).$$

Equivalently, the posterior mean update can be written in the incremental form

$$\mu_{n+1} = \mu_n + \frac{v_n}{v_n+\tau^2}\left(\overline{Y}_{n+1}-\mu_n\right).$$

This form shows that the new posterior mean moves from the old posterior mean toward the new observation, with gain

$$\frac{v_n}{v_n+\tau^2}.$$

To describe the posterior mean as a state process, we first compute the predictive distribution of the next observation. Conditioned on the current data $\mathcal{D}_n$, we may write

$$\overline{Y}_{n+1} = \theta + \varepsilon_{n+1}, \qquad \theta \mid \mathcal{D}_n \sim \mathcal{N}(\mu_n, v_n), \qquad \varepsilon_{n+1} \sim \mathcal{N}(0, \tau^2),$$

with $\theta$ and $\varepsilon_{n+1}$ conditionally independent given $\mathcal{D}_n$. Hence

$$\overline{Y}_{n+1} \mid \mathcal{D}_n \sim \mathcal{N}(\mu_n, v_n + \tau^2).$$

We now view the posterior mean as the Markov state used below. Let $S_n$ denote the posterior mean after $n$ local pulls, viewed as a random variable over future observations. If the current realized state is $S_n = s$, then the posterior update gives

$$S_{n+1} = s + \frac{v_n}{v_n + \tau^2} \left( \overline{Y}_{n+1} - s \right).$$

Since

$$\overline{Y}_{n+1} - s \mid S_n = s \sim \mathcal{N}(0, v_n + \tau^2),$$

we have

$$S_{n+1} \mid S_n = s \sim \mathcal{N}\left( s, \left( \frac{v_n}{v_n + \tau^2} \right)^2 (v_n + \tau^2) \right).$$

Therefore,

$$S_{n+1} \mid S_n = s \sim \mathcal{N}\left( s, \frac{v_n^2}{v_n + \tau^2} \right).$$

Finally, under the fixed-noise likelihood, the variance update

$$v_{n+1} = \left( v_n^{-1} + \tau^{-2} \right)^{-1}$$

does not depend on the realized observation $\overline{Y}_{n+1}$. Therefore the posterior variance follows a deterministic schedule indexed only by the local pull count $n$, while the posterior mean follows the Gaussian random-walk transition above whenever the arm is selected. Across arms, this schedule is identical only for arms with the same initial variance and fixed noise level; otherwise each arm has its own deterministic schedule.

### A.2. Proof of Bayesian Optimality

**Proposition A.1** (Restatement of Proposition 3.1)**.** *Consider the Gaussian Bayesian bandit model in Section 3.1 with independent arms, common prior $\mathcal{N}(\mu_0, v_0)$, fixed observation noise $\tau^2$, finite per-arm horizon $H$, and nonnegative effective costs of arm pulls $c_k$ for arms $k = 1, \ldots, K$. Let $\Pi_H^{\mathrm{req}}$ be the class of policies that evaluate only unfinished arms and may terminate only by recommending an arm that has reached its finite horizon. For the required-completion objective*

$$\sup_{\pi \in \Pi_H^{\mathrm{req}}} \mathbb{E}_\pi \left[ \theta_{\hat{k}_T} - \sum_{t=1}^{T} c_{A_t} \right],$$

*an optimal policy is to assign terminal index $G_{k,t} = \mu_{k,H}$ to completed arms, assign $G_{k,t} = G^{c_k}_{n_k(t)}(\mu_{k,n_k(t)})$ to unfinished arms, and stop when an arm attaining $\max_k G_{k,t}$ is completed. Otherwise, the policy evaluates an unfinished arm with largest Gittins index $G_{k,t}$. At stopping, an optimal recommendation is any completed arm with largest posterior mean among completed arms, equivalently any completed arm with largest terminal index.*

*Proof.* We prove the proposition by reducing the required-completion problem to a Markov chain selection problem. For each arm $k$, define one local chain:

**State.** The local state is $x_k = (s_k, n_k)$, where $s_k$ is the posterior mean and $n_k \in \{0, \ldots, H\}$ is the number of observed batches. The terminal states are those with $n_k = H$.

**Action.** At each decision time, choose one arm $k$. Only the chosen arm moves; every other arm keeps its current state.

**Reward.** If the chosen arm is unfinished, the immediate reward is $-c_k$. If the chosen arm is terminal, the process stops and the terminal reward is $s_k$.

**Transition.** If the chosen arm is in state $(s_k, n_k)$ with $n_k < H$, then its next state is $(S', n_k + 1)$, where

$$S' \mid s_k, n_k \sim \mathcal{N}\left( s_k, \frac{v_{n_k}^2}{v_{n_k} + \tau^2} \right).$$

The global state is the collection of local states, initialized at $((\mu_0, 0), \ldots, (\mu_0, 0))$.

By Eq. (2), the state of arm $k$ can be represented by $(\mu_{k,n_k}, n_k)$. Under fixed $\tau^2$, Appendix A.1 shows that the posterior mean follows the Gaussian random-walk transition, so this state is Markov. Conditional on this state, the next state distribution of arm $k$ is independent of the states and histories of all other arms, and pulling arm $k$ incurs only its own effective cost $c_k$. Since $s_k = \mathbb{E}[\theta_k \mid x_k]$, the terminal reward is the posterior expected reward from recommending completed arm $k$. Thus maximizing expected total reward in the constructed selection problem is exactly the required-completion objective.

The Gittins-index optimality theorem for Markov chain selection implies that an optimal policy selects a chain with largest index until a terminal chain has largest index (Dumitriu et al., 2003; Scully & Terenin, 2025). Applying this result to the single-arm value function used to define $G_n^c(s)$ gives the stated rule. Terminal chains have index equal to their posterior mean, so when a completed arm attains the largest index, recommending any completed arm with largest posterior mean among completed arms gives an optimal terminal reward. □

### A.3. One-Dimensional Gittins Index Computation

We now spell out the reduction used by the precomputation step in Algorithm 1. Fix a single arm, suppress the arm index, and let $\alpha$ be the outside terminal reward available from the other arms. With the cost $c$ fixed and suppressed in the value-function notation, the terminal value is

$$V_H(s; \alpha) = \max\{s, \alpha\},$$

and, for $n < H$,

$$\begin{aligned} Q_n^{\text{cont}}(s; \alpha) &= -c + \mathbb{E}[V_{n+1}(S_{n+1}; \alpha) \mid S_n = s]\,, \\ Q_n^{\text{stop}}(s; \alpha) &= \alpha, \\ V_n(s; \alpha) &= \max\{Q_n^{\text{cont}}(s; \alpha), Q_n^{\text{stop}}(s; \alpha)\}. \end{aligned}$$

The transition kernel depends only on differences in posterior means:

$$S_{n+1} \mid S_n = s \sim \mathcal{N}(s, \sigma_n^2), \qquad \sigma_n^2 = \frac{v_n^2}{v_n + \tau^2}.$$

*Proof of Lemma 3.2.* The terminal value satisfies

$$V_H(s + \beta; \alpha + \beta) = \max\{s + \beta, \alpha + \beta\} = V_H(s; \alpha) + \beta.$$

Assume the corresponding shift identity holds for the next-stage value. Since the transition kernel is Gaussian with additive mean, the next state from $s + \beta$ has the same distribution as $S_{n+1} + \beta$ when the current state is $s$. Therefore

$$\begin{aligned} Q_n^{\text{cont}}(s + \beta; \alpha + \beta) &= -c + \mathbb{E}[V_{n+1}(S_{n+1} + \beta; \alpha + \beta) \mid S_n = s] \\ &= -c + \mathbb{E}[V_{n+1}(S_{n+1}; \alpha) + \beta \mid S_n = s] \\ &= Q_n^{\text{cont}}(s; \alpha) + \beta. \end{aligned}$$

Taking the maximum with the shifted outside reward gives the next-stage value identity needed for the induction. The result follows by backward induction. □

Corollary 3.3 follows by applying Lemma 3.2 with shift $\beta = \alpha$ to the centered state $x = s - \alpha$ and outside reward 0. Define the centered value

$$W_n(x) = V_n(\alpha + x; \alpha) - \alpha.$$

Then the recursion no longer depends on $\alpha$:

$$W_H(x) = \max\{x, 0\},$$

and

$$\begin{aligned} W_n(x) &= \max\{0, q_n(x)\}, \\ q_n(x) &= -c + \mathbb{E}[W_{n+1}(X_{n+1}) \mid X_n = x]\,, \end{aligned}$$

where $X_{n+1} \mid X_n = x \sim \mathcal{N}(x, \sigma_n^2)$ and the recursion for $W_n$ applies for $n < H$. This is the one-dimensional dynamic program: for each stage $n$, the numerical grid is only over the centered posterior-mean advantage $x$.

For comparison with an outside reward $\alpha$, evaluating an unfinished arm is worthwhile exactly when $q_n(s - \alpha) > 0$. Let $r_n$ denote the crossing point satisfying

$$q_n(r_n) = 0.$$

The Gittins index for cost $c$ is therefore

$$G_n^c(s) = s - r_n.$$

The online policy does not need to store the full two-argument value function. During precomputation we represent the one-dimensional functions $W_n$ and $q_n$ on a grid in order to propagate the recursion backward, but the objects used online are the roots $\{r_n\}_{n=0}^{H-1}$ for each distinct cost and variance schedule.

The conditional expectation $\mathbb{E}[W_{n+1}(X_{n+1}) \mid X_n = x]$ in the recursion for $q_n$ is a convolution of the next-stage value grid with a Gaussian transition kernel. On an evenly spaced grid this can be computed by FFT-based convolution, so the index schedule is computed once offline and reused during evaluation.

## B. Dataset Details

### B.1. Dataset Description

**GSM8K and PIQA.** For GSM8K and PIQA, we use the response matrices released with BanditEval (Zhou et al., 2025). The underlying benchmarks are Grade School Math 8K (GSM8K) (Cobbe et al., 2021) and Physical Interaction: Question Answering (PIQA) (Bisk et al., 2020). GSM8K consists of grade-school math word problems, while PIQA evaluates physical commonsense reasoning through question answering.

The BanditEval response matrices are constructed from a collection of publicly available language models and sampling configurations. In particular, BanditEval considers 11 models: GPT2, GPT2-Large, CodeLLaMA, Tulu-7B, Tulu-2-7B, Gemma-7B, Phi2, Llema-7B, LLaMA-2-7B, Mistral-7B, and StarCoder-7B. For each model, responses are generated using three temperature choices $\{0, 0.5, 1\}$, two maximum decoding lengths $\{128, 512\}$, and two zero-shot prompting strategies: directly asking for the answer, and using the chain-of-thought prompt "Let's think step by step." The Cartesian product of these choices yields $11 \times 3 \times 2 \times 2 = 132$ possible model–configuration arms.

However, the released response matrices contain some missing configurations. As a result, the final matrices used in our experiments contain 122 arms for GSM8K and 103 arms for PIQA. Both GSM8K and PIQA contain 1,000 examples in the response matrices. Each benchmark is represented by five independently generated response matrices, corresponding to five random seeds used when querying the LLMs. Since the benchmark examples and arm set are fixed, the differences across these matrices mainly reflect randomness in LLM response generation.

**AlpacaEval.** We use AlpacaEval (Li et al., 2023) to evaluate instruction-following quality. Unlike GSM8K and PIQA, AlpacaEval does *not* use the BanditEval model×prompt configuration matrices. Instead, we construct a response matrix from the AlpacaEval 2.0 leaderboard comparisons annotated by `weighted_alpaca_eval_gpt4_turbo`. Each row corresponds to one AlpacaEval leaderboard model, and each column corresponds to one of the 805 fixed AlpacaEval instructions. Entry $(k, n)$ stores the model's pairwise score against the GPT-4 Turbo baseline on instruction $n$. These scores are continuous values in $[0, 1]$ derived from the auto-annotator's preference probabilities; they are *not* binary correctness labels. Accordingly, the finite-matrix target $\theta_k^{\mathrm{mat}}$ in Eq. (1) is the row mean of these pairwise scores.

The resulting matrix contains 153 leaderboard-model arms and 805 instruction examples. It keeps the AlpacaEval pairwise scores as continuous values rather than converting them to binary outcomes. Each arm is therefore a single leaderboard model rather than a model×sampling-configuration pair. In contrast to GSM8K and PIQA, we use one fixed AlpacaEval matrix rather than five independently generated response matrices; algorithm randomness enters only through the 20 randomized trial seeds used in each sweep.

For configuration-level Bayesian optimization baselines, we convert this matrix into 153 configuration-level arms. Each Bayesian optimization arm evaluates one model on all 805 instructions, revealing the row-average score and consuming that model's all-example evaluation cost. Cost-aware experiments use model-specific token prices from our Alpaca pricing table, with an input-to-output token ratio of 1:8 estimated from typical AlpacaEval prompt/response lengths (Appendix C). For Gittins-S on AlpacaEval, we use the dataset-specific prior mean/variance $(0.2,\ 0.01)$; Gittins-G uses the general prior $(0.5,\ 0.04)$ listed in Table 5.

**MMLU.** For MMLU, we use the DOVE response matrices (Habba et al., 2025). MMLU is a multiple-choice question-answering benchmark consisting of 57 subjects and approximately 14,000 examples in total. We treat each MMLU subject as a separate problem instance. Each subject is evaluated across 15 LLM configurations and 100 prompting techniques, resulting in 1500 arms per subject.

The 15 LLM configurations used in the MMLU response matrices are Llama-3-8B, Llama-3-8B-Instruct, Llama-3-70B-Instruct, CodeLlama-34B-Instruct, FLAN-T5-XL, FLAN-T5-XXL, FLAN-UL2, Merlinite-7B, Mixtral-8x7B-Instruct-v0.1, Mistral-7B-Instruct-v0.2, Gemma-7B, Gemma-7B-IT, Falcon-40B, Mistral-7B-v0.1, and Falcon-180B. The number of examples varies across subjects; the full list of subjects used in our evaluation is provided in Table 1.

### B.2. Dataset Statistics

We provide descriptive statistics for the response matrices used in our evaluation. For each dataset or subject, we compute the empirical quality of each arm as its mean score over benchmark examples. For binary correctness matrices, this score is mean accuracy; for AlpacaEval, it is the mean pairwise preference score. The following figures visualize the resulting distribution of arm qualities. These plots help illustrate the spread of arm performance, the location of the empirical mean, the best arm, and the prior mean used in informative-prior experiments.

**GSM8K and PIQA.** Figure 4 shows the per-arm quality distributions for GSM8K and PIQA across the five independently generated response matrices. The distributions are highly stable across seeds for both datasets. This indicates that, although the response matrices are generated independently, the randomness from LLM response generation introduces only limited variation at the aggregate level. GSM8K has a noticeably lower overall accuracy distribution than PIQA, while PIQA exhibits a tighter concentration of high-quality arms.

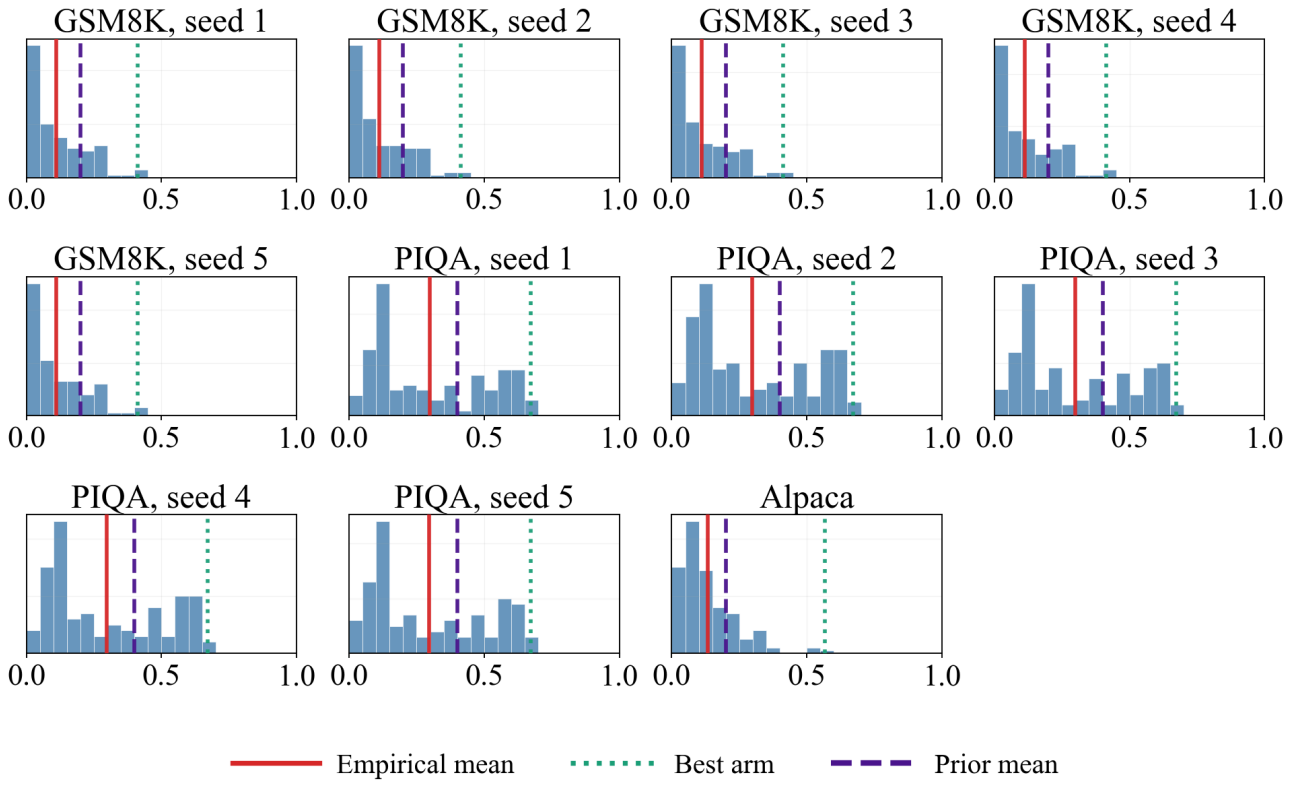



*Figure 4.* Distribution of per-arm quality for GSM8K, PIQA, and AlpacaEval. GSM8K has 122 arms and 1,000 examples, PIQA has 103 arms and 1,000 examples, and AlpacaEval has 153 arms and 805 examples. For GSM8K and PIQA, each subplot corresponds to one dataset-seed pair across five random seeds; for AlpacaEval, the subplot uses the no-rounding-debias matrix used in our experiments. Bars show the histogram of arm quality, defined as mean accuracy or win-rate over queried examples. Vertical lines mark the empirical mean (red), best arm (green), and prior mean (purple). The GSM8K and PIQA distributions are highly similar across seeds, suggesting that the observed variation is small when the only source of randomness is the LLM response generation.

**MMLU.** For MMLU, we treat each subject as a separate problem instance. We divide the subjects into three difficulty buckets according to the empirical mean arm quality of the subject. The intuition is that if the mean arm quality is high, then the subject is easier for the collection of LLM configurations and prompting techniques; conversely, a lower mean arm quality indicates a harder subject. We refer to these buckets as high-, medium-, and low-prior buckets, with prior means $\mu_0 = 0.75$, $\mu_0 = 0.6$, and $\mu_0 = 0.4$, respectively. Figures 5–7 show the per-arm quality distributions for the three buckets.

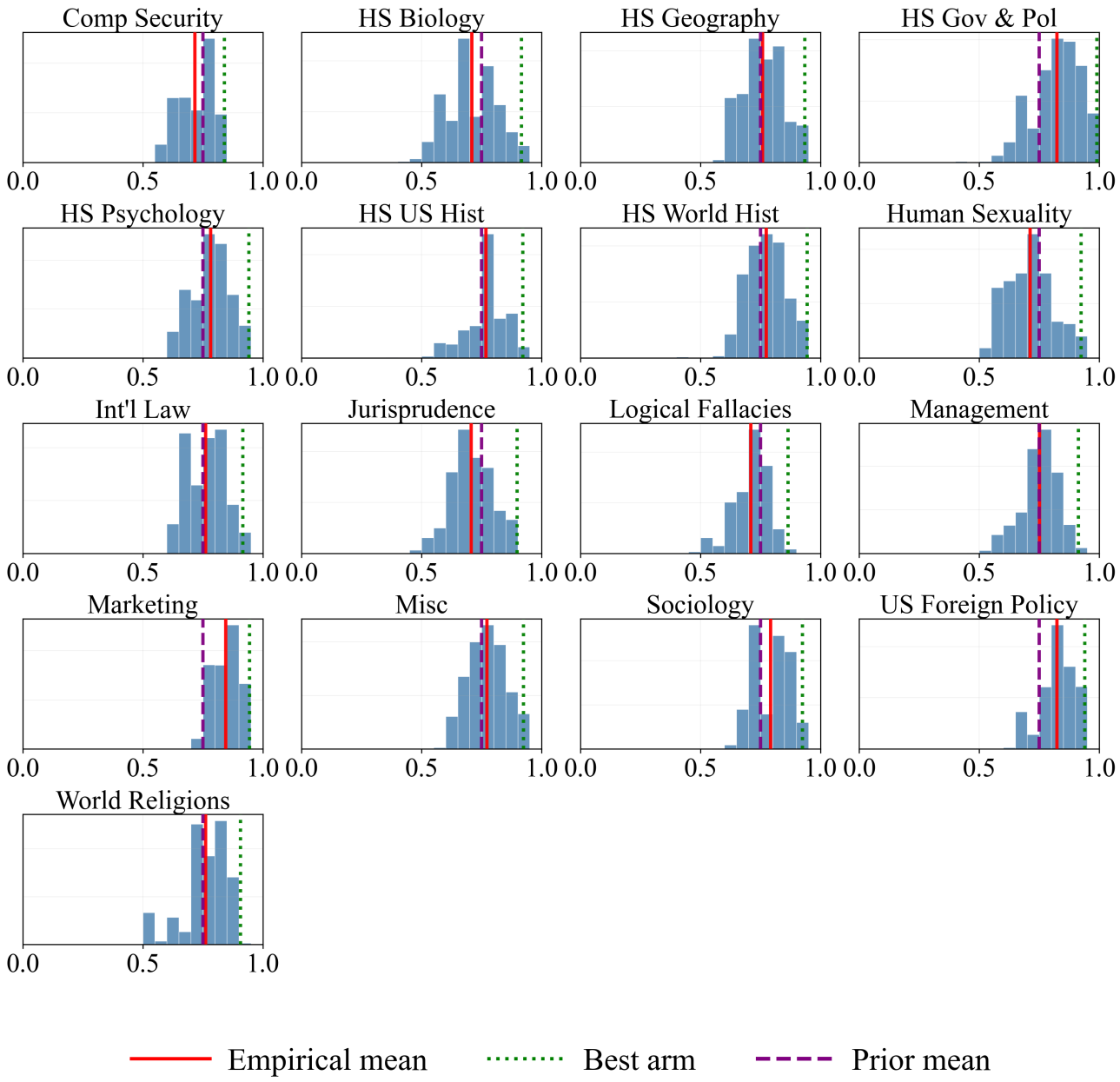



*Figure 5.* Easy subjects: Distribution of per-arm quality for MMLU subjects assigned to the high informative-prior bucket ($\mu_0 = 0.75$). Each subplot corresponds to one subject; bars show the histogram of arm quality, defined as mean accuracy over questions. Vertical lines mark the empirical mean (red), best arm (green), and prior mean (purple).

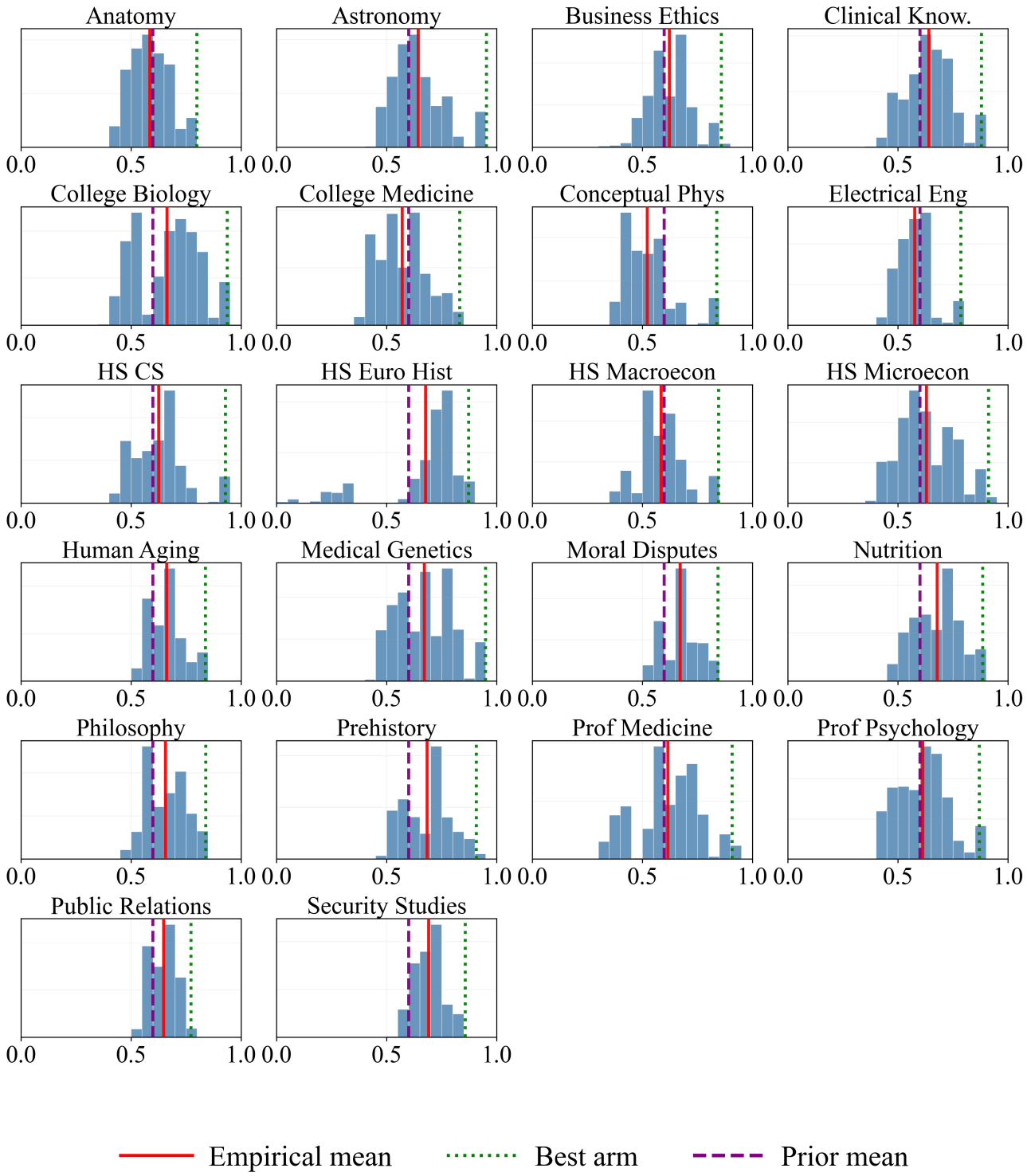



*Figure 6.* Medium subjects: Distribution of per-arm quality for MMLU subjects assigned to the medium informative-prior bucket ($\mu_0 = 0.6$). Each subplot corresponds to one subject; bars show the histogram of arm quality, defined as mean accuracy over questions. Vertical lines mark the empirical mean (red), best arm (green), and prior mean (purple).

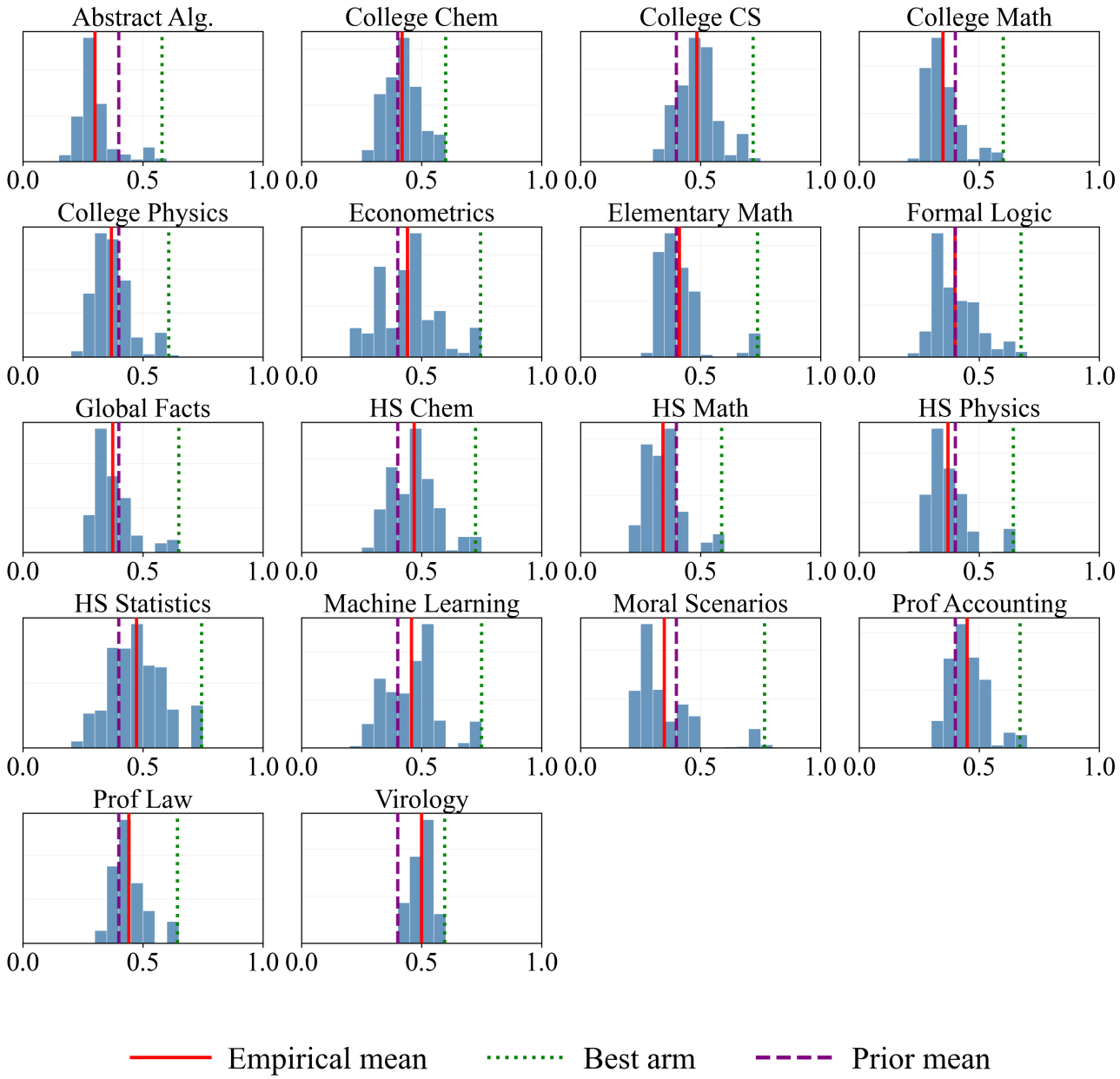



*Figure 7.* Hard subjects: Distribution of per-arm quality for MMLU subjects assigned to the low informative-prior bucket ($\mu_0 = 0.4$). Each subplot corresponds to one subject; bars show the histogram of arm quality, defined as mean accuracy over questions. Vertical lines mark the empirical mean (red), best arm (green), and prior mean (purple).

*Table 1.* MMLU datasets used in our evaluation. Each dataset is evaluated across 15 LLM configurations and 100 prompting techniques, so the number of rows is 1500 times the number of benchmark examples. Difficulty is based on prior accuracy: datasets with high prior accuracy are labeled easy, those with medium prior accuracy are labeled medium, and those with low prior accuracy are labeled hard. The size category groups datasets by the number of benchmark examples.

| Dataset | Rows | Difficulty | Size | Full Name |
|---|---|---|---|---|
| Abstract Alg. | $1500 \times 100$ | hard | small | Abstract Algebra |
| Anatomy | $1500 \times 135$ | medium | small | Anatomy |
| Astronomy | $1500 \times 152$ | medium | medium | Astronomy |
| Business Ethics | $1500 \times 100$ | medium | small | Business Ethics |
| Clinical Know. | $1500 \times 265$ | medium | medium | Clinical Knowledge |
| College Biology | $1500 \times 144$ | medium | small | College Biology |
| College Chem | $1500 \times 100$ | hard | small | College Chemistry |
| College CS | $1500 \times 100$ | hard | small | College Computer Science |
| College Math | $1500 \times 100$ | hard | small | College Mathematics |
| College Medicine | $1500 \times 173$ | medium | medium | College Medicine |
| College Physics | $1500 \times 102$ | hard | small | College Physics |
| Comp Security | $1500 \times 100$ | easy | small | Computer Security |
| Conceptual Phys | $1500 \times 235$ | medium | medium | Conceptual Physics |
| Econometrics | $1500 \times 114$ | hard | small | Econometrics |
| Electrical Eng | $1500 \times 145$ | medium | small | Electrical Engineering |
| Elementary Math | $1500 \times 378$ | hard | medium | Elementary Mathematics |
| Formal Logic | $1500 \times 126$ | hard | small | Formal Logic |
| Global Facts | $1500 \times 100$ | hard | small | Global Facts |
| HS Biology | $1500 \times 310$ | easy | medium | High School Biology |
| HS Chem | $1500 \times 203$ | hard | medium | High School Chemistry |
| HS CS | $1500 \times 100$ | medium | small | High School Computer Science |
| HS Euro Hist | $1500 \times 165$ | medium | medium | High School European History |
| HS Geography | $1500 \times 198$ | easy | medium | High School Geography |
| HS Gov & Pol | $1500 \times 193$ | easy | medium | High School Government and Politics |
| HS Macroecon | $1500 \times 390$ | medium | medium | High School Macroeconomics |
| HS Math | $1500 \times 270$ | hard | medium | High School Mathematics |
| HS Microecon | $1500 \times 238$ | medium | medium | High School Microeconomics |
| HS Physics | $1500 \times 151$ | hard | medium | High School Physics |
| HS Psychology | $1500 \times 545$ | easy | large | High School Psychology |
| HS Statistics | $1500 \times 216$ | hard | medium | High School Statistics |
| HS US Hist | $1500 \times 204$ | easy | medium | High School US History |
| HS World Hist | $1500 \times 237$ | easy | medium | High School World History |
| Human Aging | $1500 \times 223$ | medium | medium | Human Aging |
| Human Sexuality | $1500 \times 131$ | easy | small | Human Sexuality |
| Int'l Law | $1500 \times 121$ | easy | small | International Law |
| Jurisprudence | $1500 \times 108$ | easy | small | Jurisprudence |
| Logical Fallacies | $1500 \times 163$ | easy | medium | Logical Fallacies |
| Machine Learning | $1500 \times 112$ | hard | small | Machine Learning |
| Management | $1500 \times 103$ | easy | small | Management |
| Marketing | $1500 \times 234$ | easy | medium | Marketing |
| Medical Genetics | $1500 \times 100$ | medium | small | Medical Genetics |
| Misc | $1500 \times 783$ | easy | large | Miscellaneous |
| Moral Disputes | $1500 \times 346$ | medium | medium | Moral Disputes |
| Moral Scenarios | $1500 \times 895$ | hard | large | Moral Scenarios |
| Nutrition | $1500 \times 306$ | medium | medium | Nutrition |
| Philosophy | $1500 \times 311$ | medium | medium | Philosophy |
| Prehistory | $1500 \times 324$ | medium | medium | Prehistory |
| Prof Accounting | $1500 \times 282$ | hard | medium | Professional Accounting |
| Prof Law | $1500 \times 1534$ | hard | large | Professional Law |
| Prof Medicine | $1500 \times 272$ | medium | medium | Professional Medicine |
| Prof Psychology | $1500 \times 612$ | medium | large | Professional Psychology |
| Public Relations | $1500 \times 110$ | medium | small | Public Relations |
| Security Studies | $1500 \times 245$ | medium | medium | Security Studies |
| Sociology | $1500 \times 201$ | easy | medium | Sociology |
| US Foreign Policy | $1500 \times 100$ | easy | small | US Foreign Policy |
| Virology | $1500 \times 166$ | hard | medium | Virology |
| World Religions | $1500 \times 171$ | easy | medium | World Religions |

## C. Experiment Setup and Implementation Details

**Computing environment.** Experiments were run on CPU-only computing resources using precomputed response matrices rather than online LLM inference. The response matrix serves as the offline ground-truth oracle: during bandit simulation, each policy observes only the selected $(\text{arm}, \text{question})$ entries, while per-arm matrix means are used to compute simple regret. For the Bayesian optimization baselines, the response matrices are converted into configuration-level inputs, where evaluating one candidate reveals its aggregate score. For GSM8K and PIQA, each converted BanditEval configuration corresponds to a model$\times$prompt arm. For MMLU, each converted DOVE configuration corresponds to a model$\times$template arm. For AlpacaEval, each converted Bayesian optimization configuration corresponds to one leaderboard model evaluated on all $805$ instructions. Reported per-batch runtime therefore measures policy-side allocation overhead rather than model-inference time.

**Experimental repetitions.** For GSM8K and PIQA, we run each adaptive policy on each of the five independently generated response matrices. For every response matrix, we perform $20$ independent algorithm trials, each with independently randomized example orders and algorithmic randomness. We report averages over all matrix–trial pairs. For AlpacaEval, we use a single fixed pairwise score matrix with $153$ models and $805$ instructions. As in MMLU, we run $20$ independent algorithm trials on this matrix and report averages over those trials. Unlike GSM8K and PIQA, we do not average over multiple independently generated response matrices for AlpacaEval. For MMLU, we run $20$ randomized algorithm trials for each subject and aggregate results across the corresponding set of subjects. The Bayesian optimization baselines use the same randomized-trial structure on their converted configuration-level inputs.

**Stopping-rule variants.** For Gittins-based methods, we record two post-pull stopping diagnostics. The primary diagnostic used in the main figures is the *index-induced stopping rule*: after each batch is evaluated and posterior Gittins indices are updated, the policy stops when an arm attaining the largest current Gittins score is already fully observed. This corresponds to the natural termination point of the required-completion Gittins policy, since the policy's own index maximizer has become a completed arm.

We also record a *recommendation-aware stopping rule*. Let $\Gamma_i$ denote the current Gittins score of arm $i$, and let $\mu_i$ denote its posterior mean. This diagnostic stops when

$$\max_{i \in \mathcal{I}_{\text{incomplete}}} \Gamma_i < \max_j \mu_j,$$

that is, when the best remaining exploration opportunity has lower index than the current best posterior-mean recommendation. This rule is more directly tied to the anytime recommendation criterion, whereas the index-induced rule follows the exploration policy itself. We use the index-induced rule in the main figures because it is the stopping diagnostic most directly implied by the required-completion Gittins policy: it asks when the policy's own index maximizer has become a completed arm. The recommendation-aware rule is included only to clarify an alternative diagnostic tied to the anytime recommendation criterion, but it is not used for the stopping markers in the reported main figures.

**Baseline initialization and warm-up.** For a response matrix with $K$ arms and $N$ examples, the Bayesian optimization baselines use a configuration-level random-initialization phase. In our main Bayesian optimization runs, a $5\%$ random initialization means drawing approximately $0.05K$ arms uniformly without replacement and observing the full rows for those arms, i.e., evaluating the selected configurations on all $N$ examples before the acquisition-driven phase begins. This corresponds to 6 initial configurations for GSM8K ($K = 122$), 5 for PIQA ($K = 103$), 8 for AlpacaEval ($K = 153$), and 75 for each MMLU subject ($K = 1500$). Under the nominal $10\%$ Bayesian optimization budget, the corresponding total configuration budgets are 12, 10, 15, and 150 configurations, respectively. This convention is distinct from the low-rank-factorization warm-up in BanditEval (Zhou et al., 2025). There, UCB-E-LRF first draws $T_0$ individual arm–example pairs uniformly from the $K \times N$ response matrix, and the default experimental setting uses $T_0 = 0.05KN$. The standalone LRF baseline in that paper is also written with a $T_0$ entry-level warm-up parameter, so any such warm-up should be interpreted as a percentage of response-matrix entries rather than a percentage of arms evaluated on all benchmark examples.

**Evaluation costs.** The cost-aware bandit experiments assign each arm a batch cost proportional to the estimated cost of running that model configuration on $B$ benchmark examples. For GSM8K and PIQA, an arm is a model plus sampling or prompting configuration; when several arms share a base model, they share the model-level price, while the batch multiplier accounts for how many examples are queried at that allocation step. For MMLU, each arm is a model–prompt

pair, and model-level input prices are repeated across the corresponding prompt arms. For AlpacaEval, each arm is a single leaderboard model rather than a model×sampling-configuration pair. Depending on the available metadata, costs can be computed from API pricing, input/output token counts, generated-token counts, wall-clock latency, or a normalized proxy such as model size. For the Bayesian optimization baselines, each converted input row corresponds to a complete configuration, and its cost is the estimated all-example evaluation cost of that configuration on the corresponding benchmark or MMLU subject. The Gittins dynamic program uses costs rescaled to the same utility units as the index computation, while the Bayesian optimization baselines use configuration-level costs for cost-aware acquisition and budget accounting.

**AlpacaEval evaluation costs.** AlpacaEval uses model-specific API or proxy prices for the 153 leaderboard models in the response matrix. For each model, we record separate input and output costs in USD per 1M tokens and form the evaluation cost as

$$\text{cost} = c_{\text{in}} + 8\, c_{\text{out}},$$

using the estimated AlpacaEval input-to-output token ratio 1:8. These model-level costs define the per-arm cost vector for cost-aware bandit experiments and the all-example evaluation costs used by the converted Bayesian optimization inputs. Because AlpacaEval arms are individual leaderboard models rather than BanditEval-style model×prompt configurations, we do not repeat a single model price across multiple prompting arms as in MMLU. Table 2 and Table 3 therefore do not apply to AlpacaEval; the complete Alpaca pricing table is reported in Table 4.

*Table 2.* Model-level evaluation costs used for GSM8K and PIQA. Costs are reported in USD per 1M tokens. GSM8K uses an assumed input-to-output token ratio of $1:2$, so the evaluation cost is computed as input cost plus twice the output cost. PIQA uses input-only pricing. These model-level costs are used to construct the per-arm cost vectors for the cost-aware experiments.

| Model | Input cost | Output cost | GSM8K cost | PIQA cost |
|---|---|---|---|---|
| CodeLlama | 0.30 | 0.30 | 0.90 | 0.30 |
| Gemma-7B | 0.20 | 0.20 | 0.60 | 0.20 |
| GPT-2 | 0.10 | 0.10 | 0.30 | 0.10 |
| GPT-2 Large | 0.10 | 0.10 | 0.30 | 0.10 |
| LLaMA2-7B | 0.20 | 0.20 | 0.60 | 0.20 |
| Llemma-7B | 0.80 | 1.20 | 3.20 | 0.80 |
| Mistral-7B | 0.05 | 0.20 | 0.45 | 0.05 |
| Phi-2 | 0.05 | 0.10 | 0.25 | 0.05 |
| StarCoder2-7B | 0.20 | 0.20 | 0.60 | 0.20 |
| Tulu | 0.20 | 0.20 | 0.60 | 0.20 |
| Tulu2 | 0.20 | 0.20 | 0.60 | 0.20 |

*Table 3.* Model-level input costs used for MMLU. Costs are reported in USD per 1M input tokens. MMLU uses input-only pricing. Each MMLU subject contains 1500 arms, corresponding to 15 models paired with 100 prompt configurations; therefore, each model-level input cost is repeated across the corresponding prompt arms.

| MMLU model | Input cost |
|---|---|
| Llama-3-8B | 0.05 |
| Llama-3-8B-Instruct | 0.03 |
| Llama-3-70B-Instruct | 0.51 |
| CodeLlama-34B-Instruct | 0.776 |
| FLAN-T5-XL | 0.60 |
| FLAN-T5-XXL | 1.80 |
| FLAN-UL2 | 5.00 |
| Merlinite-7B | 0.60 |
| Mixtral-8x7B-Instruct-v0.1 | 0.54 |
| Mistral-7B-Instruct-v0.2 | 0.14 |
| Gemma-7B | 0.20 |
| Gemma-7B-IT | 0.07 |
| Falcon-40B | 0.84 |
| Mistral-7B-v0.1 | 0.11 |
| Falcon-180B | 1.25 |

*Table 4.* AlpacaEval model-level pricing table for the 153-model response matrix. Input and output prices are in USD per 1M tokens. The combined cost is input cost $+ 8\times$ output cost, matching the AlpacaEval input-to-output token ratio used for cost-aware runs.

| Model | Input | Output | Combined |
|---|---|---|---|
| Conifer-7B-DPO | 0.2 | 0.2 | 1.8 |
| Contextual-KTO-Mistral-PairRM | 0.2 | 0.2 | 1.8 |
| FsfairX-Zephyr-Chat-v0.1 | 0.2 | 0.2 | 1.8 |
| LMCocktail-10.7B-v1 | 0.2 | 0.2 | 1.8 |
| Meta-Llama-3-70B-Instruct | 0.9 | 0.9 | 8.1 |
| Meta-Llama-3-8B-Instruct | 0.2 | 0.2 | 1.8 |
| Mistral-7B-Instruct-v0.2 | 0.2 | 0.2 | 1.8 |
| Mistral-7B-ReMax-v0.1 | 0.2 | 0.2 | 1.8 |
| Mixtral-8x22B-Instruct-v0.1 | 1.2 | 1.2 | 10.8 |
| Mixtral-8x7B-Instruct-v0.1 | 0.5 | 0.5 | 4.5 |
| Mixtral-8x7B-Instruct-v0.1_concise | 0.5 | 0.5 | 4.5 |
| Mixtral-8x7B-Instruct-v0.1_verbose | 0.5 | 0.5 | 4.5 |
| Nanbeige-Plus-Chat-v0.1 | 0.2 | 0.2 | 1.8 |
| Nanbeige2-8B-Chat | 0.2 | 0.2 | 1.8 |
| OpenHermes-2.5-Mistral-7B | 0.2 | 0.2 | 1.8 |
| Qwen-14B-Chat | 0.2 | 0.2 | 1.8 |
| Qwen1.5-1.8B-Chat | 0.1 | 0.1 | 0.9 |
| Qwen1.5-110B-Chat | 0.9 | 0.9 | 8.1 |
| Qwen1.5-14B-Chat | 0.2 | 0.2 | 1.8 |
| Qwen1.5-72B-Chat | 0.1 | 0.1 | 0.9 |
| Qwen1.5-7B-Chat | 0.2 | 0.2 | 1.8 |
| Samba-CoE-v0.1 | 0.2 | 0.2 | 1.8 |
| Samba-CoE-v0.2-best-of-16 | 3.2 | 3.2 | 28.8 |
| Samba-CoE-v0.2 | 0.2 | 0.2 | 1.8 |
| Snorkel-Mistral-PairRM-DPO-best-of-16 | 3.2 | 3.2 | 28.8 |
| Snorkel-Mistral-PairRM-DPO | 0.2 | 0.2 | 1.8 |
| Starling-LM-7B-alpha | 0.2 | 0.2 | 1.8 |
| TempNet-LLaMA2-Chat-13B-v0.1 | 0.2 | 0.2 | 1.8 |
| TempNet-LLaMA2-Chat-70B-v0.1 | 0.9 | 0.9 | 8.1 |
| TempNet-LLaMA2-Chat-7B-v0.1 | 0.2 | 0.2 | 1.8 |
| Yi-34B-Chat | 0.9 | 0.9 | 8.1 |
| airoboros-33b | 0.9 | 0.9 | 8.1 |
| airoboros-65b | 0.9 | 0.9 | 8.1 |
| aligner-2b_claude-3-opus-20240229 | 0.1 | 0.1 | 0.9 |
| aligner-2b_qwen1.5-72b-chat | 0.1 | 0.1 | 0.9 |
| LLaMA-7B | 0.2 | 0.2 | 1.8 |
| alpaca-7b | 0.2 | 0.2 | 1.8 |
| alpaca-7b_concise | 0.2 | 0.2 | 1.8 |
| alpaca-7b_verbose | 0.2 | 0.2 | 1.8 |
| alpaca-farm-ppo-human | 0.2 | 0.2 | 1.8 |
| alpaca-farm-ppo-sim-gpt4-20k | 0.2 | 0.2 | 1.8 |
| baichuan-13b-chat | 0.2 | 0.2 | 1.8 |
| baize-v2-13b | 0.2 | 0.2 | 1.8 |
| baize-v2-7b | 0.2 | 0.2 | 1.8 |
| causallm-14b | 0.2 | 0.2 | 1.8 |
| chatglm2-6b | 0.2 | 0.2 | 1.8 |
| claude-2.1 | 8 | 24 | 200 |
| claude-2.1_concise | 8 | 24 | 200 |
| claude-2.1_verbose | 8 | 24 | 200 |
| claude-2 | 8 | 24 | 200 |
| claude-3-opus-20240229 | 15 | 75 | 615 |
| claude-3-sonnet-20240229 | 3 | 15 | 123 |
| claude-instant-1.2 | 0.8 | 2.4 | 20 |
| claude | 8 | 24 | 200 |
| claude2-alpaca-13b | 0.2 | 0.2 | 1.8 |
| cohere | 1 | 2 | 17 |
| cut-13b | 0.2 | 0.2 | 1.8 |
| dbrx-instruct | 1.2 | 1.2 | 10.8 |
| deepseek-llm-67b-chat | 0.9 | 0.9 | 8.1 |

*Continued on next page*

*Table 4.* AlpacaEval model-level pricing table (continued).

| Model | Input | Output | Combined |
|---|---|---|---|
| deita-7b-v1.0 | 0.2 | 0.2 | 1.8 |
| dolphin-2.2.1-mistral-7b | 0.2 | 0.2 | 1.8 |
| evo-7b | 0.2 | 0.2 | 1.8 |
| evo-v2-7b | 0.2 | 0.2 | 1.8 |
| falcon-40b-instruct | 0.9 | 0.9 | 8.1 |
| falcon-7b-instruct | 0.2 | 0.2 | 1.8 |
| gemini-pro | 0.5 | 1.5 | 12.5 |
| gemma-2b-it | 0.1 | 0.1 | 0.9 |
| gemma-7b-it | 0.2 | 0.2 | 1.8 |
| gpt-3.5-turbo-0301 | 1.5 | 2 | 17.5 |
| gpt-3.5-turbo-0613 | 1.5 | 2 | 17.5 |
| gpt-3.5-turbo-1106 | 1 | 2 | 17 |
| gpt-3.5-turbo-1106_concise | 1 | 2 | 17 |
| gpt-3.5-turbo-1106_verbose | 1 | 2 | 17 |
| gpt-4-0125-preview | 10 | 30 | 250 |
| gpt35_turbo_instruct | 1.5 | 2 | 17.5 |
| gpt4 | 30 | 60 | 510 |
| gpt4_0314 | 30 | 60 | 510 |
| gpt4_0613 | 30 | 60 | 510 |
| gpt4_0613_concise | 30 | 60 | 510 |
| gpt4_0613_verbose | 30 | 60 | 510 |
| gpt4_1106_preview | 10 | 30 | 250 |
| gpt4_1106_preview_concise | 10 | 30 | 250 |
| gpt4_gamed | 30 | 60 | 510 |
| guanaco-13b | 0.2 | 0.2 | 1.8 |
| guanaco-33b | 0.9 | 0.9 | 8.1 |
| guanaco-65b | 0.9 | 0.9 | 8.1 |
| guanaco-7b | 0.2 | 0.2 | 1.8 |
| humpback-llama-65b | 0.9 | 0.9 | 8.1 |
| humpback-llama2-70b | 0.9 | 0.9 | 8.1 |
| internlm2-chat-20b-ppo | 0.9 | 0.9 | 8.1 |
| jina-chat | 0.2 | 0.2 | 1.8 |
| llama-2-13b-chat-hf | 0.2 | 0.2 | 1.8 |
| llama-2-70b-chat-hf | 0.9 | 0.9 | 8.1 |
| llama-2-7b-chat-hf | 0.2 | 0.2 | 1.8 |
| llama-2-chat-7b-evol70k-neft | 0.2 | 0.2 | 1.8 |
| minichat-1.5-3b | 0.1 | 0.1 | 0.9 |
| minichat-3b | 0.1 | 0.1 | 0.9 |
| minotaur-13b | 0.2 | 0.2 | 1.8 |
| mistral-large-2402 | 8 | 24 | 200 |
| mistral-medium | 2.7 | 8.1 | 67.5 |
| mistral-orpo-beta | 0.2 | 0.2 | 1.8 |
| nous-hermes-13b | 0.2 | 0.2 | 1.8 |
| oasst-rlhf-llama-33b | 0.9 | 0.9 | 8.1 |
| oasst-sft-llama-33b | 0.9 | 0.9 | 8.1 |
| oasst-sft-pythia-12b | 0.1 | 0.1 | 0.9 |
| openbuddy-falcon-40b-v9 | 0.9 | 0.9 | 8.1 |
| openbuddy-falcon-7b-v6 | 0.2 | 0.2 | 1.8 |
| openbuddy-llama-30b-v7.1 | 0.9 | 0.9 | 8.1 |
| openbuddy-llama-65b-v8 | 0.9 | 0.9 | 8.1 |
| openbuddy-llama2-13b-v11.1 | 0.2 | 0.2 | 1.8 |
| openbuddy-llama2-70b-v10.1 | 0.9 | 0.9 | 8.1 |
| openchat-13b | 0.2 | 0.2 | 1.8 |
| openchat-v2-13b | 0.2 | 0.2 | 1.8 |
| openchat-v2-w-13b | 0.2 | 0.2 | 1.8 |
| openchat-v3.1-13b | 0.2 | 0.2 | 1.8 |
| openchat8192-13b | 0.2 | 0.2 | 1.8 |
| opencoderplus-15b | 0.2 | 0.2 | 1.8 |
| pairrm-Yi-34B-Chat | 0.9 | 0.9 | 8.1 |
| pairrm-tulu-2-13b | 0.2 | 0.2 | 1.8 |

*Continued on next page*

*Table 4.* AlpacaEval model-level pricing table (continued).

| Model | Input | Output | Combined |
|---|---|---|---|
| pairrm-tulu-2-70b | 0.9 | 0.9 | 8.1 |
| pairrm-zephyr-7b-beta | 0.2 | 0.2 | 1.8 |
| phi-2-dpo | 0.1 | 0.1 | 0.9 |
| phi-2-sft | 0.1 | 0.1 | 0.9 |
| phi-2 | 0.1 | 0.1 | 0.9 |
| platolm-7b | 0.2 | 0.2 | 1.8 |
| pythia-12b-mix-sft | 0.1 | 0.1 | 0.9 |
| recycled-wizardlm-7b-v1.0 | 0.2 | 0.2 | 1.8 |
| recycled-wizardlm-7b-v2.0 | 0.2 | 0.2 | 1.8 |
| text_davinci_001 | 20 | 20 | 180 |
| text_davinci_003 | 20 | 20 | 180 |
| tulu-2-dpo-13b | 0.2 | 0.2 | 1.8 |
| tulu-2-dpo-70b | 0.9 | 0.9 | 8.1 |
| tulu-2-dpo-7b | 0.2 | 0.2 | 1.8 |
| ultralm-13b-best-of-16 | 3.2 | 3.2 | 28.8 |
| ultralm-13b-v2.0-best-of-16 | 3.2 | 3.2 | 28.8 |
| ultralm-13b-v2.0 | 0.2 | 0.2 | 1.8 |
| ultralm-13b | 0.2 | 0.2 | 1.8 |
| vicuna-13b-v1.3 | 0.2 | 0.2 | 1.8 |
| vicuna-13b-v1.5 | 0.2 | 0.2 | 1.8 |
| vicuna-13b | 0.2 | 0.2 | 1.8 |
| vicuna-33b-v1.3 | 0.9 | 0.9 | 8.1 |
| vicuna-7b-v1.3 | 0.2 | 0.2 | 1.8 |
| vicuna-7b-v1.5 | 0.2 | 0.2 | 1.8 |
| vicuna-7b | 0.2 | 0.2 | 1.8 |
| wizardlm-13b-v1.1 | 0.2 | 0.2 | 1.8 |
| wizardlm-13b-v1.2 | 0.2 | 0.2 | 1.8 |
| wizardlm-13b | 0.2 | 0.2 | 1.8 |
| wizardlm-70b | 0.9 | 0.9 | 8.1 |
| xwinlm-13b-v0.1 | 0.2 | 0.2 | 1.8 |
| xwinlm-70b-v0.1 | 0.9 | 0.9 | 8.1 |
| xwinlm-7b-v0.1 | 0.2 | 0.2 | 1.8 |
| zephyr-7b-alpha | 0.2 | 0.2 | 1.8 |
| zephyr-7b-beta | 0.2 | 0.2 | 1.8 |

**Gaussian approximation.** For benchmark scores grouped in batches, a batch of $B$ responses produces an empirical mean score. For binary-accuracy benchmarks such as GSM8K, PIQA, and MMLU, the default observation variance is $\tau^2 = 1/(4B)$, the worst-case binary variance divided by the batch size. For AlpacaEval, matrix entries are continuous pairwise preference scores in $[0, 1]$ rather than binary correctness labels. We therefore do not use the binary worst-case variance $1/(4B)$ as the default observation noise for AlpacaEval; instead, we use the same batch-noise scale as in the main reported Gittins runs for that benchmark. For other score types, we estimate $\tau^2$ from pilot batches or historical response matrices, and we also consider plug-in variances in which $\tau^2$ is updated from the current posterior mean.

**Prior settings.** All arms use the same prior mean and variance, representing a shared belief before benchmark-specific evidence is collected. Table 5 lists the general default prior and the data-specific common priors used by Gittins-G and Gittins-S. Figures 5, 6, and 7 show the MMLU latent-accuracy distributions used to motivate the informative prior buckets.

## D. Additional Experiment Results

### D.1. Ablation Studies

We include three ablations to separate the statistical and cost-sensitive components of the Gittins index policy.

**Choice of common prior.** We compare the common priors listed in Table 5. This ablation tests whether Bayesian allocation gains come from useful shared prior information or from the index rule alone, without giving different arms different prior means.

*Table 5.* Prior settings used in the Gittins-index experiments. We report Gaussian priors as $\mathcal{N}(\mu_0, v_0)$. The default prior is a general accuracy-scale choice. Dataset-specific priors provide shared benchmark-level information without using arm-specific prior means.

| Benchmark / group | Prior | Description |
|---|---|---|
| All benchmarks | $\mathcal{N}(0.5, 0.04)$ | General default prior |
| GSM8K | $\mathcal{N}(0.2, 0.01)$ | Very hard benchmark |
| PIQA | $\mathcal{N}(0.4, 0.02)$ | Hard benchmark |
| AlpacaEval | $\mathcal{N}(0.2, 0.01)$ | Very hard benchmark |
| MMLU low-accuracy bucket | $\mathcal{N}(0.4, 0.02)$ | Hard subjects |
| MMLU medium-accuracy bucket | $\mathcal{N}(0.6, 0.02)$ | Medium subjects |
| MMLU high-accuracy bucket | $\mathcal{N}(0.75, 0.01)$ | Easy subjects |

*Table 6.* Batch-size settings used in the simple-regret experiments. For UCB-E and both Gittins variants, batch size $B$ denotes the number of examples queried for the selected arm at each allocation step. For AlpacaEval, these examples correspond to instruction-level pairwise preference scores rather than binary correctness labels. UCB-E-LRF uses $B = 32$ in all reported settings. Bayesian optimization baselines operate at the configuration level, so evaluating one candidate reveals its aggregate score and does not use a bandit batch size. For MMLU, subject matrices are grouped by the number of benchmark examples, and the batch-size grid for UCB-E and Gittins is chosen according to this subject-size bucket.

| Benchmark / group | Matrices | Example-count rule | Batch sizes |
|---|---|---|---|
| GSM8K | 5 response matrices | 1000 examples each | $B \in \{8, 16, 32\}$ |
| PIQA | 5 response matrices | 1000 examples each | $B \in \{8, 16, 32\}$ |
| AlpacaEval | 1 pairwise score matrix | 805 instructions | $B \in \{8, 16, 32\}$ |
| MMLU small | 22 subject matrices | $n_{\text{examples}} \leq 150$ | $B \in \{2, 4, 8\}$ |
| MMLU medium | 30 subject matrices | $151 \leq n_{\text{examples}} \leq 400$ | $B \in \{4, 8, 16\}$ |
| MMLU large | 5 subject matrices | $n_{\text{examples}} > 400$ | $B \in \{8, 16, 32\}$ |

**Batch size.** For UCB-E and the Gittins variants, we vary the batch size $B$ used to construct empirical batch-mean observations. Smaller batches provide more frequent adaptation but noisier observations; larger batches reduce Gaussian approximation error and posterior noise but make each allocation decision coarser. UCB-E-LRF uses $B = 32$ in the reported settings, while Bayesian optimization baselines operate at the configuration level and therefore do not use a bandit batch size. Table 6 summarizes the batch-size grids used for GSM8K, PIQA, MMLU, and AlpacaEval. On AlpacaEval, each batch query corresponds to $B$ additional pairwise comparisons for the selected model rather than $B$ independent benchmark examples with binary labels.

**Cost-scaling factor.** The Gittins index policy uses a cost-scaling factor to map evaluation costs into the continuation penalty in the index computation. In the unit-cost setting, this factor is applied to a common unit cost for each evaluation; in the cost-aware setting, it is applied to the original per-evaluation costs. This parameter controls the effective price of continuing to sample: larger values make additional evaluations more costly and therefore favor earlier stopping, whereas smaller values encourage longer exploration. In our experiments, we sweep three cost-scaling factors, $10^{-3}$, $10^{-4}$, and $10^{-5}$, for both unit-cost and cost-aware Gittins variants. Unless otherwise specified, the main reported figures use $10^{-4}$, which we found to provide a representative balance between stopping early and continuing exploration.

### D.2. Per-Subject MMLU Results

We provide additional MMLU results under informative priors in Figures 8–13. The subjects are grouped into easy, medium, and hard groups according to their empirical mean arm quality. The corresponding prior means are $\mu_0 = 0.75$, $\mu_0 = 0.6$, and $\mu_0 = 0.4$, respectively. For each bucket, we report both unit-cost results, where the horizontal axis is the number of evaluations, and cost-aware results, where the horizontal axis is cumulative evaluation cost. Across the MMLU subjects, the Gittins-based policies generally improve over UCB-E, but the stronger variant depends on the difficulty bucket. Across the MMLU subjects, Gittins-S generally reaches lower simple regret earlier than UCB-E and Gittins-G across difficulty buckets. This suggests that the data-specific prior is broadly helpful for MMLU, especially when subjects are more challenging.

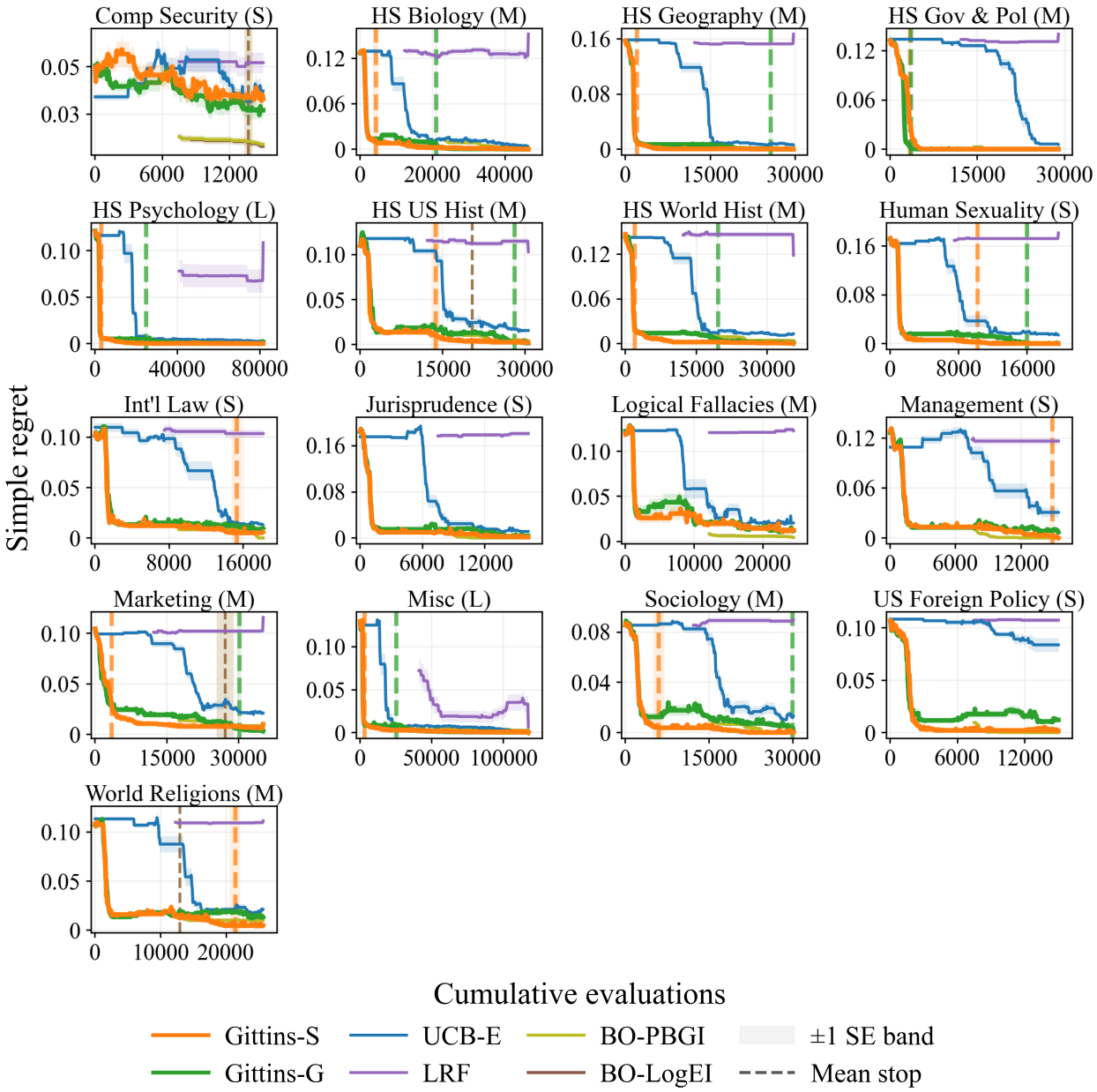



*Figure 8.* MMLU unit-cost results for easy subjects (high prior bucket). Simple regret is plotted against cumulative evaluations. Subplot labels S, M, and L indicate small, medium, and large subject matrices, corresponding to batch sizes $B = 2$, $B = 4$, and $B = 8$, respectively; LRF uses $B = 32$ throughout. Shaded regions denote $\pm 1$ standard errors across seeds. Dashed vertical lines mark mean stopping times, with faint vertical bands showing $\pm 1$ standard errors. On easier subjects, the Gittins variants usually reach low simple regret rapidly and often outperform UCB-E. Bayesian optimization is competitive on several tasks but is less uniformly strong, while LRF is generally less effective and less stable.

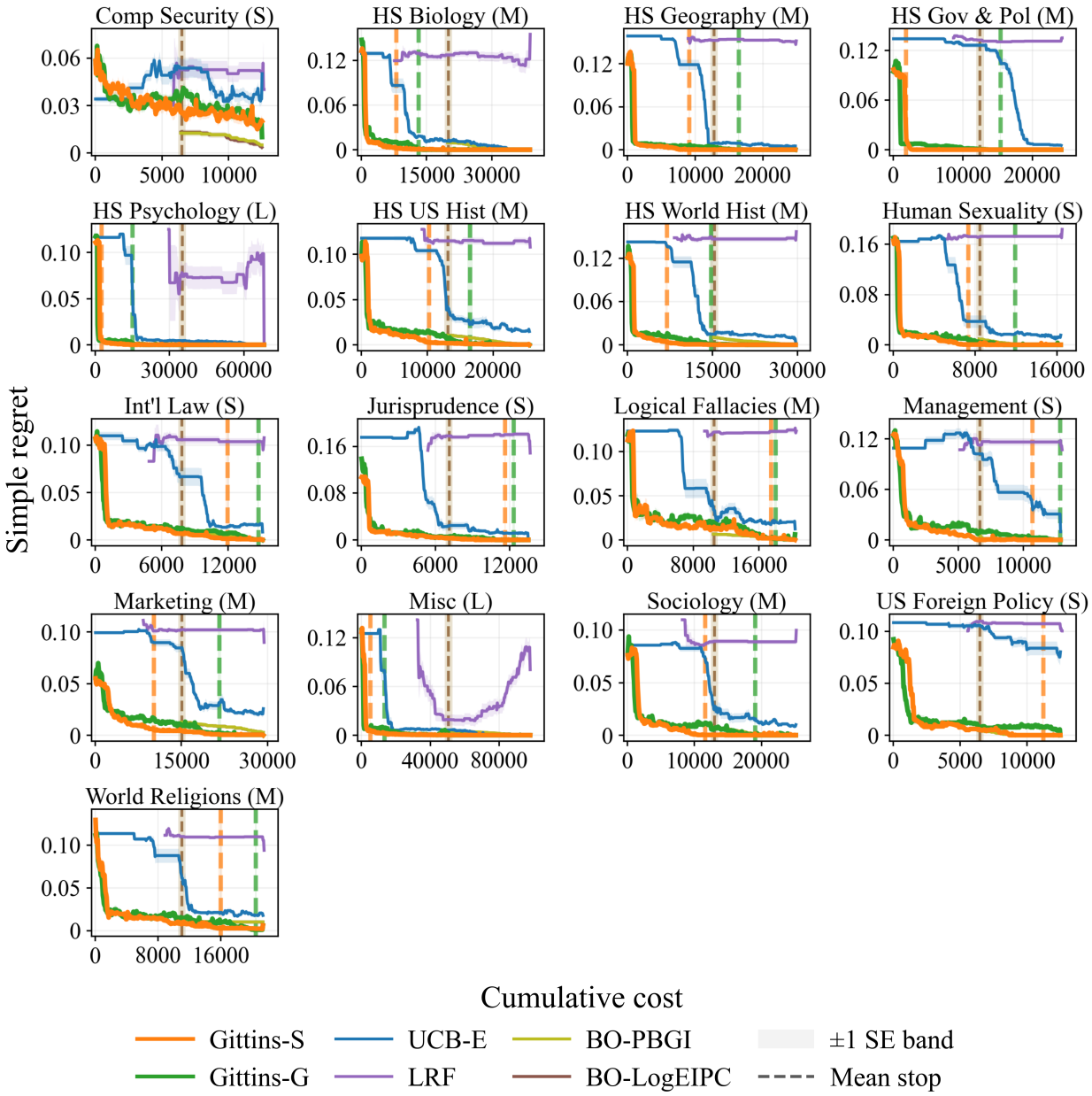



*Figure 9.* MMLU cost-aware results for easy subjects (high prior bucket). Simple regret is plotted against cumulative cost. Subplot labels S, M, and L indicate small, medium, and large subject matrices, corresponding to batch sizes $B = 2$, $B = 4$, and $B = 8$, respectively; LRF uses $B = 32$ throughout. Shaded regions denote $\pm 1$ standard errors across seeds. Dashed vertical lines mark mean stopping times, with faint vertical bands showing $\pm 1$ standard errors. In this cost-aware setting, Gittins variants typically reach low simple regret at relatively small cumulative cost. Bayesian optimization is strong on some subjects but is not consistently better than Gittins, and LRF remains comparatively weak and unstable.

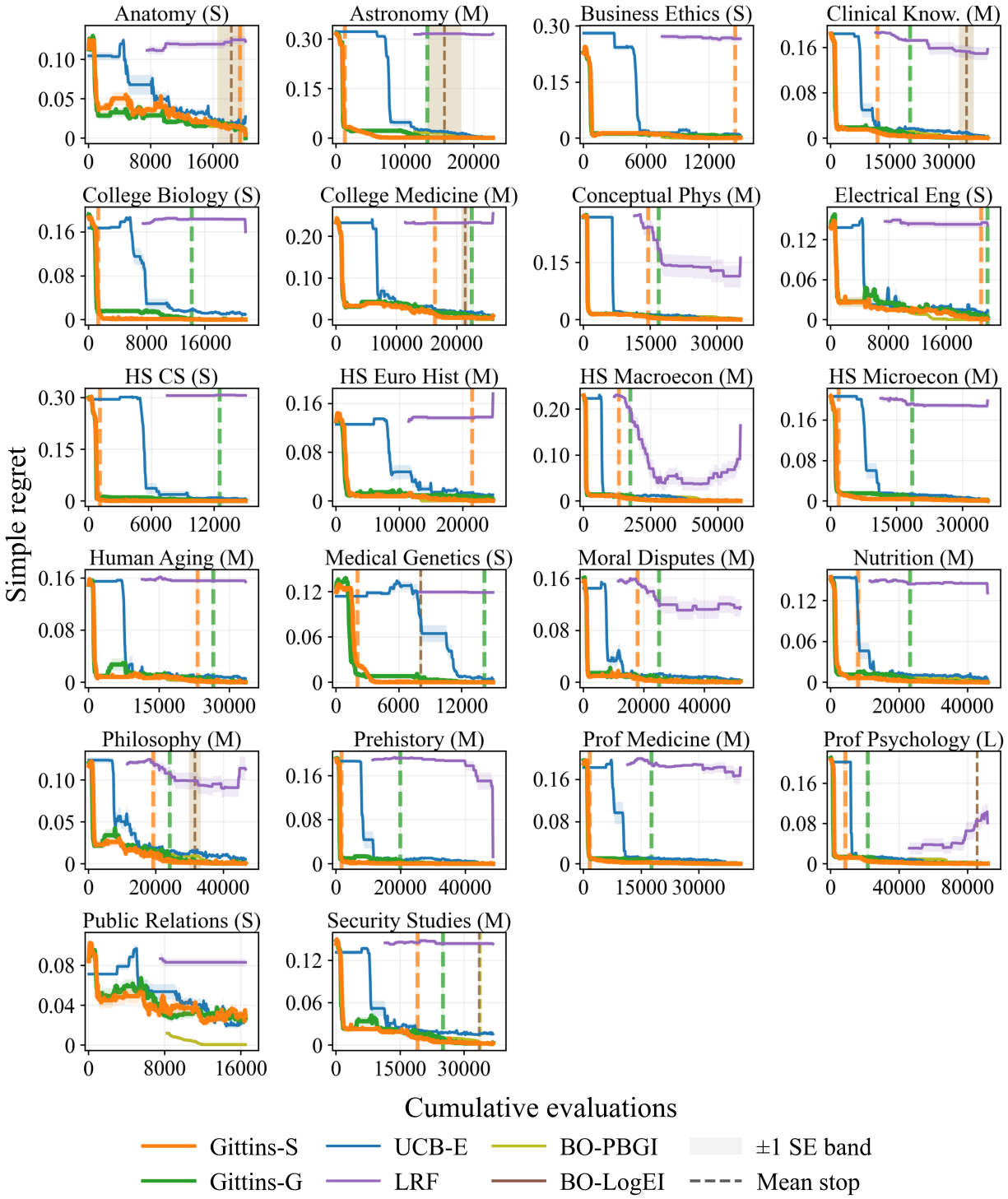



*Figure 10.* MMLU unit-cost results for medium-difficulty subjects (medium prior bucket). Simple regret is plotted against cumulative evaluations. Subplot labels S, M, and L indicate small, medium, and large subject matrices, corresponding to batch sizes $B = 2$, $B = 4$, and $B = 8$, respectively; LRF uses $B = 32$ throughout. Shaded regions denote $\pm 1$ standard errors across seeds. Dashed vertical lines mark mean stopping times, with faint vertical bands showing $\pm 1$ standard errors. Across these mixed-difficulty subjects, Gittins-S is often the most reliable variant, reaching low simple regret earlier than UCB-E on many tasks. Gittins-G remains competitive on several subjects, and Bayesian optimization performs well when the arm-level configuration structure is informative. LRF is generally less effective.

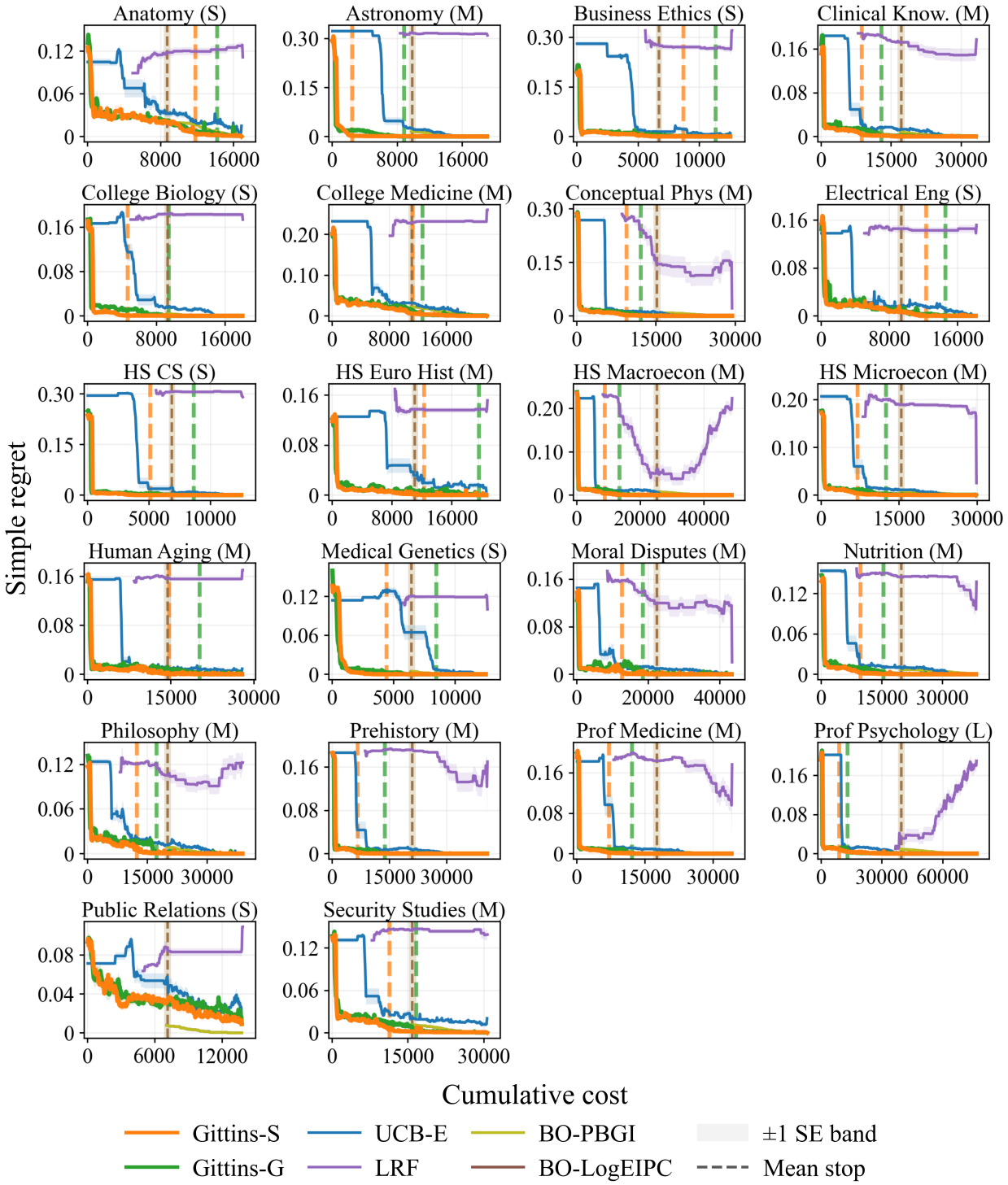



*Figure 11.* MMLU cost-aware results for medium-difficulty subjects (medium prior bucket). Simple regret is plotted against cumulative cost. Subplot labels S, M, and L indicate small, medium, and large subject matrices, corresponding to batch sizes $B = 2$, $B = 4$, and $B = 8$, respectively; LRF uses $B = 32$ throughout. Shaded regions denote $\pm 1$ standard errors across seeds. Dashed vertical lines mark mean stopping times, with faint vertical bands showing $\pm 1$ standard errors. The Gittins variants usually achieve low simple regret with substantially smaller cumulative cost than UCB-E. Gittins-S is often the strongest and most stable method in this bucket, while Bayesian optimization is competitive on selected subjects. LRF remains less stable and typically underperforms.

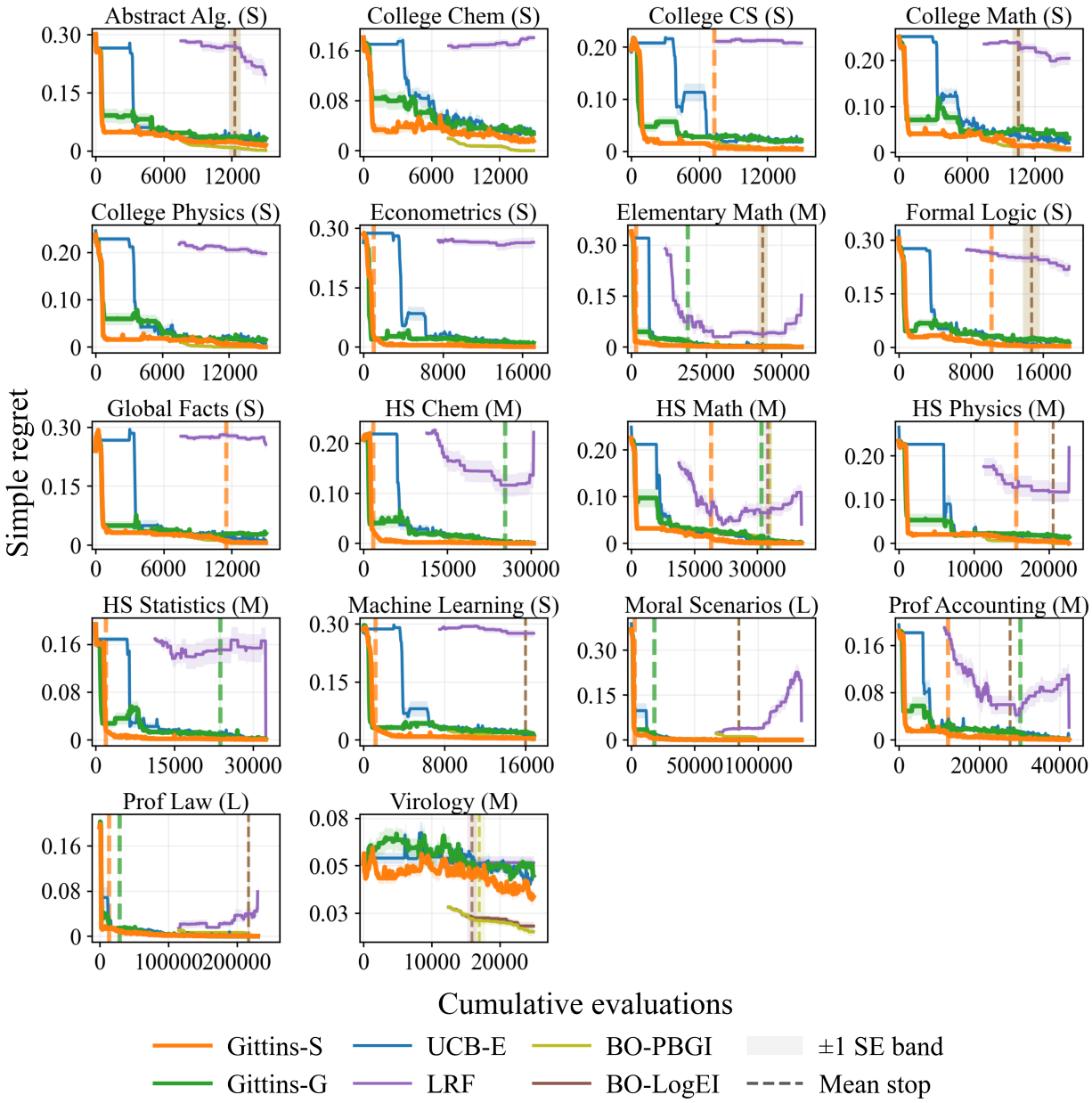



*Figure 12.* MMLU unit-cost results for hard subjects (low prior bucket). Simple regret is plotted against cumulative evaluations. Subplot labels S, M, and L indicate small, medium, and large subject matrices, corresponding to batch sizes $B = 2$, $B = 4$, and $B = 8$, respectively; LRF uses $B = 32$ throughout. Shaded regions denote $\pm 1$ standard errors across seeds. Dashed vertical lines mark mean stopping times, with faint vertical bands showing $\pm 1$ standard errors. On hard subjects, Gittins-S is usually the most effective and stable method, often reaching low simple regret earlier than UCB-E and more consistently than Gittins-G. Bayesian optimization is strong on a few subjects but is not uniformly dominant, while LRF is substantially less reliable.

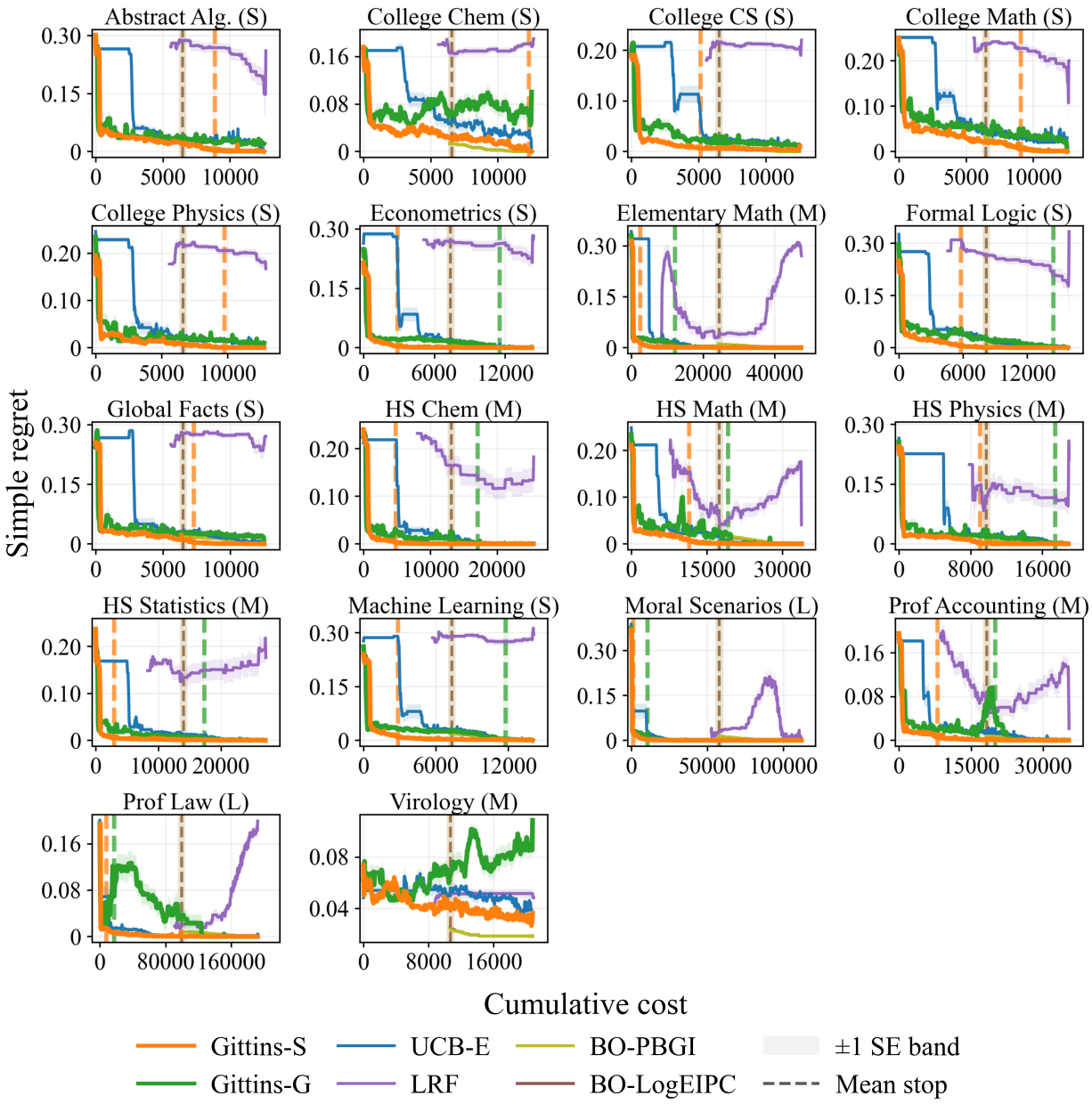



*Figure 13.* MMLU cost-aware results for hard subjects (low prior bucket). Simple regret is plotted against cumulative cost. Subplot labels S, M, and L indicate small, medium, and large subject matrices, corresponding to batch sizes $B = 2$, $B = 4$, and $B = 8$, respectively; LRF uses $B = 32$ throughout. Shaded regions denote $\pm 1$ standard errors across seeds. Dashed vertical lines mark mean stopping times, with faint vertical bands showing $\pm 1$ standard errors. In this harder cost-aware setting, Gittins-S generally provides the strongest overall performance, reaching low simple regret with smaller cumulative cost than UCB-E on most subjects. Gittins-G can be competitive on some tasks but is less stable under heterogeneous costs. Bayesian optimization performs well on selected tasks, while LRF remains the least reliable baseline.

### D.3. Computational Overhead Comparison

**Timing protocol.** Wall-clock timing measurements report the total runtime of each allocation policy over a completed simulated evaluation run. Since the response matrices are precomputed, each evaluation is implemented as an array lookup rather than an LLM inference call. Therefore, the reported runtime reflects the computational overhead of the allocation procedure itself, including method-specific setup and the repeated online computation required during the run. This includes confidence-bound computation for UCB-E, low-rank-factorization updates for UCB-E-LRF, Gittins-index computation for the Gittins variants, and Gaussian-process fitting and acquisition computation for the Bayesian optimization baselines. These measurements should be interpreted as simulation-time algorithmic overhead rather than model-serving or benchmark-construction cost.

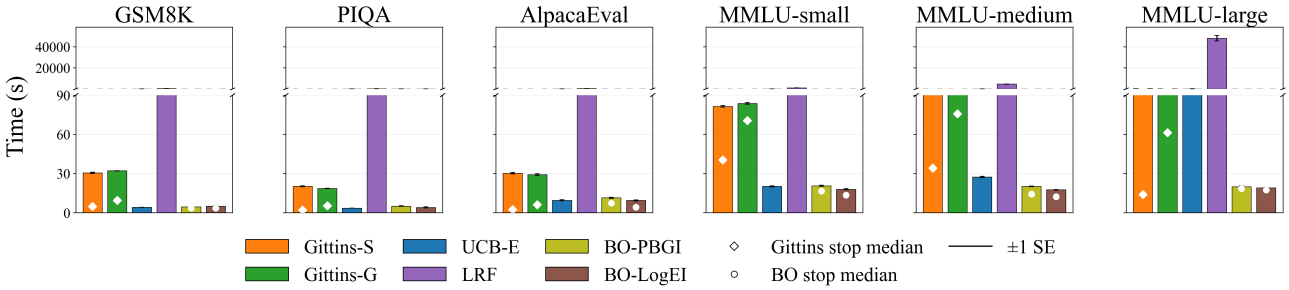



*Figure 14.* Total wall-clock runtime under the unit-cost setting. Each panel reports the total runtime for a completed simulated allocation run on GSM8K, PIQA, AlpacaEval, MMLU-small, MMLU-medium, and MMLU-large. Bars show medians across completed runs, with black error bars indicating $\pm 1$ standard error. Hollow diamonds mark median Gittins stopping times, and hollow circles mark median Bayesian optimization stopping times, estimated by scaling each run's total runtime by the fraction of the evaluation budget used at stopping. UCB-E and Gittins use $B = 8$ on GSM8K/PIQA/AlpacaEval and $B = 2, 4, 8$ on MMLU-small/medium/large, respectively; UCB-E-LRF uses $B = 32$ throughout. Bayesian optimization baselines correspond to PBGI run with unit costs and LogEI. The broken vertical axis is used because UCB-E-LRF is much slower on larger MMLU tasks. Overall, Gittins remains close to UCB-E except on the larger MMLU settings, while UCB-E-LRF incurs substantially larger overhead. The Bayesian optimization baselines remain computationally light in these total-runtime measurements, partly because their configuration-level protocol requires fewer sequential allocation decisions than the batch-level bandit methods.

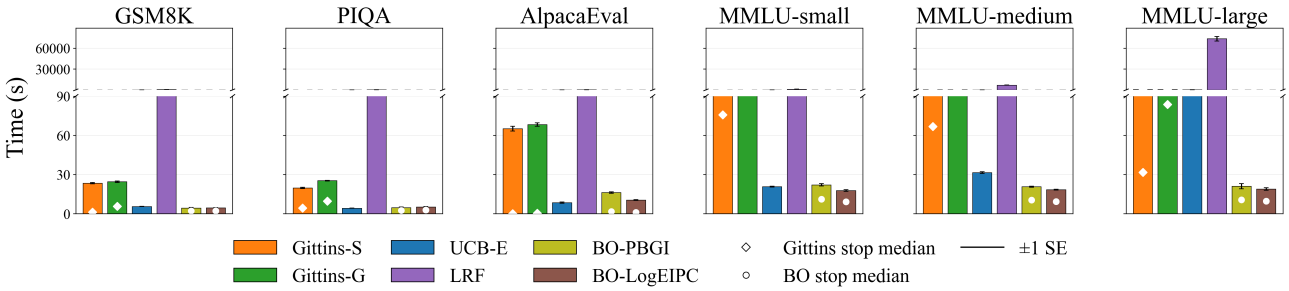



*Figure 15.* Total wall-clock runtime under the cost-aware setting. Each panel reports the total runtime for a completed simulated allocation run on GSM8K, PIQA, AlpacaEval, MMLU-small, MMLU-medium, and MMLU-large. Bars show medians across completed runs, with black error bars indicating $\pm 1$ standard error. Hollow diamonds mark median Gittins stopping times, and hollow circles mark median Bayesian optimization stopping times, estimated by scaling each run's total runtime by the fraction of the cost budget used at stopping. UCB-E and Gittins use $B = 8$ on GSM8K/PIQA/AlpacaEval and $B = 2, 4, 8$ on MMLU-small/medium/large, respectively; UCB-E-LRF uses $B = 32$ throughout. Bayesian optimization baselines correspond to PBGI and LogEIPC. The broken vertical axis is used because UCB-E-LRF is much slower on larger MMLU tasks. Overall, Gittins remains close to UCB-E except on the larger MMLU settings, while UCB-E-LRF incurs substantially larger overhead. The Bayesian optimization baselines remain computationally light in these total-runtime measurements, partly because their configuration-level protocol requires fewer sequential allocation decisions than the batch-level bandit methods.