LATENCY-AWARE ADAPTIVE BEST-OF-N

Anonymous authors
Paper under double-blind review

ABSTRACT

Best-of-N sampling is a widely deployed inference-time compute method for language models, but it can be computationally inefficient when applied uniformly across prompts without adjusting for problem difficulty. Recent adaptive variants allocate compute based on prompt-level signals such as observed rewards, curtailing generation when additional generations or tokens are unlikely to improve the final response. These methods, however, assume that the cost of additional compute is a fixed constant – an assumption violated in real serving systems, where the wall-clock time to generate a batch varies, growing with current server load, so a method using a single fixed parameter spends too little compute when generation is fast and too much when it is slow. We propose a two-stage adaptive Best-of-N algorithm that adapts jointly to prompt difficulty and to generation latency: it draws an initial batch of completions in parallel, then uses the observed rewards and the time taken to generate them to decide whether to draw a second batch and at what size by weighing expected improvement in reward against predicted cost. Concretely, the algorithm is less likely to draw a second batch when the first batch was slow, and more likely when it was fast. We use a CDF-based reward transformation – mapping each observed reward to its estimated quantile under the prompt's reward distribution – to obtain a closed-form expression for the expected improvement that requires no Monte Carlo estimation. Because reward distributions have arbitrarily different scales across prompts, language models, and reward models, the CDF transformation also allows the same tradeoff parameters to be reused across tasks without retuning. We evaluate the method on a preference-based alignment benchmark and show that, under fixed latency, it improves over parallel Best-of-N on the reward–compute Pareto frontier, and that, under variable latency, conditioning on the measured first-batch latency gives a higher *user reward* – the reward minus the cost of latency – than fixed-threshold (latency-unaware) baselines at matched numbers of generations, with the advantage growing as latency variability increases.

1 INTRODUCTION

Test-time compute (TTC) methods improve the quality of language model (LM) outputs by spending additional computation at inference time (Welleck et al., 2024; Snell et al., 2025). A central challenge is to decide *how much* compute to spend per request: too little produces lower-quality responses, while too much wastes resources and increases latency for both the current and future requests. A widely used parallel TTC method is Best-of-N sampling: given a prompt, draw N completions independently from the LM, score each with a reward model, and return the highest-scoring response. The reward model may capture human preferences for alignment (Ouyang et al., 2022; Stiennon et al., 2020), or measure correctness for math and coding tasks (Cobbe et al., 2021). Best-of-N is widely deployed in practice because of its simplicity, parallelism, and effectiveness, but the right value of N depends on the prompt: easy prompts saturate quickly, while hard prompts continue to benefit from additional generations.

A line of recent work proposes *adaptive* variants of Best-of-N that allocate compute based on prompt-level signals such as observed rewards, curtailing generation when additional generations or tokens are unlikely to improve the final response (Section 1.1). These methods reduce wasted compute relative to fixed-N Best-of-N, but they are designed under the assumption that the cost of one additional generation is a fixed constant, independent of the current load on the serving system.

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.

This gap between assumed and actual cost matters because LM serving systems run under non-trivial load: at any given moment, dozens to thousands of requests are being processed concurrently, and the time to generate a batch grows with the number of concurrent requests (Kwon et al., 2023; Yu et al., 2022; Mitzenmacher & Shahout, 2025). A method that has been tuned offline to stop at a fixed threshold therefore stops too early when generation is fast (continuing would be cheap) and too late when it is slow (continuing is expensive). LASER (Tang et al., 2026) makes such a threshold load-dependent, but for chain-of-thought (Wei et al., 2022) early exit. To our knowledge, no prior adaptive Best-of-N method conditions its stopping decisions on measured generation latency.

We propose a two-stage adaptive Best-of-N algorithm that adapts jointly to prompt difficulty and to generation latency. The algorithm draws a first batch of N_1 completions in parallel and observes their rewards together with the wall-clock time T_1 taken to generate them. The key insight is that T_1 is an informative *predictor of the second batch's latency* T_2: whatever makes the first batch slow – server load, prompt length, output length, scheduling – largely persists over the short interval before the second batch is submitted, and the two batches share the same prompt. We therefore draw a second batch, and choose its size, only when the expected reward improvement exceeds the cost predicted from T_1 – so the rule stops more readily when generation is slow, whatever the cause, with no load monitoring required. We derive it as the solution to a two-stage decision problem under a CDF-based reward transformation, which yields a closed-form expression for the expected improvement that requires no Monte Carlo estimation. Because reward distributions have arbitrarily different scales across prompts, language models, and reward models, the transformation also makes the tradeoff parameters scale-free: the same values can be reused across tasks without per-task retuning. Empirically, under fixed latency it dominates parallel Best-of-N on the reward–compute Pareto frontier, and under variable latency, at matched numbers of generations, it gives a higher user reward, the reward minus the cost of latency, than fixed-threshold (latency-unaware) baselines, with the advantage growing as latency variability increases.

1.1 Related work

Adaptive test-time compute. A growing body of work adapts the compute spent on a request to its difficulty (Welleck et al., 2024; Alomrani et al., 2025; Snell et al., 2025), using prompt-level signals such as observed rewards, verifier scores, or self-consistency (Wang et al., 2023; Aggarwal et al., 2023; Li et al., 2024; Fu et al., 2025), and treats the cost of compute as fixed.

Adaptive Best-of-N. Several recent methods are specifically tailored to Best-of-N. *Speculative Rejection* (Sun et al., 2024) prunes low-reward completions mid-generation from extrapolated partial scores, and *ST-BoN* (Wang et al., 2025) keeps only the generation whose early hidden states agree most with the others'. Both reject candidates before decoding completes rather than allocating from the rewards of completed ones, and neither conditions on latency; they could compose with our outer two-stage rule.

Damani et al. (2024) make Best-of-N allocation input-adaptive by training an auxiliary model to predict each prompt's marginal reward gain from additional samples, then distributing a fixed total budget across a batch of prompts by those predictions; the model must be retrained whenever the domain, LM, decoding strategy, or budget changes. *AdaBoN* (Raman et al., 2025) retains the input-adaptive idea but needs no auxiliary training, so it applies to any LM–reward-model pair: it draws two batches, sizing the second from the rewards observed in the first batch. Its adaptivity, however, comes from *amortizing* that choice across many prompts in a fixed offline workload; for a single prompt, the relevant case in online serving, it always draws the second batch at the maximum budget, matching parallel Best-of-N's generations at twice the latency, and so is strictly dominated by it. It also uses no reward transformation comparable across prompts.

Kalayci et al. (2025) cast inference-time alignment as an infinite-horizon Pandora's Box problem (Weitzman, 1979), generating responses one at a time and stopping once a reward exceeds an estimated Gittins index. Their main experiments are sequential, impractical in parallel serving stacks, where doubling per-request latency to halve compute is rarely worthwhile; a parallel generalization drawing a fresh batch each step would still estimate the reward distribution only from each batch's maximum. Their heuristic "acceptance probability", from a Bradley–Terry surrogate, is also less principled than our CDF transformation, which is exactly the prompt-conditional quantile. Several further adaptive Best-of-N rules share this fixed-cost structure. BEACON (Wan et al., 2025) places

a conjugate Normal prior on a prompt's rewards and stops when a Bayesian index falls below a per-sample cost c, in its parallel variant updating the posterior after each fixed-size batch; Manvi et al. (2024) resample in geometrically growing batches until a fine-tuned self-evaluation's win-or-tie probability crosses a fixed threshold; and Dughmi et al. (2026) analyze Pandora's Box opened in batches with a known per-batch setup cost. In each, the cost of further sampling is fixed in advance, as in our latency-unaware baseline.

Load- and latency-aware test-time compute. Fu et al. (2025) integrate adaptive test-time compute with an online serving stack. Their *certaindex*, a reasoning-progress signal defined separately for each TTC algorithm – self-consistency, Monte Carlo tree search (MCTS) (Hao et al., 2023; Wan et al., 2024), chain-of-thought, and Rebase (Wu et al., 2025) – serves as a stopping criterion within a fixed compute cap, and their Dynasor system schedules the workloads on SGLang (Zheng et al., 2024). The threshold is set offline by validation-accuracy profiling, not derived from a quality–cost objective, so it must be re-tuned for each (algorithm, dataset) pair. Dynasor is load-aware in scheduling but not in stopping: for self-consistency, MCTS, and Rebase, the deployed rule checks certaindex once, at a fixed step, and otherwise runs to the cap. For Best-of-N, their structure amounts to a fixed threshold on the best first-batch reward – our latency-unaware, non-adaptive variant (Section 3). Huang et al. (2025) maximize, as we do, a quality measure (accuracy) minus penalties on token count and latency. Their adaptivity lies in a trained probe that predicts accuracy from the prompt embedding and strategy features, used to choose a TTC strategy and budget per query before generation rather than mid-generation; the costs are each strategy's average token count and wall-clock time, precomputed on the training set, so they do not respond to load at request time. The only method we know of whose compute choice does is LASER (Tang et al., 2026), which ties the chain-of-thought early-exit threshold of DEER (Yang et al., 2026) to the number of active requests on an edge server through a hand-tuned map; ours instead follows from a reward–latency objective and responds to the request's own measured latency, with no access to server internals.

2 PRELIMINARIES

2.1 LANGUAGE MODELS, REWARD MODELS, AND BEST-OF-N SAMPLING

Let $\mathcal{X}$ denote the space of prompts and $\mathcal{Y}$ denote the space of responses. A *language model* (LM) $\pi^{\text{LLM}} : \mathcal{X} \to \Delta(\mathcal{Y})$ is a mapping from prompts to distributions over responses. A *reward model* is a function $r : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that maps a prompt and a response to a real number, with the property that for a fixed prompt, responses with higher reward values are preferable to responses with lower reward values. Given a prompt $x \in \mathcal{X}$, an LM π^{LLM}, and a reward model r, denote by $D_{r,\pi^{\text{LLM}}(x)} \in \Delta(\mathbb{R})$ the distribution over rewards $r(x, y)$ when $y \sim \pi^{\text{LLM}}(x)$. We will write F_x for the cumulative distribution function (CDF) of $D_{r,\pi^{\text{LLM}}(x)}$ when the LM and reward model are fixed and clear from context.

Best-of-N sampling draws N completions $y_1, \ldots, y_N \overset{\text{i.i.d.}}{\sim} \pi^{\text{LLM}}(x)$, evaluates rewards $r_i = r(x, y_i)$, and returns the response $y_{i^\star}$ with $i^\star \in \arg\max_{i \in [N]} r_i$. Additional generations have diminishing marginal returns (Raman et al., 2025). Best-of-N therefore induces a fundamental *reward–compute tradeoff*: larger N yields higher expected reward at the price of additional generation cost. In a fixed-N deployment, N is chosen offline to balance these two quantities once and for all, regardless of the prompt or the state of the serving system.

2.2 LLM SERVING AND RESPONSE LATENCY

Best-of-N runs on a serving system that hosts π^{LLM} and r and processes many concurrent requests; we take that stack as given. We measure cost by two quantities the client observes: the *number of generations* drawn, a proxy for the client's token cost and the system's compute cost, and the *response latency* T, the wall-clock time between submitting a request and receiving its outputs.

Unlike the number of generations, latency is not a function of the algorithm's compute demand alone. We submit the N completions of a draw together, as a single *batch*, and observe one wall-clock latency T for the batch to complete; whether the server decodes all N concurrently, or serializes part of the work when capacity is short, is not visible to the client. Serving systems interleave concurrent requests on a shared accelerator (Kwon et al., 2023; Yu et al., 2022), so T depends on the state of the

server – the other requests in flight – as well as on N, through queueing dynamics that are non-trivial to characterize (Mitzenmacher & Shahout, 2025) and whose form we treat as unknown to the client. Throughout, we focus on the *high-load regime*, in which the number of concurrent requests on the server is large enough that the marginal contribution of a single N-sized batch to the system state is small. This does two things. First, when the server does decode a batch's completions together, it makes T approximately independent of N: an assumption of our simulations in Section 4, which the method itself does not need – it allows the latency of a batch to depend on its size – and which Appendix L relaxes. Second, it keeps one client's stopping decisions from materially changing the load that other clients face, so we may optimize a single client's objective rather than an equilibrium among clients. We rely on only two structural facts: that one batch's latency is informative about that of later batches submitted to the same server in the near future, and that the first batch's latency is measured before the second batch must be decided.

2.3 The CDF reward transformation

Preference reward models are trained via the Bradley-Terry objective, which captures only *relative* preferences within a prompt; raw scores are therefore not comparable across prompts, LMs, or reward functions, making it unclear what it means to subtract a latency cost from a raw reward. The CDF transformation F_x maps each reward to its prompt-conditional quantile – equivalently, the probability that a fresh draw from the LM would score lower, i.e., a *win rate*. This yields two benefits: (i) *scale-free parameters* – the same λ applies across tasks without retuning, because $F_x(r) \in [0, 1]$ regardless of reward scale; and (ii) a *closed-form expected improvement* – since $F_x(r) \sim \mathrm{Unif}[0, 1]$ for $r \sim D_{r,\pi^{\mathrm{LLM}}(x)}$, the quantity $\Delta(u, N_2)$ derived in Section 3.2 has an analytic expression requiring no Monte Carlo estimation. For outcome-based reward models (e.g., math or coding accuracy), rewards are inherently cross-prompt comparable, and the natural alternative is the identity transformation with Monte Carlo computation of Δ, which we leave to future work; the CDF transform remains valid in this setting but optimizes per-prompt relative performance rather than mean accuracy. Additionally, the closed form for $\Delta(u, N)$ requires a continuous reward distribution $D_{r,\pi^{\mathrm{LLM}}(x)}$, so that $F_x(r) \sim \mathrm{Unif}[0, 1]$; in cases where the rewards are discrete, such as binary pass/fail, Δ would have to be computed from the discrete distribution instead.

3 Latency-Aware Two-Stage Best-of-N

Assume we are given finite sets $\mathcal{Z}_1, \mathcal{Z}_2 \subseteq \mathbb{N}$ of batch sizes to choose from and parameters $\lambda_{\text{gen}}, \lambda_{\text{lat}} > 0$ which control the tradeoff between obtaining a response with high final reward and incurring low generation cost and latency. Throughout, we apply the prompt-conditional CDF transformation F_x to rewards before combining them with cost terms, as motivated in Section 2.3.

When a new prompt (request) x is received, an initial batch of $N_1 \in \mathcal{Z}_1$ responses is generated in parallel and scored with a reward model, resulting in rewards $r_1^{(1)}, \ldots, r_{N_1}^{(1)} \overset{\text{i.i.d.}}{\sim} D_{r,\pi^{\mathrm{LLM}}(x)}$ and giving a maximum reward of $R_1 = \max_{1 \le i \le N_1} r_i^{(1)}$. The parallel generation is measured to take T_1 seconds and incurs a cost of $c_1 = \lambda_{\text{gen}} N_1 + \lambda_{\text{lat}} T_1$.

Next, the policy π_1 picks an action a_1 from the following:

- *Generate another batch* of a chosen size $a_1 = N_2 \in \mathcal{Z}_2$ of new responses in parallel, giving rewards $r_1^{(2)}, \ldots, r_{N_2}^{(2)} \overset{\text{i.i.d.}}{\sim} D_{r,\pi^{\mathrm{LLM}}(x)}$ and a maximum reward of $R_2 = \max_{1 \le i \le N_2} r_i^{(2)}$ for the batch. The parallel generation is measured to take T_2 seconds and incurs a cost of $c_2 = \lambda_{\text{gen}} N_2 + \lambda_{\text{lat}} T_2$, and the highest-reward response is chosen with a final utility of $\max\{F_x(R_1), F_x(R_2)\}$.
- *Stop generation* ($a_1 = \mathrm{stop}$) and output the response with highest reward observed so far, collecting a final utility of $F_x(R_1)$.

The action $a_1 = \pi_1(s_1)$ is chosen based on the *state* $s_1 = (r_1^{(1)}, \ldots, r_{N_1}^{(1)}, T_1)$, which includes the rewards observed in the first batch and the latency the first batch took. We want to choose the policy

that maximizes the profit V_π, defined as the expected net utility:

$$V_\pi \coloneqq \mathbb{E}_\pi\left[\max_{1\le i\le\tau} F_x(R_i) - \sum_{i=1}^{\tau} c_i\right] = \mathbb{E}_\pi\left[\max_{1\le i\le\tau} F_x(R_i) - \lambda_{\text{gen}}\sum_{i=1}^{\tau} N_i - \lambda_{\text{lat}}\sum_{i=1}^{\tau} T_i\right]$$

where $\tau \in \{1, 2\}$ is the number of sequential batches that are evaluated, and $\mathbb{E}_\pi$ denotes the expected value where the policy π is followed.

3.1 The latency-aware and latency-unaware policies

First let us find the optimal decision for whether to generate the second batch, and if so, what the batch size N_2 should be. Denote $Q_1(s_1, a_1)$ as the Q-function of taking action a_1 in state s_1; clearly $Q_1(s_1, \text{stop}) = F_x(R_1)$, and

$$Q_1(s_1, N_2) = -\lambda_{\text{gen}} N_2 - \lambda_{\text{lat}}\widehat{T}_2(s_1, N_2) + \mathbb{E}[\max\{F_x(R_1), F_x(R_2)\} \mid s_1,\ N_2]$$

where $\widehat{T}_2(s_1, N_2) \coloneqq \mathbb{E}[T_2 \mid s_1, N_2]$ is the expected latency of the second batch given everything observed after the first; what follows uses only that it exists. Now, define $f_1(s_1, N_2) \coloneqq Q_1(s_1, N_2) - Q_1(s_1, \text{stop})$. We can compute it as:

$$\begin{aligned} f_1(s_1, N_2) &= -\lambda_{\text{gen}} N_2 - \lambda_{\text{lat}}\widehat{T}_2(s_1, N_2) + \mathbb{E}[\max\{F_x(R_1), F_x(R_2)\} \mid s_1,\ N_2] - F_x(R_1) \\ &= -\lambda_{\text{gen}} N_2 - \lambda_{\text{lat}}\widehat{T}_2(s_1, N_2) + \mathbb{E}[[F_x(R_2) - F_x(R_1)]_+ \mid s_1,\ N_2] \end{aligned}$$

where $[\cdot]_+ = \max\{0, \cdot\}$. The optimal policy for the second batch, which we call the *latency-aware* policy, is as follows:

1. Find $N_2^\star \in \arg\max_{N_2 \in \mathcal{Z}_2} f_1(s_1, N_2)$
2. If $f_1(s_1, N_2^\star) > 0$, then generate a second batch of size $N_2^\star$. Otherwise, stop.

For baseline comparison, we also consider a *latency-unaware* policy, which does not have access to the latency measurement T_1 and therefore predicts the second batch's latency by the single constant $\overline{T}_2 \coloneqq \mathbb{E}[T_2]$, its unconditional mean, so, like existing adaptive methods, it stops once its signal, here the estimated quantile of R_1, clears a threshold fixed offline (Appendix A). Note that $\overline{T}_2$ is a parameter of this baseline alone: our policy never uses it. The latency-unaware policy similarly maximizes $\widehat{f}_1(s_1, N_2)$, where

$$\widehat{f}_1(s_1, N_2) = -\lambda_{\text{gen}} N_2 - \lambda_{\text{lat}}\overline{T}_2 + \mathbb{E}[[F_x(R_2) - F_x(R_1)]_+ \mid s_1,\ N_2].$$

3.2 Calculation and estimation of the expected reward improvement

Both $f_1(s_1, N_2)$ and $\widehat{f}_1(s_1, N_2)$ require the estimation of the expected reward improvement term $\mathbb{E}[[F_x(R_2) - F_x(R_1)]_+ \mid s_1,\ N_2]$, which can be computed as follows:

$$\begin{aligned} \mathbb{E}[[F_x(R_2) - F_x(R_1)]_+ \mid s_1,\ N_2] &= \mathbb{E}[[F_x(R_2) - F_x(R_1)]_+ \mid F_x(R_1),\ N_2] \\ &= \Delta(F_x(R_1), N_2) \end{aligned}$$

where $\Delta(u, N)$ is defined for $u \in [0, 1]$ as:

$$\Delta(u, N) \coloneqq \mathop{\mathbb{E}}_{U_1,\ldots,U_N \overset{\text{i.i.d.}}{\sim} \text{Unif}[0,1]}\left[\left(\max_{i\in[N]} U_i - u\right)_+\right] = \frac{N}{N+1} - u + \frac{u^{N+1}}{N+1}.$$

In practice, the reward CDF F_x is unknown. We adopt a plug-in approach: estimate the CDF F_x from the rewards observed in the first batch by an estimated CDF $\hat{F}_x$, resulting the approximation of $\Delta(F_x(R_1), N_2)$ by $\Delta(\hat{F}_x(R_1), N_2)$. Specifically, we fit a *shifted-exponential* model to the upper tail of the exponentiated first-batch rewards, following the approach used by Kalayci et al. (2025); we choose it for its accuracy at small N_1, its single closed form, and its having no fitting procedure that can fail to converge. Because this model is fitted to e^r, it is not invariant to the scale of the reward, and reward models differ in scale by orders of magnitude; we therefore first rescale rewards to unit median within-prompt standard deviation, using one constant estimated offline. Only one

of the three reward models we test needs this step: the other two already have a median within-prompt standard deviation close to 1, so their rewards could be fitted without rescaling and without the offline estimate. We nonetheless rescale in every configuration for methodological consistency. Appendix D defines the estimator and compares it with five alternatives in accuracy, and with the closest of them inside the stopping rule, Appendix F shows that an offline calibration removes its bias, and Appendix E treats the rescaling.

3.3 Adaptive and non-adaptive second batch size variants

In the experiments, we consider both *adaptive* and *non-adaptive* batch size policies.

The adaptive batch size policy uses $\mathcal{Z}_1 = \mathcal{Z}_2 = \mathcal{Z}$ for some chosen finite set $\mathcal{Z} \subseteq \mathbb{N}$ of allowed batch sizes. N_1 is pre-set to maximize the profit, while N_2 is chosen adaptively based on the observed rewards in the first batch, and the latency of the first batch if latency-aware.

The non-adaptive batch size policy sets $\mathcal{Z}_1 = \{N_1\}$ and $\mathcal{Z}_2 = \{N_2\}$, where N_1 and N_2 are jointly selected from the same set $\mathcal{Z}$ to maximize the profit in advance; this is the variant that lines up with fixed-threshold rules that also pre-commit to batch sizes rather than adapting them online (e.g., Fu et al., 2025; Manvi et al., 2024; Wan et al., 2025). Concretely, the pair is chosen by:

$$(N_1, N_2) \in \arg\max_{N_1, N_2 \in \mathcal{Z}} \; Q_0^{\pi}(s_0, N_1)\big|_{\mathcal{Z}_2 = \{N_2\}},$$

with the second batch's latency predicted by the constant $\overline{T}_2$ for the latency-unaware variant (the fixed-threshold baseline) and by $\widehat{T}_2(s_1, N_2)$ for the latency-aware variant; see Appendix A for the explicit computation. Here, Q_0^{π} is the profit-to-go of policy π with initial batch size N_1:

$$Q_0^{\pi}(s_0, N_1) = -\lambda_{\text{gen}} N_1 - \lambda_{\text{lat}} \overline{T}_1(N_1) + \mathbb{E}[Q_1(s_1, \pi_1(s_1)) \mid s_0,\, N_1],$$

and $\overline{T}_1(N_1) \coloneqq \mathbb{E}[T_1 \mid s_0, N_1]$ is the expected latency of a first batch of size N_1, estimable in practice from historical latency logs.

4 Experiments

We evaluate our method through two simulation experiments: (i) the basic reward–compute tradeoff against parallel Best-of-N and against an oracle version of our method that uses the true reward CDF (Section 4.2); and (ii) the value of conditioning on T_1 as the latency distribution becomes more variable, against a latency-unaware ablation (Section 4.3). An additional ablation on the value of adaptively choosing the second-batch size N_2 is reported in Appendix M.

4.1 Experimental setup

Every experiment below is run on all 24 combinations of four language models (Llama 3.1 8B, Gemma-2-9B, Mistral-7B, Qwen-2.5-7B), three reward models (FsfairX-LLaMA3-RM-v0.1, RM-Mistral-7B, ArmoRM), and two prompt datasets (AlpacaFarm (Dubois et al., 2023), HH-RLHF (Bai et al., 2022)), each with 100 prompts. Every figure and table in this section reports the *average* over the 24, so that no result rests on a single reward-distribution shape. Appendix I reports each configuration individually; Appendix H discusses why one $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pair is meaningful across all of them. Each prompt x_b, $b = 1, \ldots, 100$, has a precomputed pool of 960 rewards $\mathcal{R}_b = \{r_{b,1}, \ldots, r_{b,960}\}$ sampled from $D_{r, \pi^{\text{LLM}}(x_b)}$. The evaluation quality of a returned response is its *reward quantile* within the 960 sampled rewards for that prompt: its rank among them divided by 961, an unbiased estimate of $F_{x_b}(r)$ (Appendix B).

We sweep a geometric grid of 71 values of $\lambda_{\text{gen}} \in [10^{-10}, 10^{-3}]$ and 61 values of $\lambda_{\text{lat}} \in [10^{-8}, 10^{-2}]$, 4331 pairs in all. For each $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pair, policy and latency distribution p, we run $n_{\text{MC}} = 1000$ Monte Carlo trials per prompt x_b. Trial i draws a latency $T_{1,i} \sim p$ and a Stage 1 set $A_{b,i}$ of N_1 rewards without replacement from $\mathcal{R}_b$; if the policy continues with $N_{2,i}$ more, the second batch $B_{b,i}$ is drawn without replacement from $\mathcal{R}_b \setminus A_{b,i}$. ($N_1$ is fixed in Experiments 1 and 2; in the adaptive variant of Appendix M it depends on $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$, and $N_{2,i}$ varies across trials.) The second-batch latency $T_{2,i}$ is set equal to the first-batch latency $T_{1,i}$, so the latency-aware policy's predictor is $\widehat{T}_2(s_1, N_2) = T_1$: the simplest one-parameter choice, capturing the short-horizon autocorrelation

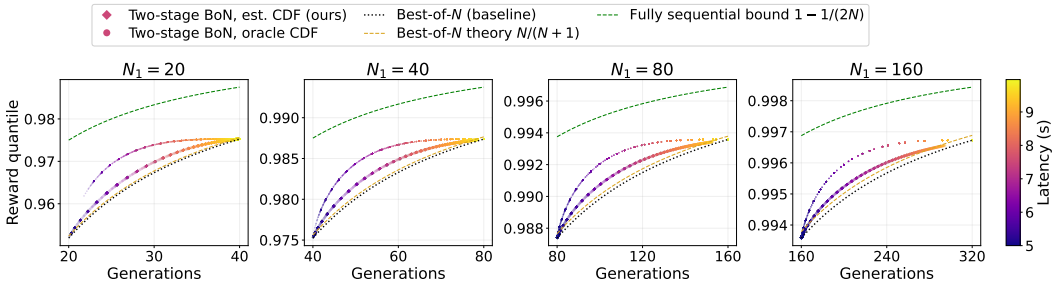


Figure 1: Experiment 1: reward–compute Pareto frontiers (mean reward quantile vs. mean number of generations) under deterministic latency ($T_1 = T_2 = 5\,\mathrm{s}$), for $N_1 \in \{20, 40, 80, 160\}$ (left to right), averaged over the 24 configurations of Appendix I. Each point is one $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pair; colour encodes mean latency. *Our method* (diamonds, estimated $\hat{F}_x$) dominates the *parallel Best-of-N baseline* (dotted black) at matched compute across all four N_1, confirming the benefit of adaptive Stage-2 stopping. Circles are the oracle variant of our method (true F_x from the full reward pool); the diamond-to-circle gap isolates the cost of estimating F_x from only N_1 Stage-1 rewards. Dashed gold: Best-of-N theory $N/(N+1)$; dashed green: fully sequential theoretical upper bound (Appendix C).

of batch latency, since a heavily loaded server tends to stay loaded for the next batch. Appendix J relaxes this and sweeps how much the reported gain depends on it. Latency is drawn independently of the batch sizes and of the rewards, so the expected first-batch latency $\overline{T}_1(N_1)$ does not vary with N_1; we write $\overline{T}$ for it, and the second batch has the same mean. Appendix L relaxes the independence from batch size.

We report means of reward quantile, latency and generations over all prompts and trials, as point estimates without error bars, which are hard to compute correctly for draws without replacement; all policies and λ pairs share the same seeded draws, so comparisons are paired (Appendix N).

In all experiments we also evaluate an *oracle* variant of each policy, with F_{x_b} replaced by the reward quantile over the full pool $\mathcal{R}_b$ instead of the estimate $\hat{F}_x$ from the Stage 1 rewards; comparing a policy with its oracle isolates the cost of estimating F_x.

4.2 Experiment 1: Reward–compute tradeoff under deterministic latency

The first experiment characterizes the basic reward–compute tradeoff achieved by our method in the simplest setting, in which latency is held fixed (so latency-aware and latency-unaware policies coincide) and the second-batch size is not chosen adaptively. We use a deterministic latency $T_1 = T_2 = \overline{T} = 5$ seconds[1], and the non-adaptive variant of Section 3 with the additional simplification $N_1 = N_2$, swept over $N_1 \in \{20, 40, 80, 160\}$. For each value of N_1 we sweep the full 71×61 grid of $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pairs described in Section 4.1 (Appendix G shows the outcome cell by cell at $N_1 = 80$), and we evaluate three policies: parallel Best-of-N (with batch size matched to the total generation count), our method using the estimate $\hat{F}_x$, and the oracle version of our method using the reward quantile over $\mathcal{R}_b$.

Figure 1 plots the reward–compute Pareto frontier of our method, its oracle variant and parallel Best-of-N at each N_1. At every N_1, our method dominates Best-of-N at matched compute, and lies below its oracle variant; the gap between the two is the cost of estimating F_x from the N_1 Stage 1 rewards.

Table 1 reads the comparison at three matched compute budgets, as the fraction of the way from Best-of-N to the oracle that our method recovers: $1.1N_1$ (the policy mostly stops at Stage 1), $1.5N_1$ (the midpoint of the $[N_1, 2N_1]$ range the non-adaptive policy spans) and $1.8N_1$. The fraction rises with compute, from 24–38% at $1.1N_1$ to 40–58% at $1.8N_1$, but the rise flattens as N_1 grows. Contrary to what one might expect, the estimation gap does not shrink monotonically as N_1 grows; the

[1]The absolute value is immaterial: rescaling $\overline{T}$ by a constant is equivalent to rescaling λ_{lat} by its inverse, and we sweep λ_{lat} over a grid.

Table 1: Experiment 1: mean reward quantile at $1.5N_1$ mean generations, and fraction of the Best-of-N-to-oracle gap recovered at $1.1N_1$, $1.5N_1$ and $1.8N_1$, read off Figure 1.

	mean reward quantile at $1.5N_1$			gap recovered at		
N_1	parallel BoN	ours, $\hat{F}_x$	ours, oracle	$1.1N_1$	$1.5N_1$	$1.8N_1$
20	0.96733	0.96982	0.97388	24%	38%	56%
40	0.98330	0.98489	0.98669	34%	47%	58%
80	0.99150	0.99229	0.99323	38%	46%	51%
160	0.99567	0.99601	0.99655	35%	38%	40%

Table 2: Experiment 2, both policies at the same $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$, averaged over all 24 combinations of four language models, three reward models, and two datasets: the three quantities they trade off and the objective $V = \overline{F_x(R)} - \lambda_{\text{gen}}\overline{N} - \lambda_{\text{lat}}\overline{T}$ of Section 3. Non-adaptive variant, $N_1 = N_2 = 80$, log-normal latency with $\overline{T} = 5\,\text{s}$, at $\lambda_{\text{gen}} = 10^{-8}$ and $\lambda_{\text{lat}} = 10^{-2.5} \approx 0.0032\,\text{s}^{-1}$, the same pair in every configuration and not retuned. Per-configuration values are in Table 15.

c_{v}	policy	mean gens	reward quantile	mean latency (s)	V	ΔV
0.5	latency-unaware	84.6	0.9886	5.288	0.97183	–
0.5	latency-aware (ours)	86.3	0.9888	5.284	0.97208	+0.00025
1	latency-unaware	84.6	0.9886	5.287	0.97183	–
1	latency-aware (ours)	89.6	0.9892	5.273	0.97253	+0.00070
2	latency-unaware	84.6	0.9886	5.295	0.97181	–
2	latency-aware (ours)	95.2	0.9898	5.254	0.97322	+0.00141

mechanism behind this N_1-dependence is unclear and we leave it to future investigation. We do not read the fraction beyond $1.8N_1$, where the oracle and Best-of-N curves nearly meet and the ratio is numerically unstable. Latency is held deterministic here, so it is simply proportional to the number of generations.

4.3 Experiment 2: Value of latency-awareness under variable latency

The second experiment isolates the contribution of conditioning the stopping rule on the measured Stage 1 latency T_1, by comparing the latency-aware and latency-unaware variants of the non-adaptive policy as the latency distribution becomes more variable. As in Experiment 1 (Section 4.2), we consider the non-adaptive variant with pre-determined batch sizes $N_1 = N_2 = 80$. The latency distribution is taken to be log-normal with mean $\overline{T} = 5$ and coefficient of variation $c_{\text{v}} = \text{std}(T_1)/\overline{T}$, related to the standard log-normal parameters by $\sigma = \sqrt{\ln(1 + c_{\text{v}}^2)}$ and $\mu = \ln \overline{T} - \sigma^2/2$. We run the experiment across three values $c_{\text{v}} \in \{0.5, 1, 2\}$, sweeping from low to high latency variance.

Because the latency-unaware baseline never sees T_1, it behaves the same at every c_{v}. Our policy continues to Stage 2 more often when T_1 is small and less often when it is large, so the more T_1 varies, the more it can gain. The quality of the returned response and the time spent waiting for it are what the user experiences, whereas the generations are a cost to whoever serves the model. Figure 2 therefore reports each policy's *user reward*, $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$ – the objective V of Section 3 without its generation cost – against its mean number of generations. At every c_{v} and every λ_{lat} plotted (10^{-3}, $10^{-2.5}$ here; also $10^{-4.5}$, $10^{-3.5}$, 10^{-2} in Figure 14, Appendix K), the latency-aware policy's user reward is at least the latency-unaware policy's at every number of generations the latter reaches, with a gap growing in c_{v} where they differ; Figure 12 (Appendix I.1) shows each of the 24 configurations.

Parallel Best-of-N waits for only one batch. At $\lambda_{\text{lat}} = 10^{-3}$ both estimated policies have a higher user reward than Best-of-N except at the largest numbers of generations they reach; at $\lambda_{\text{lat}} = 10^{-2.5}$ Best-of-N is hard to beat with $\hat{F}_x$. With the oracle F_x, both policies' user reward is higher than Best-of-N's at almost every point, so the shortfall comes from estimating F_x from the N_1 Stage-1 rewards rather than from the stopping rule.

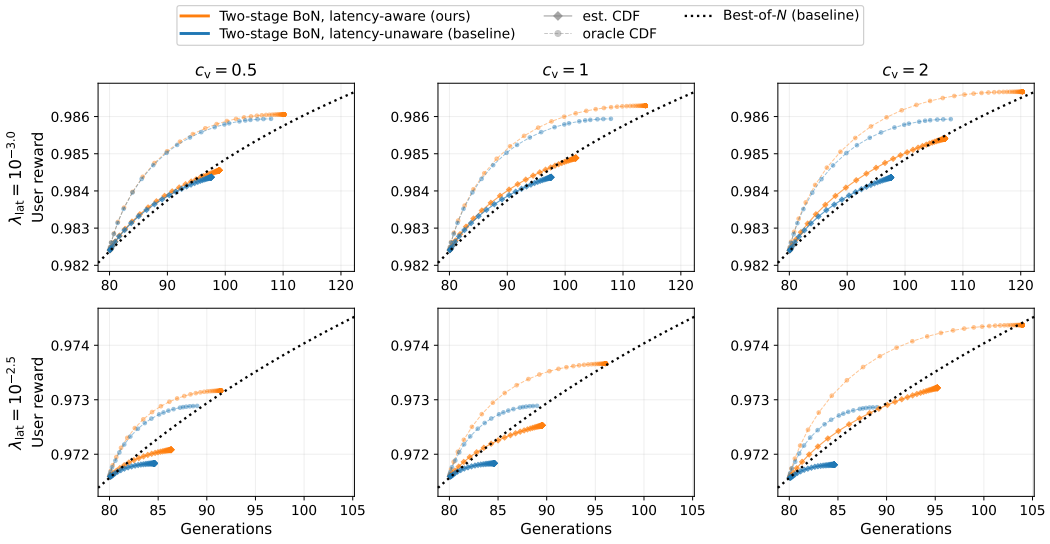


Figure 2: Experiment 2: user reward $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$ against mean generations, under log-normal latency with mean $\overline{T} = 5$ s and coefficient of variation $c_{\text{v}} \in \{0.5, 1, 2\}$ (columns), at $\lambda_{\text{lat}} = 10^{-3}$ and $10^{-2.5}$ (rows; $10^{-2.5}$ is the value of Table 2); $N_1 = N_2 = 80$; averaged over the 24 configurations of Appendix I. Each curve sweeps λ_{gen}. Orange: *our latency-aware policy*; blue: the latency-unaware baseline, a fixed-threshold stopping rule like existing adaptive methods. Diamonds: estimated $\hat{F}_x$; circles: oracle F_x; dotted black: parallel Best-of-N, charged one batch's latency.

Table 2 reports both policies at a single common $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$, giving all three quantities they trade off together with the objective V of Section 3. The latency-aware policy continues *more* often on average, yet attains both a higher mean reward and a lower mean latency, and the advantage ΔV over the latency-unaware baseline grows monotonically with c_{v}. The effect is uniform across the 24 configurations: ΔV at $c_{\text{v}} = 1$ lies between $+0.00051$ and $+0.00087$ in every one of them (Appendix I), so the average is representative rather than driven by any single configuration.

Appendix J relaxes $T_2 = T_1$: as the correlation ρ of $\log T_1$ and $\log T_2$ falls, ΔV shrinks, vanishing at $\rho = 0$, where our rule reduces to the baseline. Appendix K also plots Experiment 2 as reward against generations with mean latency as colour, and Appendix H gives guidance on choosing $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ in practice.

5 Conclusion

We introduced a two-stage adaptive Best-of-N algorithm that conditions its Stage 2 stopping decision on the wall-clock latency T_1 of the first batch, which serves as a predictor of the second batch's latency, in addition to the prompt-level signal of the observed Stage 1 rewards. A CDF transformation of the rewards yields a closed-form expected-improvement expression and scale-free tradeoff parameters that require no per-task retuning. Empirically, our method improves on parallel Best-of-N at matched compute, and conditioning on T_1 gives a higher user reward, the reward minus the cost of latency, than the latency-unaware variant (a fixed-threshold rule of the kind existing adaptive methods use) at matched numbers of generations, with the gap growing as latency variability increases. Our experiments are simulation-based. An evaluation on a live serving stack with continuous batching (vLLM) would show how well the latency assumptions of Section 2.2 hold – that one batch's latency predicts the next's, that under high load latency depends little on batch size, and that one client's batches barely shift the load others face – and identify the regimes in which our method helps. Extending the framework to fully sequential stopping, and to outcome-based reward models where the identity transformation is more appropriate, is also future work.

AI USE STATEMENT

In this work, we used generative AI tools (large language model coding and writing assistants) extensively. For software, most of the simulation code, analysis scripts, and figure and table scripts were written by AI under our direction; we did not review all of it line by line. For writing, AI drafted and revised most of the manuscript text, including the related work and appendices, and we edited it. AI also assisted in locating related work and preparing bibliography entries, which we checked, with AI assistance, for existence and correct authors, title, and venue.

The research question was developed with AI assistance; the idea of latency adaptivity, the algorithm, and the main-text experiments are the authors' own. The derivations in the appendix were mostly ours, but some of the proofs were conceived by AI, and several of the appendix studies were designed with or by AI. We did not use generative AI to generate or alter experimental data. We take responsibility for the final content of this work.

ETHICS STATEMENT

This work is a simulation study that reuses publicly released prompts, model generations, and reward-model scores from the AlpacaFarm and HH-RLHF datasets (Appendix O); it involves no human subjects, no new data collection, and no release of new model weights or generations. The method allocates inference-time compute and does not change what a language model is able to produce, so it does not introduce new capability risks. Two considerations are worth naming. First, the algorithm optimizes a reward model, and Best-of-N selection inherits whatever biases that reward model carries; our contribution is orthogonal to reward-model quality and does not mitigate such biases. Second, by conditioning on server load, the method makes the amount of compute a request receives depend on when it arrives, so response quality can vary across users for reasons unrelated to their request; a deployment that cares about equal treatment should account for this. The externality of one request's continuation decision on other users' latency is assumed negligible in the high-load regime of Section 2.2; testing this on a live serving stack is future work (Section 5).

REPRODUCIBILITY STATEMENT

All of our results are simulations over precomputed reward pools, and are fully reproducible on CPU. The algorithm and the closed-form quantities it uses are specified in Sections 3 and 3.2, with the derivation of the Q-functions and the optimal policy in Appendix A. The CDF estimator and its baselines are specified in Appendix D. The experimental setup – datasets, language models, reward models, prompt and pool sizes, the $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ grid, the number of Monte Carlo trials, and the latency model – is given in Section 4.1, and the per-configuration results behind the averages reported in the main text are in Appendix I. The datasets and the released reward scores we build on are documented in Appendix O; compute requirements are in Appendix P. Our simulation code, including the scripts that regenerate every figure and table in this paper from cached results, is submitted as supplementary material.

ACKNOWLEDGMENTS

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396, 2023.

Mohammad Ali Alomrani, Yingxue Zhang, Derek Li, Qianyi Sun, Soumyasundar Pal, Zhanguang Zhang, Yaochen Hu, Rohan Deepak Ajwani, Antonios Valkanas, Raika Karimi, Peng Cheng, Yunzhou Wang, Pengyi Liao, Hanrui Huang, Bin Wang, Jianye Hao, and Mark Coates. Reasoning on a budget: A survey of adaptive and controllable test-time compute in llms. *CoRR*, abs/2507.02076, 2025. doi: 10.48550/ARXIV.2507.02076. URL https://doi.org/10.48550/arXiv.2507.02076.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Her-

nandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, Benjamin Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback. *CoRR*, abs/2204.05862, 2022. doi: 10.48550/ARXIV.2204.05862. URL https://doi.org/10.48550/arXiv.2204.05862.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of lm computation. *arXiv preprint arXiv:2410.04707*, 2024.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36:30039–30069, 2023.

Shaddin Dughmi, Yusuf Hakan Kalayci, Vasilis Livanos, and Aditya Prasad. Batched pandora's box. *arXiv preprint arXiv:2609.04059*, 2026.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, et al. Efficiently scaling LLM reasoning programs with certaindex. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 8154–8173. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.507. URL https://doi.org/10.18653/v1/2023.emnlp-main.507.

Jenny Y. Huang, Mehul Damani, Yousef El-Kurdi, Ramon Astudillo, and Wei Sun. Latency and token-aware test-time compute. *arXiv preprint arXiv:2509.09864*, 2025.

Yusuf Kalayci, Vinod Raman, and Shaddin Dughmi. Optimal stopping vs best-of-n for inference time optimization. *arXiv preprint arXiv:2510.01394*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *International Conference on Learning Representations*, 2024.

Rohin Manvi, Anikait Singh, and Stefano Ermon. Adaptive inference-time compute: LLMs can predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*, 2024.

Michael Mitzenmacher and Rana Shahout. Queueing, predictions, and large language models: Challenges and open problems. *Stochastic Systems*, 15(3):195–219, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback.

In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Vinod Raman, Hilal Asi, and Satyen Kale. Abon: Adaptive best-of-n alignment. *arXiv preprint arXiv:2505.12050*, 2025.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. Learning to summarize with human feedback. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1f89885d556929e98d3ef9b86448f951-Abstract.html.

Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter Bartlett, and Andrea Zanette. Fast best-of-n decoding via speculative rejection. *Advances in Neural Information Processing Systems*, 37:32630–32652, 2024.

Zhiqing Tang, Size Li, Hanshuai Cui, Zilan Huang, Jianxiong Guo, Tian Wang, Yuan Wu, and Weijia Jia. LASER: Load-aware serving with early-exit for reasoning LLMs at the edge. In *International Conference on Wireless Artificial Intelligent Computing Systems and Applications (WASA)*, 2026. arXiv:2606.31580.

Guangya Wan, Zixin Stephen Xu, Sasa Zorc, Manel Baucells, Mengxuan Hu, Hao Wang, and Sheng Li. BEACON: Bayesian optimal stopping for efficient LLM sampling. *arXiv preprint arXiv:2510.15945*, 2025.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 49890–49920. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/wan24c.html.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. *CoRR*, abs/2503.01422, 2025. doi: 10.48550/ARXIV.2503.01422. URL https://doi.org/10.48550/arXiv.2503.01422.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Martin L. Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–654, 1979. doi: 10.2307/1910412.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaïd Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=eskQMcIbMS.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=VNckp7JEHn.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. In *International Conference on Learning Representations*, 2026.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In Marcos K. Aguilera and Hakim Weatherspoon (eds.), *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, pp. 521–538. USENIX Association, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

A Derivation of the latency-aware and latency-unaware policies

Q-function derivations. The Q-function of generating a second batch of size N_2 in state s_1 follows from the definition of $\widehat{T}_2(s_1, N_2) = \mathbb{E}[T_2 \mid s_1, N_2]$:

$$\begin{aligned} Q_1(s_1, N_2) &= \mathbb{E}[-\lambda_{\text{gen}} N_2 - \lambda_{\text{lat}} T_2 + \max\{F_x(R_1), F_x(R_2)\} \mid s_1,\ N_2] \\ &= -\lambda_{\text{gen}} N_2 - \lambda_{\text{lat}} \widehat{T}_2(s_1, N_2) + \mathbb{E}[\max\{F_x(R_1), F_x(R_2)\} \mid s_1,\ N_2]. \end{aligned}$$

The profit-to-go $Q_0^\pi(s_0, N_1)$ follows from the Bellman equation and the definition of $\overline{T}_1(N_1)$:

$$\begin{aligned} Q_0^\pi(s_0, N_1) &= -\lambda_{\text{gen}} N_1 - \lambda_{\text{lat}}\, \mathbb{E}[T_1 \mid s_0, N_1] + \mathbb{E}[Q_1(s_1, \pi_1(s_1)) \mid s_0,\ N_1] \\ &= -\lambda_{\text{gen}} N_1 - \lambda_{\text{lat}} \overline{T}_1(N_1) + \mathbb{E}[Q_1(s_1, \pi_1(s_1)) \mid s_0,\ N_1]. \end{aligned}$$

Optimal N_1 selection. The optimal first batch size is $N_1 \in \arg\max_{N_1 \in \mathcal{Z}_1} Q_0^\pi(s_0, N_1)$. For the latency-aware policy:

$$Q_0^\star(s_0, N_1) = -\lambda_{\text{gen}} N_1 - \lambda_{\text{lat}} \overline{T}_1(N_1) + \mathbb{E}[V_1^\star(s_1) \mid s_0,\ N_1],$$

where

$$V_1^\star(s_1) = \max\left\{F_x(R_1),\ \max_{N_2 \in \mathcal{Z}_2} Q_1(s_1, N_2)\right\} = F_x(R_1) + \left[\max_{N_2 \in \mathcal{Z}_2} f_1(s_1, N_2)\right]_+.$$

For the latency-unaware policy π_{lu}:

$$Q_0^{\pi_{\text{lu}}}(s_0, N_1) = -\lambda_{\text{gen}} N_1 - \lambda_{\text{lat}} \overline{T}_1(N_1) + \mathbb{E}[Q_1(s_1, \pi_{\text{lu}}(s_1)) \mid s_0,\ N_1],$$

where $\pi_{\text{lu}}(s_1) \in \arg\max_{a_1 \in \{\text{stop}\} \cup \mathcal{Z}_2} \widehat{Q}_1(s_1, a_1)$, with $\widehat{Q}_1(s_1, a_1) = F_x(R_1) + \widehat{f}_1(s_1, a_1)$ for $a_1 \in \mathcal{Z}_2$ and $\widehat{Q}_1(s_1, \text{stop}) = F_x(R_1)$.

Here $Q_0^{\pi_{\text{lu}}}$ is written in terms of the true Q_1: the policy is evaluated at the second-batch latency it actually incurs, though it budgeted $\overline{T}_2$ for it. We show that it agrees in expectation with $\widehat{Q}_1$ under the following condition on the latency model:

$$\mathbb{E}\big[T_2 \;\big|\; s_0,\, N_1,\, N_2,\, r_1^{(1)}, \ldots, r_{N_1}^{(1)}\big] = \overline{T}_2, \qquad N_1 \in \mathcal{Z}_1,\; N_2 \in \mathcal{Z}_2. \tag{1}$$

Of the three independences in (1), that from N_1 is the high-load regime of Section 2.2: the first batch completes before the second is submitted, so it can affect T_2 only through the state it leaves the server in, and one batch does not perturb that state. That from N_2 does not follow from the high-load regime, since a batch returns only when its slowest response does; it is vacuous in the non-adaptive variant, where $\mathcal{Z}_2$ is a singleton. That from the rewards is a separate condition, response length being correlated with both reward and latency; it still leaves T_2 free to depend on T_1, which is the dependence the latency-aware policy uses. All three hold in our experiments, where latency is drawn independently of the batch sizes and of the rewards, and from the same distribution for every prompt.

Since $Q_1(s_1, \text{stop}) = \widehat{Q}_1(s_1, \text{stop}) = F_x(R_1)$ and $Q_1(s_1, N_2) - \widehat{Q}_1(s_1, N_2) = -\lambda_{\text{lat}}\big(\widehat{T}_2(s_1, N_2) - \overline{T}_2\big)$ for $N_2 \in \mathcal{Z}_2$, the law of total expectation, splitting on the value of $\pi_{\text{lu}}(s_1)$, gives

$$\begin{aligned}
&\mathbb{E}\big[Q_1(s_1, \pi_{\text{lu}}(s_1)) - \widehat{Q}_1(s_1, \pi_{\text{lu}}(s_1)) \;\big|\; s_0, N_1\big] \\
&\quad= \Pr\big[\pi_{\text{lu}}(s_1) = \text{stop} \;\big|\; s_0, N_1\big]\; \mathbb{E}\big[Q_1(s_1, \text{stop}) - \widehat{Q}_1(s_1, \text{stop}) \;\big|\; \pi_{\text{lu}}(s_1) = \text{stop},\, s_0, N_1\big] \\
&\qquad+ \sum_{N_2 \in \mathcal{Z}_2} \Pr\big[\pi_{\text{lu}}(s_1) = N_2 \;\big|\; s_0, N_1\big]\; \mathbb{E}\big[Q_1(s_1, N_2) - \widehat{Q}_1(s_1, N_2) \;\big|\; \pi_{\text{lu}}(s_1) = N_2,\, s_0, N_1\big] \\
&\quad= -\lambda_{\text{lat}} \sum_{N_2 \in \mathcal{Z}_2} \Pr\big[\pi_{\text{lu}}(s_1) = N_2 \;\big|\; s_0, N_1\big]\; \mathbb{E}\big[\widehat{T}_2(s_1, N_2) - \overline{T}_2 \;\big|\; \pi_{\text{lu}}(s_1) = N_2,\, s_0, N_1\big] \\
&\quad= -\lambda_{\text{lat}} \sum_{N_2 \in \mathcal{Z}_2} \Pr\big[\pi_{\text{lu}}(s_1) = N_2 \;\big|\; s_0, N_1\big] \Big(\mathbb{E}\big[\mathbb{E}[T_2 \mid s_0, r_{1:N_1}^{(1)}, N_1, T_1, N_2] \;\big|\; \pi_{\text{lu}}(s_1) = N_2,\, s_0, N_1\big] - \overline{T}_2\Big) \\
&\quad= -\lambda_{\text{lat}} \sum_{N_2 \in \mathcal{Z}_2} \Pr\big[\pi_{\text{lu}}(s_1) = N_2 \;\big|\; s_0, N_1\big] \Big(\mathbb{E}\big[T_2 \;\big|\; \pi_{\text{lu}}(s_1) = N_2,\, s_0, N_1\big] - \overline{T}_2\Big) \;=\; 0.
\end{aligned}$$

Every conditional expectation above is taken on an event on which a second batch is in fact generated, so each T_2 is well defined. The fourth equality is the tower property, the event $\{\pi_{\text{lu}}(s_1) = N_2\}$ being determined by s_1. The fifth uses (1). Conditioning on $\{\pi_{\text{lu}}(s_1) = N_2\}$ is not the same as conditioning on the second batch having size N_2: $\widehat{f}_1$ does not involve T_1, so the event is a statement about the first-batch rewards, and conditioning on it reweights anything correlated with them. Condition (1) is what rules this out, holding as it does at every value of the rewards; averaging it over those values for which $\pi_{\text{lu}}(s_1) = N_2$ therefore returns the same constant,

$$\mathbb{E}\big[T_2 \;\big|\; \pi_{\text{lu}}(s_1) = N_2,\, s_0, N_1\big] = \mathbb{E}\Big[\mathbb{E}\big[T_2 \;\big|\; s_0, N_1, N_2, r_1^{(1)}, \ldots, r_{N_1}^{(1)}\big] \;\Big|\; \pi_{\text{lu}}(s_1) = N_2,\, s_0, N_1\Big] = \overline{T}_2.$$

Under (1),

$$Q_0^{\pi_{\text{lu}}}(s_0, N_1) = -\lambda_{\text{gen}} N_1 - \lambda_{\text{lat}} \overline{T}_1(N_1) + \mathbb{E}[\widehat{V}_1(s_1) \mid s_0, N_1]$$

where $\widehat{V}_1(s_1) \coloneqq F_x(R_1) + [\max_{N_2 \in \mathcal{Z}_2} \widehat{f}_1(s_1, N_2)]_+$; the two policies' Q-functions ($Q_0^\star$ and $Q_0^{\pi_{\text{lu}}}$) then have identical structure, differing only in whether the second-batch latency is $\widehat{T}_2(s_1, N_2)$ (latency-aware) or $\overline{T}_2$ (latency-unaware). Otherwise $Q_0^{\pi_{\text{lu}}}$ must be evaluated as written, with the true Q_1 inside.

Computing $Q_0^\star(s_0, N_1)$ and $Q_0^{\pi_{\text{lu}}}(s_0, N_1)$ under the CDF transformation. These quantities are computed once to determine the fixed initial batch size N_1. Under the CDF transformation, both policies admit the decomposition

$$Q_0^\pi(s_0, N_1) \;=\; -\lambda_{\text{gen}} N_1 - \lambda_{\text{lat}} \overline{T}_1(N_1) + \frac{N_1}{N_1 + 1} + W^\pi_{N_1, \mathcal{Z}_2}, \qquad \pi \in \{\pi^\star, \pi_{\text{lu}}\},$$

where $\frac{N_1}{N_1+1} = \mathbb{E}[F_x(R_1) \mid s_0, N_1] = \Delta(0, N_1)$ is the expected maximum of N_1 i.i.d. $\text{Unif}[0, 1]$ rewards, and

$$W^\star_{N_1, \mathcal{Z}_2} \coloneqq \mathbb{E}\left[\left[\max_{N_2 \in \mathcal{Z}_2} f_1(s_1, N_2)\right]_+ \;\middle|\; s_0, N_1\right], \quad W^{\pi_{\text{lu}}}_{N_1, \mathcal{Z}_2} \coloneqq \mathbb{E}\left[\left[\max_{N_2 \in \mathcal{Z}_2} \widehat{f}_1(s_1, N_2)\right]_+ \;\middle|\; s_0, N_1\right].$$

To compute these we specialize to the case simulated in Section 4, in which the second batch's expected latency depends on s_1 only through T_1 and not on N_2, so that we may write it $\widehat{T}_2(T_1)$, and in which T_1 is independent of the first-batch rewards. Writing p for the marginal distribution of T_1 and $u_1 \coloneqq F_x(R_1)$, which has density $N_1 u^{N_1-1}$ on $[0,1]$, we may then unify the computation via $G : \mathbb{R}_+ \times \mathbb{N} \to \mathbb{R}$:

$$G(\ell, N_1) \;\coloneqq\; \int_0^1 \left[\max_{N_2 \in \mathcal{Z}_2} \{-\lambda_{\text{gen}} N_2 - \lambda_{\text{lat}} \ell + \Delta(u, N_2)\} \right]_+ N_1 u^{N_1-1}\, du.$$

The two policies correspond to $\ell = \overline{T}_2$ (latency-unaware: $W^{\pi_{\text{lu}}}_{N_1,\mathcal{Z}_2} = G(\overline{T}_2, N_1)$, evaluated by quadrature) and $\ell = \widehat{T}_2(T_1)$ (latency-aware: $W^{\star}_{N_1,\mathcal{Z}_2} = \mathbb{E}_{T_1 \sim p}[G(\widehat{T}_2(T_1), N_1)]$, evaluated by Monte Carlo over samples from p); in our experiments $\widehat{T}_2(T_1) = T_1$.

If T_1 is not independent of the first-batch rewards, or if $\widehat{T}_2$ depends on N_2, this factorization does not apply; one may then either model the joint law of $(\widehat{T}_2(s_1, N_2), u_1)$ and integrate numerically over both, or retain the factorization as an approximation. Dependence of this kind is plausible in deployment, since response length correlates with both reward and latency. Either way, only the offline choice of N_1 is affected: the online stopping rule of Section 3 reads T_1 off the clock and requires no latency model.

The latency-unaware policy is a fixed-threshold rule. For every N_2, $\Delta(u, N_2)$ is strictly decreasing in u on $[0,1)$, since $\partial_u \Delta(u, N_2) = -1 + u^{N_2} < 0$, and it falls from $N_2/(N_2+1)$ at $u = 0$ to 0 at $u = 1$. Hence $\max_{N_2 \in \mathcal{Z}_2} \widehat{f}_1(s_1, N_2)$ is strictly decreasing in $u_1 = \hat{F}_x(R_1)$. The latency-unaware policy therefore continues if and only if $u_1 < u_c$, for a threshold $u_c \in [0,1)$ that depends only on λ_{gen}, λ_{lat}, $\overline{T}_2$, and $\mathcal{Z}_2$, all fixed before any request arrives. In the non-adaptive variant, u_c solves $\Delta(u_c, N_2) = \lambda_{\text{gen}} N_2 + \lambda_{\text{lat}} \overline{T}_2$. This is the fixed-threshold stopping rule of existing adaptive methods (Section 1.1), applied to the estimated quantile of the best first-batch reward. The latency-aware policy has the same form with $\overline{T}_2$ replaced by $\widehat{T}_2(s_1, N_2)$, so its threshold moves from request to request; with $\widehat{T}_2(s_1, N_2) = T_1$, it falls as T_1 grows. Every $u_c \in (0,1)$ is attained by some $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$. So, up to the range and resolution of the grid, sweeping the tradeoff parameters as in Section 4.1 traces the latency-unaware policy over all fixed thresholds on this signal.

B Measuring response quality: the reward quantile

We score a returned response by its *reward quantile* within its prompt's pool. Let k be the rank of reward r among the $M = 960$ rewards of $\mathcal{R}_b$, with tied rewards given their average rank; the reward quantile is $k/(M+1)$. The oracle policies use this quantity in place of F_{x_b}.

The reward quantile is an unbiased estimate of the true quantile $F_{x_b}(r)$. The pool is an i.i.d. sample from F_{x_b}, so F_{x_b} evaluated at its k-th smallest reward is distributed as the k-th order statistic of M i.i.d. $\text{Unif}[0,1]$ variables, whose mean is $k/(M+1)$. For Best-of-N and the oracle policies, which select by rank alone, the mean reward quantile is therefore an unbiased estimate of $\mathbb{E}[F_x(R)]$, the reward term of the objective V; for the estimated policies, which also use reward values, it is unbiased to close approximation. Normalizing ranks to span $[0,1]$ instead, as $(k-1)/(M-1)$, would overstate the quantile of the pool's best response by $1/(M+1)$.

The empirical Best-of-N curve in the figures and tables is computed exactly rather than by simulation. Sort a prompt's reward quantiles as $q_{(1)} \le \cdots \le q_{(M)}$. Best-of-$N$ draws N of the M responses without replacement and returns the best, and the best is the k-th smallest exactly when the draw contains response k and $N-1$ of the $k-1$ responses below it. Its mean reward quantile is therefore

$$\mathbb{E}[\text{Best-of-}N] = \sum_{k=N}^{M} q_{(k)} \binom{k-1}{N-1} \Big/ \binom{M}{N},$$

averaged over the prompts; tied rewards are simply equal entries of the sorted list.

The Best-of-N theory curve $N/(N+1)$ follows directly. Without ties $q_{(k)} = k/(M+1)$, so the sum is $\mathbb{E}[K]/(M+1)$ for the maximum rank K of the draw, and $\mathbb{E}[K] = N(M+1)/(N+1)$: the

mean reward quantile is exactly $N/(N+1)$, for any pool size M. The rewards, however, are stored at bfloat16 precision, so a prompt's pool holds only 6–40% distinct values on average, depending on the configuration. Average ranks are the mean of the scores over all tie-breaks, each of which gives $N/(N+1)$; since the maximum is convex in the scores, ties can only lower the expected Best-of-N reward quantile. The effect is small: for $N \geq 80$ the empirical Best-of-N curve lies below $N/(N+1)$ by about 0.001 at most in any configuration, and by at most 0.0003 in the 24-configuration average plotted in the figures.

C Fully sequential mean reward upper bound

In the plots, the dashed green curve $1 - 1/(2N)$ is the mean maximum reward of the optimal *fully sequential* (one-at-a-time) stopping policy on CDF-transformed rewards, plotted against its expected number of generations N. This is the limiting case of the multi-stage extension of Section 3 in which $\mathcal{Z}_j = \{1\}$ for every stage $j \geq 1$ and the horizon is unbounded – i.e. the Pandora's Box problem with infinitely many i.i.d. boxes considered by Kalayci et al. (2025). Since our two-stage Best-of-N setting is a constrained version of this problem, no two-stage policy can exceed this curve.

For a fixed prompt x, write $v_i = F_x(r_i) \sim \mathrm{Unif}[0,1]$ for the CDF-transformed reward of generation i. With per-generation cost $c \in (0, 1/2)$, the optimal policy $\pi^\star$ is Weitzman's index policy (Weitzman, 1979) with fair-cap value τ satisfying $\mathbb{E}_{v\sim\mathrm{Unif}[0,1]}[(v-\tau)_+] = c$, i.e. $\tau = 1 - \sqrt{2c}$. We can easily find that, in general, the optimal profit equals the fair-cap value,

$$V_{\pi^\star,x} \;=\; \mathbb{E}\Big[\max_{i \leq T_{\pi^\star,x}} v_i - c\,T_{\pi^\star,x}\Big] \;=\; \tau \;=\; 1 - \sqrt{2c},$$

and $T_{\pi^\star,x} \sim \mathrm{Geo}(\sqrt{2c})$, so $\mathbb{E}[T_{\pi^\star,x}] = 1/\sqrt{2c}$. Rearranging,

$$\mathbb{E}\Big[\max_{i \leq T_{\pi^\star,x}} v_i\Big] \;=\; \tau + c\,\mathbb{E}[T_{\pi^\star,x}] \;=\; (1 - \sqrt{2c}) + \frac{c}{\sqrt{2c}} \;=\; 1 - \frac{\sqrt{2c}}{2}.$$

Substituting $\sqrt{2c} = 1/\,\mathbb{E}[T_{\pi^\star,x}]$ and writing $N = \mathbb{E}[T_{\pi^\star,x}]$:

$$\mathbb{E}\Big[\max_{i \leq T_{\pi^\star,x}} v_i\Big] \;=\; 1 - \frac{1}{2N}.$$

We include this curve in the plots as a *reward-only* reference: it represents the most reward-adaptive policy possible – evaluating generations one at a time and deciding after each whether to continue – and so upper-bounds the mean reward attainable by any policy with the same expected number of generations N, including any two-stage policy considered in this paper. This reference says nothing about latency: a fully sequential policy that draws N generations one at a time incurs roughly N times the latency of a single batch of size N, which is precisely the regime our two-stage formulation is designed to avoid. The curve is therefore informative as a ceiling on the reward axis only, and the latency cost of actually attaining it is not reflected in the plot.

D Comparing $F_x(R_1)$ estimation methods

The shifted-exponential estimator described in Section 3 is justified by the following empirical comparison. We evaluate six candidate estimators of $F_x(R_1)$, where $R_1 = \max_i r_i^{(1)}$ is the maximum of the N_1 Stage 1 rewards:

- *In-sample KDE.* Gaussian kernel density estimator $\hat{F}_x(r) = \frac{1}{N_1}\sum_{i=1}^{N_1} \Phi\big((r - r_i^{(1)})/h\big)$ with bandwidth $h = \hat{\sigma}\, N_1^{-1/5}$ (Scott's rule), evaluated at $r = R_1$ *using the same sample* that defined the bandwidth and the kernel centers. Following Raman et al. (2025), we treat the in-sample KDE variant as the canonical KDE baseline.
- *Leave-one-out (LOO) KDE,* $\times 0.5$ *and* $\times 1.0$. Same as in-sample KDE, but the term corresponding to R_1 itself (which contributes a fixed $\Phi(0) = 1/2$) is dropped from the sum and the denominator becomes $N_1 - 1$. The two variants differ in the bandwidth multiplier ($\texttt{bw_mult} = 0.5$ and $\texttt{bw_mult} = 1.0$), and we set $h = \texttt{bw_mult}\,\hat{\sigma}\, N_1^{-1/5}$.

- *Normal.* A Gaussian distribution is fit to the Stage 1 rewards by maximum likelihood, and $\hat{F}_x(R_1) = \Phi\big((R_1 - \hat{\mu})/\hat{\sigma}\big)$.
- *GMM-2.* A two-component Gaussian mixture model (GMM) is fit to all N_1 Stage 1 rewards by maximum likelihood using the expectation–maximization (EM) algorithm. EM is initialised by splitting the sorted sample at its median: each half's mean and standard deviation start one component, with equal weights. It then runs a fixed 30 iterations, alternating between computing each reward's posterior probability of belonging to each component (E-step) and re-estimating the weights w_k, means $\hat{\mu}_k$, and standard deviations $\hat{\sigma}_k$ from those probabilities (M-step). The fitted mixture CDF is evaluated at the maximum, $\hat{F}_x(R_1) = \sum_{k=1}^{2} w_k \, \Phi\big((R_1 - \hat{\mu}_k)/\hat{\sigma}_k\big)$. Unlike the Normal fit, the mixture can represent skewed or bimodal reward distributions.
- *Shifted exponential.* Our proposed estimator (Section 3.2), fitted to the upper tail of the exponentiated Stage-1 rewards following Kalayci et al. (2025). Concretely, fix a tail level $\alpha \in (0, 1)$ (we use $\alpha = 1/2$), let $s > 0$ be the reward scale of Appendix E, let $\hat{q}_\alpha$ denote the empirical α-quantile of the rescaled rewards $r_1^{(1)}/s, \ldots, r_{N_1}^{(1)}/s$, and let $\theta = e^{\hat{q}_\alpha}$. Among the rewards above $\hat{q}_\alpha$, we fit a one-parameter exponential to the exponentiated excesses:

$$\hat{\mu} \;=\; \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \Big(e^{r_i^{(1)}/s} - \theta \Big), \qquad \mathcal{I} = \{ i : r_i^{(1)}/s > \hat{q}_\alpha \},$$

  and define the estimator

$$\hat{F}_x(r) \;=\; \alpha + (1 - \alpha) \left(1 - \exp\!\left(- \frac{e^{r/s} - \theta}{\hat{\mu}} \right) \right). \tag{2}$$

Evaluation setup. We run this evaluation separately in each of the 24 configurations of Section 4.1, each with 100 prompts, and average the results over the 24. For each prompt x_b, $b = 1, \ldots, B$, we partition the 960 precomputed rewards into a Stage-1 *training pool* of size 480 and a held-out pool of size 480. For each estimator and each value of $N_1 \in \{15, 30, 60, 120, 240\}$, we run $M = 200$ independent Monte Carlo trials. In trial m, we draw a Stage-1 set of size N_1 without replacement from the training pool of each prompt, compute the per-prompt Stage-1 maximum $R_{1,b}^{(m)}$, evaluate the estimator $\hat{F}_{x_b}(R_{1,b}^{(m)})$, and compare it to the held-out empirical CDF value

$$\tilde{F}_{x_b}(R_{1,b}^{(m)}) \;\coloneqq\; \frac{|\{ r \in \mathcal{H}_{x_b} : r \le R_{1,b}^{(m)} \}|}{|\mathcal{H}_{x_b}|},$$

where $\mathcal{H}_{x_b}$ is the held-out reward pool for prompt x_b. Splitting the rewards into disjoint training and held-out pools ensures that the held-out empirical CDF is an independent ground-truth estimate; in particular, a Stage-1 set of size $N_1 = 480$ drawn from the 480-reward training pool would be deterministic, eliminating the possibility of MC variation, which is why we cap the N_1 value at 240.

Figure 3 reports RMSE and bias as a function of N_1 (point estimates, without confidence bands, for the same reason as discussed in Section 4.1), while Table 3 reports the RMSE in tabular form. Both are averaged over the 24 configurations.

The shifted-exponential estimator has the lowest RMSE at every N_1 but 120, where the two-component Gaussian mixture is ahead by 0.0003, and is level with that mixture at $N_1 = 60$. Its margin is largest at small N_1. Its bias is positive throughout (Table 3), so it overstates $F_x(R_1)$ and errs toward stopping; the bias is smaller than the Normal fit's at every $N_1 \ge 30$ and comparable to the Gaussian mixture's at $N_1 = 240$. Appendix F removes that bias with an offline calibration map, after which the estimator has the lowest RMSE at every N_1 (dashed black in Figure 3).

We use the shifted-exponential estimator throughout the main experiments on the strength of that accuracy, of the better stopping decisions it leads to (below), of its single closed form, and of its having no fitting procedure that can fail to converge. It is fitted to the exponentiated reward, and so depends on the scale of the reward; Appendix E shows why and describes the rescaling we apply before fitting.

Table 3: RMSE and bias of $\hat{F}_x(R_1)$ against the held-out empirical CDF value for the six estimators considered, as a function of N_1, averaged over all 24 combinations of four language models, three reward models, and two datasets. Held-out rewards tied with R_1 count as half. Positive bias: the estimator overstates $F_x(R_1)$. Averaged over 100 prompts, $M = 200$ Monte Carlo trials, and the 24 configurations; best of the six per column in bold, both entries where two tie. The last row of each panel is our estimator after the calibration of Appendix F, whose RMSE is lower than all six at every N_1 and whose bias is at most 0.0002. Per-configuration values are in Table 4.

RMSE	$N_1 = 15$	30	60	120	240
In-sample KDE	0.0625	0.0398	0.0261	0.0173	0.0115
LOO KDE ($\times 0.5$)	0.0665	0.0344	0.0184	0.0104	0.0063
LOO KDE ($\times 1.0$)	0.0589	0.0353	0.0233	0.0158	0.0108
Normal	0.0601	0.0428	0.0359	0.0316	0.0279
GMM-2	0.0636	0.0315	**0.0156**	**0.0087**	0.0060
Shifted exponential (ours)	**0.0558**	**0.0296**	**0.0156**	0.0090	**0.0057**
Shifted exponential + calibration (Appendix F)	0.0527	0.0271	0.0142	0.0078	0.0047
bias	$N_1 = 15$	30	60	120	240
In-sample KDE	-0.0243	-0.0200	-0.0137	-0.0084	-0.0048
LOO KDE ($\times 0.5$)	$+0.0374$	$+0.0175$	$+0.0082$	$+0.0039$	$+0.0019$
LOO KDE ($\times 1.0$)	$+0.0052$	-0.0046	-0.0057	-0.0043	-0.0027
Normal	-0.0038	-0.0140	-0.0167	-0.0160	-0.0140
GMM-2	$+0.0301$	$+0.0116$	$+0.0030$	-0.0005	-0.0017
Shifted exponential (ours)	$+0.0192$	$+0.0119$	$+0.0058$	$+0.0028$	$+0.0014$
Shifted exponential + calibration (Appendix F)	$+0.0002$	$+0.0002$	$+0.0001$	$+0.0001$	$+0.0000$

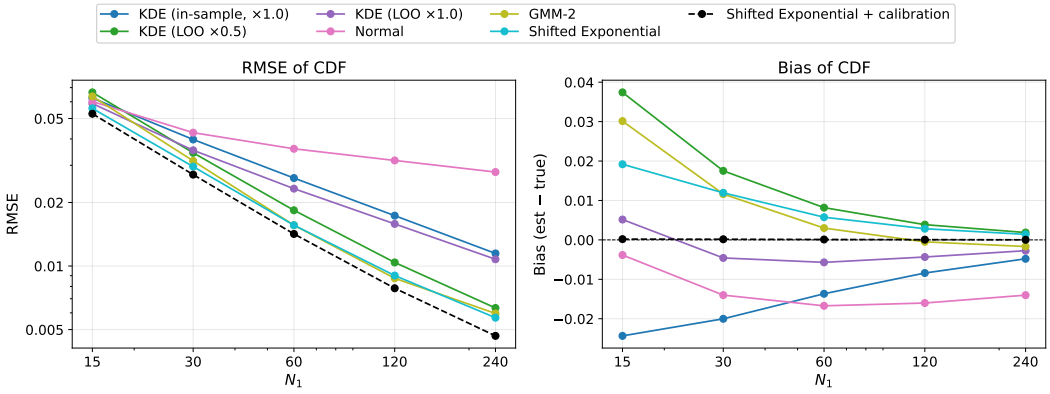


Figure 3: RMSE (left) and bias (right) of six candidate estimators of $F_x(R_1)$ as a function of the Stage-1 batch size N_1, averaged over the 24 language model, reward model, and dataset configurations of Appendix I. *Our proposed shifted-exponential estimator* has the lowest RMSE of the six at every N_1 but 120, where the two-component Gaussian mixture is ahead by 0.0003, and is level with the mixture at $N_1 = 60$. Its bias is positive throughout, smaller than the Normal fit's at $N_1 \geq 30$, and comparable to the mixture's at $N_1 = 240$. The other five (in-sample KDE – the canonical baseline of Raman et al. (2025) – two leave-one-out KDE variants, a Normal fit, and that mixture) are the baselines it is compared against. We use the shifted-exponential estimator throughout the main experiments; Figure 4 compares it with the mixture inside the policy. Dashed black: the shifted-exponential estimate after the calibration of Appendix F, which has the lowest RMSE at every N_1 and a bias of at most 0.0002.

Across language models, reward models, and datasets. Table 4 repeats this comparison in each of the 24 configurations of Appendix I, reporting our estimator's RMSE as a ratio to that of the best competing estimator in each configuration. The ranking is not uniform. The shifted-exponential estimator has the lowest RMSE in 21 of the 24 configurations at $N_1 = 15$ and at $N_1 = 30$, and in 14 at $N_1 = 60$; at $N_1 \geq 120$ it is overtaken in most, by the Gaussian mixture or the narrower leave-

Table 4: RMSE of $\hat{F}_x(R_1)$ for the shifted-exponential estimator divided by that of the best of the five competing estimators, in each of the 24 combinations of four language models, three reward models, and two datasets. Below one, ours is the most accurate; bold marks those entries. Averaged over 100 prompts and $M = 200$ Monte Carlo trials. Our estimator has the lowest RMSE in 21, 21, 14, 6, 7 of the 24 configurations at $N_1 = 15, 30, 60, 120, 240$ respectively.

dataset	RM	LM	$N_1 = 15$	30	60	120	240
AlpacaFarm	FsfairX	Llama-3.1-8B	**0.905**	**0.945**	**0.988**	**0.959**	1.020
		Gemma-2-9B	**0.979**	1.024	1.039	1.013	**0.968**
		Mistral-7B	**0.999**	**0.946**	1.017	1.076	1.028
		Qwen-2.5-7B	**0.918**	**0.943**	**0.999**	**0.963**	**0.993**
	RM-Mistral-7B	Llama-3.1-8B	**0.932**	**0.937**	1.005	1.068	1.103
		Gemma-2-9B	**0.999**	**0.969**	1.050	1.162	1.252
		Mistral-7B	1.004	**0.959**	1.044	1.146	1.136
		Qwen-2.5-7B	**0.936**	**0.942**	1.012	1.052	1.071
	ArmoRM	Llama-3.1-8B	**0.953**	**0.916**	**0.971**	**0.984**	1.027
		Gemma-2-9B	**0.948**	**0.918**	**0.997**	1.118	1.362
		Mistral-7B	**0.925**	**0.930**	**0.990**	1.001	**0.901**
		Qwen-2.5-7B	**0.875**	**0.928**	**0.993**	1.039	1.377
HH-RLHF	FsfairX	Llama-3.1-8B	**0.976**	**0.966**	1.052	1.165	1.470
		Gemma-2-9B	**0.960**	**0.939**	1.004	1.057	1.003
		Mistral-7B	1.019	1.029	1.001	1.078	1.084
		Qwen-2.5-7B	**0.923**	**0.946**	**0.966**	**0.899**	**0.974**
	RM-Mistral-7B	Llama-3.1-8B	**0.968**	**0.922**	**0.974**	1.016	1.002
		Gemma-2-9B	**0.976**	**0.932**	**0.998**	1.075	1.045
		Mistral-7B	1.034	1.015	1.042	1.167	1.250
		Qwen-2.5-7B	**0.953**	**0.924**	**0.956**	**0.928**	**0.904**
	ArmoRM	Llama-3.1-8B	**0.936**	**0.934**	**0.996**	1.036	1.175
		Gemma-2-9B	**0.945**	**0.901**	**0.962**	1.027	**0.964**
		Mistral-7B	**0.929**	**0.908**	**0.947**	**0.950**	**0.858**
		Qwen-2.5-7B	**0.880**	**0.937**	**0.979**	1.044	1.232

one-out KDE. The four configurations where it trails at small N_1 are Gemma-2-9B with FsfairX on AlpacaFarm, and three of the four Mistral-7B configurations that do not use ArmoRM. The practical reading is that the shifted-exponential fit is the right default in the small-N_1 regime our method operates in, but that once N_1 grows a more flexible estimator estimates $F_x(R_1)$ more accurately. As the next paragraph shows, that is not enough to make it the better estimator for the stopping rule.

Inside the stopping rule. RMSE measures the estimate of $F_x(R_1)$, not the decisions made from it. The Gaussian mixture is the only competitor whose RMSE matches or beats ours at any N_1 – level at $N_1 = 60$ and ahead at $N_1 = 120$ – so we re-ran Experiment 1 (Section 4.2) in all 24 configurations with the mixture in place of the shifted exponential, leaving everything else unchanged. Figure 4 overlays the two frontiers and Table 5 reads them at $1.5N_1$ mean generations, as Table 1 does. With the mixture, the policy recovers 9%, 25%, and 35% of the Best-of-N-to-oracle gap at $N_1 = 20$, 40, and 80, against 38%, 47%, and 46% with our estimator, and our estimator is ahead in every one of the 24 configurations at $N_1 \leq 40$ and in 23 at $N_1 = 80$. At $N_1 = 160$ the two are level (37% against 38%), and in no configuration do they differ by more than 0.00013, about the 0.0001 Monte Carlo resolution of the runs. So in the range where the two estimators' RMSEs are close or the mixture's is lower, the shifted exponential still gives the better policy or an equal one; accuracy in estimating $F_x(R_1)$ is not by itself a sufficient guide to which estimator makes the better stopping decisions.

E Rescaling rewards for the shifted-exponential estimator

Before fitting the shifted-exponential estimator (2), we divide every reward by a single constant, the reward scale

$$s = \mathrm{median}_x \, \mathrm{std}(r \mid x),$$

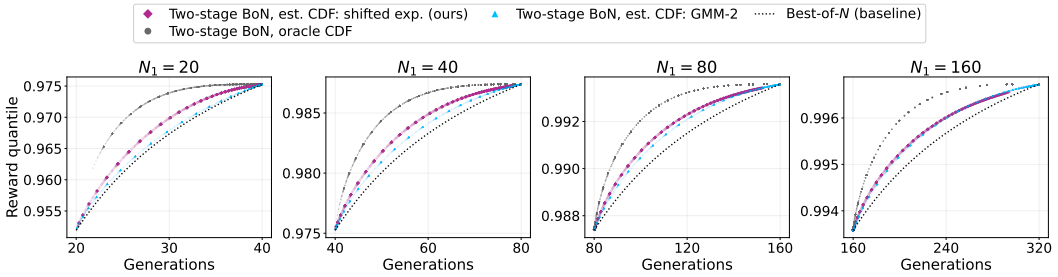


Figure 4: Experiment 1 (Figure 1) with a second CDF estimator: reward–compute Pareto frontiers of our method using the shifted-exponential estimate of F_x (magenta diamonds), using a two-component Gaussian mixture (GMM-2; cyan triangles), and using the oracle F_x (grey circles), against parallel Best-of-N (dotted black), for $N_1 = N_2 \in \{20, 40, 80, 160\}$, averaged over the 24 configurations of Appendix I. Each point is one $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pair. Latency is deterministic, so mean latency is $\overline{T}$ times one plus the fraction of trials that continue, a linear function of mean generations, and is not colour-coded. With GMM-2 the policy recovers less of the gap to the oracle at every N_1 but 160, where the two are level (Table 5), although the two estimators' RMSEs are close (Figure 3).

Table 5: Experiment 1 with the two-component Gaussian mixture (GMM-2) in place of our shifted-exponential estimator: mean reward quantile at a matched budget of $1.5N_1$ mean generations, read off the frontiers of Figure 4 as in Table 1; the fraction of the Best-of-N-to-oracle gap each estimator recovers; and the number of the 24 configurations in which ours is ahead of GMM-2 at that budget.

	mean reward quantile				gap recovered		
N_1	parallel BoN	GMM-2	ours	oracle F_x	GMM-2	ours	ours ahead
20	0.96733	0.96793	0.96982	0.97388	9%	38%	24/24
40	0.98330	0.98416	0.98489	0.98669	25%	47%	24/24
80	0.99150	0.99210	0.99229	0.99323	35%	46%	23/24
160	0.99567	0.99600	0.99601	0.99655	37%	38%	14/24

so that the median within-prompt standard deviation is 1. The scale is estimated once offline, from a sample of prompts with several scored responses each: take each prompt's sample standard deviation, then the median across prompts. In our experiments each configuration uses the s computed from its full pool of 100 prompts and 960 responses, listed in Table 6. Only the CDF estimate is affected – the reported reward and the oracle policy are rank transforms of the pool, which any rescaling leaves unchanged.

Why rescaling is needed. The estimator is fitted to the *exponentiated* reward, so without the rescaling ($s = 1$ in (2)) its fitted tail shape depends on the units in which rewards are reported, and not only on the shape of the reward distribution. Writing a first-batch reward as $r = \hat{q}_\alpha + sz$ with z of order one, the quantity the unrescaled fit treats as exponential is $e^r - \theta = \theta(e^{sz} - 1)$. When $s \approx 1$ this grows faster than linearly in z, and the implied upper tail of r decays doubly exponentially. When $s \ll 1$, $e^{sz} - 1 \approx sz$ and the fit degenerates to an exponential tail on r itself, which is far heavier than the reward distributions we observe; $\hat{F}_x(R_1)$ is then biased low, $\Delta(\hat{F}_x(R_1), N_2)$ is overstated, and the policy continues too often. Dividing by s replaces e^{sz} by e^z, whatever the reward model's units.

Table 6 shows that this matters in practice. FsfairX and RM-Mistral-7B report Bradley–Terry logits, and in the 16 configurations that use them s lies between 0.63 and 0.97: their rewards already have a median within-prompt standard deviation close to 1. Rescaling them changes little – RMSE moves by at most 0.0024 and the latency-aware policy's objective V by at most 0.0004 – so for reward models like these the offline estimate of s can be skipped and the estimator fitted to the rewards without rescaling, which makes the method simpler. We nonetheless rescale in every configuration, so that all 24 are treated by the same procedure. For these two reward models the unrescaled fit is in fact slightly the more accurate one in most configurations, as Table 7 predicts: fitting unrescaled

Table 6: The reward scale s of each of the 24 combinations of four language models, three reward models, and two datasets, and the effect of fitting the shifted-exponential estimator to the raw rewards r instead of the rescaled rewards r/s. $e_{N_1}(\cdot)$ is the RMSE of $\hat{F}_x(R_1)$ at first-batch size N_1 when fitted to those rewards, measured as in Table 7; ΔV is the change in the latency-aware policy's objective when the rewards are rescaled. s is the median within-prompt standard deviation over the full pool of 100 prompts and 960 responses. ΔV: non-adaptive variant, $N_1 = N_2 = 80$, log-normal latency with $c_{\text{v}} = 1$, $\lambda_{\text{gen}} = 10^{-8}$, $\lambda_{\text{lat}} = 10^{-2.5}$.

dataset	RM	LM	s	$e_{20}(r)$	$e_{20}(r/s)$	$e_{80}(r)$	$e_{80}(r/s)$	ΔV
AlpacaFarm	FsfairX	Llama-3.1-8B	0.968	0.0424	0.0426	0.0113	0.0113	0.00000
		Gemma-2-9B	0.627	0.0451	0.0475	0.0135	0.0145	−0.00027
		Mistral-7B	0.840	0.0421	0.0434	0.0102	0.0109	−0.00015
		Qwen-2.5-7B	0.818	0.0427	0.0437	0.0115	0.0117	−0.00006
	RM-Mistral-7B	Llama-3.1-8B	0.938	0.0420	0.0425	0.0110	0.0112	−0.00004
		Gemma-2-9B	0.670	0.0410	0.0434	0.0114	0.0126	−0.00031
		Mistral-7B	0.832	0.0430	0.0444	0.0109	0.0119	−0.00019
		Qwen-2.5-7B	0.871	0.0425	0.0434	0.0118	0.0122	−0.00009
	ArmoRM	Llama-3.1-8B	0.00992	0.0423	0.0418	0.0266	0.0106	+0.00321
		Gemma-2-9B	0.00666	0.0455	0.0457	0.0271	0.0155	+0.00266
		Mistral-7B	0.00924	0.0422	0.0422	0.0250	0.0106	+0.00299
		Qwen-2.5-7B	0.00838	0.0429	0.0424	0.0263	0.0123	+0.00294
HH-RLHF	FsfairX	Llama-3.1-8B	0.801	0.0416	0.0427	0.0122	0.0118	−0.00001
		Gemma-2-9B	0.673	0.0411	0.0433	0.0105	0.0109	−0.00019
		Mistral-7B	0.803	0.0420	0.0437	0.0099	0.0111	−0.00023
		Qwen-2.5-7B	0.748	0.0414	0.0432	0.0095	0.0105	−0.00019
	RM-Mistral-7B	Llama-3.1-8B	0.798	0.0413	0.0427	0.0097	0.0105	−0.00016
		Gemma-2-9B	0.680	0.0407	0.0431	0.0096	0.0109	−0.00025
		Mistral-7B	0.739	0.0421	0.0446	0.0100	0.0121	−0.00037
		Qwen-2.5-7B	0.795	0.0411	0.0425	0.0092	0.0103	−0.00018
	ArmoRM	Llama-3.1-8B	0.00661	0.0436	0.0420	0.0289	0.0108	+0.00379
		Gemma-2-9B	0.00533	0.0427	0.0418	0.0277	0.0108	+0.00312
		Mistral-7B	0.00654	0.0433	0.0426	0.0286	0.0109	+0.00331
		Qwen-2.5-7B	0.00654	0.0431	0.0418	0.0278	0.0112	+0.00361

rewards is the case $\kappa = s < 1$, the direction the sweep favours at small N_1. ArmoRM, whose scores are not on the logit scale, has s between 0.0053 and 0.0099, and without rescaling its fit fails as described above: RMSE at $N_1 = 80$ is between 0.025 and 0.029, against 0.011 to 0.016 once rescaled; both policies generate 131 to 140 times on average, instead of 86 to 97; and the latency-aware policy's V falls by 0.0027 to 0.0038, about four times what conditioning on T_1 gains. Without rescaling, conditioning on T_1 appears to gain between $+0.0027$ and $+0.0033$; this is an artifact, the difference of two depressed objectives.

Choice of target spread. Targeting unit spread is a choice, and Table 7 sweeps a factor κ around it, replacing s by s/κ in (2), so that the estimator is fitted to $e^{\kappa r/s}$ and $\kappa = 1$ is the rescaling above. The best κ drifts upward with N_1, from 0.5 at $N_1 \leq 20$ to 1 at $N_1 \geq 160$: a heavier fitted tail compensates the optimism of a maximum taken over few samples, while a large first batch resolves the tail well enough to prefer a lighter one. No single value is best everywhere; $\kappa = 1$ is the best at $N_1 \geq 160$, within 0.0004 of the best at $N_1 = 80$, within 0.0014 at $N_1 = 40$, and within 0.003 at the smallest first batches, so we use $\kappa = 1$ throughout and introduce no tuned quantity.

Cost of the rescaling. Estimating s needs no training and no additional model, and uses the same kind of offline sample a practitioner already needs in order to choose $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ (Appendix H). Only its rough magnitude matters, so a small sample suffices (Table 8). From 50 prompts with 20 responses each – a thousand generations, once – the estimate falls between 0.81 and 1.10 times its full-pool value 90% of the time. Dividing by such an estimate is equivalent to taking κ between roughly 0.9 and 1.2 in Table 7, which changes the RMSE by at most 0.0015. Small samples understate s, since the standard deviation of a few draws is biased low, but the bias is modest from $n = 20$ on.

Table 7: RMSE of $\hat{F}_x(R_1)$ for the shifted-exponential estimator when rewards are scaled to κ times unit median within-prompt spread, averaged over the 24 configurations of Appendix I. The paper uses $\kappa = 1$. Averaged over 100 prompts and 300 Monte Carlo draws per prompt; best per column in bold. Stage-1 batches are drawn from the full pool and scored against the reward quantile of Appendix B, as in Table 9, not against the held-out empirical CDF of Table 3, so the values differ slightly from that table's. The estimator is fitted to the exponentiated reward, so its fitted tail depends on the reward's scale.

κ	$N_1 = 10$	20	40	80	160	240
0.5	**0.0827**	**0.0404**	0.0205	0.0123	0.0092	0.0081
0.7	0.0837	0.0412	**0.0204**	**0.0111**	0.0074	0.0062
0.85	0.0845	0.0421	0.0209	0.0111	0.0069	0.0056
1	0.0853	0.0431	0.0217	0.0115	**0.0069**	**0.0054**
1.2	0.0865	0.0445	0.0230	0.0123	0.0072	0.0055
1.5	0.0883	0.0467	0.0249	0.0135	0.0077	0.0057
2	0.0911	0.0500	0.0275	0.0150	0.0084	0.0060

Table 8: Estimating s offline from m prompts with n scored responses each: the estimate as a ratio to its value on the full pool of 100 prompts and 960 responses, as median [5th, 95th percentile] over 400 resamples, for Llama-3.1-8B on AlpacaFarm scored by FsfairX.

prompts	$n = 5$	$n = 10$	$n = 20$	$n = 50$
$m = 10$	0.83 [0.55, 1.30]	0.90 [0.64, 1.29]	0.94 [0.65, 1.34]	0.99 [0.71, 1.36]
$m = 25$	0.82 [0.64, 1.06]	0.89 [0.72, 1.17]	0.92 [0.74, 1.21]	0.98 [0.78, 1.20]
$m = 50$	0.83 [0.69, 0.99]	0.90 [0.75, 1.06]	0.95 [0.81, 1.10]	0.97 [0.84, 1.11]

Scale-invariant alternatives. The scale dependence is a consequence of fixing the exponent in e^r rather than fitting it. Fitting a second tail parameter removes it: a Gompertz upper tail, with survival $\exp(-\eta(e^{by} - 1))$ above the fitted median and both parameters estimated by maximum likelihood, contains our estimator as the case $b = 1$ and is exactly invariant to any affine rescaling of the reward. Averaged over the 24 configurations it is slightly more accurate than ours at large first batches (0.006 against 0.007 at $N_1 = 160$, and level at 240) but clearly worse at small ones (0.094 against 0.085 at $N_1 = 10$), since a handful of exceedances estimate two parameters poorly – the same trade-off that led us to a one-parameter fit in Appendix D. A generalized-Pareto tail is likewise invariant and worse at every N_1. We therefore keep the one-parameter fit and set its implicit scale from offline data, but for a regime with much larger first batches the Gompertz fit would be the better choice.

F Calibrating the estimated quantile $\hat{F}_x(R_1)$

The correction. The shifted-exponential estimator overstates $F_x(R_1)$ at every N_1 (Figure 3), so the policy underrates the value Δ of a second batch and stops too early. This appendix tests an offline correction. A map g is fitted from $\hat{F}_x(R_1)$ to $\mathbb{E}[F_x(R_1) \mid \hat{F}_x(R_1)]$, and the policy uses $\Delta(g(\hat{F}_x(R_1)), N_2)$ in place of $\Delta(\hat{F}_x(R_1), N_2)$; nothing else changes. Calibrating $F_x(R_1)$ rather than Δ lets one map serve every N_2. Because Δ is convex, $\Delta(g(\hat{F}_x(R_1)), N_2)$ still understates the expected gain $\mathbb{E}[\Delta(F_x(R_1), N_2) \mid \hat{F}_x(R_1)]$; a map fitted to Δ itself, one for each N_2, would target the gain directly. With binned means the rule sees about 85% of the true value of a second batch, against 42–68% uncalibrated (Table 10). The map is fitted to pairs $(\hat{F}_x(R_1), F_x(R_1))$ from Stage-1 batches drawn from the reward pools of prompts with many scored responses, the same kind of offline data as the reward scale of Appendix E, with $F_x(R_1)$ the reward quantile of Appendix B. A separate map is fitted for each Stage-1 batch size N_1, from Stage-1 batches of that size, since the estimator's bias shrinks as N_1 grows (Table 9).

Fitting the map. Each map is fitted by binned means: the mean of $F_x(R_1)$ in each of 25 equal-count bins of $\hat{F}_x(R_1)$, interpolated linearly between bins. Isotonic regression and local-linear kernel regression give almost the same accuracy (Table 10); binned means have the lowest held-out RMSE

Table 9: Accuracy of the calibrated estimate $g(\hat{F}_x(R_1))$ of $F_x(R_1)$, against the uncalibrated $\hat{F}_x(R_1)$, averaged over all 24 combinations of four language models, three reward models, and two datasets. Errors are against the reward quantile, on prompts the map was not fitted on. "own": g fitted on the configuration's own prompts; "other": fitted on one configuration that shares none of its language model, reward model and dataset, averaged over the six such (RMSE as the root of the mean squared error).

	RMSE			R^2			bias		
N_1	none	own	other	none	own	other	none	own	other
20	0.0432	0.0389	0.0393	0.10	0.27	0.25	+0.0187	+0.0002	−0.0001
40	0.0218	0.0195	0.0200	0.17	0.33	0.30	+0.0091	+0.0001	−0.0002
80	0.0116	0.0098	0.0103	0.08	0.35	0.28	+0.0045	+0.0001	−0.0002
160	0.0069	0.0050	0.0053	−0.34	0.31	0.22	+0.0022	+0.0000	−0.0001

Table 10: Three ways of fitting the calibration map g of Appendix F, compared on prompts the map was not fitted on, and how much of the true value of a second batch the stopping rule sees. RMSE and R^2 are of $g(\hat{F}_x(R_1))$ as an estimate of $F_x(R_1)$, as in Table 9; "none" is the uncalibrated $\hat{F}_x(R_1)$. "Share of Δ seen" is the mean over Stage-1 batches of $\Delta(\cdot, N_1)$ at the estimate, divided by its mean at the true $F_x(R_1)$: 1 means the rule sees the second batch's full value, and below 1 it undervalues it. Each map is fitted on half of a configuration's prompts and scored on the other half, and vice versa; averaged over all 24 combinations of four language models, three reward models, and two datasets. Binned means (25 equal-count bins) are the map used in the paper.

	RMSE				R^2				share of Δ seen	
N_1	none	binned means	local linear	isotonic	none	binned means	local linear	isotonic	none	binned
20	0.0432	0.0389	0.0388	0.0389	0.10	0.27	0.27	0.27	0.42	0.83
40	0.0218	0.0195	0.0195	0.0196	0.17	0.33	0.33	0.33	0.51	0.85
80	0.0116	0.0098	0.0099	0.0101	0.08	0.35	0.33	0.31	0.59	0.85
160	0.0069	0.0050	0.0052	0.0053	−0.34	0.31	0.24	0.21	0.68	0.86

at $N_1 \geq 40$ and are within 0.00003 of the lowest at $N_1 = 20$, and the other two overfit more as N_1 grows (held-out R^2 at $N_1 = 160$: 0.31 for binned means, 0.24 and 0.21). So that every estimate is scored on prompts its map has not seen, the 100 prompts of each configuration are split in half, and the map fitted on one half is applied to the other, in both directions.

Accuracy. Figure 5 shows $\mathbb{E}[F_x(R_1) \mid \hat{F}_x(R_1)]$ to be increasing but flatter than the identity in every configuration: the fitted tail is overconfident when it puts R_1 high. Table 9 shows that calibration removes the bias and lowers the RMSE by 10% to 28%; a map fitted on the configuration's own prompts lowers it in all 24 configurations at every N_1. A map fitted on a configuration sharing none of the language model, reward model and dataset, from the same number of prompts, does nearly as well, with an RMSE at most 6% higher than the own-configuration map's, and still below the uncalibrated RMSE in at least 21 of the 24 configurations at every N_1, so the map need not be refitted for every deployment. Unlike Table 3, Table 9 scores against the reward quantile the policy is scored by, with Stage-1 batches drawn from the full pool, because that is the quantity the map is fitted to and the policy uses. Under the held-out protocol of Appendix D, the calibrated estimator has the lowest RMSE of all the estimators at every N_1, with a bias of at most 0.0002 (Figure 3).

Experiment 1. With N_2 fixed and latency deterministic, the policy continues exactly when $\hat{F}_x(R_1)$ falls below a threshold set by $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$, since Δ is decreasing in its first argument. An increasing g only changes which threshold each λ pair selects, so sweeping the λ grid traces the same set of policies, and the frontiers of Figure 1 are unchanged up to how the grid samples them. What calibration changes is which of these policies a given λ pair selects. It also cannot close Experiment 1's remaining gap to the oracle: the oracle stops by the true $F_x(R_1)$, and a monotone map leaves the ordering of Stage-1 batches by $\hat{F}_x(R_1)$ unchanged, so what is left of that gap is an error in ranking batches, not a bias in their level. We measure this by the *regret* relative to the oracle policy, $V_{\text{oracle}} - V$ at the same λ pair. Since each batch takes $\overline{T}$, both the decision and V depend on the λ

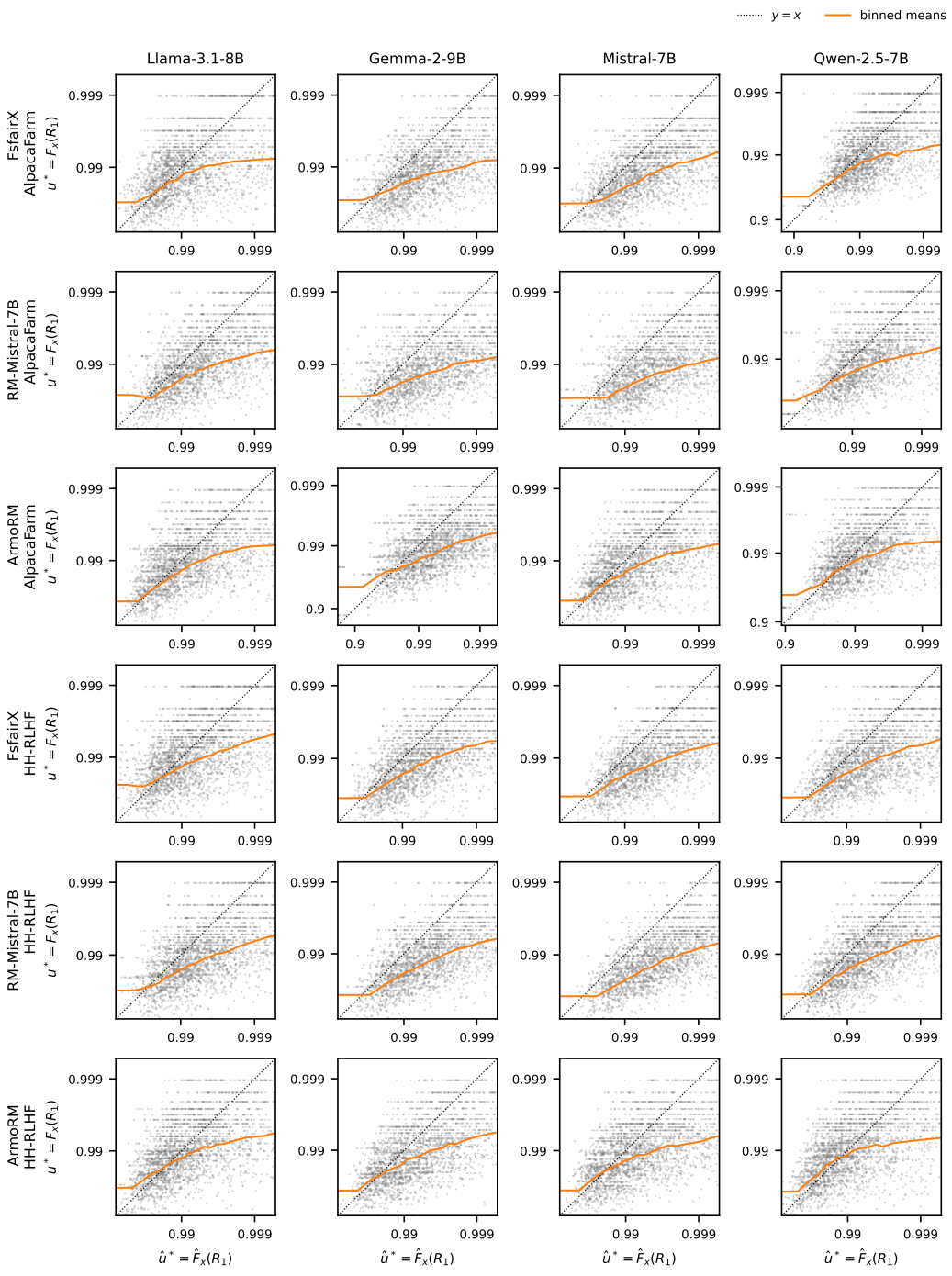


Figure 5: The estimated quantile $\hat{F}_x(R_1)$ of the best of $N_1 = 80$ Stage-1 rewards (shifted exponential) against its reward quantile $F_x(R_1)$, in each of the 24 configurations (columns: language models; rows: reward model and dataset), on logit axes. Grey points: individual Stage-1 batches. Orange: the binned-means calibration map g, fitted on half of the configuration's prompts. Dotted: the identity. Above about 0.99 the points fall below the identity: the fit overstates $F_x(R_1)$ when it puts R_1 high.

Table 11: Experiment 1: regret relative to the oracle policy, $V_{\text{oracle}} - V$, in units of 10^{-4}, averaged over all 24 combinations of four language models, three reward models, and two datasets. Its largest value over the λ grid, and its value at $\lambda_{\text{gen}} = 10^{-8}$, $\lambda_{\text{lat}} = 10^{-2.5}$ (the pair of Table 2); "own" and "other" as in Table 9.

	largest			at $\lambda_{\text{gen}} = 10^{-8}$, $\lambda_{\text{lat}} = 10^{-2.5}$		
N_1	uncal.	own	other	uncal.	own	other
20	78.0	60.3	62.3	73.17	44.51	45.90
40	32.5	26.4	28.4	31.77	26.01	28.18
80	15.7	12.6	14.4	10.55	10.38	11.67
160	8.9	6.9	8.0	2.65	1.68	1.67

pair only through the cost of a second batch, $c = \lambda_{\text{gen}} N_1 + \lambda_{\text{lat}} \overline{T}$, so Figure 6 plots the regret against c. Calibration lowers the regret at all but 128 of the 4×4331 λ pairs, where it raises it by at most 0.00005. At small c it lowers the regret to zero: the calibrated policy then always continues, as the oracle policy does, whereas the uncalibrated one still stops whenever $\hat{F}_x(R_1)$ is close to 1. It lowers the largest regret over the grid by 19% to 23%, and the regret at $\lambda_{\text{gen}} = 10^{-8}$, $\lambda_{\text{lat}} = 10^{-2.5}$ (the pair of Table 2) by 39%, 18%, 2% and 37% at $N_1 = 20$, 40, 80 and 160 (Table 11). A map fitted on another configuration does almost as well.

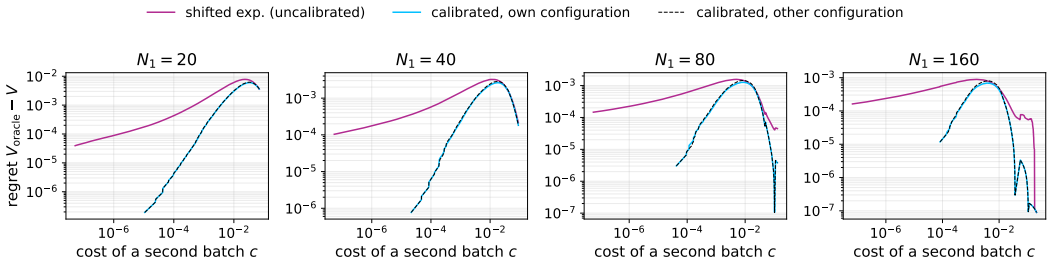


Figure 6: Experiment 1: regret relative to the oracle policy, $V_{\text{oracle}} - V$, against the cost of a second batch $c = \lambda_{\text{gen}} N_1 + \lambda_{\text{lat}} \overline{T}$, for $N_1 = N_2 \in \{20, 40, 80, 160\}$, averaged over the 24 configurations. Each point is one $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pair of the grid. Magenta: uncalibrated; blue: calibrated, own map; dashed black: calibrated, map from another configuration (both as in Table 9).

Experiment 2. Table 12 adds rows for the calibrated estimate and for the oracle to Table 2. Calibration raises ΔV at every c_{v}, from $+0.00025$, $+0.00070$ and $+0.00141$ to $+0.00033$, $+0.00088$ and $+0.00174$. It leaves the latency-unaware policy almost unchanged (84.6 mean generations with and without it) and raises the latency-aware policy's mean number of generations (from 95.2 to 106.3 at $c_{\text{v}} = 2$). That raises the latency-aware policy's reward quantile, but also its mean latency, to above the latency-unaware policy's. So with calibration the latency-aware policy has a higher reward quantile and a higher mean latency than the latency-unaware policy, whereas uncalibrated, and with the oracle, it has a higher reward quantile and a lower mean latency. Figure 7 shows the regret relative to the oracle policy across λ_{lat}, at $\lambda_{\text{gen}} = 10^{-8}$: calibration lowers it for both policies, at every λ_{lat} from $10^{-4.5}$ to 10^{-2} and every c_{v}.

Adaptive second-batch size. Appendix M compares, under deterministic latency, the adaptive policy, which chooses N_2 per instance from $\mathcal{Z} = \{10, 20, \ldots, 80\}$, with the non-adaptive one, which fixes (N_1, N_2) from $\mathcal{Z}$ offline. Here, unlike in Experiment 1, calibration can move the frontiers: the λ pair sets the batch sizes as well as the stopping threshold, so calibration changes which threshold goes with which batch sizes, and the adaptive policy's N_2 also depends on the value of $\hat{F}_x(R_1)$. Figure 8 shows that calibration raises both frontiers at low compute, by up to 0.0016 for the adaptive policy and 0.0011 for the non-adaptive one, at about 30 mean generations, and moves them by at most 0.00014 above 100. The adaptive policy's largest advantage over the non-adaptive one rises from 0.0015 to 0.0017. Under log-normal latency at the λ pair of Table 22, the two policies coincide with the calibrated estimate, as they do without it.

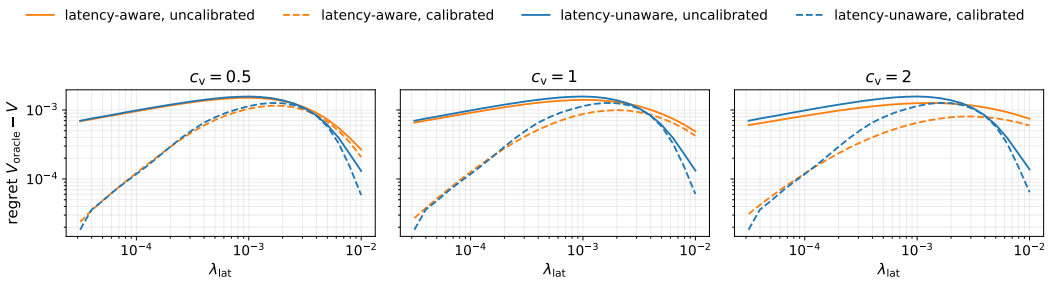


Figure 7: Experiment 2: regret relative to the oracle policy, $V_{\text{oracle}} - V$, against λ_{lat}, at $\lambda_{\text{gen}} = 10^{-8}$ and $c_{\text{v}} \in \{0.5, 1, 2\}$ (panels), averaged over the 24 configurations. Orange: latency-aware policy; blue: latency-unaware policy; solid: uncalibrated; dashed: calibrated with the configuration's own map. Each policy's regret is relative to its own oracle policy, so the comparison is between line styles of one colour.

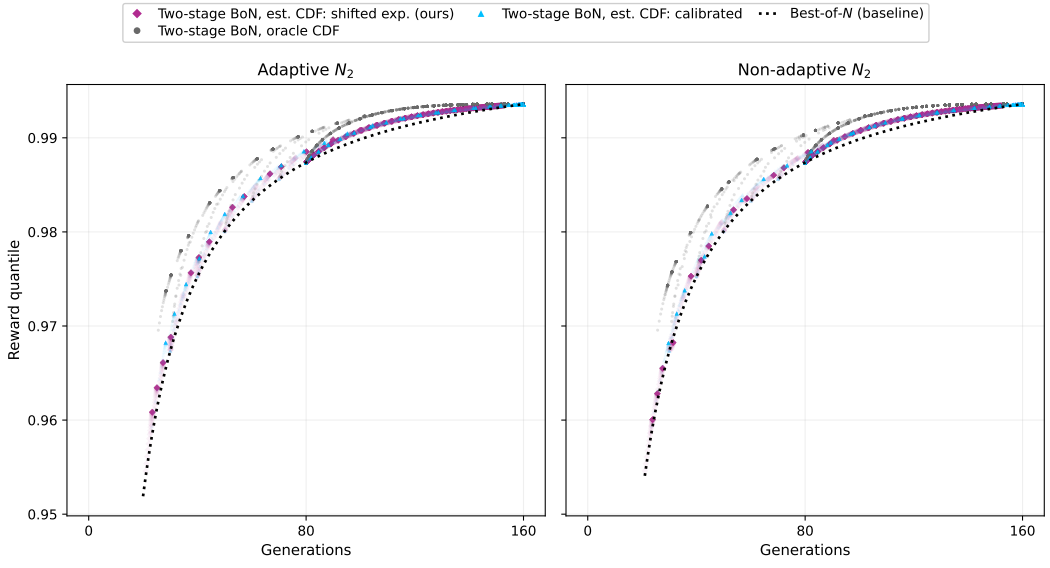


Figure 8: Figure 19 with the calibrated estimate: reward–compute Pareto frontiers of the adaptive (left) and non-adaptive (right) policies under deterministic latency, $\mathcal{Z} = \{10, 20, \ldots, 80\}$, averaged over the 24 configurations. Magenta diamonds: estimated $\hat{F}_x$; cyan triangles: calibrated estimate (own configuration); grey circles: oracle F_x; dotted black: parallel Best-of-N.

Table 12: Table 2 with rows for the calibrated estimate (own configuration) and for the oracle F_x, at the same $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$, averaged over all 24 combinations of four language models, three reward models, and two datasets. "Aware" and "unaware" are the latency-aware and latency-unaware policies; ΔV is latency-aware minus latency-unaware.

		mean gens		reward quantile		mean latency (s)		
c_{v}	estimate	aware	unaware	aware	unaware	aware	unaware	ΔV
0.5	uncalibrated	86.3	84.6	0.9888	0.9886	5.284	5.288	+0.00025
	calibrated	88.0	84.6	0.9890	0.9886	5.316	5.286	+0.00033
	oracle	91.5	89.0	0.9907	0.9905	5.536	5.561	+0.00028
1	uncalibrated	89.6	84.6	0.9892	0.9886	5.273	5.287	+0.00070
	calibrated	94.7	84.6	0.9897	0.9886	5.353	5.286	+0.00088
	oracle	96.1	89.0	0.9910	0.9905	5.494	5.561	+0.00077
2	uncalibrated	95.2	84.6	0.9898	0.9886	5.254	5.295	+0.00141
	calibrated	106.3	84.6	0.9905	0.9886	5.364	5.294	+0.00174
	oracle	104.0	89.0	0.9916	0.9905	5.435	5.569	+0.00151

Table 13: Table 20 with the calibrated estimate: the largest user-reward difference from parallel Best-of-N at matched mean generations, at $\lambda_{\text{lat}} = 10^{-2.5}$, when a batch of N takes $T\,(N/80)^{\beta}$, with the uncalibrated estimate (as in Table 20) and the calibrated one (own configuration). Policies as in Table 20: the non-adaptive ones use $N_1 = N_2 = 80$; the adaptive ones choose N_1 offline and N_2 per instance from $\mathcal{Z} = \{10, 20, 40, 80, 160, 320\}$.

			uncalibrated			calibrated		
β	N_2 rule	stopping	c_{v}=0.5	1	2	c_{v}=0.5	1	2
0	fixed 80	aware	0.0000	+0.0001	+0.0001	0.0000	+0.0001	+0.0002
		unaware	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	adaptive	aware	+0.0003	+0.0005	+0.0008	+0.0002	+0.0004	+0.0006
		unaware	+0.0002	+0.0002	+0.0002	+0.0001	+0.0001	+0.0001
0.5	fixed 80	aware	+0.0002	+0.0006	+0.0011	+0.0003	+0.0007	+0.0014
		unaware	+0.0001	+0.0001	+0.0001	+0.0001	+0.0001	+0.0001
	adaptive	aware	+0.0014	+0.0016	+0.0018	+0.0017	+0.0019	+0.0040
		unaware	+0.0013	+0.0013	+0.0013	+0.0016	+0.0016	+0.0016
1	fixed 80	aware	+0.0009	+0.0015	+0.0027	+0.0011	+0.0022	+0.0041
		unaware	+0.0005	+0.0005	+0.0005	+0.0005	+0.0005	+0.0005
	adaptive	aware	+0.0024	+0.0024	+0.0047	+0.0031	+0.0038	+0.0111
		unaware	+0.0024	+0.0024	+0.0024	+0.0031	+0.0031	+0.0031

Correlation between T_1 and T_2. Appendix J lets the second batch's latency be only partly predictable from the first's, with correlation ρ between $\log T_1$ and $\log T_2$, and compares two ways for the latency-aware policy to predict T_2: by $\mathbb{E}[T_2 \mid T_1]$ at the true ρ, and by assuming $T_2 = T_1$. With the oracle F_x, the first gives the higher ΔV at every $\rho < 1$; with $\hat{F}_x$, the second gives the higher ΔV at $\rho \in \{0.5, 0.75\}$, because the estimated policy stops too early. With the calibrated estimate the true-ρ predictor is no longer behind (Appendix J).

Latency that grows with batch size. Appendix L lets a batch of N take $T\,(N/80)^{\beta}$, so that for $\beta > 0$ larger batches take longer, and compares the policies with parallel Best-of-N in user reward at $\lambda_{\text{lat}} = 10^{-2.5}$, including adaptive policies that choose N_1 and N_2 from $\{10, 20, 40, 80, 160, 320\}$. Table 13 repeats that comparison with the calibrated estimate. At $\beta = 0$ calibration changes no policy's largest user-reward difference from Best-of-N by more than 0.0002. For $\beta > 0$ it raises that of both latency-aware policies at every c_{v}; for the adaptive one at $\beta = 1$ and $c_{\text{v}} = 2$, from 0.0047 to 0.0111.

We keep the uncalibrated estimator in the main experiments. Calibration leaves the frontiers of Experiment 1 unchanged, it adds an offline fitting step, and with it the latency-aware policy's higher V in Table 2 no longer comes with a lower mean latency.

G HEATMAPS OF THE LAMBDA GRID

Figure 9 visualises, on the full 71×61 $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ grid, the three quantities that are aggregated into the $N_1 = 80$ Pareto frontier of Figure 1: the mean reward quantile, the mean latency, and the mean number of generations, for the non-adaptive latency-aware policy with deterministic latency $\overline{T} = 5\,\text{s}$, averaged over the 24 configurations. The left column shows the policy run with the estimated CDF $\hat{F}_x$, the middle column shows the oracle policy with the reward quantile over $\mathcal{R}_b$ in place of F_{x_b}, and the right column shows their per-cell difference. The estimated policy's reward is below the oracle's in 99.9% of cells, by at most 0.0023, and never measurably above it. Its generations and latency are below the oracle's almost everywhere too – it generates more than the oracle in 2% of cells, and then by less than 0.05 generations, while generating up to 21.5 fewer elsewhere. Estimation error therefore costs reward mainly by stopping too early, not by over-generating.

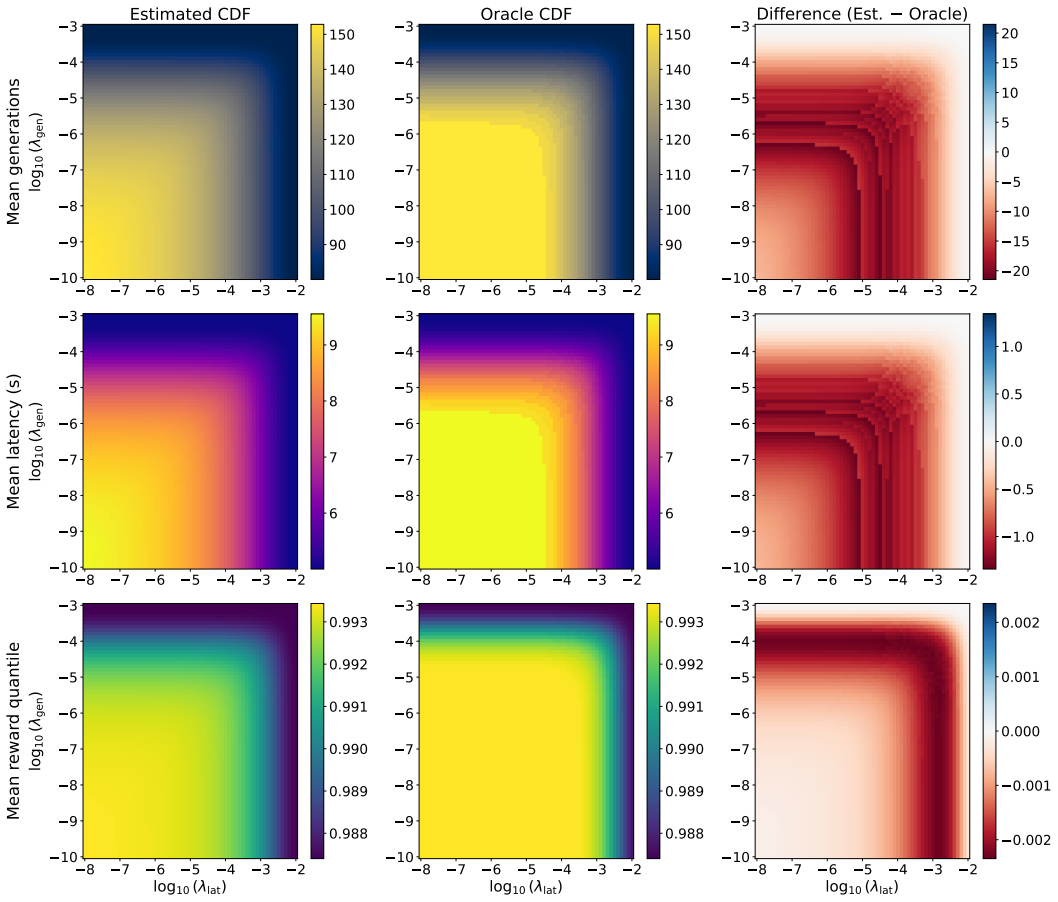


Figure 9: Heatmaps over the full 71×61 $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ grid for the non-adaptive latency-aware policy at $N_1 = 80$, deterministic latency $\overline{T} = 5\,\text{s}$, averaged over the 24 configurations; these are the three quantities aggregated into the $N_1 = 80$ panel of Figure 1. Top row: mean number of generations. Middle row: mean latency (s). Bottom row: mean reward quantile of the returned response. Left column: *our method*, run with the estimated CDF $\hat{F}_x$. Middle column: oracle variant, with the reward quantile in place of F_{x_b}. Right column: estimated minus oracle. The estimated policy's reward is at or below the oracle's everywhere, and its generations and latency are below the oracle's in all but a small fraction of cells (see text).

H SELECTING THE TRADEOFF PARAMETERS IN PRACTICE

The two tradeoff parameters are selected once, offline. Run inference on an initial serving trace or a held-out set, sweep $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ over a grid as we do above, plot the resulting tradeoff among mean reward, mean latency, and mean generation count, and select the pair that meets the deployment target. Two properties keep this cheap. First, the λ's do not need to adapt to load: adaptation to the state of the serving system is automatic through T_1, in contrast to a fixed threshold such as Dynasor's τ, which must be recalibrated per algorithm–dataset pair. Second, because the CDF transformation places the decision in scale-free quantile space, λ_{gen} and λ_{lat} should transfer across language models and reward models as long as the serving system – and hence the latency distribution – is unchanged; Appendix I checks this across the 24 configurations. λ_{lat} has a direct operational reading as an application-level preference: how much reward quantile one second of additional latency is worth.

I PER-CONFIGURATION RESULTS ACROSS LANGUAGE MODELS, REWARD MODELS, AND DATASETS

Sections 4.2 and 4.3 report results averaged over all 24 combinations of four language models (Llama 3.1 8B, Gemma-2-9B, Mistral-Instruct-v0.3-7B, Qwen-2.5-7B), three reward models (FsfairX-LLaMA3-RM-v0.1, RM-Mistral-7B, ArmoRM-Llama3-8B-v0.1), and two prompt datasets (AlpacaFarm, HH-RLHF); this appendix gives the per-configuration values behind those averages. Tables list the configurations in blocks of four language models, one block per dataset and reward model, and figures lay them out the same way, one row of panels per block. Every configuration uses 100 prompts with a 960-reward pool, and all other settings are held fixed. Reward quantiles make the configurations comparable despite their reward scales differing by orders of magnitude; each configuration's Best-of-N reference is computed on its own reward pool.

Table 14 repeats the Experiment 1 comparison in every configuration. Our method beats parallel Best-of-N at matched compute in all 24, with the fraction of the Best-of-N-to-oracle gap recovered at $1.5N_1$ ranging from 29% (Llama 3.1 8B with FsfairX on AlpacaFarm) to 63% (Mistral-7B with RM-Mistral-7B on HH-RLHF), from 30% to 52% at $1.1N_1$ and from 19% to 73% at $1.8N_1$.

Table 15 repeats Experiment 2 in every configuration, at the same $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$. The latency-aware policy improves the objective in all 24 configurations, by between $+0.00051$ and $+0.00087$, with mean generations between 80.9 and 90.3 for the latency-unaware policy and between 83.6 and 96.7 for the latency-aware one. The gain in reward quantile is similar everywhere ($+0.00050$ to $+0.00073$), but latency is not: the ten configurations that continue most often – the eight using ArmoRM, together with Llama 3.1 8B and FsfairX on either dataset – lower the mean latency by 0.02 to 0.11 s, while the rest change it by less than 0.032 s, in either direction. So the lower mean latency of Table 2 is an average over configurations, not a property of each.

These runs also give a first quantitative check on the transfer claim of Appendix H: that a λ pair selected on one language model or reward model should place a practitioner at approximately the same operating point on another, without re-sweeping. Measured on the latency-unaware baseline with the oracle F_x, which isolates the decision rule from CDF-estimation error, the 24 configurations at the fixed λ pair span 88.6 to 89.8 mean generations, with mean reward quantile in $[0.9902, 0.9906]$. The reason is structural rather than empirical: the stopping rule sees the reward only through the CDF transform $\hat{F}_x$, so under a well-specified F_x (the oracle policy), the decision is made on a $\text{Unif}[0, 1]$ scale that is identical across language models, reward models, and datasets. A λ pair therefore denominates the same tradeoff everywhere, and no re-sweeping is needed. What differs across configurations is the quality of $\hat{F}_x$: where the shifted-exponential fit misstates the upper tail, $\Delta(u, N_2)$ is biased and the operating point shifts.

The CDF-estimator comparison across configurations is reported in Table 4, and is the one place where our choices do not transfer cleanly: the shifted-exponential estimator is best in 21 of the 24 configurations at $N_1 = 15$ and again at $N_1 = 30$, but is overtaken at $N_1 \geq 120$ in most of them. We discuss this in Appendix D.

Table 14: Reward–compute tradeoff in each of the 24 combinations of four language models, three reward models, and two datasets, at a matched compute budget of $1.5N_1 = 120$ mean generations: mean reward quantile of parallel Best-of-N (BoN) and of our method with the estimated $\hat{F}_x$ and with the oracle, and the fraction of the BoN-to-oracle gap our method recovers at $1.1N_1$, $1.5N_1$ and $1.8N_1$. $N_1 = N_2 = 80$, deterministic latency. Each row's Best-of-N reference is computed on that configuration's own reward pool.

			mean reward quantile at $1.5N_1$			gap recovered at		
dataset	RM	LM	BoN	ours, $\hat{F}_x$	ours, oracle	$1.1N_1$	$1.5N_1$	$1.8N_1$
AlpacaFarm	FsfairX	Llama-3.1-8B	0.9917	0.9922	0.9934	36%	29%	27%
		Gemma-2-9B	0.9916	0.9924	0.9933	36%	47%	56%
		Mistral-7B	0.9917	0.9926	0.9934	43%	52%	60%
		Qwen-2.5-7B	0.9912	0.9918	0.9929	36%	34%	20%
	RM-Mistral-7B	Llama-3.1-8B	0.9917	0.9925	0.9934	44%	49%	45%
		Gemma-2-9B	0.9916	0.9925	0.9933	30%	50%	62%
		Mistral-7B	0.9917	0.9925	0.9933	42%	52%	59%
		Qwen-2.5-7B	0.9914	0.9921	0.9931	39%	39%	19%
	ArmoRM	Llama-3.1-8B	0.9915	0.9922	0.9932	42%	44%	38%
		Gemma-2-9B	0.9909	0.9919	0.9928	39%	55%	65%
		Mistral-7B	0.9914	0.9922	0.9932	41%	43%	47%
		Qwen-2.5-7B	0.9911	0.9918	0.9929	40%	40%	28%
HH-RLHF	FsfairX	Llama-3.1-8B	0.9917	0.9925	0.9934	35%	44%	57%
		Gemma-2-9B	0.9917	0.9925	0.9934	45%	49%	47%
		Mistral-7B	0.9917	0.9927	0.9934	47%	58%	64%
		Qwen-2.5-7B	0.9917	0.9926	0.9934	46%	51%	61%
	RM-Mistral-7B	Llama-3.1-8B	0.9916	0.9926	0.9933	43%	57%	72%
		Gemma-2-9B	0.9916	0.9926	0.9933	50%	58%	63%
		Mistral-7B	0.9917	0.9927	0.9934	52%	63%	73%
		Qwen-2.5-7B	0.9916	0.9926	0.9933	50%	57%	68%
	ArmoRM	Llama-3.1-8B	0.9913	0.9920	0.9930	40%	42%	47%
		Gemma-2-9B	0.9913	0.9922	0.9931	45%	49%	51%
		Mistral-7B	0.9914	0.9921	0.9931	42%	44%	55%
		Qwen-2.5-7B	0.9912	0.9919	0.9930	44%	38%	37%

I.1 Per-configuration figures

Figures 10 and 11 give the Pareto frontier of Experiment 1 and the reward–generation plot of Experiment 2 (Sections 4.2 and 4.3) in every configuration, and Figure 12 the user reward of Figure 2, one panel per configuration. The reward quantile makes these nearly distribution-free: without reward ties each Best-of-N reference would equal $N/(N+1)$ exactly, and ties lower them by about 0.001 at most at $N \geq 80$ (Appendix B).

J Sensitivity to the T_1–T_2 correlation

Every experiment above sets $T_2 = T_1$, and accordingly instantiates the latency model of Section 3 as $\widehat{T}_2(s_1, N_2) = T_1$, which is the most favourable possible choice for a policy that uses T_1 to predict T_2. To measure how much of the reported gain rests on it, we sweep the strength of that dependence. Let $T_1 \sim \mathrm{LogNormal}(\mu, \sigma^2)$ and introduce a correlation parameter $\rho \in [0, 1]$ acting in log-latency space, so that ρ is the Pearson correlation coefficient of $\log T_1$ and $\log T_2$:

$$\log T_2 = \mu + \rho\,(\log T_1 - \mu) + \sqrt{1-\rho^2}\;\sigma\,\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,1). \tag{3}$$

Here ε is independent of T_1. The marginal distribution of T_2 equals that of T_1 for every ρ, so varying ρ changes only how informative T_1 is about T_2 and nothing else; $\rho = 1$ recovers $T_2 = T_1$ exactly, and $\rho = 0$ makes the two independent. Two uses of T_2 must be kept distinct: the simulation charges the *sampled* T_2 into realized latency, while the policy may only use the predictor $\mathbb{E}[T_2 \mid T_1] = \exp(\mu + \rho(\log T_1 - \mu) + (1-\rho^2)\sigma^2/2)$, which is precisely $\widehat{T}_2(s_1, N_2)$ of Section 3 under this model, here a function of T_1 alone. This is the idealised predictor: the policy is handed the

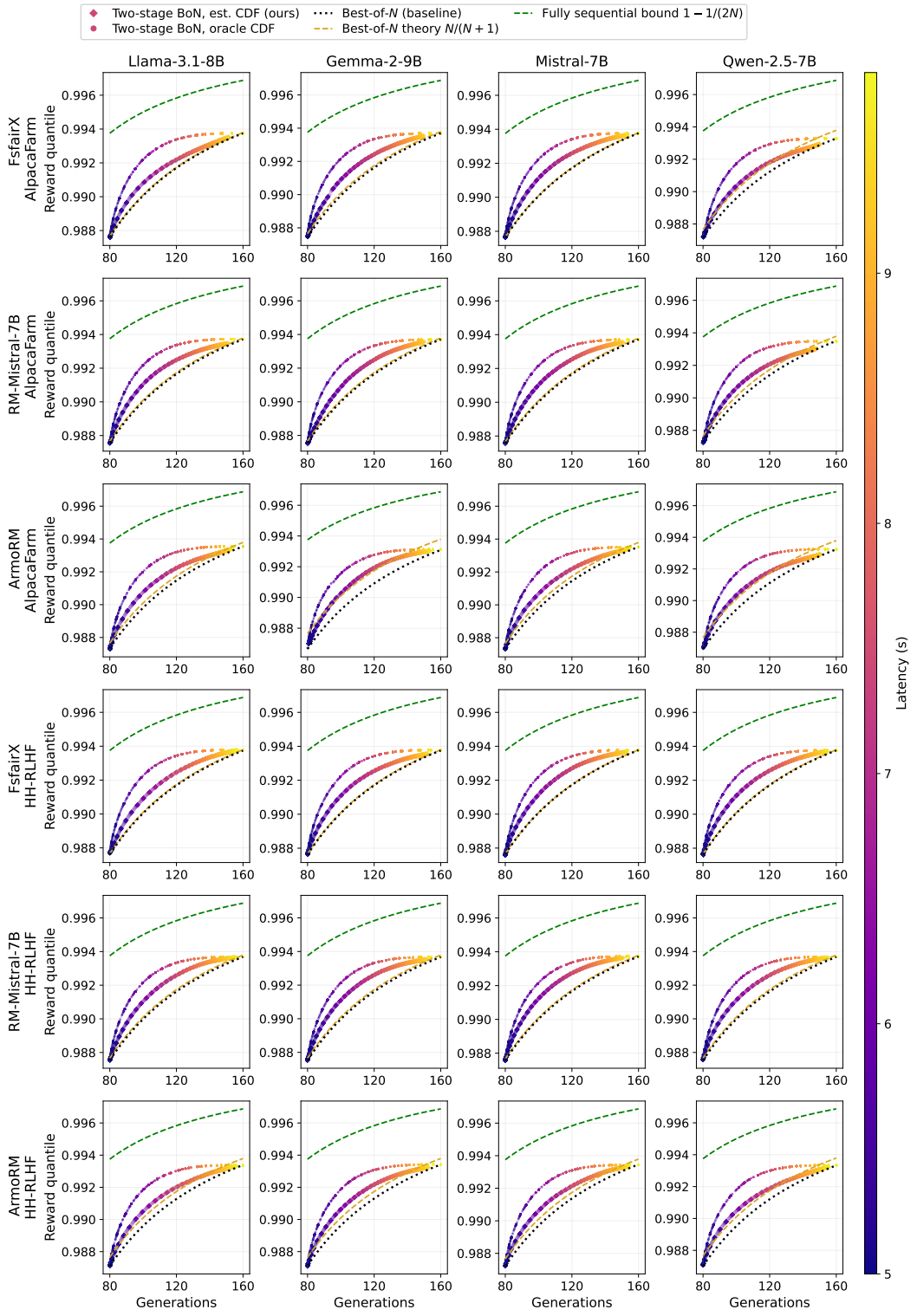


Figure 10: Reward–compute Pareto frontiers in each of the 24 configurations, under Experiment 1 conditions (deterministic latency, $N_1 = N_2 = 80$). Each row of panels is one dataset and reward model, each column one language model. Diamonds: estimated $\hat{F}_x$; circles: oracle F_x; colour encodes mean latency. Dotted black: empirical Best-of-N for that configuration's own reward pool.

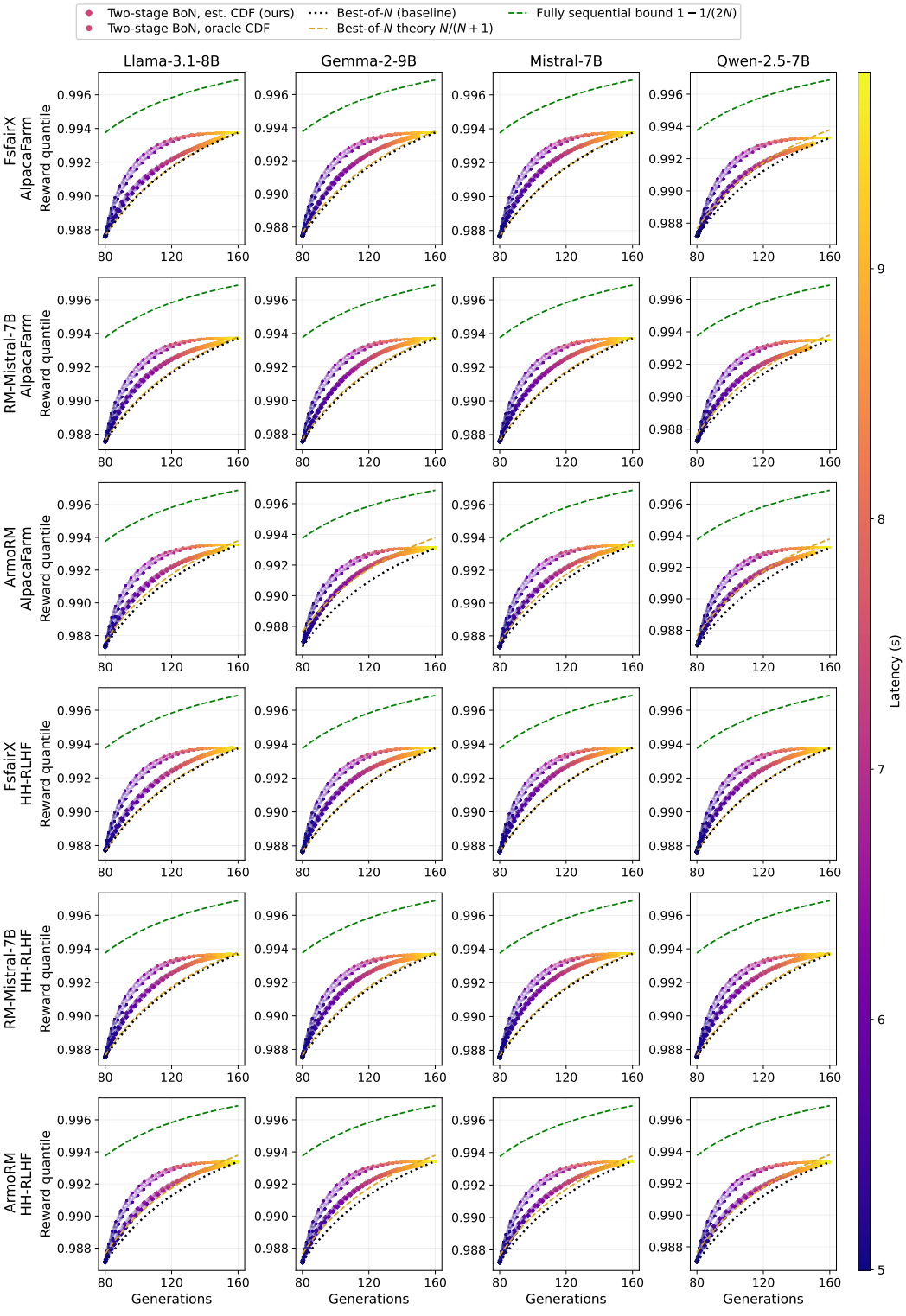


Figure 11: The same 24 configurations under Experiment 2 conditions (log-normal latency, $\overline{T} = 5\,\mathrm{s}$, $c_{\mathrm{v}} = 1$), latency-aware policy. The corresponding latency-unaware comparison is reported numerically in Table 15.

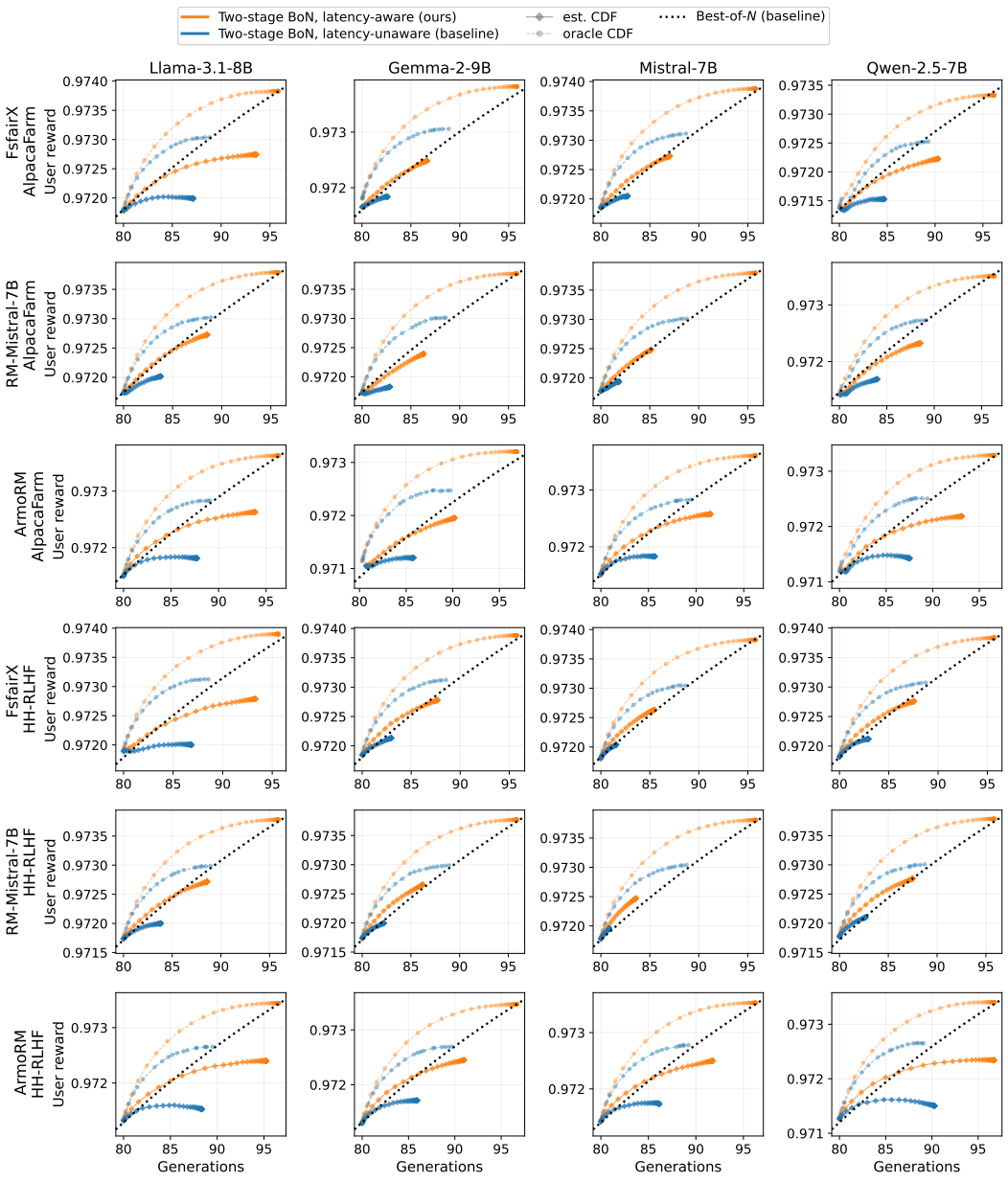


Figure 12: Figure 2 in each of the 24 configurations, at $\lambda_{\text{lat}} = 10^{-2.5}$ and $c_{\text{v}} = 1$: user reward against mean generations, orange for our latency-aware policy and blue for the latency-unaware baseline, solid with the estimated $\hat{F}_x$ and dashed with the oracle F_x; dotted black: parallel Best-of-N on that configuration's own reward pool, charged one batch's latency. Columns: language models; rows: reward model and dataset.

Table 15: Value of conditioning on T_1 in each of the 24 combinations of four language models, three reward models, and two datasets, at a common $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$. Mean generations of the latency-unaware baseline (gens_{un}) and of our latency-aware policy (gens_{aw}), and the aware-minus-unaware difference in reward quantile, latency in seconds, and the objective V of Section 3. Non-adaptive variant, $N_1 = N_2 = 80$, log-normal latency with $\overline{T} = 5\,\text{s}$ and $c_{\text{v}} = 1$, at $\lambda_{\text{gen}} = 10^{-8}$ and $\lambda_{\text{lat}} = 10^{-2.5}$ – the same pair as Table 2, not retuned per configuration.

dataset	RM	LM	gens_{un}	gens_{aw}	Δreward	Δlatency	ΔV
AlpacaFarm	FsfairX	Llama-3.1-8B	87.2	93.6	$+0.00062$	-0.043	$+0.00075$
		Gemma-2-9B	82.6	86.6	$+0.00069$	$+0.015$	$+0.00065$
		Mistral-7B	82.7	87.0	$+0.00071$	$+0.008$	$+0.00068$
		Qwen-2.5-7B	84.7	90.3	$+0.00070$	$+0.001$	$+0.00070$
	RM-Mistral-7B	Llama-3.1-8B	83.8	88.5	$+0.00071$	$+0.001$	$+0.00071$
		Gemma-2-9B	82.9	86.3	$+0.00057$	$+0.001$	$+0.00057$
		Mistral-7B	81.9	85.1	$+0.00061$	$+0.019$	$+0.00055$
		Qwen-2.5-7B	84.0	88.5	$+0.00064$	$+0.001$	$+0.00064$
	ArmoRM	Llama-3.1-8B	87.7	93.9	$+0.00061$	-0.064	$+0.00081$
		Gemma-2-9B	85.8	90.2	$+0.00063$	-0.038	$+0.00075$
		Mistral-7B	85.7	91.5	$+0.00067$	-0.023	$+0.00075$
		Qwen-2.5-7B	87.5	93.1	$+0.00057$	-0.061	$+0.00076$
HH-RLHF	FsfairX	Llama-3.1-8B	86.9	93.3	$+0.00070$	-0.027	$+0.00079$
		Gemma-2-9B	83.0	87.7	$+0.00071$	$+0.019$	$+0.00065$
		Mistral-7B	81.6	85.5	$+0.00069$	$+0.031$	$+0.00060$
		Qwen-2.5-7B	83.0	87.6	$+0.00068$	$+0.014$	$+0.00064$
	RM-Mistral-7B	Llama-3.1-8B	83.8	88.6	$+0.00071$	-0.004	$+0.00072$
		Gemma-2-9B	82.2	86.2	$+0.00072$	$+0.019$	$+0.00066$
		Mistral-7B	80.9	83.6	$+0.00060$	$+0.026$	$+0.00051$
		Qwen-2.5-7B	82.7	87.5	$+0.00073$	$+0.021$	$+0.00066$
	ArmoRM	Llama-3.1-8B	88.4	95.3	$+0.00064$	-0.074	$+0.00087$
		Gemma-2-9B	85.9	91.0	$+0.00060$	-0.042	$+0.00074$
		Mistral-7B	86.1	91.8	$+0.00064$	-0.040	$+0.00076$
		Qwen-2.5-7B	90.3	96.7	$+0.00050$	-0.108	$+0.00084$

exact conditional mean, so the sweep measures what a perfect-but-partial signal is worth, not what a predictor fitted from a server's latency logs would achieve. At $\rho = 0$ it is the constant $\overline{T}$, so the latency-aware policy becomes numerically identical to the latency-unaware baseline; a property we verify numerically rather than assume.

Figure 13 and Table 16 report ΔV against ρ for $c_{\text{v}} \in \{0.5, 1, 2\}$ at the same $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ used in Table 2. The gain decays as T_1 becomes less informative and vanishes at $\rho = 0$, as it must. The remaining panels of Table 16 show the same decay: as ρ falls, the latency-aware policy's generations, reward and latency all move back toward the latency-unaware baseline's, with mean generations falling from 95.2 to the baseline's 84.6. The difference is large in generations but negligible in the objective: at $\lambda_{\text{gen}} = 10^{-8}$, 11 extra generations cost about 10^{-7} in V, against gains of up to 1.4×10^{-3}. Reward and latency are reported separately, rather than only through ΔV, because they move in opposite directions. At every one of the twelve cells with a non-trivial gain, the latency-aware policy attains *both* a higher mean reward and a lower mean latency than the latency-unaware baseline, and both margins widen as ρ grows. A positive ΔV on its own would be equally consistent with a gain in one bought at the expense of the other; the two panels show that no such trade is being made.

The sweep above lets the policy predict T_2 at the true ρ. A practitioner following Section 3 would instead take $\widehat{T}_2 = T_1$ whatever the server's autocorrelation, so Figure 13 also plots that misspecified rule (dotted), with the oracle F_x (left) and the estimated $\hat{F}_x$ (middle). In both it does not merely lose the gain: at $\rho = 0$ it is *worse* than the latency-unaware baseline, by 0.00010 at $c_{\text{v}} = 0.5$ and 0.00078 at $c_{\text{v}} = 2$ with $\hat{F}_x$, and by 0.00028 and 0.00185 with the oracle. Reading a T_1 that carries no information about T_2 makes the stopping decision worse than not reading it at all, and the damage

grows with latency variability. Estimating ρ from latency logs is therefore not an optional refinement on a server whose latencies are weakly autocorrelated.

With the oracle, predicting at the true ρ beats the misspecified rule at every $\rho < 1$, as it must, since it is then the optimal stopping rule. With $\hat{F}_x$ the two cross: at $\rho \in \{0.5, 0.75\}$ the misspecified rule is slightly ahead, by up to 0.00014 (Table 17). This comes from estimating F_x, not from any merit of the assumption. Taking $\widehat{T}_2 = T_1$ makes the predicted latency swing more widely than $\mathbb{E}[T_2 \mid T_1]$ does, and the rule then continues to Stage 2 more often. With the oracle those extra continuations are ones the optimal rule rightly declined, so they cost value. With $\hat{F}_x$, whose fitted tail overrates the best Stage-1 reward, the rule stops too early (Appendix G), and the extra continuations partly make up for it.

The right column of Figure 13 confirms this: once the overrating is removed by the calibration of Appendix F, the true-ρ predictor gives the higher ΔV at $\rho = 0.5$ at every c_v; at $\rho = 0.75$ the two ΔV differ by at most 0.00001 for $c_v \leq 1$, and the true-ρ predictor's is higher by 0.00003 at $c_v = 2$.

The bottom row of Figure 13 repeats the comparison across λ_{lat}, at $\rho = 0.5$. With the oracle, the true-ρ predictor's ΔV is never the lower of the two, and with the calibrated estimate it is never the lower for $c_v \geq 1$. With $\hat{F}_x$, the misspecified rule's ΔV is the higher one at every λ_{lat} up to the $10^{-2.5}$ of the top row and the lower one at every λ_{lat} above it. At small λ_{lat} its extra continuations cost little latency and add reward that the estimated rule was leaving behind; at large λ_{lat} the latency it mispredicts costs more than that reward is worth.

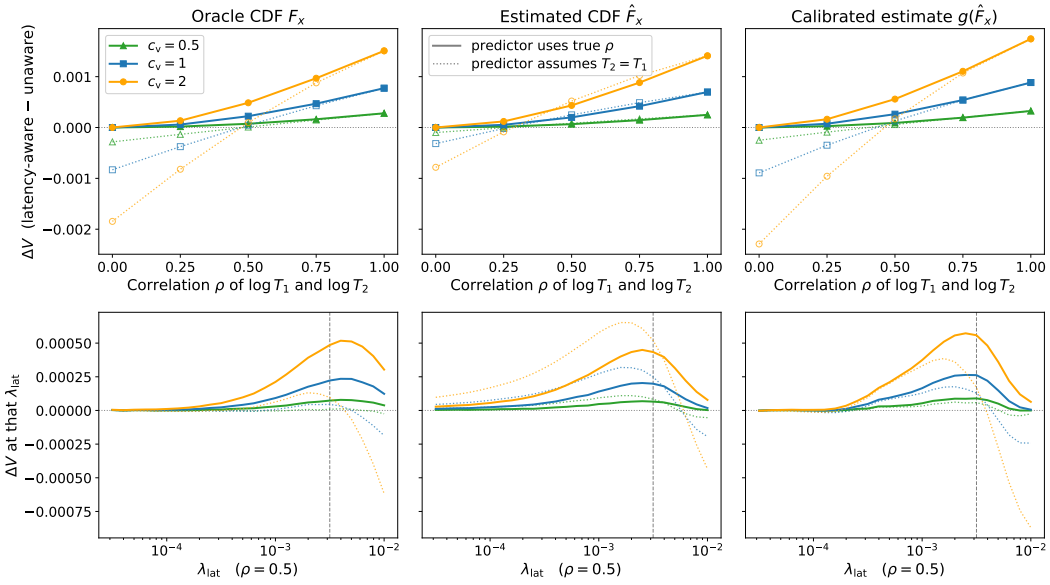


Figure 13: The latency-aware gain ΔV (the improvement in the Section 3 objective over the latency-unaware baseline) when T_1 is only a partial predictor of T_2, under the correlation model of Equation (3). All panels: $N_1 = N_2 = 80$, $\overline{T} = 5\,\text{s}$, $\lambda_{\text{gen}} = 10^{-8}$, averaged over the 24 configurations; colour is the coefficient of variation c_v of latency. *Left column:* oracle F_x. *Middle column:* estimated $\hat{F}_x$. *Right column:* the estimate after the calibration of Appendix F. Solid lines: the policy predicts T_2 by $\mathbb{E}[T_2 \mid T_1]$ at the true ρ; dotted lines: it assumes $T_2 = T_1$. *Top row:* ΔV against ρ, at $\lambda_{\text{lat}} = 10^{-2.5}$. The marginal distribution of T_2 is the same at every ρ, so only the predictability of the second batch's latency changes. At $\rho = 0$ the true-ρ predictor is the constant $\overline{T}$, and the policy provably reduces to the latency-unaware baseline. *Bottom row:* ΔV against λ_{lat}, at $\rho = 0.5$. At each point the policies are run, and V is scored, at that λ_{lat}. The dashed vertical line marks the top row's $\lambda_{\text{lat}} = 10^{-2.5}$.

Table 16: Decay of the latency-aware gain as T_1 becomes less informative about T_2, under the correlation model of Equation (3), averaged over all 24 combinations of four language models, three reward models, and two datasets. Non-adaptive variant, $N_1 = N_2 = 80$, log-normal latency with $\overline{T} = 5\,\text{s}$, at the same $(\lambda_{\text{gen}}, \lambda_{\text{lat}}) = (10^{-8}, 10^{-2.5})$ as Table 2. At $\rho = 0$ the predictor of T_2 collapses to the constant $\overline{T}$ and the policy provably reduces to the latency-unaware baseline, which the $\Delta V = 0$ row confirms numerically, in every configuration. The reward, latency and generations panels are those of the latency-aware policy; ΔV is its gain in the objective of Section 3 over the latency-unaware baseline. Reward and latency are shown separately because they move in opposite directions: at every cell with a non-trivial gain the aware policy attains *both* a higher reward and a lower latency, so the gain is not a trade of one against the other, which ΔV alone could not distinguish. That baseline never observes T_1, so it is invariant in ρ – across all 15 settings it stays within 84.6 mean generations, reward 0.9886, latency 5.287–5.297 s.

ρ	ΔV, $c_{\text{v}} = 0.5$	ΔV, $c_{\text{v}} = 1$	ΔV, $c_{\text{v}} = 2$
0	+0.00000	+0.00000	+0.00000
0.25	+0.00002	+0.00005	+0.00012
0.5	+0.00007	+0.00020	+0.00043
0.75	+0.00014	+0.00042	+0.00088
1	+0.00025	+0.00070	+0.00141
ρ	reward, $c_{\text{v}} = 0.5$	reward, $c_{\text{v}} = 1$	reward, $c_{\text{v}} = 2$
0	0.98855	0.98855	0.98855
0.25	0.98857	0.98860	0.98866
0.5	0.98861	0.98874	0.98896
0.75	0.98869	0.98895	0.98937
1	0.98879	0.98920	0.98983
ρ	latency (s), $c_{\text{v}} = 0.5$	latency (s), $c_{\text{v}} = 1$	latency (s), $c_{\text{v}} = 2$
0	5.288	5.287	5.295
0.25	5.287	5.286	5.292
0.5	5.286	5.284	5.288
0.75	5.285	5.280	5.275
1	5.284	5.273	5.254
ρ	mean gens, $c_{\text{v}} = 0.5$	mean gens, $c_{\text{v}} = 1$	mean gens, $c_{\text{v}} = 2$
0	84.6	84.6	84.6
0.25	84.7	85.0	85.4
0.5	85.1	86.0	87.6
0.75	85.6	87.5	90.9
1	86.3	89.6	95.2

Table 17: ΔV of the latency-aware policy over the latency-unaware baseline under the correlation model of Equation (3), when the policy predicts T_2 by $\mathbb{E}[T_2 \mid T_1]$ at the true ρ and when it assumes $T_2 = T_1$, with the oracle and the estimated CDF; averaged over all 24 combinations of four language models, three reward models, and two datasets. Non-adaptive variant, $N_1 = N_2 = 80$, $c_{\text{v}} = 2$, at $(\lambda_{\text{gen}}, \lambda_{\text{lat}}) = (10^{-8}, 10^{-2.5})$ as in Table 16. Bold: the better predictor of each pair, except at $\rho = 0$, where the true-ρ predictor collapses to the constant $\overline{T}$ and its ΔV is exactly zero rather than merely small. At $\rho = 1$ the two predictors coincide. With the oracle, the true-ρ predictor is ahead at every $\rho < 1$ in every configuration; with $\hat{F}_x$ it falls behind at $\rho \in \{0.5, 0.75\}$.

	oracle F_x		estimated $\hat{F}_x$	
ρ	true ρ	$T_2 = T_1$	true ρ	$T_2 = T_1$
0	+0.00000	−0.00185	+0.00000	−0.00078
0.25	**+0.00013**	−0.00082	**+0.00012**	−0.00009
0.5	**+0.00049**	+0.00010	+0.00043	**+0.00052**
0.75	**+0.00097**	+0.00088	+0.00088	**+0.00102**

K Experiment 2 in more detail: user reward and reward–compute plots

Figure 2 reports Experiment 2 in user reward, $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$, at $\lambda_{\text{lat}} \in \{10^{-3}, 10^{-2.5}\}$. Figure 14 extends it to $\lambda_{\text{lat}} \in \{10^{-4.5}, 10^{-3.5}, 10^{-3}, 10^{-2.5}, 10^{-2}\}$, from the same runs and with the same conventions: each curve sweeps λ_{gen} at a fixed λ_{lat}, and parallel Best-of-N, which draws all its generations in one batch, is charged one batch's latency, $\lambda_{\text{lat}}\overline{T}$, whereas a two-stage policy is charged a second whenever it continues. Figure 15 shows the same runs as reward–compute frontiers, as Figure 1 does for Experiment 1.

Reading the numbers. A policy's *lead* over Best-of-N is its user reward minus Best-of-N's at the same mean number of generations, so positive means the policy is better. The differences are of order 10^{-4}, too small to read off the figure, so Tables 18 and 19 give the best and worst lead along each policy's λ_{gen} sweep, and the *gap* between the latency-aware and latency-unaware policies.

Latency-aware against latency-unaware. At $\lambda_{\text{lat}} = 10^{-4.5}$, and at every smaller value on our grid, latency is too cheap to change any decision and the two policies coincide. From $10^{-3.5}$ on, the latency-aware policy's user reward is at or above the latency-unaware policy's at every generation count the latter reaches, it reaches further to the right, and the gap grows with c_{v}. With $\hat{F}_x$, the largest gap is 0.0001–0.0005 at $\lambda_{\text{lat}} = 10^{-3}$ and 0.0002–0.0006 at $10^{-2.5}$, from $c_{\text{v}} = 0.5$ to 2. It does not keep growing with λ_{lat}: at $\lambda_{\text{lat}} = 10^{-2}$ a second batch is so rarely worth its latency that the latency-unaware policy almost never takes one (at most 80.2 mean generations), the latency-aware policy takes one only when the first batch returns quickly, and the gap falls back to at most 0.0001 (Table 18). The value of conditioning on T_1 is therefore largest at intermediate λ_{lat}, where the decision to continue is close and T_1 can tip it.

Against Best-of-N. With the oracle F_x, the latency-unaware policy stays above Best-of-N at every λ_{lat}, and the latency-aware one does too apart from the right-hand ends of its curves at $c_{\text{v}} = 2$ and $\lambda_{\text{lat}} \geq 10^{-2.5}$, which dip at most 0.00013 below it. With $\hat{F}_x$ the comparison depends on λ_{lat}. At $\lambda_{\text{lat}} \leq 10^{-3.5}$ both policies are above Best-of-N at every generation count they reach, by up to 0.0009. At 10^{-3} they lead it over the left part of each curve, by up to 0.0004, but end up to 0.00024 below it; at $10^{-2.5}$ the latency-aware policy leads by at most 0.00014 and the latency-unaware one never does. The latency-unaware policy's lead is at most 0.00002 at $\lambda_{\text{lat}} = 10^{-2.6}$ and gone by $10^{-2.5}$ at every c_{v}, whereas the latency-aware policy keeps some lead up to about $10^{-2.5}$ at $c_{\text{v}} = 0.5$ and $10^{-2.1}$ at $c_{\text{v}} = 2$. The cause is the second batch's latency: every trial that continues pays λ_{lat} times a second batch's latency that Best-of-N does not, and that charge grows with λ_{lat} while the reward advantage of stopping adaptively does not. The estimated policies, whose reward advantage over Best-of-N is less than half the oracle's, are overtaken first, and first at the right-hand end of each curve, where small λ_{gen} makes them continue most often. Appendix L asks whether this is a consequence of the fixed $N_1 = 80$ or of the latency model.

Reward–compute plots. Figure 15 plots mean reward against mean generations, with mean latency as colour, as Figure 1 does. Unlike there, latency is no longer determined by the number of generations, so these are projections of a three-way tradeoff, not Pareto frontiers. Because the latency-unaware baseline never sees T_1, its curve is the same at every c_{v} and matches the deterministic-latency frontier of Figure 1. As c_{v} grows, the latency-aware policy reaches an additional cluster of operating points with substantially lower mean latency at a small reward cost, visible by $c_{\text{v}} = 2$. The per-configuration version, at $c_{\text{v}} = 1$, is Figure 11.

L User reward against Best-of-N when latency grows with batch size

Appendix K finds that, with the estimated $\hat{F}_x$, the two-stage policies lose their user-reward lead over Best-of-N once latency is expensive: at the $\lambda_{\text{lat}} = 10^{-2.5}$ of Table 2 the latency-aware policy leads it by at most 0.00014 and the latency-unaware one never does, while both oracle versions all but clear it, the latency-aware one dipping 0.00004 below at $c_{\text{v}} = 2$. Two features of the setup could

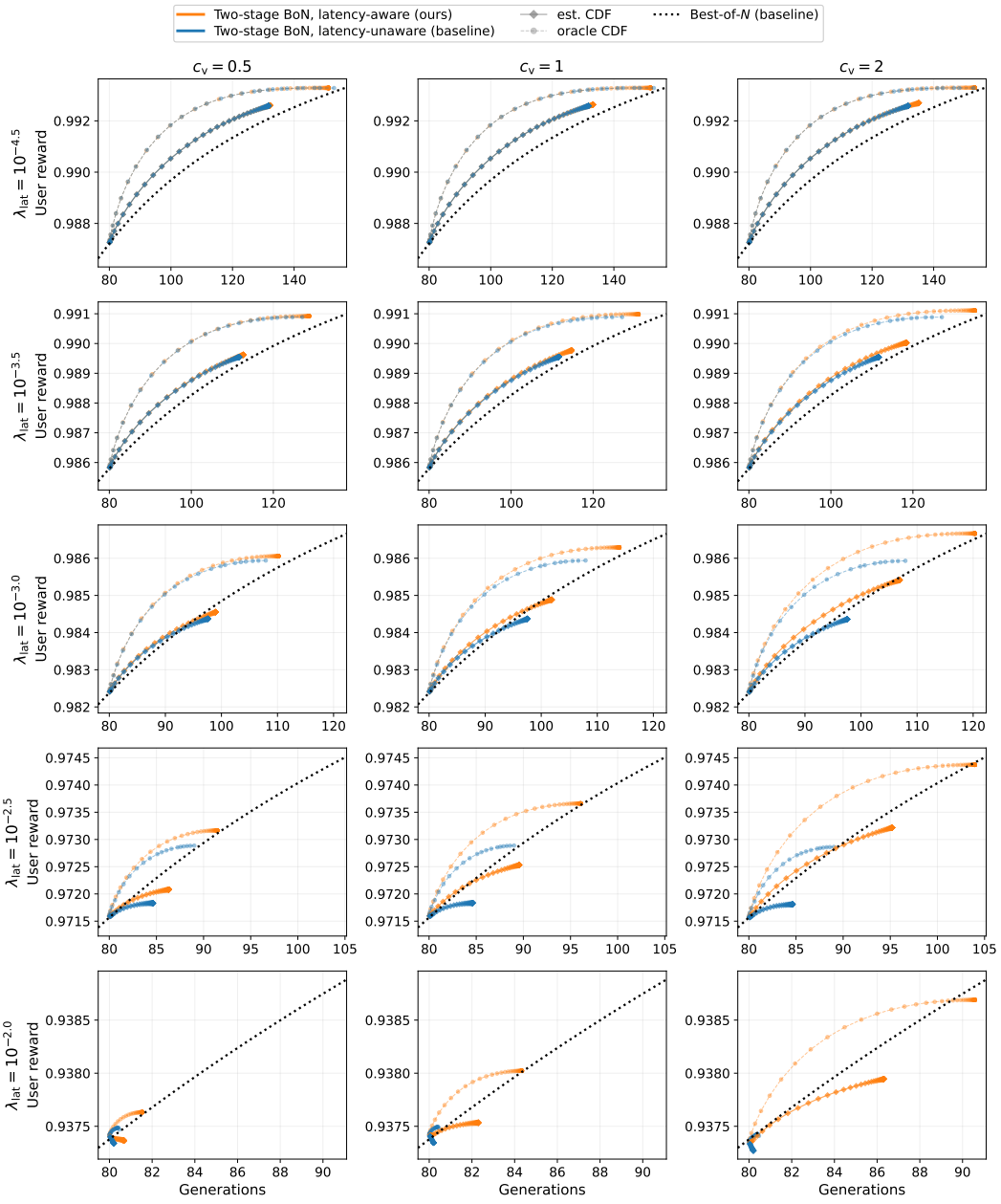


Figure 14: User reward $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$ against mean generations; the dotted line is parallel Best-of-N, charged one batch's latency. Rows: λ_{lat} ($10^{-2.5}$ is the value of Table 2), each with its own y-scale; columns: coefficient of variation c_{v}. Each curve sweeps λ_{gen} at fixed λ_{lat}, with $N_1 = N_2 = 80$; the same runs as Figure 2, averaged over the 24 configurations of Appendix I. Colour is the policy (orange: ours, latency-aware; blue: the latency-unaware baseline), not latency, as in Figure 16; in every user-reward figure warm colours are latency-aware and cool ones latency-unaware, orange and blue with a pre-committed N_2 and red and purple with an adaptive one. Solid lines with diamonds use the estimated $\hat{F}_x$; dashed lines with circles use the oracle F_x. The right-hand end of each estimated curve at $10^{-2.5}$ is the operating point reported in Table 2. Where the two policies coincide, the blue curve covers the orange.

Table 18: Experiment 2 in numbers: how much better each two-stage policy is than parallel Best-of-N, and how much better the latency-aware policy is than the latency-unaware one, at each latency price λ_{lat}. Better means higher *user reward*, $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$: the reward quantile minus the priced latency. A policy's *lead* over Best-of-N is its user reward minus Best-of-N's with the same mean number of generations, so a positive lead means the policy is better. Varying λ_{gen} traces a curve of leads, and "max" and "min" are its best and worst values. Columns: "gap" is the latency-aware policy's user reward minus the latency-unaware policy's at the same mean generations (largest value); the next four columns are the leads over Best-of-N of the two policies with the estimated $\hat{F}_x$; the next two (marked "or.") are the smallest leads of the oracle-F_x policies, and the last is the most generations the latency-unaware policy reaches. Best-of-N is charged one batch's latency. The $N_1 = 80$ end of each curve is left out, since there every policy stops after the first batch and is Best-of-N. "–": the policy never continues. $N_1 = N_2 = 80$, log-normal latency, $\overline{T} = 5$ s. Averaged over all 24 combinations of four language models, three reward models, and two datasets.

		aware − unaware	lead over Best-of-N, est. $\hat{F}_x$				lead, oracle F_x		unaware
λ_{lat}	c_{v}	gap (max)	aware max	aware min	unaware max	unaware min	aware min (or.)	unaware min (or.)	gens
$10^{-4.5}$	0.5	0.00000	+0.00086	+0.00014	+0.00086	+0.00014	+0.00022	+0.00014	131.7
	1	0.00000	+0.00086	+0.00014	+0.00086	+0.00014	+0.00020	+0.00014	131.7
	2	0.00000	+0.00086	+0.00014	+0.00086	+0.00014	+0.00014	+0.00014	131.7
$10^{-3.5}$	0.5	+0.00002	+0.00056	+0.00011	+0.00055	+0.00011	+0.00044	+0.00047	111.7
	1	+0.00006	+0.00058	+0.00011	+0.00055	+0.00011	+0.00036	+0.00047	111.7
	2	+0.00015	+0.00061	+0.00011	+0.00054	+0.00011	+0.00026	+0.00047	111.7
$10^{-3.0}$	0.5	+0.00010	+0.00024	−0.00019	+0.00021	−0.00023	+0.00029	+0.00035	97.6
	1	+0.00026	+0.00029	−0.00014	+0.00021	−0.00023	+0.00023	+0.00035	97.6
	2	+0.00047	+0.00037	−0.00008	+0.00020	−0.00024	+0.00014	+0.00035	97.6
$10^{-2.5}$	0.5	+0.00019	+0.00002	−0.00039	0.00000	−0.00041	+0.00006	+0.00007	84.6
	1	+0.00039	+0.00007	−0.00035	0.00000	−0.00041	+0.00003	+0.00007	84.6
	2	+0.00057	+0.00014	−0.00032	−0.00003	−0.00043	−0.00004	+0.00005	84.6
$10^{-2.0}$	0.5	+0.00005	−0.00008	−0.00012	–	–	+0.00002	–	80.2
	1	+0.00008	0.00000	−0.00019	–	–	+0.00001	–	80.2
	2	+0.00010	−0.00002	−0.00033	–	–	−0.00013	–	80.2

Table 19: Where the two-stage policies stop beating parallel Best-of-N: each policy's best lead (user reward minus Best-of-N's at the same mean generations, best over its λ_{gen} sweep) at every latency price λ_{lat} on the grid from 10^{-3} to 10^{-2}, with the estimated $\hat{F}_x$. Positive means better than Best-of-N; the lead turns negative once latency is expensive enough. Lead, user reward and the omitted $N_1 = 80$ end as in Table 18; $N_1 = N_2 = 80$; "–": the policy never continues. Averaged over all 24 combinations of four language models, three reward models, and two datasets.

λ_{lat}	aware, c_{v}=0.5	1	2	unaware, c_{v}=0.5	1	2
$10^{-3.0}$	+0.00024	+0.00029	+0.00037	+0.00021	+0.00021	+0.00020
$10^{-2.9}$	+0.00018	+0.00024	+0.00032	+0.00015	+0.00015	+0.00014
$10^{-2.8}$	+0.00012	+0.00018	+0.00027	+0.00009	+0.00009	+0.00008
$10^{-2.7}$	+0.00008	+0.00014	+0.00023	+0.00005	+0.00005	+0.00003
$10^{-2.6}$	+0.00004	+0.00010	+0.00018	+0.00002	+0.00002	0.00000
$10^{-2.5}$	+0.00002	+0.00007	+0.00014	0.00000	0.00000	−0.00003
$10^{-2.4}$	0.00000	+0.00005	+0.00010	−0.00003	−0.00003	−0.00006
$10^{-2.3}$	−0.00002	+0.00003	+0.00007	−0.00007	−0.00007	−0.00010
$10^{-2.2}$	−0.00004	+0.00002	+0.00004	−0.00010	−0.00010	−0.00014
$10^{-2.1}$	−0.00006	+0.00001	+0.00001	–	–	–
$10^{-2.0}$	−0.00008	0.00000	−0.00002	–	–	–

be responsible: the latency model of Section 2.2, under which a batch's latency does not depend on its size, so that Best-of-N is charged one batch's latency at every N; and the fixed $N_1 = 80$, which rules out the single larger first batch that Best-of-N uses. This appendix tests both, the first with the fixed-80 policies (Figure 16) and the second with adaptive policies that may choose any batch size up to 320 (Figures 17 and 18). As in Appendix K, a policy's *lead* is its user reward minus Best-of-N's at the same mean number of generations, so positive means the policy is better, and we report the best lead along each policy's λ_{gen} sweep: Table 20 at $\lambda_{\text{lat}} = 10^{-2.5}$, and Table 21 at the other prices of the figures.

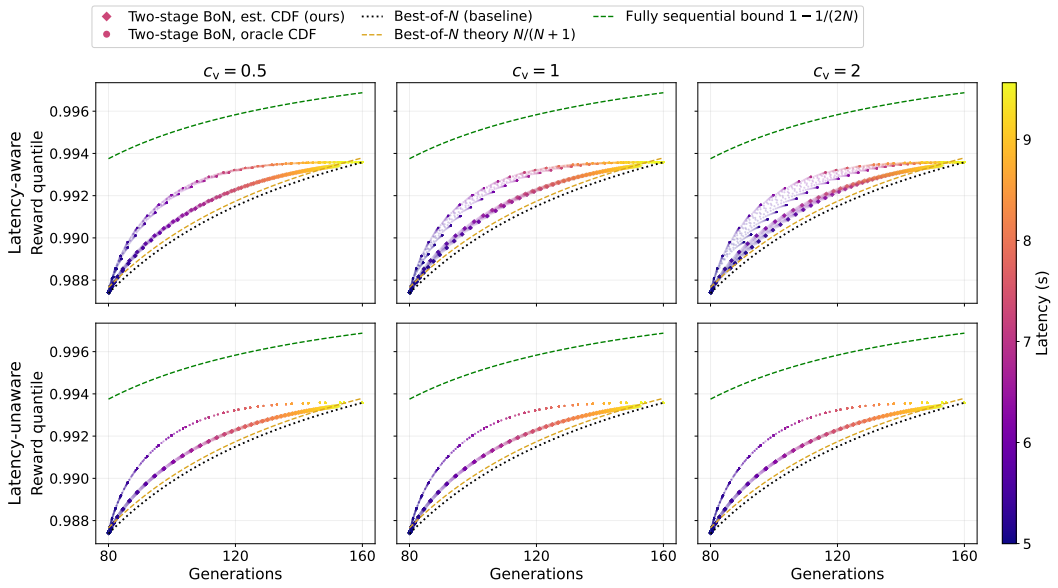


Figure 15: Experiment 2 as reward–compute plots under log-normal latency with mean $\overline{T} = 5\,\mathrm{s}$ and coefficient of variation $c_v \in \{0.5, 1, 2\}$ (left to right); $N_1 = N_2 = 80$; averaged over the 24 configurations of Appendix I. *Top row: our latency-aware policy* (stopping rule uses the measured T_1). *Bottom row: the latency-unaware baseline* (uses only the mean $\overline{T}$), a fixed-threshold stopping rule like existing adaptive methods. Diamonds: estimated $\hat{F}_x$; circles: oracle F_x; colour encodes mean latency. The latency-unaware curves do not change with c_v; as c_v grows, our policy gains operating points with lower mean latency. Reference lines as in Figure 1.

Table 20: Best user-reward lead over parallel Best-of-N at matched mean generations, at $\lambda_{\text{lat}} = 10^{-2.5}$, when a batch of N takes $T\,(N/80)^{\beta}$. User reward is $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$; the lead is the largest difference from Best-of-N's over each policy's λ_{gen} sweep, and a lead of zero or below means the policy never beats Best-of-N at this price. Best-of-N is one batch and is charged $\overline{T}\,(N/80)^{\beta}$. $\beta = 0$ is the latency model of the main experiments. The adaptive policy chooses N_1 offline and N_2 per instance, both from $\mathcal{Z} = \{10, 20, 40, 80, 160, 320\}$. Averaged over all 24 combinations of four language models, three reward models, and two datasets.

			est. $\hat{F}_x$			oracle F_x		
β	N_2 rule	stopping	c_v=0.5	1	2	c_v=0.5	1	2
0	fixed 80	aware	0.0000	+0.0001	+0.0001	+0.0006	+0.0007	+0.0010
		unaware	0.0000	0.0000	0.0000	+0.0005	+0.0005	+0.0004
	adaptive	aware	+0.0003	+0.0005	+0.0008	+0.0042	+0.0045	+0.0050
		unaware	+0.0002	+0.0002	+0.0002	+0.0041	+0.0041	+0.0041
0.5	fixed 80	aware	+0.0002	+0.0006	+0.0011	+0.0012	+0.0016	+0.0022
		unaware	+0.0001	+0.0001	+0.0001	+0.0010	+0.0010	+0.0009
	adaptive	aware	+0.0014	+0.0016	+0.0018	+0.0072	+0.0072	+0.0074
		unaware	+0.0013	+0.0013	+0.0013	+0.0072	+0.0072	+0.0071
1	fixed 80	aware	+0.0009	+0.0015	+0.0027	+0.0023	+0.0032	+0.0047
		unaware	+0.0005	+0.0005	+0.0005	+0.0019	+0.0019	+0.0018
	adaptive	aware	+0.0024	+0.0024	+0.0047	+0.0083	+0.0083	+0.0104
		unaware	+0.0024	+0.0024	+0.0024	+0.0083	+0.0083	+0.0083

Setup. A batch of N generations takes $T\,(N/80)^{\beta}$, where T is the load-dependent latency of a batch of 80, so an 80-batch keeps the mean $\overline{T} = 5\,\mathrm{s}$ of the main experiments. $\beta = 0$ is the model of the main experiments; $\beta = 1$ makes latency proportional to batch size, so two batches of 80 take as long as one of 160. The power law is not meant to model how latency depends on batch

Table 21: Does making batches slower as they grow help the two-stage policies against parallel Best-of-N? Each policy's best lead (user reward minus Best-of-N's at the same mean generations, best over its λ_{gen} sweep; positive means better than Best-of-N) at five latency prices λ_{lat}, when a batch of N takes $T\,(N/80)^{\beta}$: $\beta = 0$ is the main experiments' model, in which batch size does not matter, and $\beta = 1$ makes latency proportional to batch size. Estimated $\hat{F}_x$, $c_{\text{v}} = 1$. The $10^{-2.5}$ rows are the $c_{\text{v}} = 1$ columns of Table 20. Fixed 80: $N_1 = N_2 = 80$; adaptive: N_1 offline and N_2 per instance from $\mathcal{Z} = \{10, 20, 40, 80, 160, 320\}$. A lead that rounds to zero is a tie within Monte Carlo noise. Averaged over all 24 combinations of four language models, three reward models, and two datasets.

λ_{lat}	β	fixed 80, aware	fixed 80, unaware	adaptive, aware	adaptive, unaware
$10^{-4.5}$	0	+0.0009	+0.0009	+0.0024	+0.0024
	0.5	+0.0009	+0.0009	+0.0024	+0.0024
	1	+0.0009	+0.0009	+0.0024	+0.0024
$10^{-3.5}$	0	+0.0006	+0.0005	+0.0019	+0.0019
	0.5	+0.0007	+0.0007	+0.0023	+0.0023
	1	+0.0010	+0.0009	+0.0024	+0.0024
$10^{-3.0}$	0	+0.0003	+0.0002	+0.0013	+0.0011
	0.5	+0.0006	+0.0004	+0.0020	+0.0020
	1	+0.0012	+0.0009	+0.0024	+0.0024
$10^{-2.5}$	0	+0.0001	0.0000	+0.0005	+0.0002
	0.5	+0.0006	+0.0001	+0.0016	+0.0013
	1	+0.0015	+0.0005	+0.0024	+0.0024
$10^{-2.0}$	0	0.0000	0.0000	+0.0001	0.0000
	0.5	+0.0005	0.0000	+0.0011	+0.0005
	1	+0.0012	+0.0001	+0.0027	+0.0024

size in any real serving stack. It is a one-parameter interpolation between the two extremes, latency independent of batch size (parallel, $\beta = 0$) and latency proportional to it (sequential, $\beta = 1$); $\beta = 0.5$ is a point between them, not a measured value. This gives the latency model $\widehat{T}_2(s_1, N_2)$ of Section 3 a dependence on N_2 that spans the two extremes: the latency-aware rule predicts a second batch of N_2 to take $T_1(N_2/80)^{\beta}$, the latency-unaware rule $\overline{T}(N_2/80)^{\beta}$, and the first batch size is chosen offline against $\overline{T}(N_1/80)^{\beta}$. Best-of-$N$ is charged $\overline{T}(N/80)^{\beta}$. The fixed $N_1 = N_2 = 80$ policies of Experiment 2 need no new runs: every one of their batches is the reference size, so their results are the same under every β, and only Best-of-N's charge changes.

Adaptive policies and reporting. For the adaptive policies we run the adaptive variant of Section 3 with $\mathcal{Z}_1 = \mathcal{Z}_2 = \mathcal{Z} = \{10, 20, 40, 80, 160, 320\}$ – N_1 chosen offline from $\mathcal{Z}$, N_2 chosen per instance from $\mathcal{Z}$ – at $\beta \in \{0, 0.5, 1\}$ and $\lambda_{\text{lat}} \in \{10^{-6.5}, 10^{-5.5}, \ldots, 10^{-2.5}\}$, and at $c_{\text{v}} = 1$ also at 10^{-3} and 10^{-2}, the remaining rows of Figure 17. Everything else is as in Experiment 2, averaged over the 24 configurations. The two figures take the rows of Figure 14 and the columns β, at $c_{\text{v}} = 1$; Table 20 gives every policy's best lead over Best-of-N at $\lambda_{\text{lat}} = 10^{-2.5}$ for each c_{v}. Unless stated otherwise, the numbers below are for $c_{\text{v}} = 1$ and the estimated $\hat{F}_x$.

What β means. β indexes how much of a batch is effectively serialized, from none ($\beta = 0$: the generations are decoded together and the batch's time barely depends on its size, as in memory-bound decoding under continuous batching) to all of it ($\beta = 1$: the generations compete for the throughput of a saturated server). A real serving stack need not follow a power law in N, and a simulation cannot say where between the extremes one falls; that is one of the things an evaluation on a live serving stack should measure. What the results below show is how the policies' advantage over Best-of-N changes as latency goes from independent of batch size to proportional to it.

The latency model. Under $\beta > 0$, Best-of-N is charged more than one 80-batch's latency whenever $N > 80$, and the extra grows with N, so a fixed-80 policy gains the more, the further to the right its curve reaches. The gain is also scaled by λ_{lat} (Figure 16 and Table 21). At $\lambda_{\text{lat}} = 10^{-4.5}$ the latency-aware policy's best lead is 0.0009 under every β. From $\beta = 0$ to $\beta = 1$ it grows from 0.0006 to 0.0010 at $10^{-3.5}$, from 0.0003 to 0.0012 at 10^{-3}, from 0.0001 to 0.0015 at $10^{-2.5}$, and from nothing to 0.0012 at 10^{-2}; at $\beta = 0.5$ it lies between the two, for instance 0.0006 at $10^{-2.5}$ (Ta-

ble 20), as an interpolation should. The latency-unaware policy gains less: from nothing to 0.0005 at $10^{-2.5}$, and hardly anything at 10^{-2}, where it almost never continues past 80 generations and so never reaches the sizes at which Best-of-N's charge grows. The latency-aware policy's advantage over the latency-unaware one therefore widens with λ_{lat} once latency grows with batch size, and with c_{v} (Table 20): at $\beta = 1$ and $\lambda_{\text{lat}} = 10^{-2.5}$ it leads Best-of-N by up to 0.0009, 0.0015 and 0.0027 at $c_{\text{v}} = 0.5$, 1 and 2, against at most 0.0005 for the latency-unaware policy.

The batch sizes. Under $\beta = 0$, the larger batch-size set $\mathcal{Z}$ helps the estimated policies only modestly at $\lambda_{\text{lat}} = 10^{-2.5}$: their best lead over Best-of-N never exceeds 0.0008 at any c_{v}, against at most 0.00014 for the fixed-80 policies. With the oracle F_x the same larger action set raises the best lead from 0.0004–0.0010 to 0.004–0.005, so under this latency model what holds the estimated policies back is the estimate of F_x, not the batch sizes available. Figure 17 shows the adaptive policies' user reward, and Figure 18 its gap to Best-of-N, which shows where their lead comes from. At $\lambda_{\text{lat}} = 10^{-4.5}$ it is largest, up to 0.0024, at about 25 mean generations, where the policy draws a small first batch and continues only when its best reward is poor. Under $\beta = 0$ that lead shrinks as λ_{lat} grows (Table 21), to 0.0005 for the latency-aware policy and 0.0002 for the latency-unaware one at $10^{-2.5}$, because every continuation pays a second batch's latency. Under $\beta = 1$ a first and a second batch cost no more latency, in expectation, than one batch of their combined size, and at $10^{-2.5}$ both adaptive policies recover the 0.0024 lead of the small-λ_{lat} rows; the latency-aware one leads by up to 0.0047 at $c_{\text{v}} = 2$.

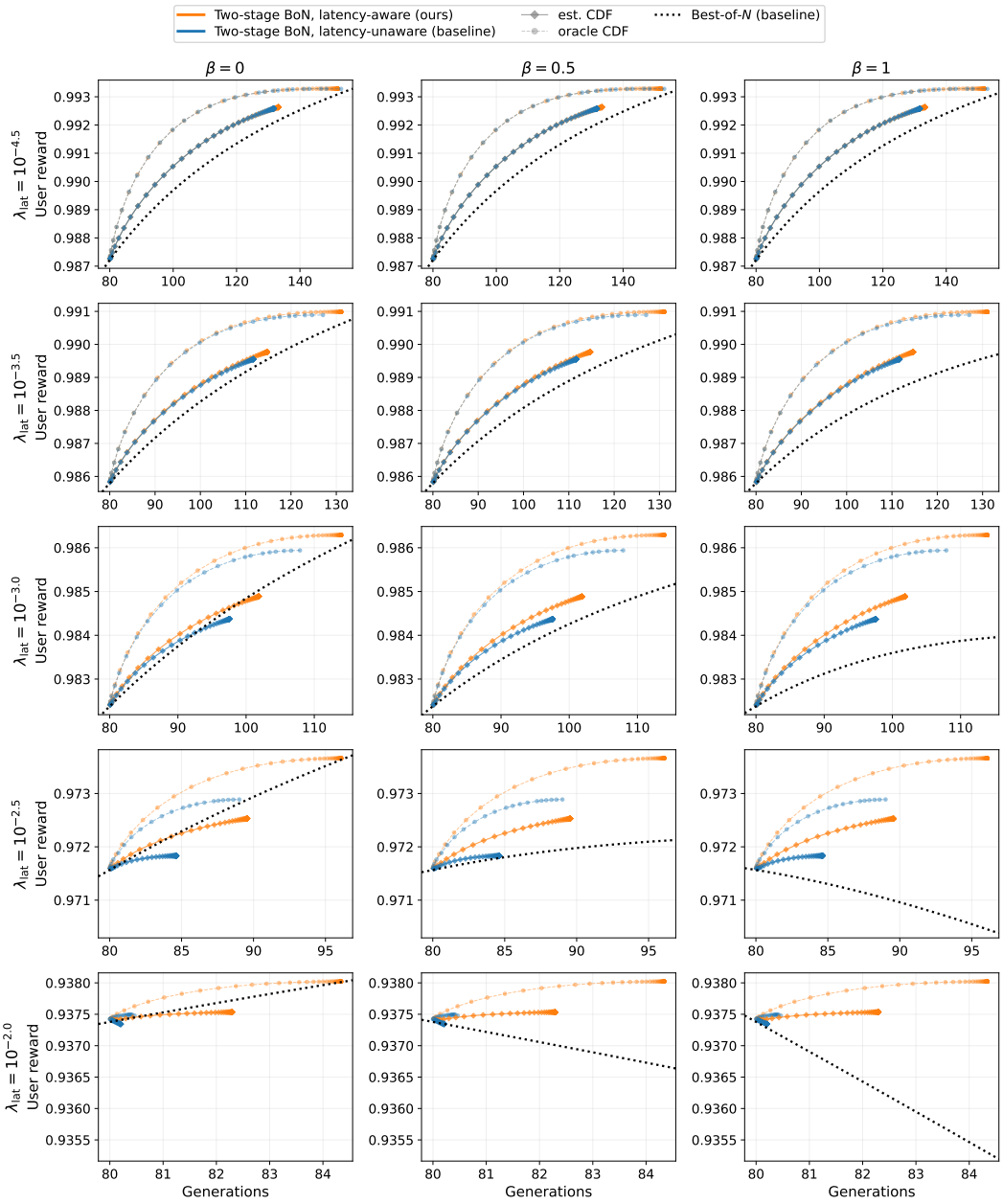


Figure 16: User reward against mean generations for the fixed $N_1 = N_2 = 80$ policies of Experiment 2, when a batch of N takes $T\,(N/80)^{\beta}$; the dotted line is parallel Best-of-N, charged $\lambda_{\text{lat}}\overline{T}(N/80)^{\beta}$. Rows: λ_{lat}, as in Figure 14, each with its own y-scale; columns: β; $c_{\text{v}} = 1$. Averaged over the 24 configurations; colours and line styles as in Figure 14. Within each row the policies' curves are the same in every column, since all their batches are the reference size; only Best-of-N's line moves, falling as β grows and the more so the larger λ_{lat}. The $\beta = 0$ column is the $c_{\text{v}} = 1$ column of Figure 14.

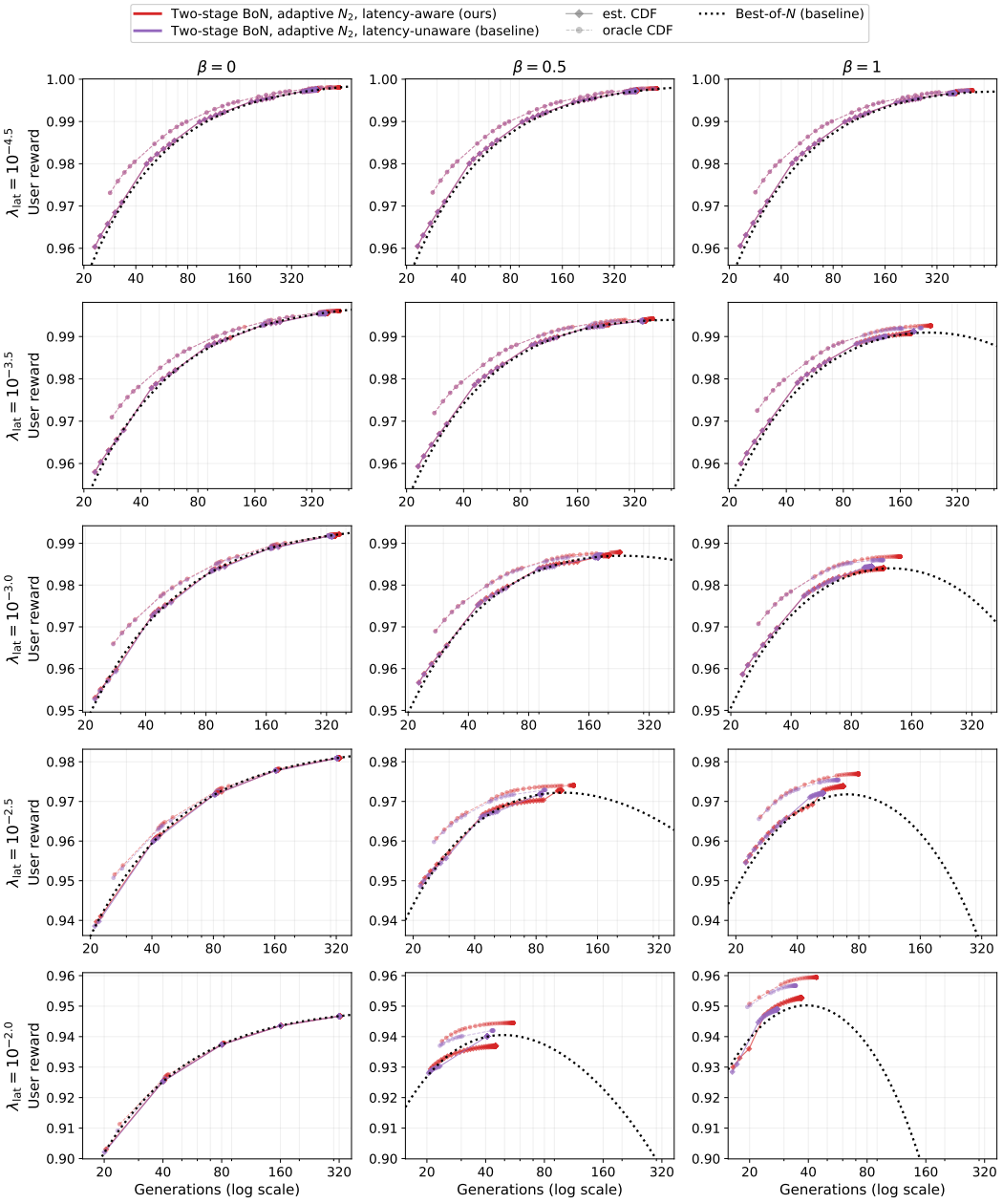


Figure 17: As Figure 16, for the adaptive policies, which choose N_1 offline and N_2 per instance from $\mathcal{Z} = \{10, 20, 40, 80, 160, 320\}$, in red (latency-aware) and purple (latency-unaware); mean generations on a log scale. Unlike the fixed-80 policies, these weigh each batch size against the latency it costs, so their curves change with β as well as Best-of-N's line does. Under $\beta > 0$ Best-of-N's user reward rises and then falls: each further generation adds less reward, but at $\beta = 1$ the same latency charge. The y-scale follows the policies' curves, and beyond them Best-of-N's line runs off the axes. Figure 18 shows the same curves as the gap to Best-of-N.

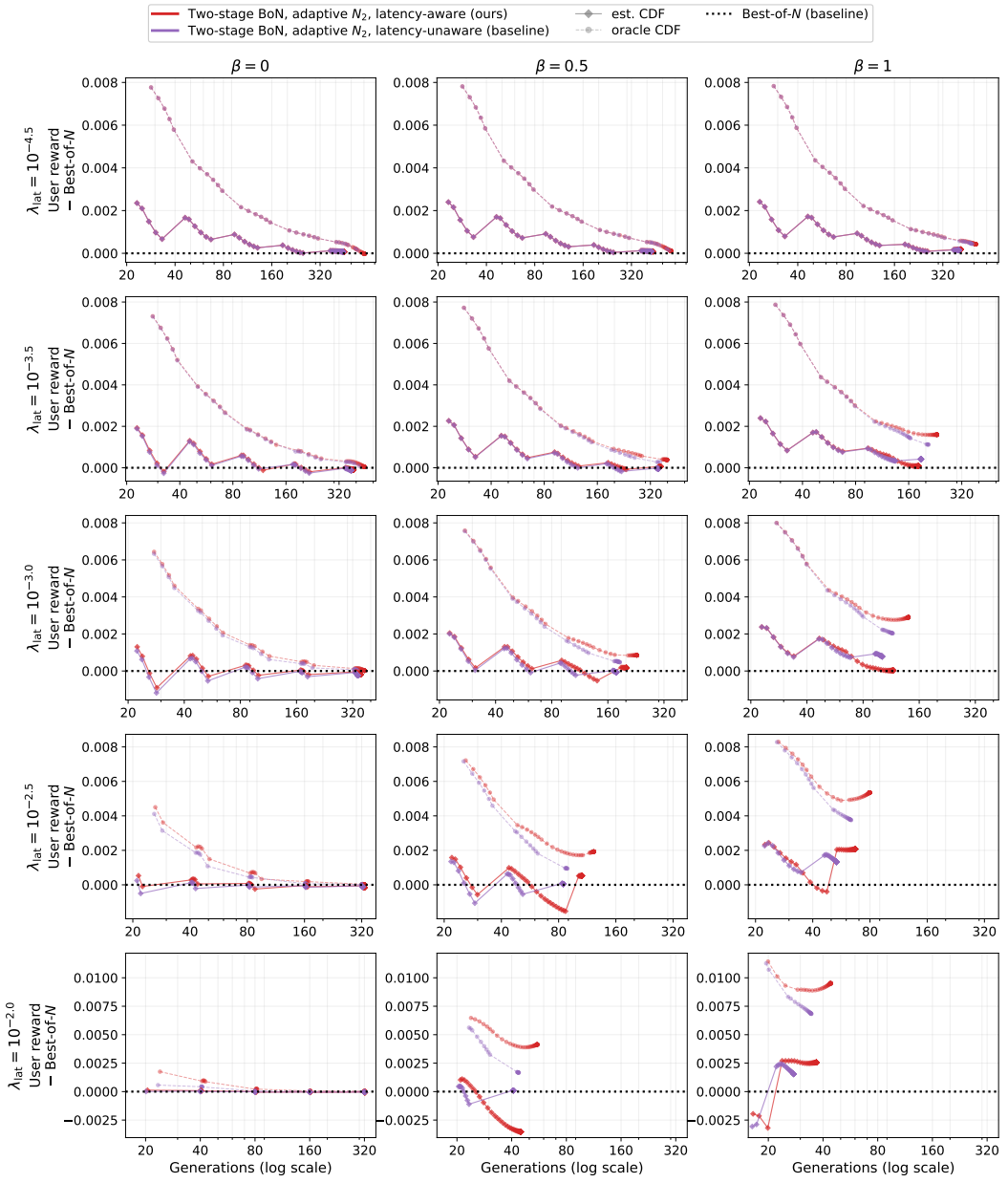


Figure 18: The curves of Figure 17 minus Best-of-N's user reward at the same mean generations, so that the dotted zero line is Best-of-N; the gaps of 0.0005–0.0025 are small beside Best-of-N's rise of about 0.045 over 20 to 400 generations. The sawtooth comes from the policy switching its first batch between the sizes in $\mathcal{Z}$ as λ_{gen} varies; the dips below Best-of-N come from the estimate of F_x, and are absent from the oracle curves, which stay at or above Best-of-N, to within 0.0001, throughout.

M VALUE OF AN ADAPTIVE SECOND-BATCH SIZE

We compare the adaptive and non-adaptive N_2 policies under deterministic latency, $T_1 = T_2 = \overline{T} = 5$ s, so that latency-aware and latency-unaware coincide and the comparison is purely about N_2 adaptivity. The action set is $\mathcal{Z} = \{10, 20, 30, \ldots, 80\}$ for both stages. In the adaptive variant, N_1 is selected by maximizing $Q_0^\pi(s_0, N_1)$ over $N_1 \in \mathcal{Z}$ as in Section 3, and $N_2 \in \mathcal{Z}$ is selected on a per-trial basis from the realized Stage 1 rewards; in the non-adaptive variant, (N_1, N_2) are jointly selected offline from $\mathcal{Z}$. Each policy is evaluated with both the estimated $\hat{F}_x$ and the oracle reward quantile over $\mathcal{R}_b$, as in Experiment 1.

Figure 19 shows the resulting Pareto frontiers. Averaged over the 24 configurations, the adaptive variant's frontier is at or above the non-adaptive one's almost everywhere – never below it by more than 0.0002 – and ahead by up to 0.0015 in reward quantile with the estimated $\hat{F}_x$ (0.0009 with the oracle). The advantage is concentrated at low compute, between about 20 and 70 mean generations, where generations are expensive and a smaller second batch pays; at higher compute the two frontiers nearly coincide. This anticipates the finding below: adapting N_2 matters when generations are expensive, and conditioning on T_1 when latency varies.

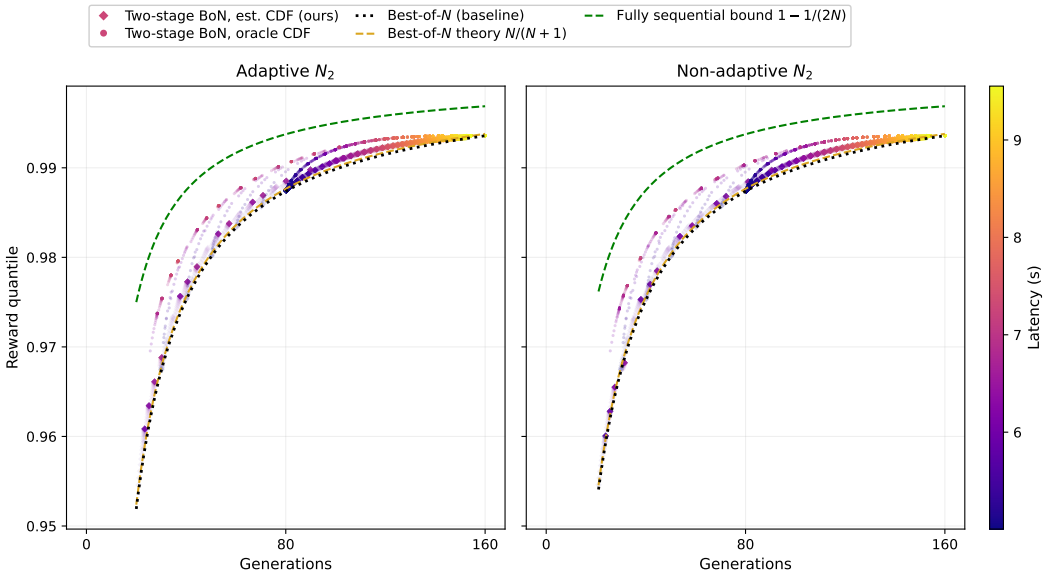


Figure 19: Ablation on the value of an adaptive second-batch size, under deterministic latency ($T_1 = T_2 = 5$ s, so latency-aware and latency-unaware coincide and the comparison is purely about N_2 adaptivity), averaged over the 24 configurations. Left: *our adaptive variant*, with N_2 selected per-instance from the observed Stage 1 rewards. Right: the non-adaptive variant, with (N_1, N_2) jointly pre-selected offline. Diamonds: estimated $\hat{F}_x$; circles: oracle F_x; colour encodes mean latency. The adaptive variant's frontier is at or above the non-adaptive one's almost everywhere, ahead by up to 0.0015 at low compute, and the two nearly coincide at higher compute (see text). Dashed reference lines as in Figure 1; dashed green is the fully sequential upper bound $1 - 1/(2N)$ (Appendix C).

Under variable latency. Deterministic latency is the case in which the two rules have the least room to differ, since every trial faces the same latency. We therefore repeat the comparison under the log-normal latency model of Experiment 2, with $c_v \in \{0.5, 1, 2\}$ and $\mathcal{Z} = \{10, 20, \ldots, 80\}$, running all four combinations of {adaptive, non-adaptive} N_2 with {latency-aware, latency-unaware} stopping. Figures 20 and 21 and Table 22 report the result. At the $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ of Table 2 the two rules coincide *exactly*, at every c_v. This is a property of that operating point rather than evidence that adaptivity is worthless: at $\lambda_{\text{gen}} = 10^{-8}$ the cost of a generation is negligible, so whenever the rule continues, the N_2 maximising $\Delta(u^*, N_2) - \lambda_{\text{gen}} N_2 - \lambda_{\text{lat}} T_1$ is the largest element of $\mathcal{Z}$ – which is exactly the pre-selected non-adaptive N_2. The adaptive rule has nothing to adapt there.

Table 22: Reward-adaptive versus non-adaptive N_2 under log-normal latency, for the latency-aware policy, averaged over all 24 combinations of four language models, three reward models, and two datasets. Figure 19 compared these only under deterministic latency; this tests whether adapting N_2 still pays once latency is stochastic. Entries are the adaptive policy's *lead* in user reward $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$ over the non-adaptive one at the same mean number of generations, so positive means adapting N_2 is better. Sweeping λ_{gen} traces the curves of Figure 21; their largest lead is given, with the mean generations at which it occurs. The smallest is zero everywhere: adapting N_2 is never worse, anywhere on the grid, by more than 4×10^{-6}. In units of 10^{-4}. $\mathcal{Z} = \{10, 20, \ldots, 80\}$, log-normal latency with $\overline{T} = 5$ s. At the single $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pair of Table 2 the two rules coincide *exactly*: $\lambda_{\text{gen}} = 10^{-8}$ makes a generation nearly free, so whenever the adaptive rule continues it selects the largest $N_2 \in \mathcal{Z}$, which is the pre-selected non-adaptive N_2; the gain of Section 4.3 is therefore attributable to T_1-conditioned stopping alone.

		largest lead		
λ_{lat}	c_{v}	est. $\hat{F}_x$	oracle F_x	at gens
$10^{-4.5}$	0.5	+13.9	+3.4	30
	1	+13.9	+3.4	30
	2	+13.9	+3.4	30
$10^{-3.5}$	0.5	+10.3	+3.4	25
	1	+10.6	+3.4	25
	2	+11.1	+3.5	25
$10^{-3.0}$	0.5	+5.6	+2.0	24
	1	+6.4	+2.2	24
	2	+8.0	+2.4	24
$10^{-2.5}$	0.5	+1.0	+0.7	31
	1	+1.0	+0.9	32
	2	+2.0	+1.1	33
$10^{-2.0}$	0.5	0.0	0.0	40
	1	+0.1	+0.2	40
	2	+1.1	+0.6	31

Away from that pair it does help, and the driver is λ_{gen}, not latency variability. Sweeping λ_{gen} at fixed c_{v}, the two rules are identical on every grid cell with $\lambda_{\text{gen}} \leq 10^{-8}$; by $\lambda_{\text{gen}} \approx 10^{-3}$ the adaptive rule is ahead on every cell, by $+0.00101$ on average and $+0.00125$ at best – larger than the $+0.00070$ that conditioning on T_1 earns at $c_{\text{v}} = 1$. The profile is almost unchanged across latency variability (mean gain $+0.00099$, $+0.00101$, $+0.00104$ at $c_{\text{v}} = 0.5, 1, 2$), so the value of adapting N_2 is essentially orthogonal to how variable latency is. Run with the oracle F_x the gain is smaller and never negative (at most $+0.00036$); the larger differences under the estimated $\hat{F}_x$, in both directions, are estimator error rather than a property of adaptivity.

Figure 20 shows the same comparison in user reward, $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$, as in Appendix K, and Figure 21 its gap to Best-of-N, on which the two policies separate visibly. The adaptive policy is at or above the non-adaptive one at every generation count, and its lead is concentrated between about 25 and 35 mean generations, where generations are expensive: up to 0.0014 at $\lambda_{\text{lat}} = 10^{-4.5}$ and 0.0010–0.0011 at $10^{-3.5}$, whatever c_{v}. It shrinks as latency gets dearer, to at most 0.0002 at $10^{-2.5}$, because a dear second batch is taken rarely, whatever its size. With the oracle F_x the lead is at most 0.0004. Table 22 gives these leads at every latency price.

Two conclusions follow. The improvement reported in Section 4.3 is attributable to T_1-conditioned stopping, since adaptive N_2 is exactly inert at that λ pair. But adapting N_2 is worth roughly as much as conditioning on T_1 once generations are expensive, and the two mechanisms are complementary rather than substitutes; a policy for the expensive-generation regime should use both.

N Random draws and seeding

Every random draw in the simulations comes from a generator with a fixed seed, so each run reproduces exactly, and the same draws are shared across all the policies being compared (common

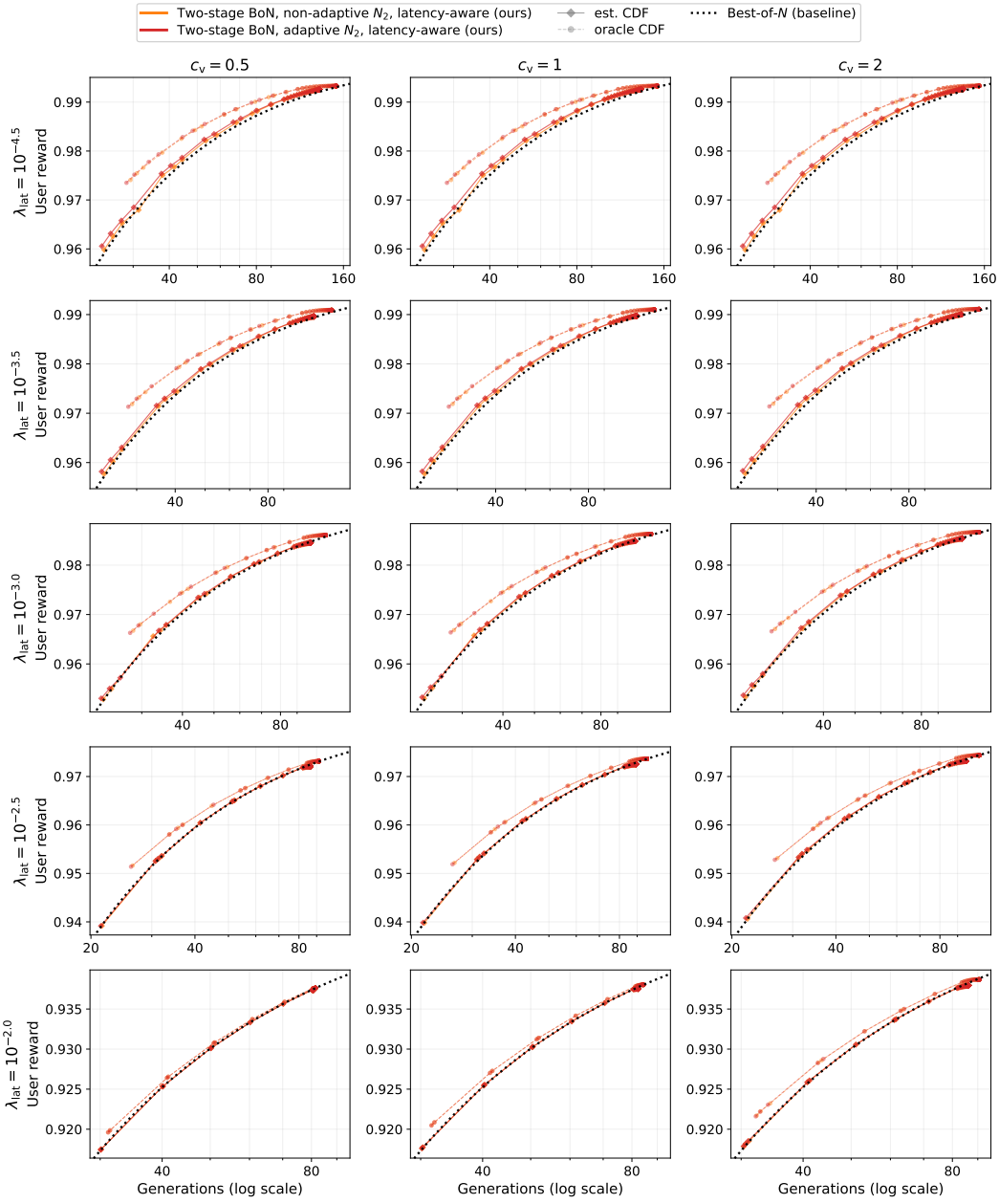


Figure 20: Adaptive against non-adaptive N_2 under log-normal latency, in user reward $\overline{F_x(R)} - \lambda_{\text{lat}}\overline{T}$ against mean generations; the dotted line is parallel Best-of-N, charged one batch's latency. Latency-aware policies with $\mathcal{Z} = \{10, 20, \ldots, 80\}$: red, N_2 chosen per instance; orange, (N_1, N_2) pre-selected offline. Rows: λ_{lat}, as in Figure 14, each with its own y-scale; columns: c_v; mean generations on a log scale. Each curve sweeps λ_{gen}; averaged over the 24 configurations. Solid lines with diamonds use the estimated $\hat{F}_x$; dashed lines with circles use the oracle F_x. Where the two rules coincide, the red curve covers the orange. The y-scale follows the policies' curves, and beyond them Best-of-N's line runs off the axes. The gaps between the policies are small beside Best-of-N's own rise; Figure 21 shows them as the difference from Best-of-N.

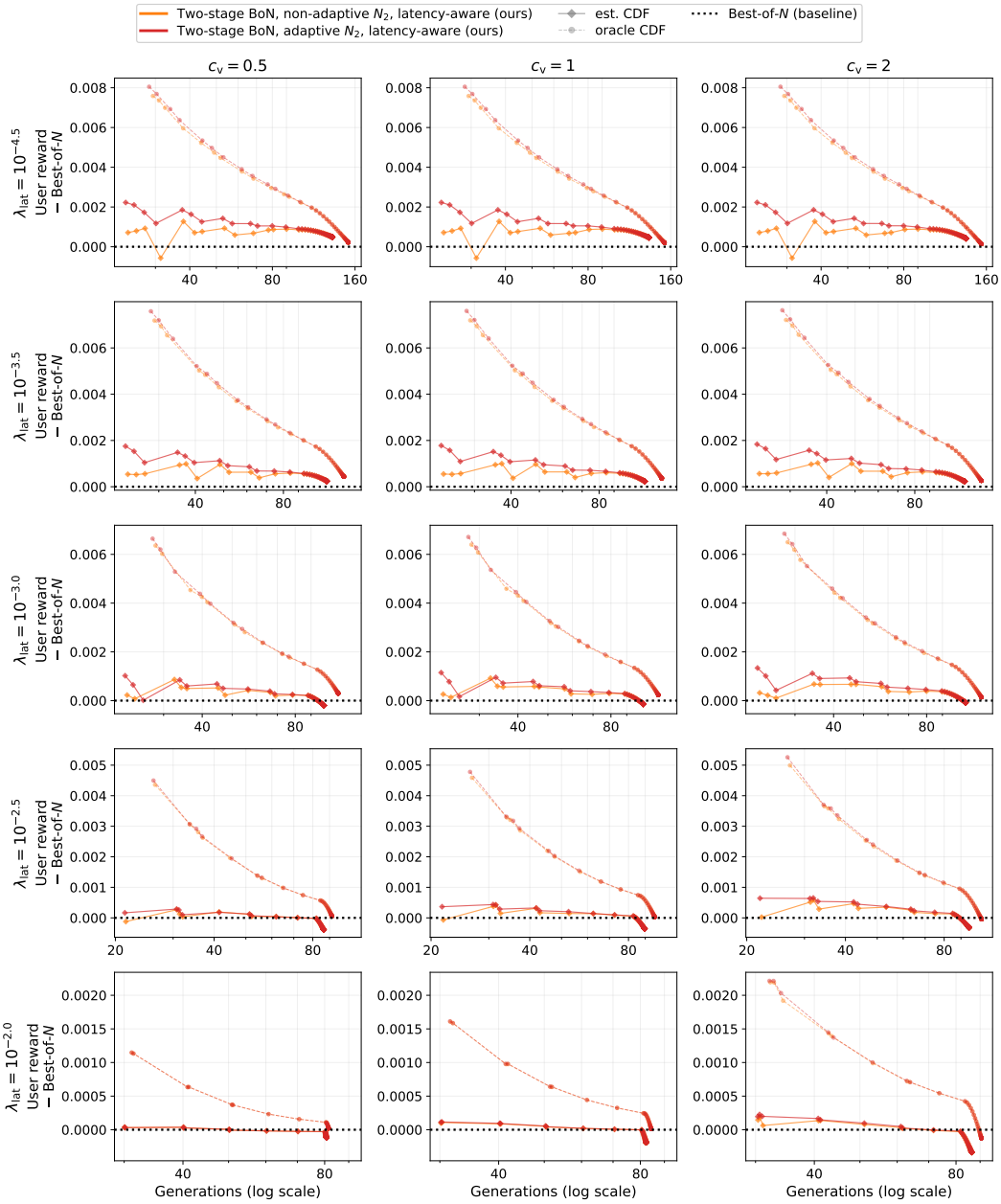


Figure 21: The curves of Figure 20 minus Best-of-N's user reward at the same mean generations, so that the dotted zero line is Best-of-N; colours, line styles, rows and columns as there. Where the two rules coincide, the red curve covers the orange. This isolates the interaction between latency *variability* and adaptivity of N_2. It does not show the chosen positive N_2 varying with T_1: under the Section 3 formulation $-\lambda_{\text{lat}} T_1$ is constant across all positive N_2, so T_1 moves the stop-versus-continue decision only, and capturing that interaction would need a $\widehat{T}_2$ that depends on N_2, as in Appendix L.

random numbers). The latencies $T_{1,i}$, and $T_{2,i}$ when $\rho < 1$ (Appendix J), are drawn once per prompt and trial. Trial i draws one uniformly random ordering of the pool $\mathcal{R}_b$: its first N_1 entries form the Stage-1 set $A_{b,i}$ and the next $N_{2,i}$ the second batch $B_{b,i}$, which is equivalent to the two draws without replacement of Section 4.1. The same orderings index every prompt's pool; since each pool holds independent generations, this leaves each prompt's draws with the same distribution. These latencies and orderings are reused for every policy, every $(\lambda_{\text{gen}}, \lambda_{\text{lat}})$ pair, and both the estimated and oracle variants. The empirical Best-of-N reference curves involve no random draws: they are computed exactly (Appendix B). Because the second batch continues the first batch's ordering, trial i sees the same rewards whatever N_1 and $N_{2,i}$ a policy chooses. A difference between two policies therefore reflects only their decisions, not different random draws, which is why the latency-aware policy at $\rho = 0$ matches the latency-unaware baseline exactly rather than to within Monte Carlo noise. The latency samples used for the offline choice of batch sizes (Appendix A) come from a separately seeded generator, so the chosen N_1 and N_2 are also reproducible. All 24 configurations of Appendix I use the same seed.

O Dataset and Model Asset Details

All experiments reuse the pools of generations and reward scores released by Kalayci et al. (2025): for each of 100 prompts from each dataset, 960 responses sampled from each language model and scored by each reward model. We use every combination of the four language models, three reward models, and two datasets below, 24 configurations in all (Section 4.1), and generate or score no new responses. The three reward models are those used by Sun et al. (2024).

Datasets. We use prompts from the following datasets, 100 from each:

- **AlpacaFarm**: `https://github.com/tatsu-lab/alpaca_farm`, Apache 2.0 License.
- **HH-RLHF**: `https://huggingface.co/datasets/Anthropic/hh-rlhf`, MIT License.

We did not perform any additional data scraping.

Language Models (LMs). Responses were generated by the following publicly available language models:

- **Meta-Llama-3.1-8B-Instruct**: `https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`, Llama 3.1 Community License.
- **Gemma-2-9B-it**: `https://huggingface.co/google/gemma-2-9b-it`, Gemma Terms of Use.
- **Mistral-7B-Instruct-v0.3**: `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3`, Apache 2.0 License.
- **Qwen2.5-7B-Instruct**: `https://huggingface.co/Qwen/Qwen2.5-7B-Instruct`, Apache 2.0 License.

Reward Models (RMs). Responses were scored by the following publicly available reward models:

- **FsfairX-LLaMA3-RM-v0.1**: `https://huggingface.co/sfairXC/FsfairX-LLaMA3-RM-v0.1`, CC-BY-NC-4.0 License.
- **RM-Mistral-7B**: `https://huggingface.co/weqweasdas/RM-Mistral-7B`; no license is stated on the model card.
- **ArmoRM-Llama3-8B-v0.1**: `https://huggingface.co/RLHFlow/ArmoRM-Llama3-8B-v0.1`, Llama 3 Community License.

For all models and datasets, we follow their licensing terms and acknowledge the original sources.

P Compute Infrastructure

All experiments are simulations over precomputed rewards and run on an internal university CPU cluster; no GPU is needed. A run fixes one configuration, latency model, and set of policies, and is one SLURM job with 20 CPU cores and 20 GB of memory. Within it, every combination of policy, λ_{gen}, λ_{lat}, and estimated or oracle CDF is an independent task of 1,000 Monte Carlo trials for each of the 100 prompts – between 2,840 and 34,648 tasks per run, depending on how many policies and λ_{lat} values it covers – and the tasks are spread over the job's cores. Covering the 24 configurations for every experiment in the paper takes about 1,400 such runs. Their median running time is 15 minutes, 90% finish within 34 minutes, and the longest takes under 4 hours; together they use about 8,800 core-hours. The reported tables and figures are computed from the stored results of these runs, with no further simulation.