SPECULATIVE EVALUATION OF STOCHASTIC LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating a stochastic large language model is costly: benchmark scores estimate expected performance from randomized rollouts, yet uniform repetition ignores sharp differences in task-level rollout variance. We ask how to minimize the variance of a fixed-benchmark mean under an exact rollout budget. We develop Speculative Evaluation with a Hierarchical Bayesian Neyman (HBN) policy with pilot size and stage weight jointly chosen ex ante. It runs a short uniform pilot, pools per-task success counts with a hierarchical Bayesian model, and uses posterior expectations of task-level sampling variances for exact positive-integer Neyman allocation. To mitigate the pilot synchronization barrier, HBN-async speculatively executes continuations from partial pilot feedback and retains those selected by the final allocation. Across six checkpoints and 18 benchmark groups, we evaluate 107 nondegenerate benchmark–checkpoint profiles. For rollout budgets of 8–64 per task, Speculative Evaluation reduces variance relative to Uniform by 12.8%–33.6% on average across profiles, outperforming hindsight-tuned empirical and independent Bayesian baselines. Real-generation experiments that account for the pilot synchronization barrier show that HBN-async mitigates its overhead, helping translate statistical efficiency into practical evaluation benefits.

1 INTRODUCTION

Benchmarking a stochastic large language model (LLM) is a statistical estimation problem. Rollouts on the same task can produce different answers, so a benchmark score estimates expected performance rather than a deterministic quantity. Quantifying this uncertainty requires repeated generation, which is often omitted because of its cost (Blackwell et al., 2024). Even with substantial repetition, uncertainty can remain appreciable. For the AIME24–Qwen3.5-4B profile studied in Section 4, uniform evaluation with $b = 32$ rollouts per task—a common choice for AIME—still yields a normal-reference 95% confidence interval with a half-width of 1.85 benchmark percentage points. Under uniform evaluation with b rollouts per task, the standard error decreases only as $b^{-1/2}$; halving an error bar therefore requires roughly four times as many rollouts. Reliable stochastic evaluation can therefore require large rollout budgets.

Automated research and self-improvement make both evaluation cost and precision especially important. Systems such as AFlow (Zhang et al., 2025), SICA (Robeyns et al., 2025), and the Darwin Gödel Machine (Zhang et al., 2026) repeatedly propose and evaluate changes to LLM workflows or agents, making evaluation a recurring cost rather than a one-time measurement. Their candidate changes may yield only small improvements, so evaluation must also provide sufficiently precise feedback to distinguish genuine progress from rollout noise. Otherwise, search may retain ineffective changes or discard useful ones. Increasing repetition improves precision but raises the cost of every iteration, motivating more informative evaluation under a fixed rollout budget.

This motivates a more basic question that applies beyond auto-research: *can a fixed rollout budget be allocated more efficiently?* Uniform evaluation gives every task the same number of samples, although task-level rollout variances can differ sharply. For a binary outcome with pass probability p_i, the variance is $p_i(1-p_i)$: tasks near $p_i = 1/2$ are intrinsically noisy, while tasks near zero or one require fewer samples to estimate precisely. If known, these variances would let classical Neyman allocation (Neyman, 1992) concentrate rollouts on noisier tasks and minimize benchmark-mean variance. In practice they are unknown, and learning them consumes the same budget that the allocation is meant to save. A useful policy must therefore balance the budget spent learning task variances against the gains from adapting the allocation.

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

Following feasible two-stage Neyman designs (Sukhatme & Tang, 1975; Cai & Rafi, 2024), we develop *Speculative Evaluation* for stochastic LLM benchmarks, using a *Hierarchical Bayesian Neyman (HBN)* policy. A uniform pilot observes every task, and a hierarchical Bayesian model pools its noisy counts to construct posterior expected sampling-variance scores. An exact integer Neyman allocator distributes the remaining budget while retaining at least one fresh rollout per task. Pilot size and stage weight are jointly chosen ex ante; fresh continuation samples permit unbiased reuse of the pilot through a weighted estimator.

Statistical efficiency alone does not ensure runtime efficiency. Finalizing the allocation requires all pilot rewards, creating a synchronization barrier that can leave compute idle. We therefore distinguish *HBN-sync*, which waits before dispatching continuations, from *HBN-async*, which uses partial pilot feedback to speculatively pre-allocate and execute them. The final allocation determines which requests are retained. Although unselected speculation wastes computation, executing continuations early can reduce idle time at the barrier, improving throughput and overall evaluation efficiency.

Speculation thus operates at two levels. At the statistical level, speculation uses pilot evidence to predict which tasks will benefit from more samples. At the systems level, speculation uses partial pilot feedback to predict which continuations will be retained and executes them early. The first can reduce estimator variance, while the second helps translate statistical efficiency into wall-clock efficiency without changing the estimator.

Our contributions are threefold:

- We formulate stochastic LLM evaluation as variance-minimizing integer allocation. We develop an unbiased two-stage HBN policy with ex-ante pilot and weight design, hierarchical information sharing, and exact integer allocation.
- We introduce HBN-async, which uses partial pilot feedback to speculatively execute continuations, mitigating the synchronization barrier so that statistical gains can translate into practical runtime benefits.
- For rollout budgets of 8–64 per task across 107 benchmark–checkpoint profiles, HBN reduces variance by 12.8%–33.6% on average relative to Uniform, outperforming tuned empirical and independent Bayesian baselines on this metric. Real-generation experiments show that HBN-async mitigates barrier overhead, helping turn statistical efficiency into practical gains.

2 Background

Problem formulation. Fix a checkpoint and a benchmark of N tasks. Rollout t on task i produces $Y_{i,t} \sim \text{Bernoulli}(p_i)$, for $i = 1, \dots, N$, independently conditional on the pass-probability vector $\mathbf{p} = (p_1, \dots, p_N)$. Our target is the benchmark mean $\mu(\mathbf{p}) = N^{-1} \sum_{i=1}^{N} p_i$. We call $\mathbf{p}$ the task-probability profile of this benchmark–checkpoint pair.

Let $b \in \mathbb{Z}_{\geq 1}$ be the average rollout budget per task. An allocation $\mathbf{n} \in \mathbb{Z}_{\geq 1}^{N}$ satisfies $\sum_{i=1}^{N} n_i = Nb$. Under this allocation, define $\widehat{p}_i = n_i^{-1} \sum_{t=1}^{n_i} Y_{i,t}$ and $\widehat{\mu}(\mathbf{n}) = N^{-1} \sum_{i=1}^{N} \widehat{p}_i$. The estimator is conditionally unbiased, with

$$\text{Var}(\widehat{\mu} \mid \mathbf{p}, \mathbf{n}) = \frac{1}{N^2} \sum_{i=1}^{N} \frac{\sigma_i^2}{n_i}, \qquad \sigma_i^2 = p_i(1 - p_i). \tag{1}$$

Hence the oracle allocation solves

$$\min_{\mathbf{n} \in \mathbb{Z}_{\geq 1}^{N}} \sum_{i=1}^{N} \frac{\sigma_i^2}{n_i} \quad \text{subject to} \quad \sum_{i=1}^{N} n_i = Nb. \tag{2}$$

Oracle Neyman allocation. Uniform allocation sets $n_i = b$ and has variance $V_{\text{unif}}(\mathbf{p}) = (N^2 b)^{-1} \sum_i \sigma_i^2$. For an analytic benchmark, relax integrality and the per-task lower bound in (2). If an oracle knows $\mathbf{p}$, the classical Neyman allocation (Neyman, 1992) is $n_i^* = Nb\sigma_i / \sum_j \sigma_j$. For profiles with positive total variance, its variance relative to Uniform is

$$R_N := \frac{V_{\text{oracle}}}{V_{\text{unif}}} = \frac{(\sum_i \sigma_i)^2}{N \sum_i \sigma_i^2} \leq 1. \tag{3}$$

The inequality follows from Cauchy–Schwarz. Thus task heterogeneity creates an opportunity to reduce variance, but the required σ_i's are unknown. A practical policy must first learn them from pilot observations and then allocate the remaining budget.

Two-stage evaluation. Following classical feasible Neyman designs based on preliminary samples (Sukhatme & Tang, 1975; Cai & Rafi, 2024), our two-stage policy requires $b \geq 2$ and spends an integer pilot budget $1 \leq m \leq b-1$ on every task. Let $S_i = \sum_{t=1}^{m} Y_{i,t}^{(0)}$ be the pilot successes on task i, and write $S = (S_1, \ldots, S_N)$ for the complete pilot vector. The pilot determines a positive integer second-stage allocation $\mathbf{L}(S)$ satisfying $\sum_{i=1}^{N} L_i(S) = N(b-m)$. Thus the allocation may adapt to the pilot, while the second-stage rollouts $Y_{i,t}^{(1)}$ are fresh and conditionally independent of it.

Define the stage-wise estimators $\widehat{\mu}_0 = (Nm)^{-1} \sum_i S_i$ and $\widehat{\mu}_1 = N^{-1} \sum_i L_i(S)^{-1} \sum_{t=1}^{L_i(S)} Y_{i,t}^{(1)}$. The pilot can be recycled without bias using any deterministic stage weight $w \in [0,1]$ fixed before the target evaluation:

$$\widehat{\mu}_w = w\widehat{\mu}_0 + (1-w)\widehat{\mu}_1. \tag{4}$$

Since the pilot estimator is unbiased and the fresh continuation estimator is unbiased conditional on the pilot, any stage weight w fixed before evaluation yields an unbiased combined estimate.

Within this two-stage policy class, we seek precommitted m, w, and an allocation rule $\mathbf{L}(\cdot)$ whose realized allocation depends only on the pilot:

$$\min_{\substack{m \in \mathbb{Z},\ 1 \leq m \leq b-1 \\ 0 \leq w \leq 1 \\ \mathbf{L}(\cdot)\ \text{pilot-measurable}}} \mathrm{Var}(\widehat{\mu}_w \mid \mathbf{p}) \quad \text{subject to} \quad \mathbf{L}(S) \in \mathbb{Z}_{\geq 1}^{N}, \quad \sum_{i=1}^{N} L_i(S) = N(b-m). \tag{P}$$

This design introduces one synchronization point before the final continuation allocation. More stages allow finer adaptation but add coordination overhead; Section 3.2 addresses execution around the two-stage barrier. Appendix D provides a comprehensive discussion of related work, including how this choice relates to classical two-stage designs and recent online allocation methods.

3 Speculative Evaluation with Asynchronous Hierarchical Bayesian Neyman Allocation

Because the task-probability profile in Problem (P) is unknown, we choose the precommitted pilot size and stage weight by minimizing prior-averaged risk rather than profile-specific conditional variance. A hierarchical Bayesian model yields the Hierarchical Bayesian Neyman (HBN) policy.

3.1 Statistical speculation: Hierarchical Bayesian Neyman allocation

Hierarchical variance estimation. Let $\xi \in (0,1)$ represent the latent mean task success probability for the benchmark–checkpoint pair, and let $\delta \in (0,1)$ represent normalized between-task dispersion. We place independent hyperpriors π_ξ and π_δ on them, set $\kappa = (1-\delta)/\delta$, and use

$$p_i \mid \xi, \delta \overset{\text{iid}}{\sim} \mathrm{Beta}(\xi\kappa, (1-\xi)\kappa)\,. \tag{5}$$

Together, the hyperpriors and conditional task model induce a joint hierarchical prior π on $(\xi, \delta, \mathbf{p})$. This parameterization gives $\mathbb{E}[p_i \mid \xi, \delta] = \xi$, $\mathrm{Var}(p_i \mid \xi, \delta) = \xi(1-\xi)\delta$, and expected sampling variance $\mathbb{E}[p_i(1-p_i) \mid \xi, \delta] = \xi(1-\xi)(1-\delta)$.

The derivations allow any prespecified hyperprior, which affects allocation efficiency but not unbiasedness. All reported HBN analyses and experiments use the untuned reference specification $\xi, \delta \overset{\text{ind}}{\sim} \mathrm{Beta}(1,1)$. Only the pilot updates the hyperparameter posterior.

For the pilot counts $S = (S_1, \ldots, S_N)$ from Section 2, let $\mathrm{B}(a,b)$ denote the beta function. Beta–binomial conjugacy yields the hyperposterior

$$\pi(\xi, \delta \mid S) \propto \pi_\xi(\xi)\pi_\delta(\delta) \prod_{i=1}^{N} \frac{\mathrm{B}(\xi\kappa + S_i, (1-\xi)\kappa + m - S_i)}{\mathrm{B}(\xi\kappa, (1-\xi)\kappa)}. \tag{6}$$

Neyman allocation depends on the conditional sampling variance $p_i(1-p_i)$ of a fresh rollout. Since p_i is unknown, we use its posterior expectation:

$$\widetilde{v}_i(S) := \mathbb{E}[p_i(1-p_i) \mid S] = \mathbb{E}_{\xi,\delta|S}\left[\frac{(\xi\kappa + S_i)((1-\xi)\kappa + m - S_i)}{(\kappa+m)(\kappa+m+1)}\right]. \tag{7}$$

The full vector S updates the shared hyperparameters, while S_i provides task-specific evidence. Consequently, $\widetilde{v}_i(S)$ remains positive even when a task has only pilot successes or only pilot failures.

Posterior Neyman allocation. After observing the complete pilot, minimizing posterior expected variance gives $L_i \propto \sqrt{\widetilde{v}_i(S)}$ when integrality and the per-task lower bound are relaxed. For actual evaluation, we solve

$$\mathbf{L}^{\mathrm{B}}(S) \in \arg\min_{\substack{L_i \in \mathbb{Z}_{\geq 1} \\ \sum_i L_i = N(b-m)}} \sum_{i=1}^{N} \frac{\widetilde{v}_i(S)}{L_i}. \tag{8}$$

Starting from $L_i = 1$, assign each remaining rollout to the task with the largest marginal reduction $\Delta_i(L_i) = \widetilde{v}_i(S)/[L_i(L_i+1)]$. These gains decrease with L_i, so the greedy procedure is the exact integer optimum. It preserves the budget and guarantees a fresh rollout for every task.

Joint pilot-and-weight selection. For fixed $\mathbf{p}$, the pilot and continuation estimators are uncorrelated: the conditional mean of $\widehat{\mu}_1$ given the pilot is the nonrandom quantity $\mu(\mathbf{p})$. Therefore any fixed (m, w) and continuation rule $\mathbf{L}$ have exact conditional variance

$$\begin{aligned} \mathrm{Var}(\widehat{\mu}_w \mid \mathbf{p}) &= w^2 A_m(\mathbf{p}) + (1-w)^2 B_m(\mathbf{p}; \mathbf{L}), \\ A_m(\mathbf{p}) = \frac{1}{N^2 m}\sum_i p_i(1-p_i), &\qquad B_m(\mathbf{p};\mathbf{L}) = \frac{1}{N^2}\mathbb{E}_{S|\mathbf{p}}\left[\sum_i \frac{p_i(1-p_i)}{L_i(S)}\right]. \end{aligned} \tag{9}$$

HBN selects (m, w) ex ante using only (b, N) and the reference prior π. Let $\overline{v}_\pi = \mathbb{E}_\pi[p_i(1-p_i)]$. For each candidate pilot, define the prior-averaged variance masses without the common factor N^{-2}:

$$A_{\pi,m} = \frac{N\overline{v}_\pi}{m}, \qquad B_{\pi,m} = \mathbb{E}_{S\sim\pi}\left[\sum_{i=1}^{N} \frac{\widetilde{v}_i(S)}{L_i^{\mathrm{B}}(S)}\right]. \tag{10}$$

The expectation covers the hyperparameters, task probabilities, and pilot outcomes. Relative to the prior-averaged Uniform risk, the Bayes risk is

$$\mathcal{R}_{\pi,N}(b,m,w) = \frac{b}{N\overline{v}_\pi}\left\{w^2 A_{\pi,m} + (1-w)^2 B_{\pi,m}\right\}. \tag{11}$$

For each integer m, the Bayes-optimal weight and minimized risk are available in closed form:

$$w^*_{\pi,m} = \frac{B_{\pi,m}}{A_{\pi,m} + B_{\pi,m}}, \qquad \mathcal{R}^*_{\pi,N}(b,m) = \frac{b}{N\overline{v}_\pi}\frac{A_{\pi,m}B_{\pi,m}}{A_{\pi,m}+B_{\pi,m}}. \tag{12}$$

We select m^*_{B} by minimizing $\mathcal{R}^*_{\pi,N}(b,m)$ over integers $m = 1, \ldots, b-1$, then set $w^*_{\mathrm{B}} = w^*_{\pi,m^*_{\mathrm{B}}}$. Both are fixed offline under the reference prior without target outcomes. Algorithm 1 in Appendix A summarizes the statistical policy.

3.2 Systems speculation: Asynchronous pre-allocation

HBN introduces one statistical synchronization point: the continuation allocation is finalized only after every pilot reward is available. We distinguish two execution schemes for the same statistical procedure. ***HBN-sync*** executes all pilot requests, waits for their rewards, computes the final allocation, and only then dispatches continuations. This execution barrier can leave workers idle while waiting for the long-running pilot tail. ***HBN-async*** mitigates this barrier by using partial pilot feedback to speculatively execute continuations early, without changing the statistical design or final allocation rule.

At a pre-barrier time t, task i has $n_i(t) \leq m$ scored pilot rollouts and $S_i(t)$ successes, with at least one $n_i(t) < m$. Write $S(t)$ and $\mathbf{n}(t)$ for the corresponding vectors. Replacing the common count m

in Eq. (6) by the task-specific counts $n_i(t)$ gives a provisional posterior for pre-allocation. Applying Eq. (7) under this distribution gives the partial allocation scores:

$$\widetilde{v}_i^{(t)} = \mathbb{E}_{\xi,\delta|S(t),\mathbf{n}(t)}\left[\frac{(\xi\kappa + S_i(t))((1-\xi)\kappa + n_i(t) - S_i(t))}{(\kappa + n_i(t))(\kappa + n_i(t) + 1)}\right]. \tag{13}$$

Substituting $\widetilde{v}_i^{(t)}$ into Eq. (8) gives a provisional allocation $\mathbf{L}^{\mathrm{B},(t)}$, updated as pilot rewards arrive. This scheduling signal can be cost-sensitive if generation latency depends on the outcome. Once all designated pilot rewards are available, it recovers the same final allocation rule as HBN-sync, without selection by completion time.

The partial quantities above provide an online signal for managing HBN-async's request queue. We apply the following rules in priority order:

1. **Pilot first.** Every pilot request is submitted before any continuation request; pending pilot admission always takes precedence.
2. **Allocation-gated continuations.** For the ℓ-th continuation of task i, let $r_{i,\ell}^{(t)} = \ell / L_i^{\mathrm{B},(t)}$. Only requests with $r_{i,\ell}^{(t)} \le 1$ are queued, in increasing $r_{i,\ell}^{(t)}$ order with deterministic tie breaking. Partial updates refresh the pending queue without cancelling work already in flight.
3. **Fixed-ID recovery.** Pilot and continuation results are recovered by request IDs fixed before execution. Even if another request for the same task finishes earlier, it cannot replace a designated ID, because such substitution can introduce latency-dependent selection bias.

Once all pilot rewards are ready, the partial construction reduces to the official $\mathbf{L}^{\mathrm{B}}(S)$. For each task, the router reuses the first $L_i^{\mathrm{B}}(S)$ continuation requests, submits any missing requests in that prefix, and discards completed surplus work, cancels queued surplus work, or aborts running surplus work. Under the conditional-independence assumption in Section 2, unbiasedness therefore remains unchanged. Algorithm 2 in Appendix A summarizes the asynchronous execution scheme.

4 Experiments

Our experiments separate statistical efficiency from execution cost. Section 4.2 evaluates estimator variance conditional on fixed empirical task-probability profiles. Section 4.3 measures generation and coordination costs using real inference, with coupled rewards that give HBN-sync and HBN-async identical final allocations. We then combine the replay variance ratios with measured runtimes in a variance-equivalent cost comparison.

4.1 Benchmarks and rollout data

Models and benchmarks. We form the complete grid of six checkpoints and 18 benchmark groups, giving 108 benchmark–checkpoint pairs. The checkpoints are Qwen2.5-3B, 7B, and 32B (Qwen Team, 2025), and Qwen3.5-4B, 9B, and 27B (Qwen Team, 2026). The benchmarks are AIME 2024–2026 (Dekoninck et al., 2026), HMMT (Dekoninck et al., 2026), and MATH-100 (Hendrycks et al., 2021); the biology, chemistry, and physics partitions of GPQA Diamond (Rein et al., 2023), MMLU Science (Hendrycks et al., 2020), and SuperGPQA Science (Team et al., 2025); a 100-question text-only HLE subset (Phan et al., 2025); and the easy, medium, and hard partitions of LiveCodeBench (Jain et al., 2025) restricted to problems released after March 30, 2024. Together the groups contain 1,418 problems; their sizes are $N \in \{19, 30, 86, 93, 100\}$.

Rollout corpus and task-probability profiles. For every problem–checkpoint combination, we independently sample $K = 1024$ responses, yielding 8,712,192 responses. Let $Y_{ik} \in \{0, 1\}$ indicate whether rollout k on task i is correct under the benchmark-specific scoring rule. Appendix B.1 documents the rollout and scoring protocol. For each benchmark–checkpoint pair, we estimate $\widehat{p}_i^{(K)} = K^{-1}\sum_{k=1}^{K} Y_{ik}$. Each of the 108 resulting vectors $\widehat{\mathbf{p}}^{(K)}$ estimates the task-probability profile of one benchmark–checkpoint pair. The estimated profile for GPQA Diamond Biology with Qwen2.5-3B is degenerate: every $\widehat{p}_i^{(K)}$ is zero or one, so evaluation under this fixed empirical profile has zero variance and the Uniform-normalized ratio is undefined. We exclude this pair and retain 107 estimated task-probability profiles.

4.2 VARIANCE REDUCTION AND ROBUSTNESS

We evaluate the statistical benefit of HBN through two questions:

- *RQ1: Does HBN reduce evaluation variance relative to Uniform across budgets?*
- *RQ2: Do Bayesian regularization and hierarchical information sharing improve allocation?*

Experimental design. We fix the 107 retained task-probability profiles and evaluate budgets $b \in \{8, 16, 32, 64\}$. For each profile and candidate, we average 8192 independent pilot draws and compute continuation variance analytically using (9). The metric is Uniform-normalized variance, averaged equally across profiles at each budget; Uniform corresponds to 100%. Oracle knows the profile and solves the exact positive-integer Neyman problem, providing an infeasible full-information benchmark rather than a deployable competitor. Appendix B.4 gives the replay and random-seed details.

A1: Average variance gains increase with budget. Table 1 shows that HBN improves on Uniform at every tested budget, with its mean normalized variance decreasing from 87.2% at $b = 8$ to 66.4% at $b = 64$. Larger budgets therefore yield greater average statistical benefit. Oracle remains substantially lower, at 40.7%–39.3%, indicating that full knowledge of task probabilities would permit further gains; HBN must instead infer allocation scores from a limited pilot.

HBN reduces variance relative to Uniform on nearly all profiles, but not every profile. Appendix C.1 reports the distribution of HBN's variance gains and the single adverse profile, where HBN's variance exceeds Uniform's by at most 2.4%.

A2: Bayesian regularization and hierarchical sharing improve the observed allocations. To examine Bayesian regularization and hierarchical information sharing, we compare HBN with two ablation variants, EN and IBN. All three share the same two-stage estimator and exact integer continuation allocator. Each variant receives its own pilot-and-weight design, since changing the allocation scores also changes the balance between learning from the pilot and allocating continuation samples.

- **Tuned Empirical Neyman (EN)** removes Bayesian regularization and hierarchical information sharing, using the empirical pilot score $S_i(m - S_i)/[m(m + 1)]$. It selects (m, w) post hoc using the profiles with the same task count at each budget.
- **Tuned Independent Bayesian Neyman (IBN)** retains Bayesian regularization but removes hierarchical information sharing, using independent $\mathrm{Beta}(\alpha, \alpha)$ priors. It designs (m, w) under each matched prior and selects only α post hoc across profiles at each budget.

Both controls select their best-performing designs retrospectively on the evaluation profiles, yielding hindsight envelopes over the searched candidates. This deliberately avoids tying the comparison to an arbitrary default and gives the ablation variants an information advantage over HBN. HBN fixes its reference hyperpriors $\xi, \delta \sim \mathrm{Beta}(1, 1)$ and selects (m, w) ex ante using only (N, b), without target evaluation outcomes. Only its posterior and continuation allocation adapt to the pilot. Appendices B.2 and B.3 provide the HBN design and baseline implementation details.

	Oracle	Tuned EN	Tuned IBN	HBN
$b = 8$	40.7%	94.3%	89.6%	**87.2%**
$b = 16$	39.9%	92.6%	81.9%	**80.1%**
$b = 32$	39.5%	91.9%	74.2%	**73.0%**
$b = 64$	39.3%	91.5%	67.0%	**66.4%**

Table 1: Mean Uniform-normalized variance across 107 benchmark–checkpoint profiles (Uniform = 100%); lower is better. EN and IBN use hindsight tuning; HBN uses ex-ante design.

Table 1 shows successively lower mean variance from Tuned EN to Tuned IBN to HBN at every budget. HBN outperforms both hindsight envelopes despite committing to its design ex ante.

Even with hindsight tuning, EN reduces variance by only 5.7%–8.5%. Its score is zero for all-failure or all-success pilots, although the true task variance may be positive. When other tasks have positive

scores, such tasks typically receive only the mandatory one fresh rollout. Longer pilots reduce false zeros but shrink the continuation budget; tuning (m, w) cannot eliminate this tradeoff.

IBN's positive α prevents zero scores, alleviating this failure and yielding normalized variances of 89.6%–67.0%. However, its independent priors do not share pilot information across tasks: tuning one α per budget adjusts overall shrinkage, not each profile's latent mean and dispersion. HBN infers these shared quantities from the complete pilot, allowing tasks to borrow strength from one another. This mechanism is consistent with its advantage, although the comparison evaluates complete designs rather than isolating information sharing alone.

Beyond these ablations, we adapt TS-Neyman (Morikawa et al., 2026) as an external baseline for our Bernoulli setting. HBN achieves lower mean Uniform-normalized MSE at all four budgets, even with hindsight tuning of this baseline (Appendix C.3).

4.3 Computation, system overhead, and practical benefit

The variance gains in Section 4.2 must be weighed against the cost of generating a non-uniform workload and waiting for pilot feedback. The systems question is whether a concrete asynchronous implementation can reduce the pilot barrier sufficiently for the statistical gains to remain beneficial after accounting for execution costs. Using the HBN-sync and HBN-async schemes defined in Section 3.2, we study three questions:

- *RQ3: What additional computation does HBN introduce?*
- *RQ4: Does HBN-async mitigate the impact of the pilot synchronization point through speculative pre-allocation?*
- *RQ5: When does speculative evaluation offer a clear practical benefit?*

Experimental design. We use the same 107 task-probability profiles and budgets $b \in \{8, 16, 32, 64\}$, giving 428 profile–budget settings. Generation runs on eight H20 GPUs per worker, with a shared concurrency limit of $C = 256$ outstanding requests (32 per replica) for all three strategies. We use the frozen prompts and model settings of the rollout corpus. To compare synchronous and asynchronous execution under the same statistical allocation, we couple their pilot rewards. Using generated-answer rewards would generally produce different pilot outcomes and hence different continuation allocations across runs, mixing execution-policy effects with allocation changes. We therefore assign each logical request ID a fixed Bernoulli reward with probability $\widehat{p}_i^{(K)}$, shared across execution strategies and revealed only when its real generation completes. HBN-sync and HBN-async consequently receive identical pilot rewards and select the same final continuation IDs. This controls the allocation decision while retaining real generation and execution costs; realized response lengths may still differ across runs. Benchmark judging and code-execution latency are excluded.

Uniform makes all Nb uniformly allocated requests eligible from the start. We compare it with HBN-sync and HBN-async under this common concurrency limit. Each strategy has one successful measured execution per profile–budget setting, for 1,284 policy executions. We summarize execution costs across the 107 profiles at each budget, targeting aggregate behavior over the evaluated workload suite. These measurements do not quantify repeated-run variability within an individual setting. We additionally measure 428 Uniform executions at rounded-down variance-equivalent budgets for the benefit–cost comparison below. Appendix B.5 gives the execution and accounting details.

A3: Useful computation and wasted speculative computation. HBN's computation relative to Uniform can be decomposed into two parts. We distinguish estimated *useful FLOPs*, F^{useful}, for accepted rollouts from *wasted FLOPs*, F^{waste}, for discarded or running-aborted requests. Their sum is the estimated actual computation, denoted by F. This decomposition applies to both HBN-sync and HBN-async. The first is the change in useful computation associated with task allocation: HBN changes the distribution of accepted rollouts across tasks, whose generation costs differ, while preserving the total count Nb. The second, present in HBN-async, is wasted speculative computation from pre-allocated rollouts that are not selected by the final allocation, including completed-discarded and running-aborted requests. Since Uniform's actual computation is entirely useful, the accounting identity is

$$F_{\text{HBN}} - F_{\text{Uniform}} = \left(F_{\text{HBN}}^{\text{useful}} - F_{\text{Uniform}}\right) + F_{\text{HBN}}^{\text{waste}}. \tag{14}$$

Here HBN denotes either execution scheme; for HBN-sync, $F_{\text{HBN}}^{\text{waste}} = 0$. Table 2 reports this decomposition for HBN-async. Useful computation increases by 2.00%–5.62% in aggregate over Uniform, whereas wasted speculative computation falls from 2.23% of Uniform's computation at $b = 8$ to 0.026% at $b = 64$. Thus, at larger budgets, the additional computation is predominantly in accepted generations, even though the number of accepted rollouts is identical.

b	Extra rollouts vs. Uniform	Useful FLOPs vs. Uniform	Wasted FLOPs vs. Uniform	Actual FLOPs vs. Uniform
8	+2.89%	+2.00%	+2.23%	+4.23%
16	+0.98%	+3.03%	+0.51%	+3.54%
32	+0.25%	+4.41%	+0.073%	+4.49%
64	+0.077%	+5.62%	+0.026%	+5.64%

Table 2: HBN-async computation across 107 profiles at each budget. Extra rollouts report the relative increase in actual rollout count over Uniform. FLOPs are estimated using the method in Appendix B.5. The Useful, Wasted, and Actual FLOP columns correspond to the first right-hand term, second right-hand term, and left-hand side of Eq. (14), respectively, each summed across profiles at each budget and divided by summed Uniform FLOPs.

A4: Pre-allocation reduces the observed execution overhead. Despite its wasted FLOPs, HBN-async reduces both actual time and useful-FLOP-normalized time relative to HBN-sync at every tested budget.

	Actual time vs. Uniform			Useful-FLOP-normalized time vs. Uniform		
b	HBN-sync	HBN-async	Δ	HBN-sync	HBN-async	Δ
8	+33.73%	+11.15%	22.57%	+30.48%	+8.45%	22.03%
16	+16.47%	+2.85%	13.61%	+13.39%	+0.14%	13.25%
32	+13.38%	+5.49%	7.89%	+9.27%	+1.33%	7.94%
64	+11.10%	+6.56%	4.54%	+5.61%	+1.38%	4.23%

Table 3: System time across 107 profiles at each budget. Strategy columns report percentage increases over Uniform, computed from ratios of total times across all 107 profiles at the same budget, with useful-FLOP normalization applied per profile where indicated. Δ denotes the HBN-sync–HBN-async difference in percentage points.

To better assess the execution overhead associated with the pilot barrier, we approximately adjust runtime for differences in useful computation. Otherwise, a time difference can reflect a different amount of accepted generation rather than a difference in execution efficiency. For strategy p and profile j at a fixed budget, *useful-FLOP-normalized time* is $\widetilde{T}_{p,j} = T_{p,j} F_{\text{Uniform},j}^{\text{useful}} / F_{p,j}^{\text{useful}}$: the measured time rescaled to Uniform's useful computation in that setting. We normalize each profile–budget setting, then average equally over the 107 profiles at each budget. Wasted computation remains included in runtime but does not count toward useful FLOPs in this adjustment.

Table 3 shows that HBN-async reduces both actual and useful-FLOP-normalized overhead relative to HBN-sync at every budget, despite speculative waste. Similar Δ values before and after normalization indicate that the reduction persists after accounting for useful computation. The gap is largest at $b = 8$ and narrows with budget.

For $b \geq 16$, HBN-async's mean useful-FLOP-normalized time is less than 1.4% above Uniform's. Thus, under this hardware and concurrency configuration, speculative execution reduces the remaining normalized system overhead to a marginal level at the larger budgets. At $b = 8$, the residual overhead is still 8.45%. Intuitively, at small budgets, waiting for the final pilot rewards can account for a substantial fraction of runtime, with relatively little continuation computation available to hide that wait. Larger budgets provide more continuation computation to overlap with the pilot and spread the remaining synchronization cost over a longer run, leaving a small relative overhead.

A5: Sufficient statistical gain and a total rollout budget well above the concurrency limit. Let $r_{j,b}$ be HBN's Uniform-normalized variance from the replay in Table 1. By Uniform's $1/b$ variance scaling, its *continuous-equivalent budget*, $b/r_{j,b}$, matches HBN's variance under these replay ratios. Its *rounded-equivalent budget*, $b_{j,b}^{\downarrow} = \lfloor b/r_{j,b} \rfloor$, rounds this value down to an integer for execution,

leaving Uniform with variance no lower than HBN's. We freshly measure Uniform at this budget with unchanged generation and concurrency settings.

b	Equivalent rollout budget			Actual time vs. Uniform		
	Avg. continuous	Avg. rounded	Rounded / continuous	Rounded-equivalent Uniform	HBN-async	Δ
8	9.37	8.87	94.63%	+3.79%	+11.15%	-7.36%
16	20.88	20.39	97.66%	+13.64%	+2.85%	$+10.79\%$
32	47.04	46.56	98.98%	+27.95%	+5.49%	$+22.46\%$
64	107.22	106.75	99.56%	+42.81%	+6.56%	$+36.25\%$

Table 4: Measured benefit–cost comparison across 107 profiles. Average budgets are task-count-weighted. The budget ratio reports the fraction of continuous-equivalent rollouts actually executed after rounding down: $\sum_j N_j b^{\downarrow}_{j,b} / \sum_j N_j (b/r_{j,b})$, as a percentage. Time increases use summed original-budget Uniform time as the denominator; HBN-async values are from Table 3. Δ is rounded-equivalent Uniform minus HBN-async, in percentage points.

At $b \geq 16$, rounded-equivalent Uniform takes 13.64%–42.81% more total time than original-budget Uniform, versus 2.85%–6.56% for HBN-async (Table 4). HBN-async thus retains a clear aggregate time advantage despite rounding in Uniform's favor. Rounding removes 5.37% of the continuous rollout budget at $b = 8$, falling to 0.44% at $b = 64$.

At $b = 8$, HBN-async is slower than rounded-equivalent Uniform. Two factors limit its benefit. First, the statistical gain is smaller, and rounding further reduces the comparator's additional budget: it retains only 63.36% of the extra rollouts prescribed by continuous variance matching. Second, the smaller total budget Nb provides less continuation computation relative to the concurrency limit $C = 256$, leaving less opportunity to overlap generation with the pilot barrier. HBN-async consequently retains 8.45% useful-FLOP-normalized overhead (Table 3). These effects leave no clear overall benefit in this comparison.

The widening margin in Table 4 reflects a combination of statistical and systems effects. Statistically, the opportunity for nonuniform allocation depends on the task-probability profile, while larger budgets allow HBN to exploit that opportunity more effectively, yielding greater average variance reductions in our experiments. On the systems side, increasing b raises the total rollout budget Nb relative to the fixed concurrency limit C, providing more continuation computation to overlap with the pilot and reducing the relative impact of residual barrier delays. Speculative evaluation is therefore most beneficial when **profiles offer substantial allocative opportunity** and **the budget is sufficient both to exploit it statistically and to amortize synchronization overhead**. Under our experimental setup, this comparison indicates a clear benefit at $b \geq 16$.

5 Conclusion and Limitations

Speculative Evaluation combines hierarchical Bayesian allocation and asynchronous pre-allocation to improve stochastic LLM evaluation. HBN reduces variance through hierarchical information sharing and ex-ante pilot-and-weight design; HBN-async mitigates the pilot barrier despite speculative waste. Across 107 benchmark–checkpoint profiles, their combination yields clear aggregate benefits at $b \geq 16$ under our setup. Several limitations remain:

- **Statistical scope.** We assume conditionally independent Bernoulli outcomes. Profile structure and prior misspecification affect efficiency, and gains are not guaranteed for every profile.
- **Systems scope.** Coupled synthetic rewards control the final allocation across HBN execution schemes. They do not reproduce within-task dependence between correctness and generation latency, and the timing excludes benchmark scoring. Practical benefits depend on generation and scoring costs as well as concurrency.
- **Timing precision.** Given the cost of real generation, each policy is measured once per profile–budget setting. The reported time benefits are aggregate observations under the tested hardware and concurrency; repeated-run variability within individual settings remains unquantified.

Future work could extend the framework to nonbinary or correlated outcomes and develop cost-aware allocation with heterogeneous generation costs and real scoring feedback.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in the design of the hierarchical Bayesian method, designing and providing feedback on research methodology and experiments, implementing methods, cleaning and reformatting datasets, and interpreting results. We have not used generative AI tools for generating synthetic datasets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, or assisting with translation, and qualitative and thematic data analysis is not applicable to this work. We have reviewed all AI-assisted work. We checked the theoretical arguments, implementation, references, and final manuscript, and verified the experimental results through tests, input and output consistency checks, and table reproduction. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Robert E Blackwell, Jon Barry, and Anthony G Cohn. Towards reproducible llm evaluation: Quantifying uncertainty in llm benchmark scores. *arXiv preprint arXiv:2410.03492*, 2024.

Yong Cai and Ahnaf Rafi. On the performance of the neyman allocation with small pilots. *Journal of Econometrics*, 242(1):105793, 2024.

Charlie Cowen-Breen, Alekh Agarwal, Stephen Bates, William W Cohen, Jacob Eisenstein, Amir Globerson, and Adam Fisch. Multiple-prediction-powered inference. *arXiv preprint arXiv:2603.27414*, 2026.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: MathArena as an evaluation platform for mathematics with LLMs. 2026. URL https://arxiv.org/abs/2605.00674.

Max Hamilton, Jinlin Lai, Wenlong Zhao, Subhransu Maji, and Daniel Sheldon. Active measurement: Efficient estimation at scale, 2025. URL https://arxiv.org/abs/2507.01372.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260):663–685, 1952.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94074dd5a072d28ff75a76dabed43767-Paper-Conference.pdf.

Heyang Jiang, Henry Liu, and Baharan Mirzasoleiman. Learning as reasoning unfolds: Progressive rollout allocation for efficient reinforcement learning. *arXiv preprint arXiv:2607.22002*, 2026.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth. Active testing: Sample-efficient model evaluation. In *International Conference on Machine Learning*, pp. 5753–5763. PMLR, 2021.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Thomas Rainforth. Active surrogate estimators: An active learning approach to label-efficient model evaluation. *Advances in Neural Information Processing Systems*, 35:24557–24570, 2022.

Junghyun Lee, Sanghwa Kim, Yassir Jedra, Alexandre Proutière, and Se-Young Yun. Instance-optimal estimation with multiple llm judges on a budget. *arXiv preprint arXiv:2605.23362*, 2026.

Gili Lior, Eliya Habba, Shahar Levy, Avi Caciularu, and Gabriel Stanovsky. Reliableeval: A recipe for stochastic llm evaluation via method of moments. In *EMNLP (Findings)*, pp. 11146–11153, 2025.

Zeli Liu, Jiancheng Zhang, Cong Liu, and Yinglun Zhu. Active testing of large language models via approximate neyman allocation. *arXiv preprint arXiv:2605.10075*, 2026.

Kosuke Morikawa, Mst Moushumi Pervin, and Jae Kwang Kim. Ts-neyman: Posterior sampling for adaptive stratified estimation. *arXiv preprint arXiv:2606.08418*, 2026.

Jerzy Neyman. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. In *Breakthroughs in statistics: Methodology and distribution*, pp. 123–150. Springer, 1992.

Yotam Perlitz, Elron Bandel, Ariel Gera, Ofir Arviv, Liat Ein Dor, Eyal Shnarch, Noam Slonim, Michal Shmueli-Scheuer, and Leshem Choshen. Efficient benchmarking (of language models). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2519–2536, 2024.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples. *arXiv preprint arXiv:2402.14992*, 2024.

Qwen Team. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Aadirupa Saha, Aniket Wagde, and Branislav Kveton. Llm-as-judge on a budget. *arXiv preprint arXiv:2602.15481*, 2026.

BV Sukhatme and Victor KT Tang. Allocation in stratified sampling subsequent to preliminary test of significance. *Journal of the American Statistical Association*, 70(349):175–179, 1975.

M-A-P Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, King Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, Kaixin Deng, Shawn Gavin, Shian Jia, Sichao Jiang, Yiyan Liao, Rui Li, Qinrui Li, Sirun Li, Yizhi Li, Yunwen Li, David Ma, Yuansheng Ni, Haoran Que, Qiyao Wang, Zhoufutu Wen, Siwei Wu, Tyshawn Hsing, Ming Xu, Zhenzhu Yang, Zekun Moore Wang, Junting Zhou, Yuelin Bai, Xingyuan Bu, Chenglin Cai, Liang Chen, Yifan Chen, Chengtuo Cheng, Tianhao Cheng, Keyi Ding, Siming Huang, Yun Huang, Yaoru Li, Yizhe Li, Zhaoqun Li, Tianhao Liang, Chengdong Lin, Hongquan Lin, Yinghao Ma, Tianyang Pang, Zhongyuan Peng, Zifan Peng, Qige Qi, Shi Qiu, Xingwei Qu, Shanghaoran Quan, Yizhou Tan, Zili Wang, Chenqing Wang, Hao Wang, Yiya Wang, Yubo Wang, Jiajun Xu, Kexin Yang, Ruibin Yuan, Yuanhao Yue, Tianyang Zhan, Chun Zhang, Jinyang Zhang, Xiyue Zhang, Xingjian Zhang, Yue Zhang, Yongchi Zhao, Xiangyu Zheng, Chenghua Zhong, Yang Gao, Zhoujun Li, Dayiheng Liu, Qian Liu, Tianyu Liu, Shiwen Ni, Junran Peng, Yujia Qin, Wenbo Su, Guoyin Wang, Shi Wang, Jian Yang, Min Yang, Meng Cao, Xiang Yue, Zhaoxiang Zhang, Wangchunshu Zhou, Jiaheng Liu, Qunshu Lin, Wenhao Huang, and Ge Zhang. Supergpqa: Scaling llm evaluation across 285 graduate disciplines, 2025. URL https://arxiv.org/abs/2502.14739.

Skyler Wu, Yash Nair, and Emmanuel J. Candès. Efficient evaluation of LLM performance with statistical guarantees. *arXiv preprint arXiv:2601.20251*, 2026.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025.

Bowen Zuo and Yinglun Zhu. Strategic scaling of test-time compute: A bandit learning approach. In *International Conference on Learning Representations*, volume 2026, pp. 134105–134128, 2026.

A COMPLETE ALGORITHM

Algorithm 1 Speculative evaluation with Hierarchical Bayesian Neyman (HBN) allocation

Require: Tasks $1{:}N$, integer budget $b \geq 2$ per task, reference hyperprior $\pi_\xi = \pi_\delta = \mathrm{Beta}(1,1)$, and rollout procedure ROLLOUT(i)
Ensure: Unbiased estimate $\widehat{\mu}_w$

▷ OFFLINE DESIGN
Jointly choose $(m, w) \leftarrow (m^*_{\mathrm{B}}, w^*_{\mathrm{B}})$ by (12) and minimizing over feasible pilot sizes ▷ uses no target outcomes
▷ STAGE 1: UNIFORM PILOT
In parallel, draw $Y^{(0)}_{i,1:m} \leftarrow$ ROLLOUT(i) and set $S_i \leftarrow \sum_{t=1}^{m} Y^{(0)}_{i,t}$ for all i
▷ POSTERIOR UPDATE AND EXACT INTEGER ALLOCATION
synchronize; compute $\pi(\xi, \delta \mid S)$ by (6) and $\widetilde{v}_i(S)$ by (7) ▷ only synchronization
Initialize $L_i \leftarrow 1$ for all i ▷ guarantees a fresh rollout per task
while $\sum_i L_i < N(b-m)$ **do**
  $k \leftarrow \arg\max_i \widetilde{v}_i(S)/[L_i(L_i+1)]$; $L_k \leftarrow L_k + 1$ ▷ largest marginal reduction
end while
▷ STAGE 2: ADAPTIVE CONTINUATION
In parallel, draw fresh $Y^{(1)}_{i,1:L_i} \leftarrow$ ROLLOUT(i) for all i
▷ UNBIASED AGGREGATION
Form $\widehat{\mu}_0, \widehat{\mu}_1$ from both stages and **return** $\widehat{\mu}_w$ in (4) ▷ recycles the pilot

Algorithm 2 HBN-async: speculative pre-allocation and fixed-ID recovery

Require: Tasks $1{:}N$, integer budget $b \geq 2$ per task, reference prior π, concurrency limit C, and rollout procedure ROLLOUT(i)
Ensure: Estimate $\widehat{\mu}_w$ using the final HBN allocation

▷ OFFLINE DESIGN AND PILOT SUBMISSION
Fix (m, w) as in Algorithm 1
Fix pilot IDs and continuation IDs (i, ℓ) before execution
Initialize scored pilot counts $n_i \leftarrow 0$, successes $S_i \leftarrow 0$, and result store
Submit all pilot requests before any continuation request ▷ at most C active requests
▷ PARTIAL-PILOT SPECULATION
while some pilot reward is unavailable **do**
  Receive a completion and store its result by its designated ID
  if the result is a pilot reward for task i **then**
    Update n_i and S_i
    if all pilot rewards are available **then**
      BREAK
    end if
    Compute partial scores $\widetilde{v}_i^{(t)}$ by (13)
    Compute $(L_i^{\mathrm{B},(t)})_{i=1}^N$ with the exact integer allocator
    Refresh pending continuations to unsubmitted IDs with $\ell \leq L_i^{\mathrm{B},(t)}$
    Order them by increasing $\ell / L_i^{\mathrm{B},(t)}$, with deterministic ties
    Leave running and completed continuations unchanged
  end if
  Fill available capacity with pending pilots first, then eligible continuations
end while
▷ FINAL ALLOCATION AND RECOVERY
Compute $(L_i^{\mathrm{B}}(S))_{i=1}^N$ from the complete pilot by (8)
Set required continuation IDs $\mathcal{I} \leftarrow \{(i,\ell) : 1 \leq \ell \leq L_i^{\mathrm{B}}(S)\}$
Retain completed results and running requests in $\mathcal{I}$
Discard completed surplus, cancel queued surplus, and abort running surplus
Submit missing IDs in $\mathcal{I}$, respecting the concurrency limit
Await every required ID; never substitute an earlier-finishing surplus result
▷ AGGREGATION
Form $\widehat{\mu}_0$ from pilot results and $\widehat{\mu}_1$ from IDs in $\mathcal{I}$
return $\widehat{\mu}_w = w\widehat{\mu}_0 + (1-w)\widehat{\mu}_1$

B Experimental and Implementation Details

B.1 Rollout generation and scoring

We generate every response with vLLM 0.23.0 using temperature 0.6, top-$p = 0.95$, no top-k truncation, a 16,384-token cap, and bfloat16 inference. The model context limit is 32,768 tokens and prefix caching is disabled. Qwen2.5 runs in non-thinking mode; Qwen3.5 uses thinking mode with at most 8,192 thinking tokens within the total cap. We use master seed 42 with deterministic per-request seeds.

Generation workers each contain eight NVIDIA H20 GPUs with 96 GB per GPU. Each GPU hosts one persistent tensor-parallel-size-one replica, while a routing process dynamically distributes requests among the eight replicas. The GPU-memory-utilization limit is 0.9. The engine sequence limits for Qwen2.5-3B/7B/32B and Qwen3.5-4B/9B/27B are respectively 256/192/128 and 128/128/64.

Prompts, extraction, and scoring are benchmark-specific: mathematical tasks use normalized numeric or symbolic matching, and multiple-choice tasks use option extraction. HLE uses the CAIS judge prompt with a fixed Qwen3.6-35B-A3B checkpoint in non-thinking mode at temperature zero. LiveCodeBench programs are evaluated against test cases on separate CPU workers in network-isolated, resource-limited environments.

Generation of the complete 108-pair corpus required approximately 411 active model-job hours, equivalent to 17.1 days if all model-level jobs were run sequentially on one independent eight-H20 worker. This estimate excludes scoring and is not an exact GPU-hour count because some jobs used multiple eight-GPU workers concurrently.

B.2 HBN design and numerical implementation

For each candidate pilot, HBN estimates the continuation mass $B_{\pi,m}$ in (10), obtains the optimal weight in closed form from (12), and compares the resulting Bayes risks. It evaluates the full feasible grid $m = 1, \ldots, b-1$; every selected solution is interior to that search range. It averages eight independent 8192-draw replicates under its hierarchical reference prior, using common random numbers across candidate pilots within each replicate. Its tensor-product Gauss–Legendre quadrature has order 16 in each of ξ and δ. The same quadrature grid is used for complete- and partial-pilot posterior updates online. Posterior weights are computed in the log domain using log-beta evaluations and log-sum-exp normalization. Table 5 reports the resulting ex-ante schedules.

N	$b = 8$	$b = 16$	$b = 32$	$b = 64$
19	$(4, 0.49)$	$(6, 0.35)$	$(10, 0.29)$	$(17, 0.24)$
30	$(4, 0.48)$	$(6, 0.35)$	$(10, 0.29)$	$(16, 0.22)$
86	$(4, 0.48)$	$(6, 0.35)$	$(10, 0.28)$	$(16, 0.22)$
93	$(4, 0.48)$	$(6, 0.35)$	$(10, 0.29)$	$(16, 0.22)$
100	$(4, 0.48)$	$(6, 0.35)$	$(10, 0.28)$	$(16, 0.22)$

Table 5: Ex-ante HBN design selected under the reference prior. Each cell reports $(m^*_{\mathrm{B}}, w^*_{\mathrm{B}})$ for task count N and average budget b. We display weights to two decimal places; all experiments use their full-precision values. No target-pair outcome enters the selection.

B.3 Baseline scores and tuning

For $\alpha > 0$, Independent Bayesian Neyman (IBN) uses

$$\widetilde{v}_i^{\mathrm{IBN}} = \frac{(S_i + \alpha)(m - S_i + \alpha)}{(m + 2\alpha)(m + 2\alpha + 1)}. \tag{15}$$

This is the posterior expectation of $p_i(1 - p_i)$ under the matched prior $p_i \overset{\mathrm{iid}}{\sim} \mathrm{Beta}(\alpha, \alpha)$. The $\alpha \downarrow 0$ allocation-rule limit is Empirical Neyman (EN):

$$\widetilde{v}_i^{\mathrm{EN}} = \frac{S_i(m - S_i)}{m(m + 1)}. \tag{16}$$

The factor $m/(m+1)$ relative to the plug-in Bernoulli variance is common to all tasks and therefore does not change the allocation.

Every method passes its nonnegative scores to the decreasing-marginal-gain allocator in Section 3.1. It starts at one continuation rollout per task and assigns the remaining budget exactly. If all EN scores are zero, its continuation is uniform.

EN uses the evaluation profiles to select (m, w) for each (N, b). IBN designs (m, w) under each candidate prior, then uses the profiles to select α at each budget.

EN design pooled over task-probability profiles. Index the retained task-probability profiles by j, and let $\mathcal{J}_N = \{j : N_j = N\}$. The five task counts $N \in \{19, 30, 86, 93, 100\}$ contain respectively $5, 24, 6, 6, 66$ task-probability profiles. For task-probability profile j, write $v_{ji} = \widehat{p}_{ji}^{(K)}(1 - \widehat{p}_{ji}^{(K)})$ and $V_j = \sum_i v_{ji}$. For every b and every $m = 1, \ldots, b-1$, we draw 8192 pilot vectors from the fixed task-probability profile, form the EN scores in (16), and compute the exact continuation allocation $\mathbf{L}_j^{\mathrm{EN}}(S; m, b)$. This estimates

$$C_{j,b,m}^{\mathrm{EN}} = \mathbb{E}_{S|\widehat{\mathbf{p}}_j^{(K)}}\left[\frac{1}{V_j}\sum_{i=1}^{N}\frac{v_{ji}}{L_{ji}^{\mathrm{EN}}(S; m, b)}\right], \qquad \overline{C}_{N,b,m}^{\mathrm{EN}} = \frac{1}{|\mathcal{J}_N|}\sum_{j\in\mathcal{J}_N} C_{j,b,m}^{\mathrm{EN}}. \tag{17}$$

Because the pilot contribution to each task-probability profile's Uniform-normalized variance is bw^2/m, the group-mean candidate objective is

$$\mathcal{R}_{N,b}^{\mathrm{EN}}(m, w) = b\left\{\frac{w^2}{m} + (1-w)^2\overline{C}_{N,b,m}^{\mathrm{EN}}\right\}. \tag{18}$$

No weight-grid search is needed. For each candidate m, the minimizing weight and risk are

$$w_{N,b,m}^{\mathrm{EN}} = \frac{\overline{C}_{N,b,m}^{\mathrm{EN}}}{m^{-1} + \overline{C}_{N,b,m}^{\mathrm{EN}}}, \qquad \mathcal{R}_{N,b}^{\mathrm{EN},*}(m) = b\frac{m^{-1}\overline{C}_{N,b,m}^{\mathrm{EN}}}{m^{-1} + \overline{C}_{N,b,m}^{\mathrm{EN}}}, \tag{19}$$

The minimizing integer m is then selected, breaking an exact tie toward the smaller pilot. This produces one (m, w) schedule for each (N, b).

Table 6 reports the selected schedules.

N	$b = 8$	$b = 16$	$b = 32$	$b = 64$
19	$(6, 0.718)$	$(14, 0.814)$	$(27, 0.764)$	$(53, 0.728)$
30	$(6, 0.726)$	$(14, 0.835)$	$(29, 0.869)$	$(59, 0.887)$
86	$(6, 0.685)$	$(13, 0.722)$	$(27, 0.740)$	$(54, 0.735)$
93	$(6, 0.710)$	$(13, 0.755)$	$(27, 0.771)$	$(51, 0.707)$
100	$(6, 0.702)$	$(14, 0.804)$	$(28, 0.797)$	$(55, 0.779)$

Table 6: Tuned EN's post-hoc selected (m, w) for each task count and budget. Weights are displayed to three decimal places; replay uses full-precision values.

IBN inner matched-prior design. The positive prior-strength grid is

$$\mathcal{A} = \{0.01, 0.02, \ldots, 0.20, 0.25, 0.50, 1.00\}.$$

For each $\alpha \in \mathcal{A}$, the inner design treats $p_i \overset{\mathrm{iid}}{\sim} \mathrm{Beta}(\alpha, \alpha)$, for which $\overline{v}_\alpha = \mathbb{E}[p_i(1-p_i)] = \alpha/[2(2\alpha+1)]$. For every (α, N, b, m), with $m = 1, \ldots, b-1$, it estimates

$$A_{\alpha,N,m} = \frac{N\overline{v}_\alpha}{m}, \qquad B_{\alpha,N,b,m} = \mathbb{E}_{S\sim\pi_\alpha}\left[\sum_{i=1}^{N}\frac{\widetilde{v}_i^{\mathrm{IBN}}(S)}{L_i^{\mathrm{IBN}}(S; m, b)}\right]. \tag{20}$$

The expectation is evaluated with eight independent replicates of 8192 prior-predictive panels. We generate the Beta–Bernoulli pilot paths through the equivalent Pólya-urn predictive law, thereby integrating over both $\mathbf{p}$ and S. Within a replicate, common uniforms give nested paths across pilot sizes and are also reused across α values.

As with HBN, the matched-prior weight and risk are then computed in closed form:

$$w^{\mathrm{IBN}}_{\alpha,N,b,m} = \frac{B_{\alpha,N,b,m}}{A_{\alpha,N,m} + B_{\alpha,N,b,m}}, \qquad \mathcal{R}^{\mathrm{IBN},*}_{\alpha,N,b}(m) = \frac{b}{N\overline{v}_\alpha} \frac{A_{\alpha,N,m} B_{\alpha,N,b,m}}{A_{\alpha,N,m} + B_{\alpha,N,b,m}}. \tag{21}$$

For every (α, N, b), we select the risk-minimizing integer m, again breaking exact ties toward smaller m, and retain its closed-form weight.

IBN outer hindsight selection. Each of the 23 matched-prior schedules is next replayed on every retained task-probability profile. For task-probability profile j, budget b, and strength α, let $\widehat{r}_{j,b,\alpha}$ be the Uniform-normalized variance averaged over 8192 independent pilot vectors. The displayed IBN entry selects separately at each budget

$$\alpha^*_b \in \arg\min_{\alpha \in \mathcal{A}} \widehat{r}_{b,\alpha}, \qquad \widehat{r}_{b,\alpha} = \frac{1}{107} \sum_{j=1}^{107} \widehat{r}_{j,b,\alpha}, \tag{22}$$

with an exact tie broken toward smaller α.

The selected strengths are $\alpha^*_b = 0.13, 0.13, 0.12, 0.11$ at $b = 8, 16, 32, 64$, respectively. These are the IBN settings used in Table 1.

B.4 Variance evaluation protocol

Design and replay use distinct deterministic random-seed namespaces. For each benchmark–checkpoint pair, budget, and candidate policy, replay averages 8192 independent pilot vectors. It then evaluates the continuation contribution analytically via (9); no second-stage Bernoulli outcomes are simulated. EN candidate pilots share nested paths during design. During IBN replay, all α candidates for a fixed task-probability profile and budget share one nested Bernoulli path through the largest selected pilot, reducing Monte Carlo noise in the outer comparison. Allocation ties use task-index order; an all-zero EN score vector uses uniform continuation. All 107 retained task-probability profiles receive equal weight in Table 1.

B.5 System execution and cost accounting

Frozen workload and reward protocol. The system study in Section 4.3 uses the 107 retained profiles, all four budgets, and the full-precision ex-ante HBN designs in Table 5. To compare execution schemes under the same statistical allocation, we couple their pilot rewards: rewards from generated answers could otherwise change the pilot and final allocation across runs. Bernoulli rewards with probability $\widehat{p}^{(K)}_i$ are fixed by logical request ID and shared across execution strategies. They are revealed only after real generation completes. This gives HBN-sync and HBN-async identical pilot rewards and final continuation IDs while measuring real generation time, excluding benchmark scoring latency. Useful-FLOP differences reflect both task allocation and schedule-dependent response lengths.

Serving and timing. We use the generation configuration in Appendix B.1, with eight persistent single-GPU replicas per worker. The common router admits at most 32 outstanding requests per replica, for a total window of 256. It assigns eligible requests to the least-loaded replica and replenishes capacity as completion events arrive.

The main timer starts before initial dispatch and ends when all formally selected request IDs have completed and their rewards are available. Pilot-dependent posterior inference, integer allocation, and online queue management are included. Model loading, warmup, ex-ante design, static quadrature setup, and between-policy reset are excluded. Between policies, we drain outstanding work and clear request and cache state while retaining the loaded models.

Execution policies. The execution policies follow Section 3.2 and Algorithm 2. HBN-async computes provisional allocations in the background, starting from the prior and updating from the latest available partial-pilot snapshot; intermediate snapshots may be skipped while generation proceeds. The exact integer allocator uses threshold search followed by marginal-gain corrections.

Rounded-equivalent Uniform measurements. For each of the 428 profile–budget settings, we derive $b^{\downarrow}_{j,b} = \lfloor b/r_{j,b} \rfloor$ from the unrounded HBN replay ratio and perform a fresh Uniform execution, including settings where the integer budget is unchanged. We retain the prompts, generation settings, eight replicas, concurrency limit, frozen Bernoulli reward protocol, and result-ready timing definition. Each execution accepts exactly $N_j b^{\downarrow}_{j,b}$ rollouts. The time increase in Table 4 is $\sum_j T_{\mathrm{U},j}(b^{\downarrow}_{j,b}) / \sum_j T_{\mathrm{U},j}(b) - 1$, using the original Uniform measurements in the denominator.

Rollout and token accounting. At the original budgets, every policy accepts exactly Nb rollouts. Actual rollout counts add completed-discarded and running-aborted requests to accepted rollouts. The extra-rollout percentage in Table 2 divides the sum of HBN-async's actual counts minus Uniform's actual counts by the sum of Uniform's actual counts within each budget. Uniform's actual and accepted counts coincide. Each running-aborted request counts as one rollout. For completed requests, we record prompt and completion token counts. For aborted requests, we record the observed prompt and generated tokens before cancellation. A queued cancellation with no observed model compute contributes zero to the compute estimate. Cancellation before vLLM reports its first output can conceal already executed computation; hence aborted compute and aggregate waste are observable lower bounds.

Sequence-length-aware FLOP proxy. Let P_r and C_r denote the prompt and observed completion lengths of request r, and write $Q_r = P_r + C_r$. We estimate its cost as

$$\widehat{F}_r = 2\theta_{\mathrm{core}} Q_r + 2\theta_{\mathrm{head}} C_r + 2a d_{\mathrm{attn}} Q_r (Q_r + 1). \tag{23}$$

Here a multiply–add counts as two FLOPs, θ_{core} is the stored text-core parameter count excluding input embeddings, the output head, vision, and multi-token-prediction modules; θ_{head} is the output projection size (the embedding matrix size for tied weights); a is the number of full-attention layers; and d_{attn} is query-head count times head dimension. Counts come from the checkpoint weights and model configurations. Table 7 lists the coefficients.

Checkpoint	$\theta_{\mathrm{core}}/10^9$	$\theta_{\mathrm{head}}/10^9$	a	d_{attn}
Qwen2.5-3B	2.774774	0.311165	36	2048
Qwen2.5-7B	6.525622	0.544997	28	3584
Qwen2.5-32B	31.206741	0.778568	64	5120
Qwen3.5-4B	3.570052	0.635699	8	4096
Qwen3.5-9B	6.919566	1.017119	8	4096
Qwen3.5-27B	24.353202	1.271398	16	6144

Table 7: Coefficients for the estimated FLOP proxy. Parameter counts are displayed in billions and rounded; attention width need not equal the model hidden size.

The first term approximates text-core work as parameter-proportional per token, the second accounts for output projection, and the third approximates causal query–key and attention–value operations over the sequence. We omit explicit GDN recurrent-state updates, whose estimated contribution to total FLOPs is marginal across the evaluated Qwen3.5 models and budgets. This analytical proxy approximates model computation rather than hardware execution cost.

We sum Eq. (23) over accepted requests to obtain F^{useful}, over completed-discarded requests using their full observed lengths, and over running-aborted requests using their recorded partial lengths to obtain F^{waste}. Useful and actual FLOP increases in Table 2 are ratios of summed costs at a given budget. The wasted-FLOP column divides HBN-async's wasted compute by Uniform's useful FLOPs, which equal Uniform's actual FLOPs. With this common denominator, the useful-compute increase and wasted-compute contribution add to the actual-compute increase. In contrast, time is normalized separately within each profile–budget setting as defined in Section 4.3, then averaged with equal weight over the 107 profiles at each budget. The residual overhead is the ratio of mean times minus one, equivalently $\sum_j \widetilde{T}_{\text{HBN-async},j} / \sum_j T_{\mathrm{Uniform},j} - 1$, rather than the mean of per-profile ratios at that budget. This keeps each workload's model-specific reference time in the calculation.

C Additional Experimental Results

C.1 Variance gains across profiles and failure boundary

We complement Section 4.2 with the question: *How broadly do HBN's gains hold across profiles, and where can it lose?*

Gains are widespread, with a small adverse corner case. Figure 1 shows that the average gains are not driven by a few profiles. Requiring the Monte Carlo estimate plus 1.96 standard errors to remain below one, HBN improves on Uniform for 107/107, 106/107, 106/107, and 107/107 profiles at $b = 8, 16, 32, 64$, respectively.

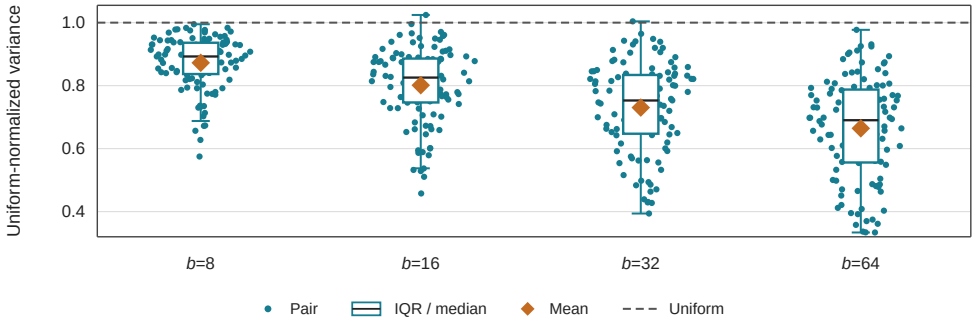


Figure 1: Distribution of HBN's Uniform-normalized variance across 107 benchmark–checkpoint task-probability profiles. Points are individual profiles, boxes show the interquartile range and median, diamonds show the mean, and the dashed line is Uniform.

The only profile with an adverse result is MATH-100 with Qwen3.5-4B, where HBN has mean variance ratios of 1.024 and 1.004 relative to Uniform at $b = 16$ and 32, respectively. This profile is highly concentrated near the endpoints, with 96 of 100 tasks outside $[0.2, 0.8]$. Although its integer Oracle ratio of approximately 0.774 indicates substantial allocative opportunity, a short pilot provides limited information for reliably ranking the task variances. HBN nevertheless reallocates according to these noisy posterior differences, and the resulting misallocation can hurt rather than improve performance. Even in this worst case, HBN is at most 2.4% worse than Uniform. Thus, the observed variance benefit is broad but not a per-profile guarantee: allocative opportunity alone does not ensure that a limited pilot can exploit it. Appendix C.2 provides a complementary detailed AIME24–Qwen3.5-4B case study.

C.2 Profile case study: AIME24 with Qwen3.5-4B

We examine AIME24 with Qwen3.5-4B as a heterogeneous task-probability profile. Among its 30 tasks, 33.3% have $\widehat{p}_i^{(K)} < 0.2$, 36.7% have $\widehat{p}_i^{(K)} > 0.8$, and four are empirical endpoints.

For this profile, HBN's variance ratio falls from 0.937 at $b = 8$ to 0.801 at $b = 64$, while the Oracle ratio is nearly flat at 0.655–0.644. At $b = 32$, HBN attains 0.842, compared with one for Uniform and 0.646 for Oracle. Figure 2(d) confirms that all three estimators remain centered, with HBN narrower than Uniform under the fixed-weight estimator.

C.3 A sequential posterior-sampling baseline

Motivation and adaptation. Inspired by TS-Neyman (Morikawa et al., 2026), we construct a sequential posterior-sampling baseline for the same fixed-profile Bernoulli estimation problem as HBN. TS-Neyman allocates each new observation using a posterior-sampled marginal variance-reduction index. We retain this principle while adapting the observation model and target weights to our setting; we refer to the resulting baseline as Bernoulli TS-Neyman.

The original method uses a normal working model and an inverse-χ^2 posterior over stratum variances. Here outcomes are binary, so we instead use independent $\mathrm{Beta}(\alpha, \alpha)$ working priors over task success

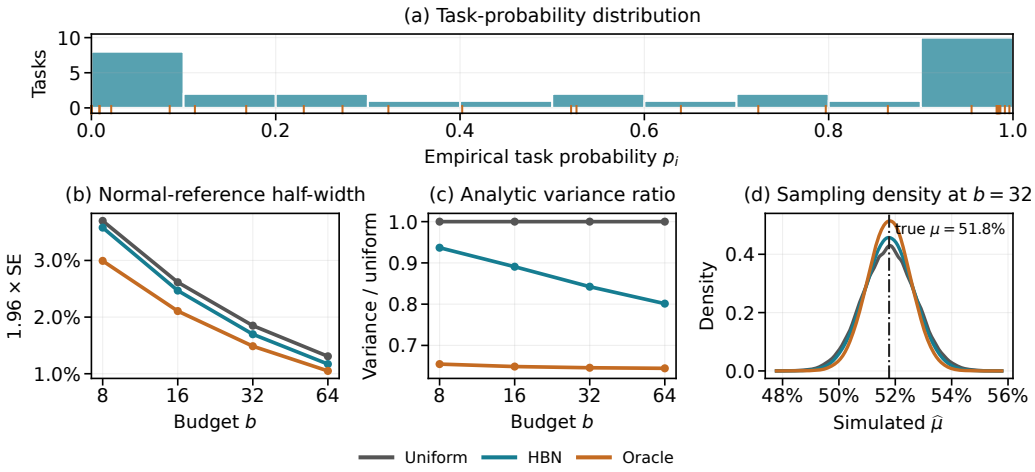


Figure 2: AIME24–Qwen3.5-4B under the jointly designed HBN policy. Panels show (a) the empirical task-probability distribution, (b) the normal-reference half-width, (c) the analytic fixed-profile variance ratio, and (d) sampling densities at $b = 32$ from 200,000 exact Bernoulli repetitions. HBN uses the precommitted $(m, w) = (10, 0.286)$ at $b = 32$; Oracle is the full-information integer Neyman allocation.

probabilities. Beta–Bernoulli conjugacy provides posterior samples of each task's sampling variance, $\widetilde{p}_i(1-\widetilde{p}_i)$. Each task is an equally weighted stratum, with fresh independent Bernoulli observations drawn from the fixed profile as in the main replay.

Sequential allocation and estimator. We first collect m_0 observations per task, charging all Nm_0 observations to the total budget Nb. After t observations in total, let $n_i(t)$ and $S_i(t)$ be the observation and success counts for task i. At each step, independently across tasks, we draw $\widetilde{p}_i(t) \sim \mathrm{Beta}(\alpha + S_i(t), \alpha + n_i(t) - S_i(t))$ and select

$$i_{t+1} \in \arg\max_i \frac{\widetilde{p}_i(t)(1-\widetilde{p}_i(t))}{n_i(t)(n_i(t)+1)}. \tag{24}$$

The common target-weight factor N^{-2} cancels from the index. We break exact ties by task index, observe one new outcome for the selected task, update its posterior, and repeat until $t = Nb$. The final estimate is $\widehat{\mu}_{\mathrm{TS}} = N^{-1}\sum_i S_i(Nb)/n_i(Nb)$, including the initial observations. Because outcomes also influence subsequent sample counts, this estimator need not be unbiased at finite budgets; we therefore evaluate mean squared error (MSE), not variance alone.

Aligned replay and hindsight search. We use the same 107 fixed task-probability profiles and budgets $b \in \{8, 16, 32, 64\}$ as Section 4.2. For every profile and candidate, we simulate 8192 complete adaptive trajectories, recording the four budgets along each trajectory where feasible. HBN evaluates continuation variance analytically, whereas TS simulates the interleaved allocation and observation sequence. We use common random numbers across TS candidates within each profile.

We search $\alpha \in \{0.01, 0.02, \ldots, 0.20, 0.25, 0.50, 1.00\}$, matching the positive prior grid used for IBN, jointly with $m_0 \in \{2, 5, 10\}$, the warm-start values examined in the TS-Neyman paper. The infeasible choice $m_0 = 10$ is excluded at $b = 8$. For each profile j, we normalize the empirical MSE by its analytic Uniform MSE, $V_{\mathrm{U},j}(b) = \sum_i p_{ji}(1-p_{ji})/(N_j^2 b)$, and average these ratios equally over the 107 profiles. At each budget, the hindsight envelope selects one common (α, m_0) minimizing this average. HBN's reference variance equals its MSE because its estimator is unbiased. Selection and reporting use the same replays, so the TS envelope may be optimistic due to Monte Carlo selection noise.

Results. Table 8 reports the completed search over all 23 prior strengths and feasible warm starts. HBN has lower mean Uniform-normalized MSE at all four budgets, with gaps of 0.79–1.97

percentage points despite the hindsight selection afforded to TS. The selected prior strengths are small ($\alpha = 0.04$–0.07), favoring endpoint-concentrated task probabilities.

This equal-rollout comparison does not charge TS's sequential feedback overhead. Each allocation decision in the evaluated TS policy depends on the preceding observation, limiting concurrent generation when rewards become available only after rollout completion. HBN instead concentrates this dependence at the pilot boundary, where speculative pre-allocation can overlap generation with feedback collection. Batched or asynchronous TS could relax the sequential dependency, but would change the policy evaluated here.

b	α	m_0	TS-Neyman	HBN	Δ (pp)
8	0.07	5	88.60%	87.25%	1.35
16	0.06	2	82.09%	80.12%	1.97
32	0.06	2	74.72%	73.03%	1.69
64	0.04	10	67.22%	66.44%	0.79

Table 8: Bernoulli TS-Neyman hindsight envelope versus HBN. MSE entries are mean Uniform-normalized MSE across 107 profiles (lower is better). Each budget selects one common (α, m_0); Δ is TS minus HBN in percentage points.

D Related Work

Allocation and adaptive sampling. The oracle allocation $n_i \propto \sigma_i$ is Neyman allocation (Neyman, 1992), and inverse-probability estimation under unequal sampling traces to Horvitz & Thompson (1952). Feasible two-stage Neyman designs based on preliminary variance estimates are also classical: Sukhatme & Tang (1975) use fixed preliminary samples to choose between proportional and estimated Neyman allocation, while Cai & Rafi (2024) show that allocation noise from a small fixed pilot can make plug-in Neyman allocation less efficient than balanced allocation.

Many recent unknown-variance methods instead update allocation online. Modern active measurement constructs unbiased per-round estimates (Hamilton et al., 2025), while TS-Neyman uses posterior-sampled marginal variance reductions for sequential stratified model evaluation and reports severe sparse-pilot failures of plug-in rules (Morikawa et al., 2026). Several close analyses emphasize eventual large-budget behavior: Cai & Rafi (2024) hold the pilot fixed while the main wave tends to infinity, TS-Neyman proves convergence to the Neyman target and asymptotic optimality as its sequential budget grows, and EST-IVWE matches its oracle rate up to lower-order terms in the total budget (Lee et al., 2026). These results complement our operational target: $b = 8$–64 target-model rollouts per task, where short pilots provide limited task-level evidence. Online policies can adapt as feedback arrives, but their parallel execution must account for delayed feedback and in-flight requests. Our two-stage design fixes the continuation allocation after one complete pilot, leaving execution before that barrier to speculation.

Repeated stochastic LLM evaluation. ReliableEval treats prompt-preserving perturbations as a source of evaluation randomness and uses method-of-moments analysis to determine how many repeats are needed for stable conclusions (Lior et al., 2025). More directly, Saha et al. (2026) and Lee et al. (2026) study allocation for vectors of LLM-as-a-judge scores, whereas we estimate one fixed-benchmark mean from target-model rollouts. Saha et al. minimize worst-coordinate (ℓ_∞) error for a single judge, yielding the oracle allocation $n_i \propto \sigma_i^2$. Their unknown-variance ROBIN-HOOD method follows a uniform warm-up with round-by-round allocation using upper confidence bounds on the variances. Lee et al. extend the vector problem to multiple judges with heterogeneous costs and general ℓ_p error; with one equal-cost judge and $p = 2$, their oracle reduces to Neyman allocation, $n_i \propto \sigma_i$. Our setting instead targets a fixed-benchmark mean from low-budget Bernoulli rollouts, with hierarchical pooling of pilot evidence across tasks.

Active and proxy-assisted model evaluation. Active Testing and Active Surrogate Estimators reduce label cost by selecting which examples to label and correcting the resulting adaptive sample (Kossen et al., 2021; 2022). Factorized Active Querying uses historical model–question factors to

choose a subset of questions and an augmented inverse-probability estimator to recover fixed-bank accuracy with statistical guarantees (Wu et al., 2026). Approximate Neyman Allocation partitions a finite input pool using surrogate semantic entropy and allocates target labels across strata according to proxy Neyman weights (Liu et al., 2026); the surrogate generations are outside the reported target-label budget, and there is no paid target pilot to recycle. MultiPPI instead allocates budget across expensive measurements and multiple cheaper, correlated autoraters or proxies (Cowen-Breen et al., 2026). Benchmark-compression methods such as tinyBenchmarks and Efficient Benchmarking reduce the evaluated suite itself or use coarse-to-fine evaluation (Polo et al., 2024; Perlitz et al., 2024). These approaches change which items or measurement sources are queried. We retain every task in the fixed estimand and change only its number of target-model replications.

Adaptive inference and training compute. A neighboring line of work treats repeated generations as compute used to improve the model's output rather than to estimate its performance. Strategic Scaling formulates test-time allocation across queries as a bandit problem and maximizes the fraction of queries ultimately answered correctly under a fixed generation budget (Zuo & Zhu, 2026). On the training side, VIGOR progressively assigns RLVR rollouts to examples with high group-reward variance because that variance controls the training signal (Jiang et al., 2026). These methods and ours exploit heterogeneous returns to additional rollouts, but their objectives are answer quality or learning progress; ours is the precision of an unbiased fixed-benchmark mean.