MODEL PRIMING: UNVERIFIED HANDOFFS FOR EFFICIENT DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

The fast adoption of large language models is straining global compute, making inference efficiency both an economic and environmental imperative. We investigate *primed decoding*: a scheduled handoff policy in which a small model M_q continues a generation initiated by a larger, more capable model M_p. Unlike speculative decoding, primed decoding performs no verification and requires no router, retraining, or shared serving infrastructure. We evaluate it on five model pairs from two model families and six reasoning, mathematics, and code benchmarks, measuring accuracy together with throughput, cost, and energy on H200 GPUs. For instance, generating only the first 1024 tokens with `Qwen3.5-122B-A10B` and the rest with `Qwen3.5-4B` recovers 92% of `Qwen3.5-122B-A10B`'s performance and reduces single-request latency by 1.6–$1.8\times$ on long generations. At the same time, it cuts cost by 60% and energy consumption by 68%. These results suggest that a short high-capability prefix can establish a reasoning trajectory that a smaller model often sustains, enabling substantial offloading of generation to cheaper models without token-level coordination.

1 INTRODUCTION

The worldwide adoption of large language models is growing faster than the compute and energy infrastructure that supports it, turning inference efficiency into both an economic bottleneck and an ecological concern: every token generated by an LLM has a measurable cost in electricity, water, and carbon. Reducing the compute spent per useful generation is a prerequisite for scaling LLM use sustainably.

Within this landscape, deploying capable language models at inference time is fundamentally a cost-efficiency problem. Strong reasoning models are expensive to serve; smaller models are cheap but weaker. The gap between them is large, and existing approaches to bridging it like speculative decoding, query routing and ensembled models either require shared infrastructure, per-token coordination, or still invoke the large model on demand throughout generation.

How much of the large model's advantage persists in the small model if the large model speaks first? This question motivates **primed decoding**: the larger model M_p generates the first k tokens, then hands over entirely to M_q for the remaining $N - k$. In contrast to prior work, our approach uses no verifier, no retraining, and no routing, in the simplest possible setup.

The intuition is grounded in two properties of long-form reasoning chains. Early tokens disproportionately determine whether a reasoning trajectory succeeds (Sun et al., 2025), and the tail of a long chain is often within a smaller model's reach once a good trajectory is established (Chen et al., 2025). A short high-capability prefix can therefore anchor the generation, leaving the bulk of token production to the faster, cheaper model.

Empirically, this holds. On GPQA, for `Qwen3.5-122B-A10B` $\Rightarrow$ `Qwen3.5-9B`, a prefix of 256 tokens corresponds to approximately 2.2% of the mean output length under this configuration. It has an inference cost equal to only 8.6% of pure M_P decoding and recovers 50.0% of the performance gap between the small and large model baselines. The same model pair and priming length use only 40.4% of the GPU energy per output token required by pure M_P decoding, corresponding to a reduction of 59.6%. Across our pairs, PRIMING cuts serving cost and energy and lowers latency with respect to pure M_P.

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.

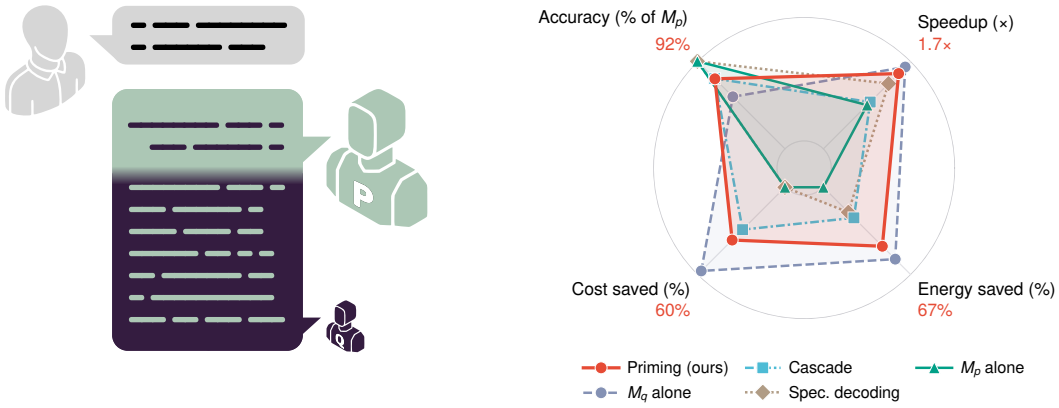

Figure 1: **(left)**: Illustration of primed decoding, where a large model M_p generates an initial prefix that is then handed off to a smaller model M_q for the remaining generation. **(right)**: `Qwen3.5-122B-A10B` $\Rightarrow$ `Qwen3.5-4B` primed with $k = 1024$ tokens, compared with pure large- and small-model decoding, a confidence cascade, and speculative decoding. Every axis points outward for better: accuracy relative to M_p, speedup in single-request latency over M_p, and cost per query and energy per token saved relative to M_p.

2 Related Work

Speculative decoding. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) uses a small draft model M_Q to propose γ tokens that a large target model M_P verifies in a single parallel forward pass; rejected tokens are resampled from a residual distribution that preserves M_P's output exactly. Subsequent work has refined the speedup under this distribution-preserving constraint: tree-structured drafts and learned decoding heads (Cai et al., 2024; Li et al., 2024), parallel and Jacobi-style drafting (Stern et al., 2018; Fu et al., 2024), online distillation of the draft (Liu et al., 2023; Zhou et al., 2024), and self-speculative decoding via early-exit layers (Elhoushi et al., 2024). A key structural difference distinguishes our method from this entire family: speculative decoding *verifies* every draft token against M_P and resamples failures, guaranteeing exact distributional fidelity at the cost of a speedup tied to the acceptance rate α. Our method instead accepts the distributional shift from handing off to M_Q.

Collaborative and token-level routing. Several approaches mix large- and small-model continuations within a single response without the draft–verify loop. BiLD (Kim et al., 2023) runs M_Q by default and escalates to M_P on low-confidence tokens, reporting an ensembling effect that can exceed either model alone. Co-LLM (Shen et al., 2024) learns a token-level deferral classifier that routes difficult tokens to a domain-expert assistant. CITER (Zheng et al., 2025) extends this with reinforcement learning over the routing policy. R2R (Fu et al., 2026) focuses on *path-divergent* tokens, positions where small- and large-model continuations lead to qualitatively different reasoning trajectories, and routes only those to M_P. RelayLLM (Huang et al., 2026) trains the small model to emit a command token that invokes M_P for a fixed number of steps before resuming. All of these methods use M_Q as the *default* generator and escalate to M_P on demand; our schedule runs in the opposite direction, using M_P to establish the initial reasoning trajectory before handing off to M_Q.

Query-level cascading and routing. At the coarser granularity of entire queries, cascading systems commit each prompt to a single model. FrugalGPT (Chen et al., 2024) routes through a sequence of increasingly capable APIs, stopping when a small model's answer passes a confidence filter (Chuang et al., 2025). Hybrid LLM (Ding et al., 2024) and RouteLLM (Ong et al., 2025) train classifiers that assign each prompt to a small or large model based on predicted difficulty. AutoMix (Aggarwal et al., 2024) uses few-shot self-verification to decide whether to escalate. These methods cannot exploit within-sequence variation in token difficulty because they allocate an entire generation to one model. Speculative cascades (Narasimhan et al., 2025) combine cascade deferral with speculative verification, but the deferral decision is still made at the query level. Our method operates within a single response, drawing on both models across different segments.

Algorithm 1 Primed Decoding

Require: Models M_P, M_Q; prompt $x_{1:n_0}$; output length N; priming length k
$x \leftarrow x_{1:n_0}$; prefill M_P on x.
Generate up to k tokens from M_P:
for $i = 1$ to k **do**
  $z \sim M_P(\cdot \mid x)$
  append z to x
  if z is <eos> **then return** x ▷ no handoff
  end if
end for
handoff: prefill M_Q on x
Generate remaining tokens from M_Q:
while $|x| < n_0 + N$ **and** last token of $x \neq$ <eos> **do**
  $z \sim M_Q(\cdot \mid x)$
  append z to x
end while
return x

3 METHODOLOGY

We introduce *scheduled decoding*, a framework that distributes generation across two models: a large target model M_P and a smaller approximation model M_Q. Each output position is assigned to one of the two by a schedule $S \in \{\mathsf{P}, \mathsf{Q}\}^N$.

Let $\mathcal{V}$ denote the shared vocabulary. The two models induce autoregressive next-token distributions $p(\cdot|x_{<t})$ and $q(\cdot|x_{<t})$ over $\mathcal{V}$, with corresponding length-N joints $p(x_{1:N}) = \prod_{t=1}^{N} p(x_t \mid x_{<t})$ and $q(x_{1:N}) = \prod_{t=1}^{N} q(x_t \mid x_{<t})$. The schedule then induces the joint

$$\pi^S(x_{1:N}) \;=\; \prod_{t=1}^{N} r_t^S(x_t \mid x_{<t}), \qquad r_t^S \;=\; \begin{cases} p & \text{if } S_t = \mathsf{P}, \\ q & \text{if } S_t = \mathsf{Q}. \end{cases} \tag{1}$$

Every schedule pays a *context-switching* cost: each transition between models forces the incoming model to prefill its KV cache from all previously generated tokens, a one-time but non-negligible overhead per handoff. Schedules with frequent handoffs pay this cost repeatedly; schedules with few handoffs pay it rarely.

3.1 PRIMED DECODING

The simplest non-trivial schedule has a single handoff: M_P generates the first k tokens, then M_Q generates the remaining $N - k$:

$$S = \underbrace{\mathsf{PPP}\ldots\mathsf{P}}_{k}\underbrace{\mathsf{QQQ}\ldots\mathsf{Q}}_{N-k} \qquad \text{with } k \ll N. \tag{2}$$

We call this *primed decoding*: the prefix from M_P primes a reasoning trajectory that M_Q continues. With only one handoff, the schedule pays the context-switching cost exactly once. Algorithm 1 gives the full procedure.

Two recent findings motivate placing M_P at the start. The early tokens of a reasoning chain disproportionately determine its outcome: Sun et al. (2025) shows that fixing the first 15% from a correct solution strongly constrains the remainder toward correctness. Long chains, meanwhile, exhibit *overthinking*, where models revisit and degrade correct answers (Chen et al., 2025). Therefore we posit that the marginal value of large-model decoding is concentrated at the start, leaving the smaller model well-suited to finish decoding the end.

3.2 Design Choices

Priming length. The priming length k is the only hyperparameter. We find empirically (Section 4) that $k \in [256, 2048]$ suffices to anchor the trajectory across our tasks, after which M_Q sustains high-quality generation for thousands of tokens.

Provider compatibility. Unlike speculative decoding (Leviathan et al., 2023; Kim et al., 2023), primed decoding requires no specialized inference engine and runs entirely at the API-caller level: issue a request to M_P, retrieve the first k tokens (including reasoning tokens), and pass them as a prompt prefix to M_Q. The only provider-side requirement is that reasoning tokens be exposed in the response. When M_P and M_Q are served by different providers or hosted on separate nodes, the handoff adds one network call round-trip.

4 Experiments

We evaluate primed decoding across reasoning, mathematics, and programming benchmarks. Our goal is to measure whether a short prefix generated by a stronger model M_P improves the continuation quality of a smaller model M_Q, and how this effect varies with the priming length k. We compare PRIMING against a number of baselines: pure M_P decoding, denoted by $k = \infty$, pure M_Q decoding, denoted by $k = 0$, speculative decoding (Leviathan et al., 2023) with each model's official drafter, and a confidence cascade (Gupta et al., 2024; Chen et al., 2024) in which M_Q answers every query and the given share of least-confident queries is regenerated by M_P. Appendix B details the baselines, the drafters and our vLLM implementation. For each model-task configuration, we sweep the priming length $k \in \{256, 512, 1024, 2048\}$, and evaluate more priming lengths in Section 4.2.

The central question we aim to answer is whether a small amount of M_P computation can move M_Q substantially toward M_P performance while leaving most of the generation to the cheaper model. Table 1 reports accuracy, speedup and cost averaged over six benchmarks while a detailed per-benchmark breakdown is in Table 8. For each metric we normalize by pure M_P decoding ($k = \infty$), which serves as the accuracy and cost ceiling. Normalized accuracy ($\tilde{Acc}$) measures how much of M_P's accuracy a method retains, cost measures its computational expense in dollars relative to M_P, and speedup measures its throughput gain over M_P; every method is scored against pure M_P; speedup is the ratio of single-request latencies on the controlled workload of Appendix H.

$$\widetilde{Acc} = \frac{Acc_{\mathrm{k}}}{Acc_{\infty}} \qquad \text{Speedup} = \frac{\text{Throughput}_k}{\text{Throughput}_{\infty}} \qquad \widetilde{\text{Cost}} = \frac{\text{Cost}_k}{\text{Cost}_{\infty}} \tag{3}$$

Models. We evaluate the proposed methodology on `Gemma-4-31B` $\rightarrow$ `Gemma-4-E4B` and `Qwen3.5-122B-A10B` $\rightarrow$ `Qwen3.5-4B`. More pairs and per-benchmark results are reported in Table 8. We use open-weight models served through vLLM, which allows us to control generation, expose intermediate tokens, and implement the handoff explicitly.

Benchmarks. We evaluate on six benchmarks spanning graduate-level science, mathematical reasoning, and code generation: GPQA (Rein et al., 2024), AIME'26 (Zhang & Math-AI, 2026), MATH-500 (Lightman et al., 2024), HumanEval (Chen et al., 2021), MBPP (Odena et al., 2021), and HLE (Center for AI Safety et al., 2026). These benchmarks differ substantially in response length and reasoning structure. GPQA, AIME, and MATH-500 test long-form reasoning and are the most favorable setting for primed decoding because the expensive prefix can be amortized over many continuation tokens. HumanEval and MBPP are shorter programming tasks, where the cost advantage is smaller but the handoff still tests whether the small model can exploit a high-quality partial solution. Each configuration is run with 5 seeds on the aforementioned benchmarks, and sampling parameters recommended settings for each model.

4.1 Results

To assess primed decoding, we analyze throughput, cost, and recovered accuracy relative to standalone M_P decoding as a function of the priming length k.

Table 1: **Accuracy, speedup and cost of PRIMING against baselines.** Acc.: accuracy relative to pure M_P, averaged over six benchmarks; speedup: single-request latency of pure M_P over that of the method; cost relative to pure M_P. $K = \infty$ and $K = 0$ are pure M_P and M_Q (gray); cascade rows give the share of queries deferred to M_P. $\pm$: s.e.m. Other pairs in Table 8.

$(M_p \Rightarrow M_q)$	K	Acc. ↑	Speedup ↑	Cost ↓
`Gemma-4-31B` ⇓ `Gemma-4-E4B`	∞	100.0%	1.00×	100.0%
	2048	90.1%$^{\pm 1.3}$	1.50×	64.0%$^{\pm 0.3}$
	1024	83.2%$^{\pm 0.7}$	1.76×	49.6%$^{\pm 0.2}$
	512	77.9%$^{\pm 0.5}$	1.90×	38.2%$^{\pm 0.1}$
	256	74.9%$^{\pm 0.8}$	1.99×	31.7%$^{\pm 0.1}$
	0	71.3%$^{\pm 0.7}$	2.29×	19.2%$^{\pm 0.2}$
Cascade	80%	95.6%$^{\pm 1.3}$	0.81×	104.7%
	60%	89.8%$^{\pm 1.1}$	0.97×	87.5%
	40%	85.9%$^{\pm 0.8}$	1.20×	70.3%
	20%	79.2%$^{\pm 1.8}$	1.57×	47.6%
Spec. decoding		100.0%	1.96×	100.0%
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-4B`	∞	100.0%	1.00×	100.0%
	2048	92.3%$^{\pm 0.5}$	1.60×	48.1%$^{\pm 1.0}$
	1024	92.0%$^{\pm 0.6}$	1.72×	39.7%$^{\pm 1.0}$
	512	87.3%$^{\pm 1.1}$	1.77×	28.8%$^{\pm 0.8}$
	256	86.4%$^{\pm 0.6}$	1.80×	18.8%$^{\pm 0.5}$
	0	83.8%$^{\pm 0.6}$	1.87×	4.4%$^{\pm 0.1}$
Cascade	80%	97.8%$^{\pm 0.7}$	0.75×	89.8%
	60%	96.0%$^{\pm 0.7}$	0.88×	72.1%
	40%	92.1%$^{\pm 0.7}$	1.07×	51.6%
	20%	89.0%$^{\pm 0.5}$	1.36×	30.4%
Spec. decoding		100.0%	1.49×	100.0%

PRIMING recovers M_Q performance. Across both model families, a short M_P prefix substantially improves M_Q continuations, showing that scheduled unverified handoffs can effectively bridge the performance gap between small and large models. However, the magnitude of recovered accuracy depends strongly on the priming budget k and the underlying task, creating a direct, controllable trade-off between output quality and inference efficiency.

Efficiency trade-offs. PRIMING trades raw performance for reductions in serving cost and latency. By varying the priming prefix length k, we can flexibly navigate the frontier trade-off between generation quality and inference efficiency. For example, on `Gemma-4-31B`, operating at approximately 90% of standalone M_P accuracy ($k = 2048$) yields a 36% reduction in serving cost and a 1.5× lower single-request latency.

The efficacy of priming varies noticeably across task profiles. Moderate prefixes recover a large fraction of standalone M_P accuracy on GPQA, HumanEval, MBPP, and MATH-500, where early guidance effectively anchors the reasoning trajectory. In contrast, on the hardest benchmarks, AIME'26 and HLE, the small model must carry the reasoning far beyond the prefix: at $k = 256$, M_P writes about 1% of the AIME'26 and HLE tokens for the Qwen pairs and under 4% of the AIME'26 tokens for the Gemma pairs (Table 10), and recovery is more difficult for some pairs.

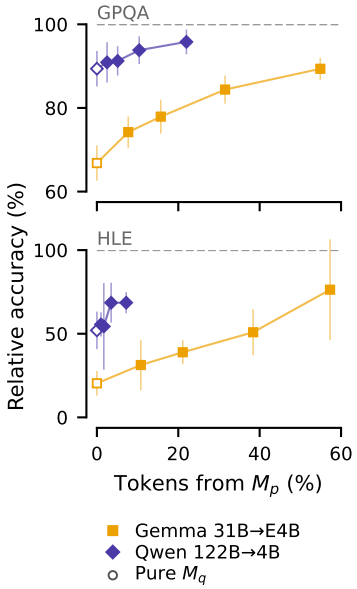

Figure 2: Accuracy relative to pure M_P against the share of output tokens written by M_P (Table 10), for $k \in \{256, 512, 1024, 2048\}$ on GPQA and HLE.

Table 2: **Effect of Ablations** Performance trade-offs for alternative handoff schedules on GPQA, with `Gemma-4-31B` and `Gemma-4-E4B` (cross-family: `Qwen3.5-122B-A10B`). $\widetilde{Acc}$ and Cost are relative to pure M_P on GPQA.

	Schedule	K	$\widetilde{\text{GPQA}}$ ↑	Cost ↓
PRIMING	PPPQQQQQQ...	2048	$89.4\%^{\pm 0.5}$	$45.9\%^{\pm 0.1}$
		1024	$84.5\%^{\pm 0.9}$	$30.0\%^{\pm 0.1}$
		512	$77.9\%^{\pm 1.2}$	$22.6\%^{\pm 0.1}$
INTERLEAVED	PPPQQQPPP...	2048	$93.1\%^{\pm 1.2}$	$64.5\%^{\pm 0.6}$
		1024	$90.7\%^{\pm 0.2}$	$85.8\%^{\pm 0.8}$
		512	$88.6\%^{\pm 0.8}$	$133.0\%^{\pm 4.3}$
Reverse PRIMING	QQQPPPPPP...	2048	$87.0\%^{\pm 0.6}$	$76.5\%^{\pm 0.7}$
		1024	$95.6\%^{\pm 1.1}$	$92.9\%^{\pm 0.7}$
		512	$97.5\%^{\pm 0.6}$	$96.4\%^{\pm 0.4}$
Cross-family	`Qwen3.5-122B-A10B` ⇓ `Gemma-4-E4B`	2048	$85.0\%^{\pm 0.4}$	$23.9\%^{\pm 0.0}$
		1024	$79.2\%^{\pm 1.0}$	$12.8\%^{\pm 0.0}$
		512	$70.9\%^{\pm 2.4}$	$7.4\%^{\pm 0.0}$

Consequently, k controls the trade-off: larger prefixes recover more accuracy at a higher cost and a higher latency. On `Gemma-4-31B` → `Gemma-4-E4B`, single-request latency falls by 1.9–2.0× for $k \leq 512$ and by 1.5× at $k = 2048$, against 2.3× for pure M_Q. Against the baselines, at the matched accuracies PRIMING has lower single-request latency than the confidence cascade on every pair, whose deferred queries pay for a full generation by M_Q and then by M_P, and is cheaper. Against lossless speculative decoding, PRIMING has often lower or equal latency, while on one tested pair (`Qwen3.5-122B-A10B` → `Qwen3.5-9B`) speculative decoding is faster (Appendix B.3). The two techniques are also orthogonal: speculative decoding accelerates a single model and can be applied to M_P while it writes the prefix and to M_Q after the handoff, compounding the savings of PRIMING.

4.2 ABLATIONS

Cross-family priming. Because our API-level handoff works with plain decoded text, it naturally bypasses tokenizer and chat template mismatches, meaning that PRIMING is not limited to pairings of the same family. To show this we test a cross-family configuration with `Qwen3.5-122B-A10B` → `Gemma-4-E4B` and observe that PRIMING remains effective: at $k = 2048$, it recovers 85.0% of standalone M_P accuracy while reducing compute costs to less than a quarter (23.9%) of pure M_P decoding.

Reverse-PRIMING. Throughout this paper we assume that the most important part of the generation is the early reasoning tokens. To test this, we evaluate a "Reverse-PRIMING" schedule where the smaller draft model M_Q generates the initial k tokens before handing off the remainder to the bigger M_P. Reverse priming is more accurate (87.0–97.5% on GPQA), but only because M_P then writes almost all of the response: it pays 76.5–96.4% of the cost of pure M_P. As shown in Table 2, reverse priming is therefore unfavorable.

INTERLEAVED schedule. We also explore an alternating schedule that switches between M_P and M_Q throughout generation. Although interleaving achieves stronger relative accuracy (up to 93.1% at $k = 2048$), it introduces repeated context-switching penalties. Each handover forces the target model to prefill its KV cache over the accumulated history, inflating the overall compute expense. In fact, for $k = 512$, the cumulative prefill overhead drives serving costs to 133.0% of standalone M_P decoding.

Varying Priming length. To determine the minimal prefix required to anchor generation, we evaluate very short priming budgets $k \in \{16, 32, 64, 128\}$ (Table 3, detailed per benchmark in Table 13). For `Gemma-4-31B` → `Gemma-4-E4B` on GPQA, prefixes of up to 128 tokens retain 65.8–68.4% of M_P's accuracy, no better than pure M_Q (66.8%), and the first clear gain appears at $k = 256$ (74.2%). We hypothesize that such short prefixes rarely get past restating the problem, so they leave M_Q without a plan to follow; we did not verify this on the generated traces.

Table 3: **Short priming lengths.** $\widetilde{\text{GPQA}}$ is the GPQA accuracy relative to pure M_P decoding, and Cost the GPQA serving cost relative to pure M_P.

$(M_p \Rightarrow M_q)$	K	$\widetilde{\text{GPQA}} \uparrow$	**Speedup** $\uparrow$	**Cost** $\downarrow$
	∞	100.0%	1.00×	100.0%
	256	$74.2\%^{\pm 1.3}$	1.99×	$19.1\%^{\pm 0.1}$
`Gemma-4-31B` ⇓	128	$67.5\%^{\pm 1.0}$	2.04×	$17.3\%^{\pm 0.2}$
`Gemma-4-E4B`	64	$65.8\%^{\pm 0.8}$	2.06×	$16.6\%^{\pm 0.0}$
	32	$66.4\%^{\pm 1.4}$	2.08×	$16.3\%^{\pm 0.2}$
	16	$68.4\%^{\pm 1.7}$	2.08×	$16.3\%^{\pm 0.1}$
	0	$66.8\%^{\pm 1.5}$	2.29×	$14.7\%^{\pm 0.1}$

5 Discussion

5.1 Throughput

The throughput speed of primed decoding is piecewise. During the priming phase, tokens are generated at the throughput of M_P; once the handoff occurs at position k, throughput transitions to that of M_Q. Let τ_p and τ_q denote the throughput (tokens/second) of M_P and M_Q respectively, with $\tau_q > \tau_p$ by assumption. The average throughput over a full generation of length N is then:

$$\bar{\tau}(k) = \frac{N}{\frac{k}{\tau_p} + t_{\text{switch}} + \frac{N-k}{\tau_q}}$$

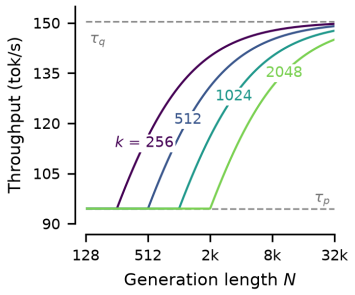

Figure 3: Theoretical throughput $\bar{\tau}(k)$ against generation length N.

where t_{switch} is the latency incurred at the handoff point, primarily due to prefilling M_Q's KV cache with the full context of length $I + k$. This prefill cost scales roughly linearly with $I + k$ and constitutes a one-time latency penalty at the switching point.

The context-switch latency t_{switch} becomes non-trivial when M_P and M_Q are hosted by different providers, adding a network round-trip on top of the prefill cost.

As it can be seen in Figure 3, the throughput $\bar{\tau}(k)$ converges to τ_q as generation length N increases. Figure 4 shows that the trade-off is favorable against the cascade: averaged over benchmarks, no primed configuration is dominated by a cascade in latency or in cost. Against speculative decoding, PRIMING is faster on three pairs and cheaper on all, while speculative decoding is faster on `Qwen3.5-122B-A10B` → `Qwen3.5-9B` and, except at $k = 256$, on `Gemma-4-31B` → `Gemma-4-E4B`.

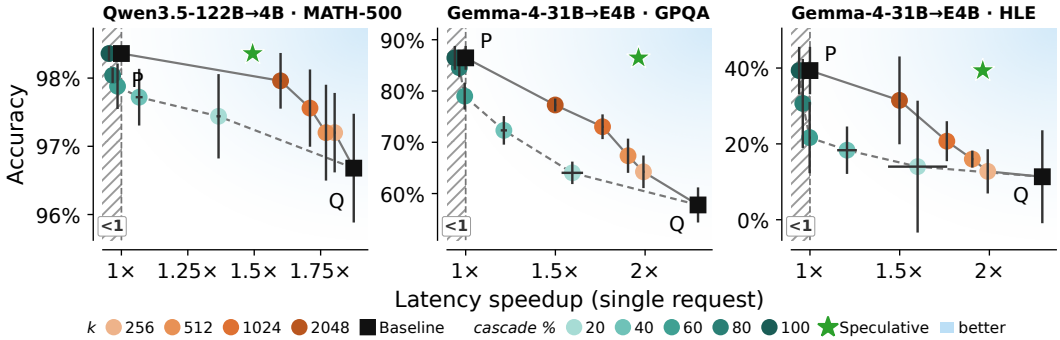

Figure 4: **Latency speedup–accuracy trade-offs for PRIMING and baselines.** Black squares: pure M_P (P) and pure M_Q (Q); orange: PRIMING at increasing k. Speedup is the single-request latency of pure M_P over that of each method.

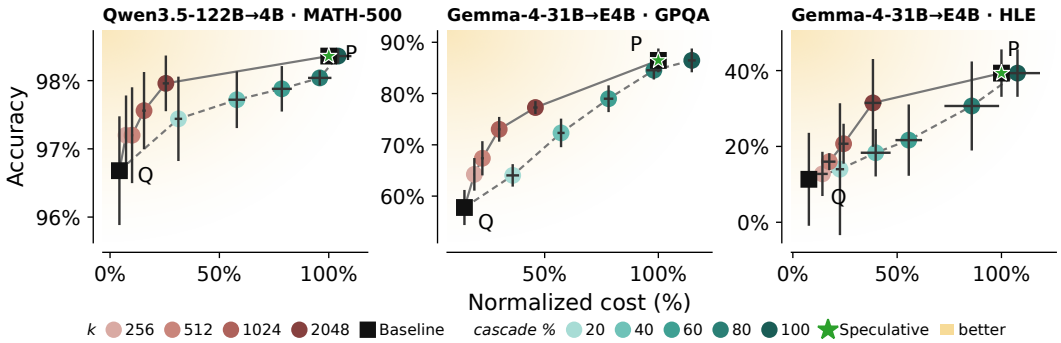

Figure 6: Cost–accuracy trade-offs for primed decoding across Qwen and Gemma model pairs. Black squares denote baseline evaluations: full large-model decoding (M_P) and full small-model decoding (M_Q). Colored markers denote primed decoding runs across varying prefix lengths k, with costs normalized to M_P.

5.2 TOKENOMICS

Let $c^{(i)}$ and $c^{(o)}$ denote the per-token input and output prices for a given model. For a prompt of length I and total generated length N, pure M_P decoding costs $\mathcal{C}_{\text{full}} = I \cdot c_p^{(i)} + N \cdot c_p^{(o)}$.

Under primed decoding, M_P generates the first k tokens, and M_Q is then prefilled with the $I + k$ context to generate the remaining $N - k$ tokens. This approach is cost-beneficial when the savings on small-model output tokens exceed the context-switching overhead:

$$\underbrace{(N-k)\left(c_p^{(o)} - c_q^{(o)}\right)}_{\text{savings on output tokens}} > \underbrace{(I+k)\cdot c_q^{(i)}}_{\text{overhead of context switch}} \quad (4)$$

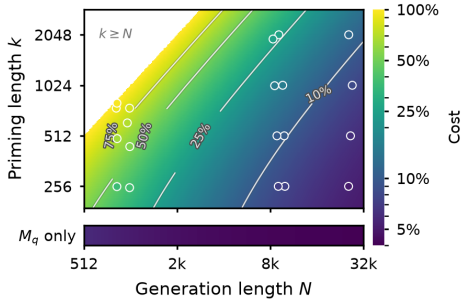

Figure 5: Cost of primed decoding as a percentage of pure M_P decoding, for `Qwen3.5-122B-A10B` → `Qwen3.5-4B` at $I = 1024$; Markers: measured operating points.

The left-hand side grows with $N - k$, meaning longer generations amplify the cost advantage. The right-hand side is the one-off charge for re-reading the prompt and prefix into M_Q, determined by prompt I and prefix k. This makes primed decoding particularly attractive for the long-reasoning tasks we consider (GPQA, AIME, HLE), where $N \gg k$ and output costs dominate. We report representative prices from OpenRouter and the resulting cost breakdowns per benchmark in Appendix G.

Figure 5 illustrates cost of PRIMING as a percentage of full M_P decoding for the `Qwen3.5-122B-A10B` → `Qwen3.5-4B` pair. For $k = 512$, primed decoding reduces costs by 40% after ∼1024 generated tokens, reaching almost 90% savings at ∼8000 tokens.

Figure 6 highlights the cost–accuracy trade-off across three benchmarks. Primed decoding forms a favorable Pareto frontier between the two model baselines: increasing the prefix length k recovers accuracy close to M_P while retaining substantial cost reductions.

5.3 ENERGY EFFICIENCY

Beyond latency and price, the handoff also changes how much energy each generated token consumes. To measure it, we read the GPUs energy counters during the throughput runs.

Table 4 reports the energy per generated token for representative pairs, with all pairs in Appendix D. Relative to pure M_P decoding, PRIMING saves energy at every priming length: for `Gemma-4-31B` → `Gemma-4-E4B` at $k = 2048$, it uses 62% less energy per token while retaining 90% of M_P's accuracy.

Table 4: Normalized accuracy and GPU energy per generated token of primed decoding; saving is relative to pure M_P. All pairs in Table 9.

$(M_p \Rightarrow M_q)$	K	$\tilde{Acc}$ ↑	**Energy (J/token)** ↓	**Saving** ↑
`Gemma-4-31B` ⇓ `Gemma-4-E4B`	∞	100.0%	1.775	0.0%
	2048	90.1%$^{\pm 1.3}$	0.672	62.1%
	1024	83.2%$^{\pm 0.7}$	0.567	68.1%
	512	77.9%$^{\pm 0.5}$	0.485	72.7%
	256	74.9%$^{\pm 0.8}$	0.488	72.5%
	0	71.3%$^{\pm 0.7}$	0.327	81.6%
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-4B`	∞	100.0%	1.761	0.0%
	2048	92.3%$^{\pm 0.5}$	0.708	59.8%
	1024	92.0%$^{\pm 0.6}$	0.565	67.9%
	512	87.3%$^{\pm 1.1}$	0.512	70.9%
	256	86.4%$^{\pm 0.6}$	0.457	74.0%
	0	83.8%$^{\pm 0.6}$	0.297	83.1%

6 LIMITATIONS

Primed decoding is not distribution-preserving: PRIMING does not verify the tokens of M_Q against M_P, so the continuation can depart from what M_P would have written, and unlike speculative decoding it trades accuracy for lower serving cost and energy. How much accuracy is retained depends on the task and on the pair. At $k = 2048$, HLE retains only 60–76% of M_P's accuracy for every pair except `Qwen3.5-122B-A10B` → `Qwen3.5-9B` (91%), and AIME'26 retains 60–79% for the Gemma pairs, against 89–98% for the Qwen pairs. The handoff is simplest within a model family, where M_P and M_Q share tokenizer and chat template and the prefix is passed as token IDs. Across families the prefix is passed as text and re-rendered with the chat template of M_Q; this works in our cross-family experiment, but retains less accuracy on GPQA than the same-family pair with the same M_P (85.0% with `Gemma-4-E4B` against 95.8% with `Qwen3.5-4B` at $k = 2048$), and relies on M_Q continuing a reasoning trace written by another model. Part of this difference reflects the weaker `Gemma-4-E4B`: measured as the share of the M_Q–M_P gap that priming closes, the two pairs are close (54% and 60%). Finally, the priming length k is a cost-quality hyperparameter that we select by sweep rather than adaptively; with models hosted by different providers, the handoff also adds a network round trip (Section 5.1).

7 CONCLUSION

We presented PRIMING, a simple scheduled-handoff decoding strategy for cost-efficient long-form generation. By using a larger model M_P to generate an initial prefix and then transferring generation to a smaller model M_Q, PRIMING requires no verifier, routing policy, retraining, or shared serving infrastructure. Empirically, priming substantially improves small-model continuations across reasoning, mathematics, and code benchmarks, though the gains depend on the task, model pair, and handoff length. The method is most effective when the prefix establishes useful problem structure and the remaining continuation lies within M_Q's capabilities; harder tasks such as HLE, and AIME'26 for the Gemma pairs, show that priming is not a universal substitute for large-model decoding.

Our ablations also show that where the large model intervenes matters: letting M_Q write the beginning instead is slower than pure M_P on the same GPUs, interleaving the two models buys accuracy at the price of repeated prefills, and prefixes shorter than 256 tokens do not improve on M_Q alone. The beginning of the generation is where a few tokens from M_P are worth the most. The benefits of PRIMING are lower serving cost and energy and lower latency than pure M_P; it is not faster than the small model alone. Overall, PRIMING provides a lightweight, deployment-friendly way to trade a small amount of high-capability computation for improved small-model performance at reduced inference cost.

AI USE STATEMENT

We used generative AI tools to assist with editing and restructuring the manuscript along with code development and debugging. All AI-assisted research code and manuscript text were reviewed and verified by the authors. The authors take responsibility for the final content of this work, including all claims and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the full implementation of primed decoding in an anonymized code repository in `https://anonymous.4open.science/r/model-priming`. It's built directly on top of the standard vLLM repository and is ready to use. Implementation details and complete lists of evaluated models and benchmarks are provided in the appendix.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam Mausam. Automix: Automatically mixing language models. 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. 131000–131034. Curran Associates, Inc., 2024. doi: 10.52202/079017-4164. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/ecda225cb187b40ea8edc1f46b03ffda-Paper-Conference.pdf`.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL `https://arxiv.org/abs/2401.10774`.

Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL `https://arxiv.org/abs/2501.14249`.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023. URL `https://arxiv.org/abs/2302.01318`.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research (TMLR)*, 2024. URL `https://openreview.net/forum?id=XUZ2S0JVJP`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT think that much for 2+3=? on the overthinking of long reasoning models. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=MSbU3L7V00`.

Yu-Neng Chuang, Leisheng Yu, Guanchu Wang, Lizhe Zhang, Ling Chang, Hongyi Liu, Zirui Liu, Xuanting Cai, Yang Sui, Vladimir Braverman, and Xia Hu. Confident or seek stronger: Exploring uncertainty-based small LM routing. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025. URL https://openreview.net/forum?id=pyjLRuheNU.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=02f3mUtqnM.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Tianyu Fu, Yi Ge, Yichen You, Enshu Liu, Zhihang Yuan, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. R2r: Efficiently navigating divergent reasoning paths with small-large model token routing. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=DpeJYRFRQY.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.02057.

Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. Language model cascades: Token-level uncertainty and beyond. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2404.10136.

Chengsong Huang, Tong Zheng, Langlin Huang, Jinyuan Li, Haolin Liu, and Jiaxin Huang. Relayllm: Efficient reasoning via collaborative decoding, 2026.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://openreview.net/forum?id=EfMyf9MC3t.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP)*, 2023. doi: 10.48550/arXiv.2309.06180. URL https://arxiv.org/abs/2309.06180.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://arxiv.org/abs/2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2401.15077.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. Online speculative decoding. *arXiv preprint arXiv:2310.07177*, 2023. URL https://arxiv.org/abs/2310.07177.

Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Seungyeon Kim, Neha Gupta, Aditya Krishna Menon, and Sanjiv Kumar. Faster cascades via speculative decoding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vo9t20wsmd.

Augustus Odena, Charles Sutton, David Martin Dohan, Ellen Jiang, Henryk Michalewski, Jacob Austin, Maarten Paul Bosma, Maxwell Nye, Michael Terry, and Quoc V. Le. Program synthesis with large language models. In *n/a*, pp. n/a, n/a, 2021. n/a.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Shannon Zejiang Shen, Hunter Lang, Bailin Wang, Yoon Kim, and David Sontag. Learning to decode collaboratively with multiple language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12974–12990, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.701/.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Yiliu Sun, Zicheng Zhao, Yang Wei, Yanfang Zhang, and Chen Gong. Well begun, half done: Reinforcement learning with prefix optimization for llm reasoning, 2025. URL https://arxiv.org/abs/2512.15274.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2026, 2026.

Wenhao Zheng, Yixiao Chen, Weitong Zhang, Souvik Kundu, Yun Li, Zhengzhong Liu, Eric P. Xing, Hongyi Wang, and Huaxiu Yao. CITER: Collaborative inference for efficient large language model decoding with token-level routing. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=nqX9UYW9Af.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*, 2024. URL https://arxiv.org/abs/2310.08461.

A Model Names

Table 5 maps the model names used throughout the paper to the HuggingFace checkpoints that were served in all experiments. All checkpoints are the instruction-tuned releases; the `-IT`/`-it` suffix is dropped in the text for brevity.

B Serving Stack and Baselines

All methods, PRIMING and baselines alike, are served with the same stack, on the same type of GPU (NVIDIA H200; the number of GPUs per method is given in Table 12) and with the same sampling parameters, so that they differ only in how the generation is split between the models.

Name in paper	HuggingFace checkpoint	Role
`Qwen3.5-122B-A10B`	`Qwen/Qwen3.5-122B-A10B`	M_P
`Qwen3.5-27B`	`Qwen/Qwen3.5-27B`	M_P
`Qwen3.5-9B`	`Qwen/Qwen3.5-9B`	M_Q
`Qwen3.5-4B`	`Qwen/Qwen3.5-4B`	M_Q
`Gemma-4-31B`	`google/gemma-4-31B-it`	M_P
`Gemma-4-26B-A4B`	`google/gemma-4-26B-A4B-it`	M_P
`Gemma-4-E4B`	`google/gemma-4-E4B-it`	M_Q
`Gemma-4-E2B`	`google/gemma-4-E2B-it`	M_Q

Table 5: Model names used in the paper and the corresponding HuggingFace checkpoints. M_P denotes models used as the priming model, M_Q models used as the approximation model.

B.1 vLLM IMPLEMENTATION

Serving stack. Every model is served with vLLM (Kwon et al., 2023), from a fork that adds primed decoding. This fork is available for you to test at `https://anonymous.4open.science/r/model-priming`. Each server runs with thinking enabled in the chat template, the model family's reasoning parser, prefix caching, and at most 8 concurrent sequences. All measurements are taken on NVIDIA H200 GPUs (141 GB). Each model sits on its own GPUs: one GPU per model, except `Qwen3.5-122B-A10B`, which is served with tensor parallelism 2 on two GPUs. In the accuracy runs, sampling follows each family's recommended settings, and is the same for every method: temperature 1.0, top-p 0.95 and top-k 64 for Gemma; temperature 1.0, top-p 0.95, top-k 20 and presence penalty 1.5 for Qwen.

Throughput and energy measurements. Speedups (Appendix H) are ratios of single-request latency, measured at concurrency 1 with 12000 forced output tokens per request (EOS ignored, greedy decoding). Energy is measured in the same runs, reading the GPUs' NVML energy counters. Only the throughput of Table 12 is measured at a concurrency of 32. Every method is measured in this same setup, with each model on its own GPUs as within the priming pair.

B.2 BASELINES

Pure decoding ($K = \infty$, $K = 0$). M_P or M_Q alone, served as above with their respective recommended parameters.

Speculative decoding. We use lossless speculative decoding (Leviathan et al., 2023) as implemented in vLLM, with the drafter released by each model's developer (Table 6). For Gemma, the drafter is Google's companion "assistant" model, which proposes 6 tokens per step (the `num_assistant_tokens` of its generation config). For Qwen, the drafter is the multi-token-prediction (MTP) module shipped in the checkpoint, with 2 speculative tokens per step, the value recommended by Qwen for vLLM. Since speculative sampling leaves the output distribution of M_P unchanged, its accuracy is that of M_P, and only its throughput is measured.

Table 6: Drafters used for speculative decoding.

Target (M_P)	Drafter	Size	Tokens / step
`Gemma-4-31B`	`google/gemma-4-31B-it-assistant`	470M (4 layers)	6
`Gemma-4-26B-A4B`	`google/gemma-4-26B-A4B-it-assistant`	420M (4 layers)	6
`Qwen3.5-122B-A10B`	built-in MTP module	1 layer	2
`Qwen3.5-27B`	built-in MTP module	1 layer	2

Cascade. We follow the confidence cascade of Gupta et al. (2024) with its Chow-Average deferral rule. M_Q answers every query, and its confidence is the mean token log-probability over the whole generated sequence (reasoning included). The queries with the lowest confidence are regenerated by M_P, and M_P's answer replaces M_Q's. The percentage reported for the cascade in Table 1 is the target share of queries deferred to M_P. To avoid tuning on the test set, the deferral thresholds are set by two-fold cross-fitting (Chuang et al., 2025): each benchmark is split into two fixed halves by

question id, the threshold matching the target share is set on one half and applied unchanged to the other, and the roles are then swapped. The cost of a cascade is that of M_Q on all queries plus that of M_P on the deferred ones. Its speedup is composed from the measured pure-M_P and pure-M_Q throughputs, with M_Q generating all answers and M_P regenerating the deferred ones, one after the other.

B.3 Comparison with Baselines

Table 7 pairs, for each model pair, a primed configuration with the cascade of closest accuracy (all values from Table 8, averaged over the six benchmarks). At comparable accuracy, PRIMING is 1.3–1.7× faster in single-request latency than the cascade and 9–24 points cheaper relative to pure M_P. The cascade pays for M_Q on every query and for M_P on the deferred ones, while PRIMING pays for M_P only on the prefix. More generally, no primed configuration is dominated by a cascade: none of them reaches at least the accuracy of a primed configuration of the same pair with a lower latency or at a lower cost.

Cascade. The cascade reaches higher accuracy only when it defers most queries: at 80% deferral it retains 95.6–99.3% of M_P's accuracy, but it is slower than pure M_P (0.57–0.97× in single-request latency) and costs 90–105% of it, since most queries are answered twice.

Speculative Decoding. Speculative decoding is exact: its accuracy is that of M_P, and so is its cost, since it serves the same model at the same price. Its speedup, however, is bounded by the acceptance rate of the drafter and ranges from 0.75× to 2.1× in single-request latency across pairs. On a single request, PRIMING has lower latency than speculative decoding on `Gemma-4-26B-A4B` → `Gemma-4-E2B` (up to 1.04 against 0.75×, the drafter slowing this model down), `Qwen3.5-122B-A10B` → `Qwen3.5-4B` (1.60–1.80 against 1.49×) and `Qwen3.5-27B` → `Qwen3.5-4B` (e.g. 3.72 against 2.11× at $k = 256$), at 45–49% of its cost at $k = 2048$. Speculative decoding is faster on `Qwen3.5-122B-A10B` → `Qwen3.5-9B`, where the small model gains little over the large one, and on `Gemma-4-31B` → `Gemma-4-E4B`, where it is on par with PRIMING at $k = 256$. Forcing 12000 output tokens can drive a model into repetitive text after its answer, which its drafter predicts almost perfectly; this inflates speculative decoding, so these comparisons are, if anything, favourable to it. Since speculative decoding accelerates a single model, it could also be applied to M_P during the prefix and to M_Q after the handoff; we leave this combination to future work.

Table 7: PRIMING against the cascade of closest accuracy (speedup: single-request latency), for each model pair. $\tilde{Acc}$, speedup and cost are relative to pure M_P and averaged over the six benchmarks (Table 8); the cascade percentage is the share of queries deferred to M_P.

	PRIMING			Cascade		
($M_p \Rightarrow M_q$)	K	$\tilde{Acc}$	Speedup / Cost	Deferred	$\tilde{Acc}$	Speedup / Cost
`Gemma-4-26B-A4B` → `Gemma-4-E2B`	2048	83.5%	0.98× / 45.3%	40%	82.3%	0.73× / 64.2%
`Gemma-4-31B` → `Gemma-4-E4B`	2048	90.1%	1.50× / 64.0%	60%	89.8%	0.97× / 87.5%
`Qwen3.5-122B-A10B` → `Qwen3.5-9B`	2048	97.7%	1.20× / 48.5%	60%	97.4%	0.73× / 72.1%
`Qwen3.5-122B-A10B` → `Qwen3.5-4B`	1024	92.0%	1.72× / 39.7%	40%	92.1%	1.07× / 51.6%
`Qwen3.5-27B` → `Qwen3.5-4B`	2048	91.1%	2.61× / 45.5%	40%	91.9%	1.55× / 54.4%

C Complete Results

Table 8 extends Table 1 with per-benchmark accuracies and three additional model pairs.

Table 8: Complete results for all model pairs. HLE: 100 text-only multiple-choice questions of the HLE-Verified Gold subset. Avg. Acc. is the mean over the six benchmarks of the accuracy relative to pure M_P ($K = \infty$). Speedup is the ratio of the single-request latency of pure M_P to that of the method (Appendix H; for the cascade, composed per query from the pure-model runs); Cost is relative to pure M_P; $\pm$ is the s.e.m., over seeds for accuracy and cost and over requests for speedup. Pure `Qwen3.5-122B-A10B` (and pure `Qwen3.5-4B`) appear in two blocks from separate runs of the same model; their differences reflect run-to-run variance.

$(M_p \Rightarrow M_q)$	K	GPQA	AIME'26	HumanEval	MBPP	MATH-500	HLE	Avg. Acc. ↑	Speedup ↑	Cost ↓
`Gemma-4-26B-A4B` ⇓ `Gemma-4-E2B`	∞	80.1%$^{\pm0.7}$	86.7%$^{\pm2.6}$	97.0%$^{\pm0.4}$	96.3%$^{\pm0.3}$	97.6%$^{\pm0.2}$	34.3%$^{\pm1.2}$	100.0%	1.00×	100.0%
	2048	68.2%$^{\pm1.2}$	52.0%$^{\pm1.5}$	95.5%$^{\pm0.4}$	93.9%$^{\pm0.3}$	94.1%$^{\pm0.2}$	21.7%$^{\pm1.5}$	83.5%$^{\pm1.0}$	0.98×	45.3%$^{\pm0.3}$
	1024	61.2%$^{\pm0.8}$	48.0%$^{\pm2.0}$	92.2%$^{\pm0.5}$	93.9%$^{\pm0.3}$	92.8%$^{\pm0.3}$	16.0%$^{\pm2.0}$	77.7%$^{\pm1.1}$	1.01×	30.5%$^{\pm0.2}$
	512	54.7%$^{\pm0.9}$	43.3%$^{\pm2.3}$	89.9%$^{\pm0.5}$	92.0%$^{\pm0.5}$	92.1%$^{\pm0.3}$	14.3%$^{\pm2.2}$	73.8%$^{\pm1.2}$	1.02×	21.7%$^{\pm0.1}$
	256	51.3%$^{\pm1.0}$	42.0%$^{\pm1.5}$	88.8%$^{\pm0.6}$	90.7%$^{\pm0.2}$	91.0%$^{\pm0.4}$	15.0%$^{\pm1.2}$	72.5%$^{\pm0.8}$	1.04×	17.2%$^{\pm0.1}$
	0	43.1%$^{\pm1.4}$	39.3%$^{\pm2.2}$	86.7%$^{\pm0.8}$	91.2%$^{\pm0.3}$	88.1%$^{\pm0.3}$	13.3%$^{\pm1.8}$	68.7%$^{\pm1.1}$	1.03×	10.4%$^{\pm0.0}$
Cascade `Gemma-4-26B-A4B` ⇓ `Gemma-4-E2B`	80%	76.6%$^{\pm0.5}$	82.7%$^{\pm2.9}$	96.7%$^{\pm0.4}$	96.0%$^{\pm0.4}$	97.6%$^{\pm0.1}$	28.7%$^{\pm1.8}$	95.7%$^{\pm1.3}$	0.57×	97.5%
	60%	68.2%$^{\pm1.4}$	78.0%$^{\pm3.9}$	96.2%$^{\pm0.4}$	95.6%$^{\pm0.5}$	97.4%$^{\pm0.2}$	22.0%$^{\pm2.6}$	89.6%$^{\pm1.6}$	0.64×	81.5%
	40%	59.7%$^{\pm1.6}$	66.0%$^{\pm3.6}$	94.9%$^{\pm0.3}$	94.9%$^{\pm0.4}$	96.8%$^{\pm0.2}$	16.3%$^{\pm0.3}$	82.3%$^{\pm0.9}$	0.73×	64.2%
	20%	50.9%$^{\pm1.1}$	53.3%$^{\pm3.3}$	92.4%$^{\pm0.6}$	93.4%$^{\pm0.7}$	95.6%$^{\pm0.2}$	13.3%$^{\pm0.9}$	75.7%$^{\pm0.9}$	0.86×	41.8%
SD `Gemma-4-26B-A4B`								100.0%	0.75×	100.0%
`Gemma-4-31B` ⇓ `Gemma-4-E4B`	∞	86.5%$^{\pm0.8}$	89.3%$^{\pm1.2}$	97.1%$^{\pm0.4}$	96.5%$^{\pm0.0}$	98.5%$^{\pm0.3}$	39.3%$^{\pm1.5}$	100.0%	1.00×	100.0%
	2048	77.3%$^{\pm0.4}$	70.7%$^{\pm2.0}$	95.6%$^{\pm0.4}$	95.7%$^{\pm0.2}$	96.4%$^{\pm0.3}$	30.0%$^{\pm2.5}$	90.1%$^{\pm1.3}$	1.50×	64.0%$^{\pm0.3}$
	1024	73.0%$^{\pm0.8}$	64.0%$^{\pm1.7}$	93.2%$^{\pm0.2}$	95.3%$^{\pm0.3}$	96.3%$^{\pm0.2}$	20.0%$^{\pm1.0}$	83.2%$^{\pm0.7}$	1.76×	49.6%$^{\pm0.2}$
	512	67.4%$^{\pm1.1}$	62.0%$^{\pm1.2}$	86.1%$^{\pm0.7}$	93.6%$^{\pm0.3}$	94.2%$^{\pm0.2}$	15.3%$^{\pm0.3}$	77.9%$^{\pm0.5}$	1.90×	38.2%$^{\pm0.1}$
	256	64.2%$^{\pm1.0}$	58.0%$^{\pm2.2}$	84.3%$^{\pm0.8}$	93.2%$^{\pm0.4}$	94.2%$^{\pm0.2}$	12.3%$^{\pm1.3}$	74.9%$^{\pm0.8}$	1.99×	31.7%$^{\pm0.1}$
	0	57.8%$^{\pm1.2}$	59.3%$^{\pm3.1}$	83.3%$^{\pm0.9}$	90.7%$^{\pm0.3}$	92.8%$^{\pm0.2}$	8.0%$^{\pm0.6}$	71.3%$^{\pm0.7}$	2.29×	19.2%$^{\pm0.2}$
Cascade `Gemma-4-31B` ⇓ `Gemma-4-E4B`	80%	84.5%$^{\pm0.7}$	88.7%$^{\pm1.7}$	95.7%$^{\pm0.5}$	96.5%$^{\pm0.0}$	98.2%$^{\pm0.1}$	30.7%$^{\pm2.7}$	95.6%$^{\pm1.3}$	0.81×	104.7%
	60%	79.0%$^{\pm0.9}$	85.3%$^{\pm1.7}$	94.5%$^{\pm0.5}$	96.3%$^{\pm0.1}$	98.0%$^{\pm0.1}$	21.7%$^{\pm2.2}$	89.8%$^{\pm1.1}$	0.97×	87.5%
	40%	72.3%$^{\pm1.0}$	82.0%$^{\pm1.7}$	92.4%$^{\pm0.6}$	95.6%$^{\pm0.1}$	97.6%$^{\pm0.1}$	18.3%$^{\pm1.5}$	85.9%$^{\pm0.8}$	1.20×	70.3%
	20%	64.0%$^{\pm0.8}$	70.0%$^{\pm2.4}$	88.7%$^{\pm0.6}$	94.2%$^{\pm0.2}$	96.8%$^{\pm0.2}$	14.0%$^{\pm4.0}$	79.2%$^{\pm1.8}$	1.57×	47.6%
SD `Gemma-4-31B`								100.0%	1.96×	100.0%
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-9B`	∞	87.2%$^{\pm1.0}$	92.7%$^{\pm1.2}$	95.7%$^{\pm0.7}$	94.9%$^{\pm0.3}$	98.1%$^{\pm0.1}$	42.3%$^{\pm0.9}$	100.0%	1.00×	100.0%
	2048	84.6%$^{\pm0.6}$	90.7%$^{\pm2.9}$	97.2%$^{\pm0.5}$	93.2%$^{\pm0.4}$	98.2%$^{\pm0.1}$	38.7%$^{\pm1.5}$	97.7%$^{\pm0.9}$	1.20×	48.5%
	1024	83.3%$^{\pm1.2}$	88.6%$^{\pm2.3}$	94.6%$^{\pm0.5}$	93.2%$^{\pm0.3}$	98.1%$^{\pm0.1}$	35.0%$^{\pm1.0}$	95.2%$^{\pm0.8}$	1.23×	40.4%$^{\pm0.8}$
	512	83.1%$^{\pm0.5}$	89.1%$^{\pm2.0}$	93.8%$^{\pm0.6}$	91.6%$^{\pm0.4}$	97.8%$^{\pm0.2}$	30.3%$^{\pm1.8}$	92.9%$^{\pm0.9}$	1.25×	29.5%$^{\pm0.7}$
	256	84.0%$^{\pm1.0}$	89.3%$^{\pm3.0}$	92.4%$^{\pm0.8}$	89.4%$^{\pm0.3}$	97.6%$^{\pm0.2}$	34.7%$^{\pm1.5}$	94.2%$^{\pm0.9}$	1.25×	19.7%$^{\pm0.4}$
	0	80.8%$^{\pm0.4}$	86.7%$^{\pm1.8}$	91.1%$^{\pm0.8}$	86.9%$^{\pm0.5}$	97.8%$^{\pm0.2}$	33.0%$^{\pm1.0}$	91.8%$^{\pm0.7}$	1.28×	5.3%
Cascade `Qwen3.5-122B-A10B` ⇓ `Qwen3.5-9B`	80%	87.3%$^{\pm1.0}$	92.7%$^{\pm1.2}$	95.9%$^{\pm0.7}$	94.7%$^{\pm0.4}$	97.8%$^{\pm0.1}$	40.7%$^{\pm1.2}$	99.3%$^{\pm0.7}$	0.63×	90.8%
	60%	86.3%$^{\pm0.8}$	92.0%$^{\pm0.8}$	95.2%$^{\pm0.8}$	94.4%$^{\pm0.3}$	97.8%$^{\pm0.2}$	37.0%$^{\pm2.1}$	97.4%$^{\pm1.0}$	0.72×	72.1%
	40%	84.5%$^{\pm0.6}$	92.0%$^{\pm0.8}$	94.5%$^{\pm0.9}$	93.6%$^{\pm0.5}$	97.7%$^{\pm0.2}$	34.7%$^{\pm1.8}$	95.9%$^{\pm0.9}$	0.85×	52.8%
	20%	82.5%$^{\pm0.7}$	89.3%$^{\pm1.2}$	93.0%$^{\pm0.9}$	91.9%$^{\pm0.6}$	97.8%$^{\pm0.1}$	35.0%$^{\pm2.1}$	94.6%$^{\pm1.0}$	1.02×	31.0%
SD `Qwen3.5-122B-A10B`								100.0%	1.49×	100.0%
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-4B`	∞	85.7%$^{\pm0.8}$	94.0%$^{\pm0.7}$	96.0%$^{\pm0.5}$	93.7%$^{\pm0.5}$	98.4%$^{\pm0.0}$	42.3%$^{\pm0.9}$	100.0%	1.00×	100.0%
	2048	82.1%$^{\pm0.5}$	83.3%$^{\pm1.8}$	96.5%$^{\pm0.4}$	94.2%$^{\pm0.7}$	98.0%$^{\pm0.1}$	29.0%$^{\pm0.0}$	92.3%$^{\pm0.5}$	1.60×	48.1%$^{\pm1.0}$
	1024	80.4%$^{\pm0.7}$	86.7%$^{\pm1.8}$	95.1%$^{\pm0.3}$	92.8%$^{\pm0.4}$	97.6%$^{\pm0.2}$	29.0%$^{\pm1.0}$	92.0%$^{\pm0.6}$	1.72×	39.7%$^{\pm1.0}$
	512	78.2%$^{\pm0.8}$	83.3%$^{\pm1.5}$	92.0%$^{\pm0.8}$	88.7%$^{\pm0.6}$	97.2%$^{\pm0.3}$	23.0%$^{\pm2.5}$	87.3%$^{\pm1.1}$	1.77×	28.8%$^{\pm0.8}$
	256	77.9%$^{\pm1.3}$	84.7%$^{\pm1.7}$	87.7%$^{\pm0.9}$	85.8%$^{\pm0.5}$	97.2%$^{\pm0.2}$	23.5%$^{\pm0.5}$	86.4%$^{\pm0.6}$	1.80×	18.8%$^{\pm0.5}$
	0	76.6%$^{\pm1.1}$	83.3%$^{\pm1.8}$	84.5%$^{\pm0.6}$	81.2%$^{\pm0.9}$	96.7%$^{\pm0.3}$	22.0%$^{\pm1.0}$	83.8%$^{\pm0.6}$	1.87×	4.4%$^{\pm0.1}$
Cascade `Qwen3.5-122B-A10B` ⇓ `Qwen3.5-4B`	80%	85.4%$^{\pm0.7}$	93.3%$^{\pm0.0}$	95.5%$^{\pm0.5}$	92.9%$^{\pm0.3}$	98.0%$^{\pm0.0}$	38.0%$^{\pm1.5}$	97.8%$^{\pm0.7}$	0.75×	89.8%
	60%	84.2%$^{\pm0.4}$	92.0%$^{\pm0.8}$	93.9%$^{\pm0.4}$	91.8%$^{\pm0.3}$	97.9%$^{\pm0.1}$	35.7%$^{\pm1.5}$	96.0%$^{\pm0.7}$	0.88×	72.1%
	40%	81.9%$^{\pm0.6}$	90.0%$^{\pm1.1}$	92.3%$^{\pm0.5}$	90.4%$^{\pm0.5}$	97.7%$^{\pm0.1}$	29.3%$^{\pm1.5}$	92.1%$^{\pm0.7}$	1.07×	51.6%
	20%	79.1%$^{\pm0.9}$	87.3%$^{\pm1.2}$	89.0%$^{\pm0.5}$	87.0%$^{\pm0.8}$	97.4%$^{\pm0.2}$	27.3%$^{\pm0.7}$	89.0%$^{\pm0.5}$	1.36×	30.4%
SD `Qwen3.5-122B-A10B`								100.0%	1.49×	100.0%
`Qwen3.5-27B` ⇓ `Qwen3.5-4B`	∞	85.6%$^{\pm1.1}$	94.6%$^{\pm1.7}$	94.9%$^{\pm0.6}$	92.9%$^{\pm0.6}$	98.2%$^{\pm0.1}$	43.0%$^{\pm1.5}$	100.0%	1.00×	100.0%
	2048	81.7%$^{\pm0.9}$	86.0%$^{\pm1.2}$	95.6%$^{\pm0.6}$	92.1%$^{\pm0.5}$	98.0%$^{\pm0.1}$	26.0%$^{\pm1.2}$	91.1%$^{\pm0.8}$	2.61×	45.5%$^{\pm1.3}$
	1024	79.6%$^{\pm1.1}$	84.0%$^{\pm1.2}$	93.4%$^{\pm0.6}$	92.1%$^{\pm0.4}$	97.7%$^{\pm0.2}$	24.3%$^{\pm1.9}$	89.2%$^{\pm0.9}$	3.14×	37.6%$^{\pm1.4}$
	512	79.3%$^{\pm0.7}$	84.0%$^{\pm1.2}$	91.2%$^{\pm0.9}$	88.2%$^{\pm0.8}$	97.5%$^{\pm0.1}$	25.3%$^{\pm2.6}$	88.4%$^{\pm1.2}$	3.49×	28.2%
	256	78.2%$^{\pm0.5}$	80.7%$^{\pm1.9}$	88.2%$^{\pm0.9}$	86.5%$^{\pm1.3}$	97.2%$^{\pm0.2}$	20.7%$^{\pm1.2}$	85.0%$^{\pm0.8}$	3.72×	20.2%
	0	76.6%$^{\pm1.0}$	85.2%$^{\pm2.8}$	84.5%$^{\pm0.6}$	81.3%$^{\pm1.0}$	96.7%$^{\pm0.3}$	22.0%$^{\pm1.0}$	84.3%$^{\pm0.8}$	4.01×	8.7%$^{\pm0.2}$
Cascade `Qwen3.5-27B` ⇓ `Qwen3.5-4B`	80%	85.3%$^{\pm1.1}$	94.0%$^{\pm1.9}$	94.5%$^{\pm0.5}$	92.1%$^{\pm0.5}$	97.9%$^{\pm0.1}$	38.3%$^{\pm0.9}$	97.7%$^{\pm0.8}$	0.95×	93.8%
	60%	84.2%$^{\pm1.2}$	93.3%$^{\pm2.4}$	92.8%$^{\pm0.4}$	91.0%$^{\pm0.5}$	97.8%$^{\pm0.1}$	35.7%$^{\pm1.8}$	95.9%$^{\pm1.1}$	1.18×	76.1%
	40%	81.3%$^{\pm1.6}$	91.2%$^{\pm1.6}$	91.1%$^{\pm0.6}$	90.1%$^{\pm0.8}$	97.7%$^{\pm0.1}$	29.0%$^{\pm0.6}$	91.9%$^{\pm0.7}$	1.54×	54.4%
	20%	78.5%$^{\pm1.1}$	87.9%$^{\pm2.4}$	88.7%$^{\pm0.5}$	86.8%$^{\pm0.9}$	97.4%$^{\pm0.2}$	27.3%$^{\pm0.7}$	89.0%$^{\pm0.8}$	2.23×	33.3%
SD `Qwen3.5-27B`								100.0%	2.11×	100.0%

D ENERGY RESULTS

Table 9 extends Table 4 to all model pairs.

Measurement methodology. Energy is measured in the same single-request runs that give the speedups (Appendix H): GPQA prompts sent one at a time (concurrency 1), each forced to generate 12000 output tokens, at temperature zero, on NVIDIA H200 GPUs. For each of these GPUs we read the cumulative energy counter exposed by NVML (`nvmlDeviceGetTotalEnergyConsumption`, in millijoules) immediately before the first request is dispatched and after the last response is received, over the same wall-clock window used for throughput. The energy of a run is the sum over GPUs of the counter differences, and the energy per token is this total divided by the number of output tokens of the run. Because the whole window is counted, the figure includes the power a GPU draws while waiting, e.g. M_Q's GPU while M_P writes the prefixes, and the prefill of the prefix by M_Q. It covers GPU energy only; CPU, memory, networking, and cooling are excluded.

Table 9: Normalized accuracy and GPU energy of primed decoding for all model pairs. $\tilde{Acc}$ is the accuracy relative to pure M_P decoding, averaged over the six benchmarks (Table 1). Energy represents the average consumption per generated token (Joule/token). Saving is computed relative to standalone M_P decoding. The reference is the most energy-efficient placement of pure M_P that we measured (for the Gemma pairs, one GPU rather than the two used by the primed pipeline).

$(M_p \Rightarrow M_q)$	K	$\tilde{Acc}\uparrow$	**Energy (J/token)** $\downarrow$	**Saving** $\uparrow$
	∞	100.0%	0.520	0.0%
`Gemma-4-26B-A4B` $\Downarrow$ `Gemma-4-E2B`	2048	$83.5\%^{\pm 1.0}$	0.300	42.3%
	1024	$77.7\%^{\pm 1.1}$	0.293	43.6%
	512	$73.8\%^{\pm 1.2}$	0.265	49.0%
	256	$72.5\%^{\pm 0.8}$	0.264	49.2%
	0	$68.7\%^{\pm 1.1}$	0.182	65.0%
	∞	100.0%	1.775	0.0%
`Gemma-4-31B` $\Downarrow$ `Gemma-4-E4B`	2048	$90.1\%^{\pm 1.3}$	0.672	62.1%
	1024	$83.2\%^{\pm 0.7}$	0.567	68.1%
	512	$77.9\%^{\pm 0.5}$	0.485	72.7%
	256	$74.9\%^{\pm 0.8}$	0.488	72.5%
	0	$71.3\%^{\pm 0.7}$	0.327	81.6%
	∞	100.0%	1.761	0.0%
`Qwen3.5-122B-A10B` $\Downarrow$ `Qwen3.5-9B`	2048	$97.7\%^{\pm 0.9}$	0.889	49.5%
	1024	$95.2\%^{\pm 0.8}$	0.777	55.9%
	512	$92.9\%^{\pm 0.9}$	0.715	59.4%
	256	$94.2\%^{\pm 0.9}$	0.712	59.6%
	0	$91.8\%^{\pm 0.7}$	0.458	74.0%
	∞	100.0%	1.761	0.0%
`Qwen3.5-122B-A10B` $\Downarrow$ `Qwen3.5-4B`	2048	$92.3\%^{\pm 0.5}$	0.708	59.8%
	1024	$92.0\%^{\pm 0.6}$	0.565	67.9%
	512	$87.3\%^{\pm 1.1}$	0.512	70.9%
	256	$86.4\%^{\pm 0.6}$	0.457	74.0%
	0	$83.8\%^{\pm 0.6}$	0.297	83.1%
	∞	100.0%	1.367	0.0%
`Qwen3.5-27B` $\Downarrow$ `Qwen3.5-4B`	2048	$91.1\%^{\pm 0.8}$	0.571	58.2%
	1024	$89.2\%^{\pm 0.9}$	0.475	65.3%
	512	$88.4\%^{\pm 1.2}$	0.422	69.1%
	256	$85.0\%^{\pm 0.8}$	0.414	69.7%
	0	$84.3\%^{\pm 0.8}$	0.297	78.2%

E PRIMED TOKENS

The priming budget K is an upper bound: when M_P completes its answer within the first K tokens, no handoff takes place and fewer tokens are primed. Table 10 reports the share of all output tokens

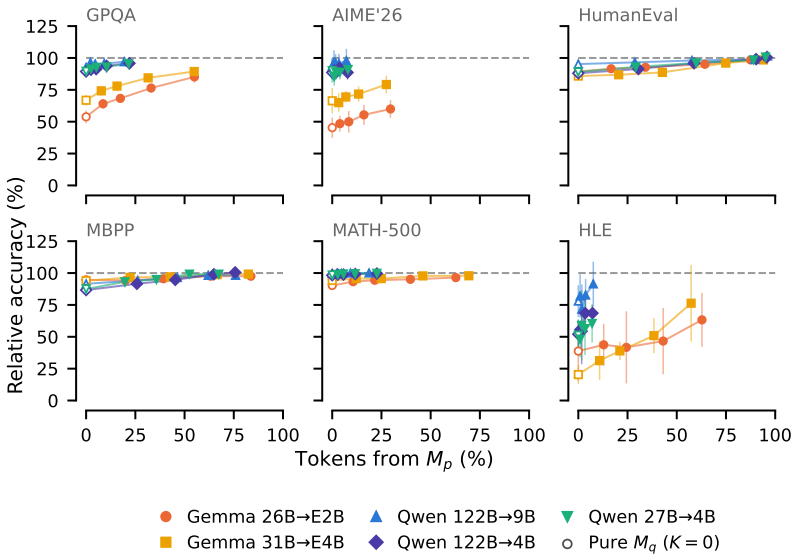

Figure 7: Accuracy relative to pure M_P decoding against the share of output tokens generated by M_P (Table 10), per benchmark. Each marker is one priming length $K \in \{256, 512, 1024, 2048\}$, joined in order of K; hollow markers at 0% are pure M_Q decoding. Error bars are 95% t-intervals over seeds.

generated by M_P, and Figure 7 relates it to the accuracy retained. On GPQA, AIME'26, MATH-500 and HLE the prefix is a small fraction of the generation: at $K = 1024$ it is 13–46% of the output tokens for the Gemma pairs, and only 3–12% for the Qwen pairs, whose reasoning traces are much longer. On the short code benchmarks (HumanEval, MBPP) many answers end within the prefix at larger K, so M_P writes most of the output.

Table 10: Share of all output tokens generated by M_P (%). Mean and s.e.m. over seeds. Realized primed tokens fall short of K when M_P finishes the answer during priming.

$(M_p \Rightarrow M_q)$	K	GPQA	AIME'26	HumanEval	MBPP	MATH-500	HLE
`Gemma-4-26B-A4B` ⇓ `Gemma-4-E2B`	256	$8.6^{\pm 0.0}$	$3.9^{\pm 0.1}$	$16.7^{\pm 0.2}$	$20.0^{\pm 0.1}$	$10.6^{\pm 0.0}$	$12.9^{\pm 0.1}$
	512	$17.4^{\pm 0.1}$	$8.4^{\pm 0.1}$	$34.1^{\pm 0.2}$	$39.3^{\pm 0.3}$	$21.6^{\pm 0.1}$	$24.4^{\pm 0.3}$
	1024	$33.0^{\pm 0.2}$	$16.1^{\pm 0.2}$	$64.3^{\pm 0.1}$	$63.3^{\pm 0.3}$	$39.7^{\pm 0.3}$	$43.1^{\pm 0.2}$
	2048	$55.0^{\pm 0.3}$	$29.6^{\pm 0.4}$	$87.5^{\pm 0.2}$	$83.7^{\pm 0.4}$	$62.8^{\pm 0.4}$	$62.8^{\pm 0.3}$
`Gemma-4-31B` ⇓ `Gemma-4-E4B`	256	$7.7^{\pm 0.1}$	$3.3^{\pm 0.1}$	$20.5^{\pm 0.1}$	$22.3^{\pm 0.3}$	$12.2^{\pm 0.1}$	$10.8^{\pm 0.4}$
	512	$15.7^{\pm 0.1}$	$6.9^{\pm 0.1}$	$42.7^{\pm 0.6}$	$42.7^{\pm 0.5}$	$24.9^{\pm 0.2}$	$21.1^{\pm 0.9}$
	1024	$31.5^{\pm 0.2}$	$13.4^{\pm 0.2}$	$74.9^{\pm 0.5}$	$66.6^{\pm 0.6}$	$46.0^{\pm 0.3}$	$38.4^{\pm 0.2}$
	2048	$54.9^{\pm 0.2}$	$27.5^{\pm 0.7}$	$94.0^{\pm 0.3}$	$82.4^{\pm 0.3}$	$69.4^{\pm 0.7}$	$57.3^{\pm 0.7}$
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-9B`	256	$2.2^{\pm 0.0}$	$0.9^{\pm 0.0}$	$28.7^{\pm 0.2}$	$26.2^{\pm 2.2}$	$2.3^{\pm 0.0}$	$0.9^{\pm 0.0}$
	512	$4.6^{\pm 0.1}$	$1.7^{\pm 0.1}$	$57.7^{\pm 0.6}$	$46.3^{\pm 3.4}$	$4.7^{\pm 0.0}$	$1.8^{\pm 0.0}$
	1024	$9.2^{\pm 0.0}$	$3.4^{\pm 0.1}$	$89.6^{\pm 2.3}$	$62.0^{\pm 3.9}$	$9.3^{\pm 0.1}$	$3.7^{\pm 0.1}$
	2048	$19.4^{\pm 0.1}$	$7.2^{\pm 0.2}$	$96.2^{\pm 1.7}$	$75.9^{\pm 3.1}$	$18.7^{\pm 0.2}$	$7.6^{\pm 0.2}$
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-4B`	256	$2.5^{\pm 0.0}$	$1.0^{\pm 0.0}$	$30.5^{\pm 0.4}$	$25.8^{\pm 2.0}$	$2.8^{\pm 0.1}$	$1.0^{\pm 0.0}$
	512	$5.1^{\pm 0.1}$	$1.9^{\pm 0.0}$	$58.8^{\pm 1.0}$	$45.3^{\pm 3.7}$	$5.7^{\pm 0.1}$	$1.7^{\pm 0.0}$
	1024	$10.4^{\pm 0.1}$	$3.7^{\pm 0.1}$	$90.3^{\pm 1.0}$	$64.9^{\pm 4.5}$	$11.7^{\pm 0.1}$	$3.5^{\pm 0.0}$
	2048	$22.0^{\pm 0.1}$	$7.8^{\pm 0.1}$	$95.7^{\pm 1.5}$	$75.7^{\pm 3.8}$	$22.7^{\pm 0.3}$	$7.2^{\pm 0.1}$
`Qwen3.5-27B` ⇓ `Qwen3.5-4B`	256	$2.5^{\pm 0.0}$	$1.0^{\pm 0.0}$	$29.2^{\pm 0.7}$	$19.7^{\pm 4.2}$	$2.8^{\pm 0.0}$	$0.9^{\pm 0.0}$
	512	$5.2^{\pm 0.0}$	$1.9^{\pm 0.0}$	$59.9^{\pm 1.2}$	$35.7^{\pm 7.0}$	$5.7^{\pm 0.1}$	$1.7^{\pm 0.0}$
	1024	$10.4^{\pm 0.1}$	$3.9^{\pm 0.0}$	$89.0^{\pm 1.3}$	$52.5^{\pm 6.5}$	$11.6^{\pm 0.2}$	$3.4^{\pm 0.0}$
	2048	$21.6^{\pm 0.1}$	$8.0^{\pm 0.1}$	$95.1^{\pm 1.6}$	$67.4^{\pm 5.4}$	$22.7^{\pm 0.3}$	$7.0^{\pm 0.1}$

F Implementation Details

F.1 Chat Templating and Prefill Injection

This section describes the text-level handoff used for the cross-family and interleaved schedules of Section 4.2. Within a model family, the handoff passes token IDs inside vLLM (Appendix B.1), and none of the re-rendering below is needed.

A core engineering challenge in primed decoding is correctly injecting the prefix generated by M_p into M_q's context. Since the two models may use different chat templates, we cannot simply concatenate token sequences; instead, we re-render the full conversation from scratch using each model's own tokenizer at every handoff.

Concretely, after M_p produces k tokens, the `completions/` client decomposes the raw completion text into a structured `Message` object containing separate `reasoning` and `content` fields, using the model-specific vLLM reasoning parser. The resulting message is appended to the conversation history as a partial assistant turn. When M_q is called next, its tokenizer re-applies its own chat template to the full history via `apply_chat_template`, dynamically passing the target field name (e.g. `"reasoning"` or `"content"`) to the `continue_final_message` argument, which directs the tokenizer to reopen and extend exactly that segment of the partial turn in M_q's native format.

The target field is resolved adaptively: if the prefill contains only reasoning tokens (i.e. the ⟨*think*⟩ block is still open), the argument is set to the model's reasoning field name so the tokenizer correctly reopens the reasoning segment; if content tokens are already present, it is set to `"content"` instead; if no prefill exists, it is set to `False` and a fresh generation prompt is added. This distinction is non-trivial: passing an incorrect field name - or `True` instead of the field string - causes the tokenizer to emit a malformed prompt.

After re-templating the *prefill text*, the contiguous suffix of the rendered prompt corresponding to tokens already generated by M_p is recovered via exact string matching of the generated reasoning and content segments against the newly rendered prompt. This text is then passed to the parser so that repeated prefix tokens are correctly stripped from M_q's raw output, preventing duplication in the final response.

G Model Pricing

We estimate the dollar cost of every method from the token counts of our runs and the per-token prices of Table 11. Prices are taken from OpenRouter at the time of release of the models. `Qwen3.5-4B`, `Gemma-4-E4B`, and `Gemma-4-E2B` have no public price; we assign them the price of publicly listed models of comparable size. All three are used only as M_Q.

Model	Input ($/M tok)	Output ($/M tok)
`Qwen3.5-122B-A10B`	$0.45	$3.50
`Qwen3.5-27B`	$0.195	$1.56
`Qwen3.5-9B`	$0.10	$0.15
`Qwen3.5-4B`	$0.06	$0.14
`Gemma-4-31B`	$0.25	$0.60
`Gemma-4-26B-A4B`	$0.15	$0.50
`Gemma-4-E4B`	$0.06	$0.14
`Gemma-4-E2B`	$0.03	$0.10

Table 11: Per-token prices (USD per million tokens) used for all cost estimates in this paper, as configured on 13 September 2026.

Cost of a query. Let I be the size of a query and N the number of tokens generated in its response, reasoning tokens included; $c^{(i)}$ and $c^{(o)}$ are the input and output prices of a model. Pure M_P decoding costs $I\,c_p^{(i)} + N\,c_p^{(o)}$, and pure M_Q decoding is priced in the same way. With a priming budget k, M_P reads the prompt and writes the first k tokens, and M_Q reads the prompt and the

prefix and writes the remaining $N - k$ tokens:

$$\mathcal{C}_{\text{prime}} = \underbrace{I\, c_p^{(i)} + k\, c_p^{(o)}}_{M_P} + \underbrace{(I + k)\, c_q^{(i)} + (N - k)\, c_q^{(o)}}_{M_Q}. \tag{5}$$

If M_P completes its response within k tokens ($N \leq k$), no handoff takes place and the query costs $I\, c_p^{(i)} + N\, c_p^{(o)}$. Every term uses the token counts measured on that query, so primed runs are charged for their own response lengths, which can differ from those of pure M_P decoding.

Reported cost. For each benchmark and seed we divide the mean cost per query of a method by that of pure M_P decoding on the same benchmark and seed. The Cost column of our tables averages this ratio over benchmarks; its $\pm$ is the s.e.m. over seeds, shown where the per-seed costs are available.

Why output tokens dominate. On the reasoning benchmarks, responses are one to two orders of magnitude longer than prompts. Across the pure-M_P baselines, GPQA prompts contain 269–274 tokens and responses 5836–11725 tokens; AIME prompts contain 229–232 tokens and responses 8035–26105 tokens; HumanEval is shorter, with 180–191 prompt tokens and 832–2320 response tokens. Output prices are also higher than input prices for every model in Table 11, so the cost is set almost entirely by the output term. Since PRIMING moves the $N - k$ tokens after the handoff to M_Q's lower output price, while the extra charge is only M_Q reading $I+k$ tokens once (Equation (4)), its relative savings are largest on the long-reasoning benchmarks.

H THROUGHPUT ANALYSIS

All speedups and energies in the paper are measured at concurrency 1, on a controlled workload of $N = 12000$ output tokens per request, and are relative to pure M_P decoding measured in the same way. Table 12 additionally reports throughput under 32 concurrent requests, which is used nowhere else in the paper.

Single-request latency. Speedups in the paper are ratios of single-request latency: one request at a time, 12000 forced output tokens, each model on its own GPUs as in Table 12; the speedup of a condition is the mean latency of pure M_P divided by its own. With one request at a time the number of GPUs does not matter, since a request only ever runs on one copy of each model. The cascade's latency is composed per query as that of M_Q plus, for the deferred share, that of M_P. The cascade has no pipeline measurement, and its speedup is composed from the measured pure-model runs (see below).

All controlled measurements are collected on NVIDIA H200 GPUs with 141 GB of HBM3e memory. We serve both M_P and M_Q with `vLLM`. For model pairs in which each model fits on a single GPU, M_P and M_Q are placed on separate H200 GPUs with tensor parallelism set to one. For the 122B Qwen model, M_P is distributed across two H200 GPUs with tensor parallelism set to two, while M_Q is served on a third H200 GPU.

Throughput at concurrency 32. This second measurement, reported only in Table 12, uses temperature zero, 50 requests, client concurrency 32, and 12000 output tokens per request; each vLLM engine decodes at most 8 sequences at once, so the other requests wait in its queue. For any model condition, aggregate throughput is computed as $\tau = \frac{600000}{T_{\text{wall}}}$, where T_{wall} is measured from the dispatch of the batch until all HTTP responses have been received. This interval includes client-side queueing but excludes server startup. The same measurement is taken for pure M_P (τ_p), pure M_Q (τ_q), and for the full primed pipeline at each priming length k, with both models served as in the experiments. The throughput speedup of a condition is its measured throughput divided by τ_p.

Table 12 reports, for every pair, the measured throughput, the speedup over pure M_P, and the mean latency of a request, i.e. the time from its dispatch to the end of its response, queueing included. These speedups are much larger than the single-request ones used in the rest of the paper (e.g. $3.39\times$ against $1.50\times$ for `Gemma-4-31B` $\rightarrow$ `Gemma-4-E4B` at $k = 2048$), because the two measure different things. With one request at a time, only one model decodes at any moment, so the pipeline is at best as fast as M_Q. Under 32 concurrent requests, M_P writes the prefixes of some requests

while M_Q continues others on its own GPU, so the pipeline, which occupies the GPUs of both models (column GPUs), adds up the throughput of both engines, and requests also spend less time queued behind the at most 8 sequences decoded by each engine.

Table 12: **Measured throughput and latency at concurrency 32.** Aggregate output throughput, speedup over pure M_P, and mean latency of a request, on the controlled workload of Appendix H (50 GPQA prompts, 12000 output tokens each, client concurrency 32, at most 8 sequences decoded at once per engine). Latency includes queueing. Pure M_P ($K = \infty$) and pure M_Q ($K = 0$) in gray; SD is speculative decoding. GPUs is the number of GPUs a run occupies: a primed run uses those of both models; the baselines are re-measured on the same number of GPUs where available, and rows with fewer GPUs come from the original campaign. $\pm$ is the s.e.m. over requests.

($M_p \Rightarrow M_q$)	K	GPUs	Throughput (tok/s) ↑	Speedup ↑	Latency (s) ↓
`Gemma-4-26B-A4B` ⇓ `Gemma-4-E2B`	∞	1	$918^{\pm 87}$	$1.00\times$	$294^{\pm 16}$
	2048	2	$1747^{\pm 123}$	$1.90\times^{\pm 0.22}$	$153^{\pm 7}$
	1024	2	$1654^{\pm 132}$	$1.80\times^{\pm 0.22}$	$160^{\pm 8}$
	512	2	$1657^{\pm 128}$	$1.81\times^{\pm 0.22}$	$161^{\pm 8}$
	256	2	$1599^{\pm 135}$	$1.74\times^{\pm 0.22}$	$165^{\pm 9}$
	0	1	$1630^{\pm 140}$	$1.78\times^{\pm 0.23}$	$162^{\pm 8}$
	SD	1	$1146^{\pm 141}$	$1.25\times^{\pm 0.19}$	$205^{\pm 12}$
`Gemma-4-31B` ⇓ `Gemma-4-E4B`	∞	1	$327^{\pm 29}$	$1.00\times$	$809^{\pm 44}$
	2048	2	$1107^{\pm 54}$	$3.39\times^{\pm 0.35}$	$275^{\pm 11}$
	1024	2	$1056^{\pm 45}$	$3.23\times^{\pm 0.32}$	$265^{\pm 9}$
	512	2	$1116^{\pm 82}$	$3.42\times^{\pm 0.40}$	$241^{\pm 11}$
	256	2	$1082^{\pm 79}$	$3.31\times^{\pm 0.38}$	$247^{\pm 12}$
	0	1	$1108^{\pm 98}$	$3.39\times^{\pm 0.43}$	$237^{\pm 13}$
	SD	1	$1144^{\pm 108}$	$3.50\times^{\pm 0.46}$	$232^{\pm 13}$
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-9B`	∞	2	$609^{\pm 54}$	$1.00\times$	$452^{\pm 24}$
	2048	3	$1123^{\pm 38}$	$1.84\times^{\pm 0.18}$	$256^{\pm 8}$
	1024	3	$1151^{\pm 63}$	$1.89\times^{\pm 0.20}$	$240^{\pm 9}$
	512	3	$1124^{\pm 79}$	$1.85\times^{\pm 0.21}$	$241^{\pm 11}$
	256	3	$1125^{\pm 88}$	$1.85\times^{\pm 0.22}$	$238^{\pm 12}$
	0	1	$1168^{\pm 104}$	$1.92\times^{\pm 0.24}$	$226^{\pm 12}$
	SD	2	$837^{\pm 74}$	$1.38\times^{\pm 0.17}$	$328^{\pm 17}$
`Qwen3.5-122B-A10B` ⇓ `Qwen3.5-4B`	∞	2	$609^{\pm 54}$	$1.00\times$	$452^{\pm 24}$
	2048	3	$1501^{\pm 42}$	$2.47\times^{\pm 0.23}$	$197^{\pm 6}$
	1024	3	$1539^{\pm 71}$	$2.53\times^{\pm 0.25}$	$182^{\pm 6}$
	512	3	$1582^{\pm 100}$	$2.60\times^{\pm 0.28}$	$173^{\pm 7}$
	256	3	$1662^{\pm 119}$	$2.73\times^{\pm 0.31}$	$163^{\pm 8}$
	0	1	$1623^{\pm 143}$	$2.67\times^{\pm 0.33}$	$163^{\pm 9}$
	SD	2	$837^{\pm 74}$	$1.38\times^{\pm 0.17}$	$328^{\pm 17}$
`Qwen3.5-27B` ⇓ `Qwen3.5-4B`	∞	1	$418^{\pm 37}$	$1.00\times$	$630^{\pm 34}$
	2048	2	$1346^{\pm 40}$	$3.22\times^{\pm 0.30}$	$224^{\pm 7}$
	1024	2	$1535^{\pm 57}$	$3.67\times^{\pm 0.36}$	$186^{\pm 6}$
	512	2	$1584^{\pm 88}$	$3.79\times^{\pm 0.40}$	$176^{\pm 7}$
	256	2	$1578^{\pm 110}$	$3.77\times^{\pm 0.43}$	$172^{\pm 8}$
	0	1	$1623^{\pm 143}$	$3.88\times^{\pm 0.49}$	$163^{\pm 9}$
	SD	1	$872^{\pm 78}$	$2.09\times^{\pm 0.26}$	$307^{\pm 16}$

Sampling. The accuracy runs sample with each family's recommended settings (Appendix B), whereas the throughput workload decodes greedily and ignores EOS, so that every request produces exactly N output tokens. The cost of a decoding step is set by the model, the batch size and the context length, not by which token is selected, and applying temperature and top-p/top-k to the logits is a small fraction of a step; fixing the output length removes the variance due to response length without changing the per-token cost. The measured speedups therefore carry over to the sampled runs, up to their response lengths, which vary by benchmark (Appendix G): shorter responses give the prefix more weight and lower the speedup of PRIMING (Section 5.1). The exception is speculative

decoding, whose acceptance rate is typically higher under greedy decoding than under sampling, so its measured speedups are, if anything, optimistic.

Why a fixed-length workload. The speedup of PRIMING is not a constant of the method but depends on each query. For a response of L tokens, M_P writes $\min(k, L)$ of them and M_Q the remaining $\max(L - k, 0)$, so with per-token latencies t_p and t_q the latency of a request is about $\min(k, L)\, t_p + \max(L - k, 0)\, t_q$. The same k therefore gives close to the speed of M_Q on a long reasoning trace and no speedup at all on a short answer that ends within the prefix. L varies with the query, the model and the sampling seed, and its distribution differs sharply across benchmarks (Table 10): at $K = 1024$, M_P writes a small fraction of the output on GPQA, AIME'26, MATH-500 and HLE, but most of it on HumanEval and MBPP. A latency averaged over the runs of one benchmark would thus mostly describe the length distribution of that benchmark, and would not carry over to another workload. We therefore measure the per-token cost of each condition on a fixed long generation, which is the regime PRIMING targets, and report the realized share of tokens written by M_P separately (Table 10); with the formula above, the two give the expected speedup for any response length. The reported speedups are thus an upper bound for queries whose answers are short compared with k, where PRIMING reduces to M_P in both speed and accuracy. Costs, instead, are computed from the realized token counts of each benchmark run (Appendix G), and so already reflect this dependence.

I SHORT PRIMING LENGTHS PER BENCHMARK

Table 13 breaks down by benchmark the short-priming-length results summarised in Table 3.

Table 13: Short priming lengths, per benchmark. Accuracy per benchmark, and $\tilde{Acc}$ (accuracy relative to pure M_P decoding, averaged over the five benchmarks; HLE was not run at these lengths), Speedup and Cost relative to pure M_P decoding

$(M_p \Rightarrow M_q)$	K	GPQA	AIME'26	HumanEval	MBPP	MATH-500	$\tilde{Acc}\uparrow$	Speedup ↑	Cost ↓
	∞	$80.1\%^{\pm 0.7}$	$86.7\%^{\pm 2.6}$	$97.0\%^{\pm 0.4}$	$96.3\%^{\pm 0.3}$	$97.6\%^{\pm 0.2}$	100.0%	1.00×	100.0%
Gemma-4-26B-A4B ⇓ Gemma-4-E2B	256	$51.3\%^{\pm 1.0}$	$42.0\%^{\pm 1.5}$	$88.8\%^{\pm 0.6}$	$90.7\%^{\pm 0.2}$	$91.0\%^{\pm 0.4}$	$78.3\%^{\pm 0.6}$	1.04×	$17.2\%^{\pm 0.1}$
	128	$44.8\%^{\pm 1.1}$	$44.0\%^{\pm 1.7}$	$88.8\%^{\pm 0.6}$	$90.2\%^{\pm 0.6}$	$90.0\%^{\pm 0.4}$	$76.8\%^{\pm 0.6}$	1.02×	$15.0\%^{\pm 0.1}$
	64	$45.4\%^{\pm 2.0}$	$44.0\%^{\pm 1.7}$	$88.3\%^{\pm 0.6}$	$90.4\%^{\pm 0.3}$	$89.1\%^{\pm 0.0}$	$76.7\%^{\pm 0.7}$	1.03×	$13.8\%^{\pm 0.1}$
	32	$45.9\%^{\pm 1.3}$	$48.0\%^{\pm 3.3}$	$88.4\%^{\pm 0.7}$	$90.3\%^{\pm 0.6}$	$88.4\%^{\pm 0.3}$	$77.6\%^{\pm 0.9}$	1.05×	$13.2\%^{\pm 0.1}$
	16	$43.1\%^{\pm 0.5}$	$45.3\%^{\pm 2.0}$	$87.0\%^{\pm 0.6}$	$90.0\%^{\pm 0.3}$	$88.0\%^{\pm 0.5}$	$75.9\%^{\pm 0.6}$	1.03×	$13.0\%^{\pm 0.1}$
	0	$43.1\%^{\pm 1.4}$	$39.3\%^{\pm 2.2}$	$86.7\%^{\pm 0.8}$	$91.2\%^{\pm 0.3}$	$88.1\%^{\pm 0.3}$	$74.7\%^{\pm 0.7}$	1.03×	$10.4\%^{\pm 0.0}$
	∞	$86.5\%^{\pm 0.8}$	$89.3\%^{\pm 1.2}$	$97.1\%^{\pm 0.4}$	$96.5\%^{\pm 0.0}$	$98.5\%^{\pm 0.3}$	100.0%	1.00×	100.0%
Gemma-4-31B ⇓ Gemma-4-E4B	256	$64.2\%^{\pm 1.0}$	$58.0\%^{\pm 2.2}$	$84.3\%^{\pm 0.8}$	$93.2\%^{\pm 0.4}$	$94.2\%^{\pm 0.2}$	$83.6\%^{\pm 0.6}$	1.99×	$31.7\%^{\pm 0.1}$
	128	$58.4\%^{\pm 0.7}$	$58.7\%^{\pm 1.2}$	$84.6\%^{\pm 0.7}$	$92.5\%^{\pm 0.4}$	$93.1\%^{\pm 0.3}$	$82.1\%^{\pm 0.4}$	2.04×	$28.2\%^{\pm 0.2}$
	64	$56.9\%^{\pm 0.5}$	$61.3\%^{\pm 2.2}$	$81.8\%^{\pm 1.1}$	$91.0\%^{\pm 0.4}$	$93.0\%^{\pm 0.4}$	$81.5\%^{\pm 0.6}$	2.06×	$26.2\%^{\pm 0.1}$
	32	$57.4\%^{\pm 1.1}$	$60.0\%^{\pm 1.9}$	$85.0\%^{\pm 0.7}$	$90.0\%^{\pm 0.2}$	$92.0\%^{\pm 0.1}$	$81.6\%^{\pm 0.6}$	2.08×	$25.6\%^{\pm 0.1}$
	16	$59.2\%^{\pm 1.4}$	$59.3\%^{\pm 1.1}$	$83.5\%^{\pm 0.6}$	$91.5\%^{\pm 0.4}$	$92.7\%^{\pm 0.1}$	$82.0\%^{\pm 0.5}$	2.08×	$25.2\%^{\pm 0.2}$
	0	$57.8\%^{\pm 1.2}$	$59.3\%^{\pm 3.1}$	$83.3\%^{\pm 0.9}$	$90.7\%^{\pm 0.3}$	$92.8\%^{\pm 0.2}$	$81.4\%^{\pm 0.8}$	2.29×	$19.2\%^{\pm 0.2}$

J PLOTS

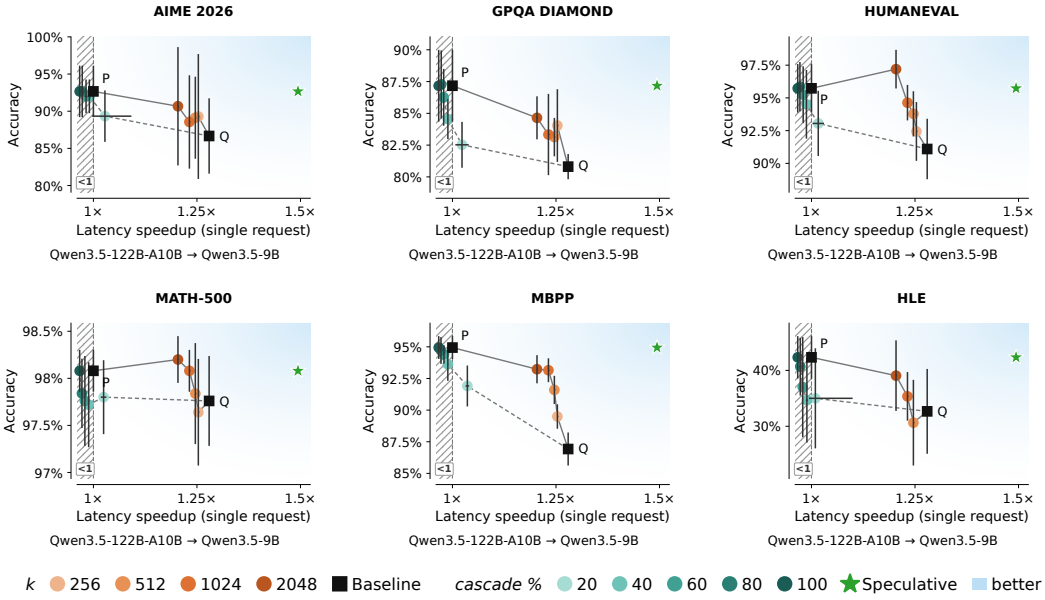

Figure 8: Accuracy vs latency speedup, `Qwen3.5-122B-A10B` → `Qwen3.5-9B`.

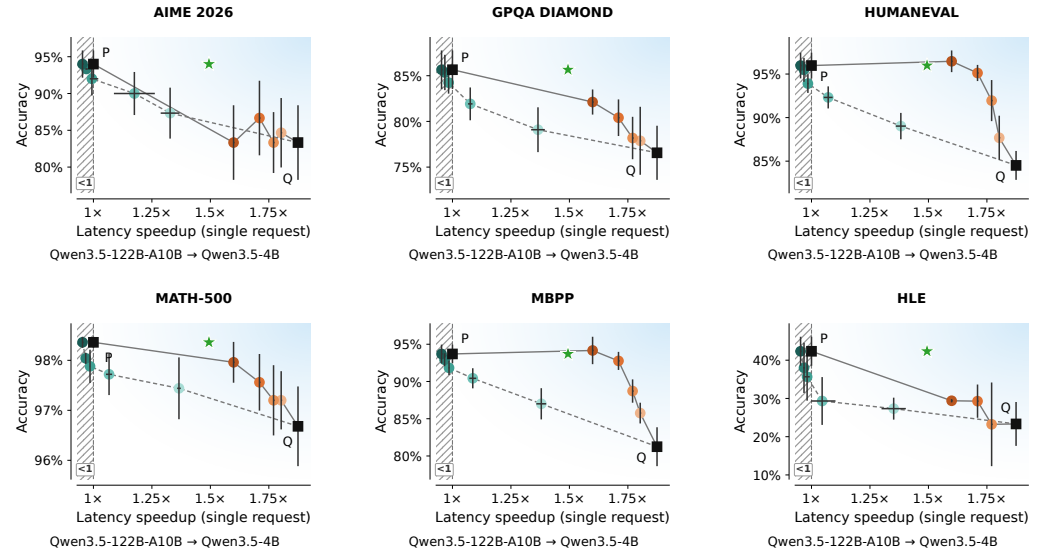

Figure 9: Accuracy vs latency speedup, `Qwen3.5-122B-A10B` → `Qwen3.5-4B`.

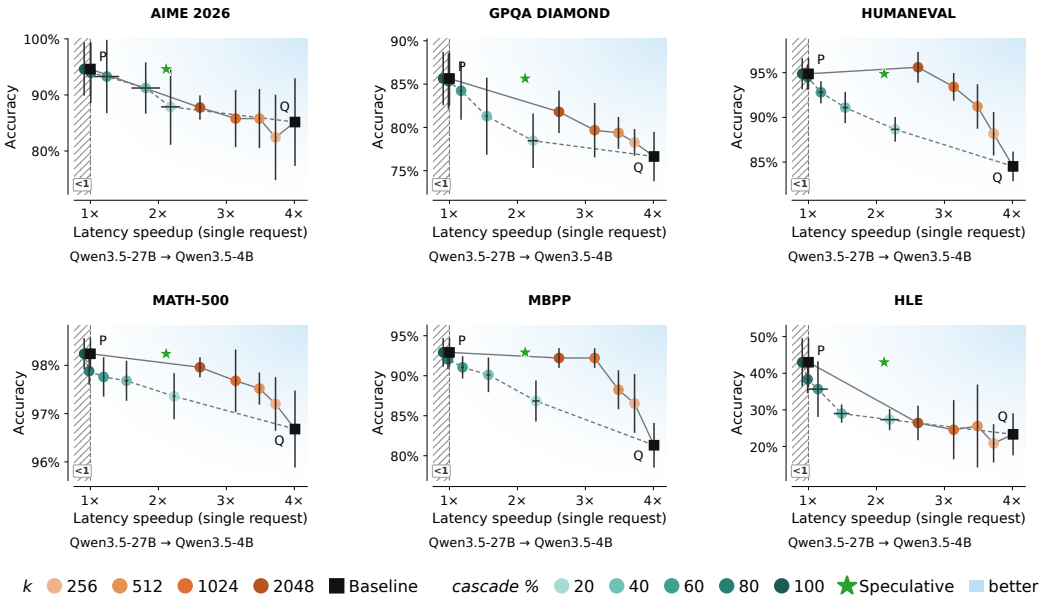

Figure 10: Accuracy vs latency speedup, `Qwen3.5-27B` → `Qwen3.5-4B`.

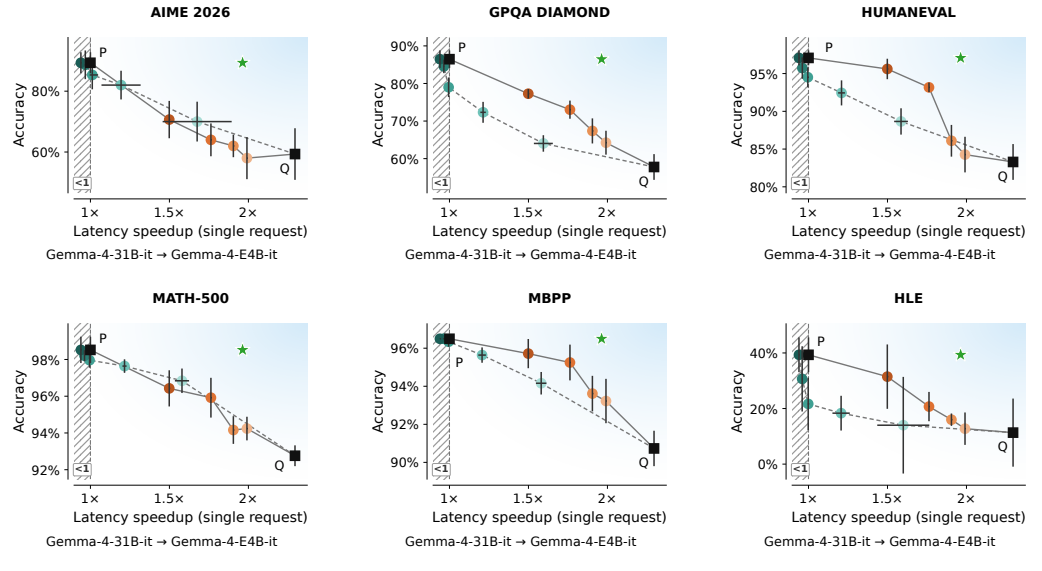

Figure 11: Accuracy vs latency speedup, `Gemma-4-31B-it` → `Gemma-4-E4B-it`.

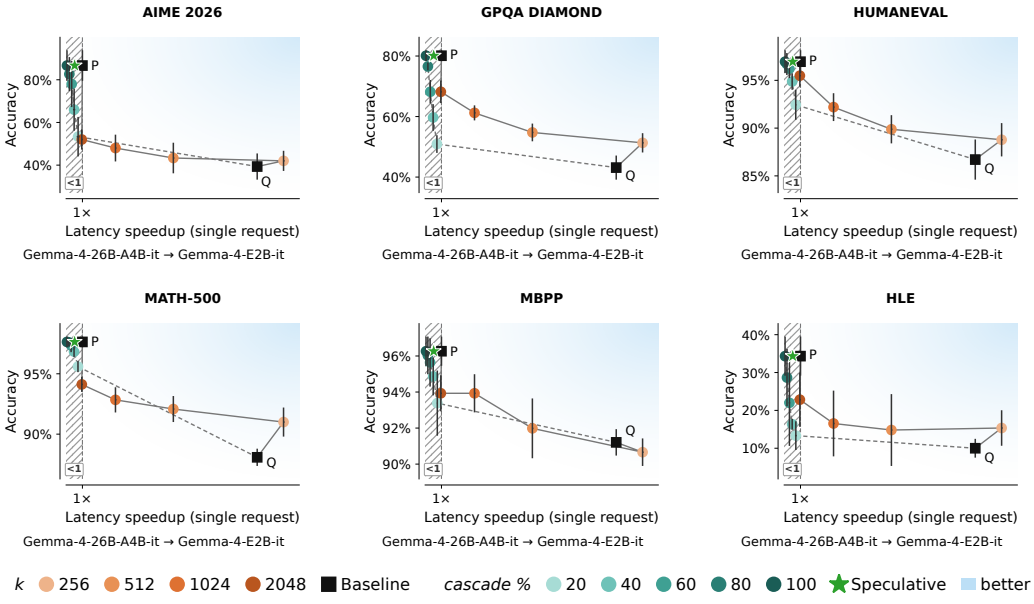

Figure 12: Accuracy vs latency speedup, `Gemma-4-26B-A4B-it` → `Gemma-4-E2B-it`.

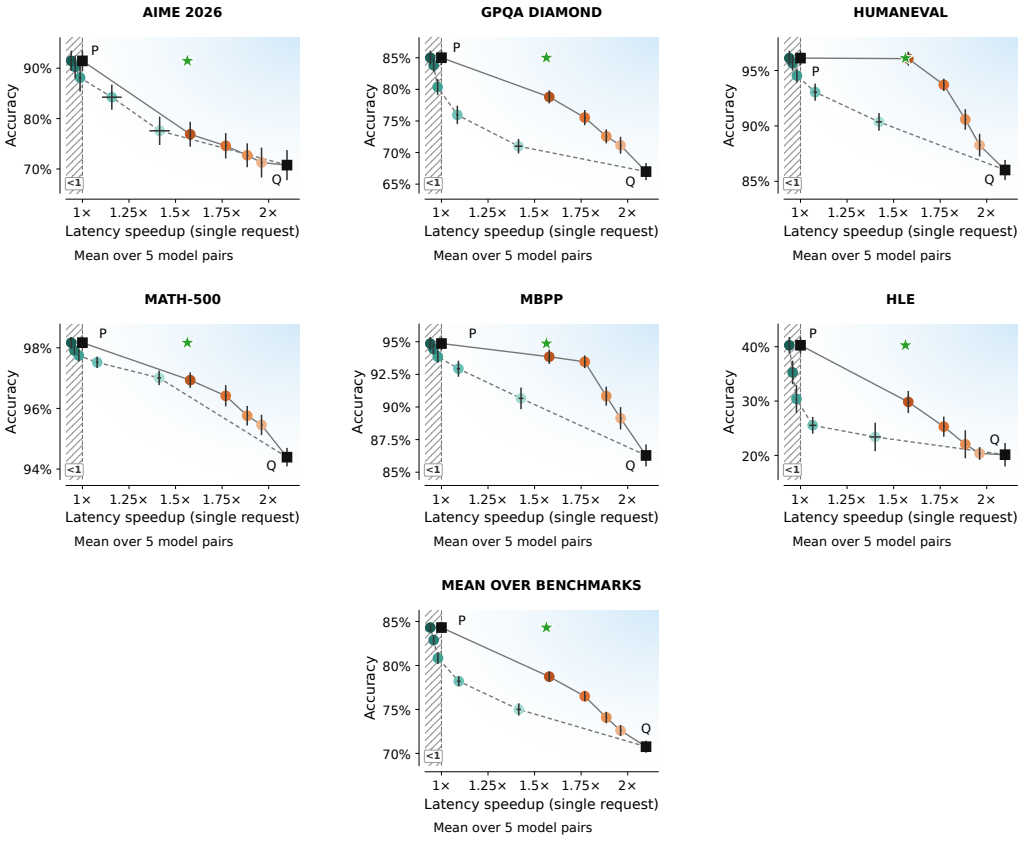

Figure 13: Accuracy vs latency speedup, mean over the five model pairs.

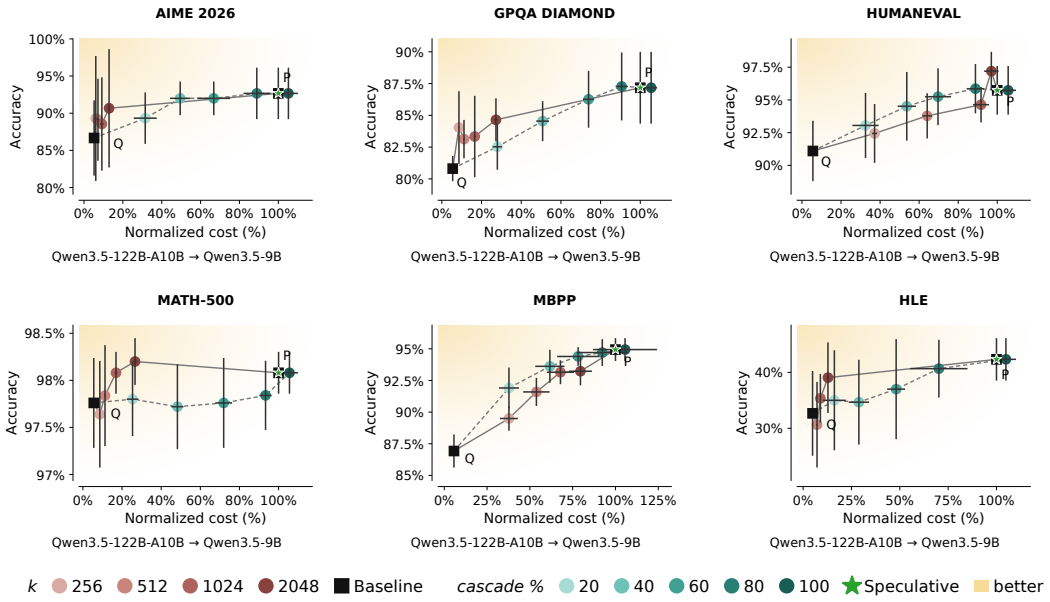

Figure 14: Accuracy vs normalized cost, `Qwen3.5-122B-A10B` → `Qwen3.5-9B`.

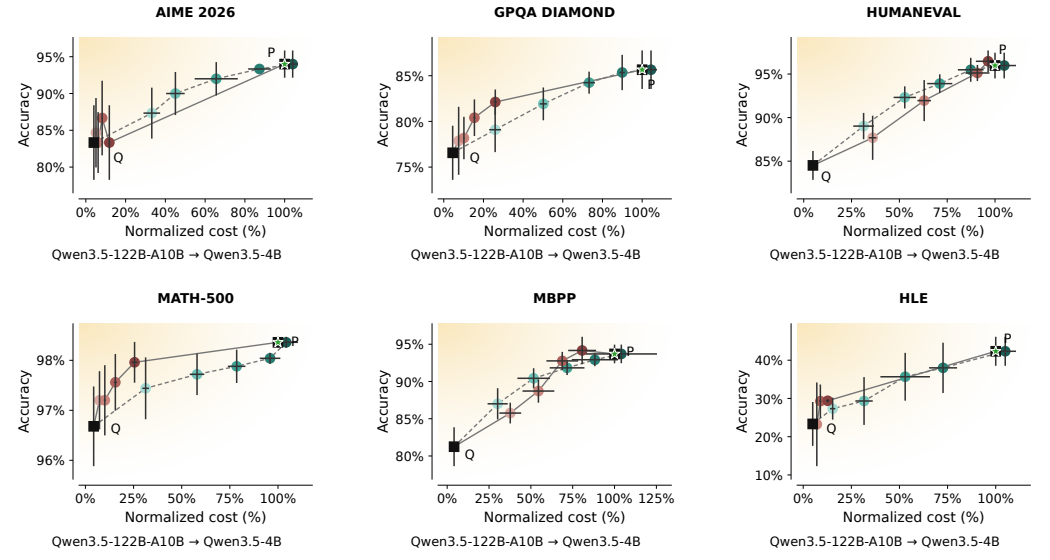

Figure 15: Accuracy vs normalized cost, `Qwen3.5-122B-A10B` → `Qwen3.5-4B`.

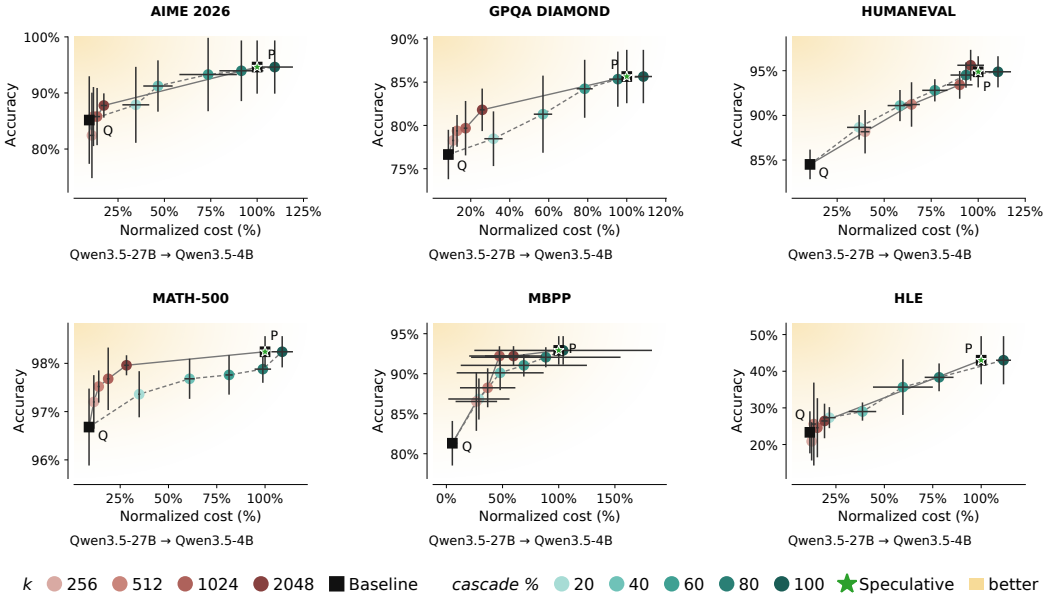

Figure 16: Accuracy vs normalized cost, `Qwen3.5-27B` → `Qwen3.5-4B`.

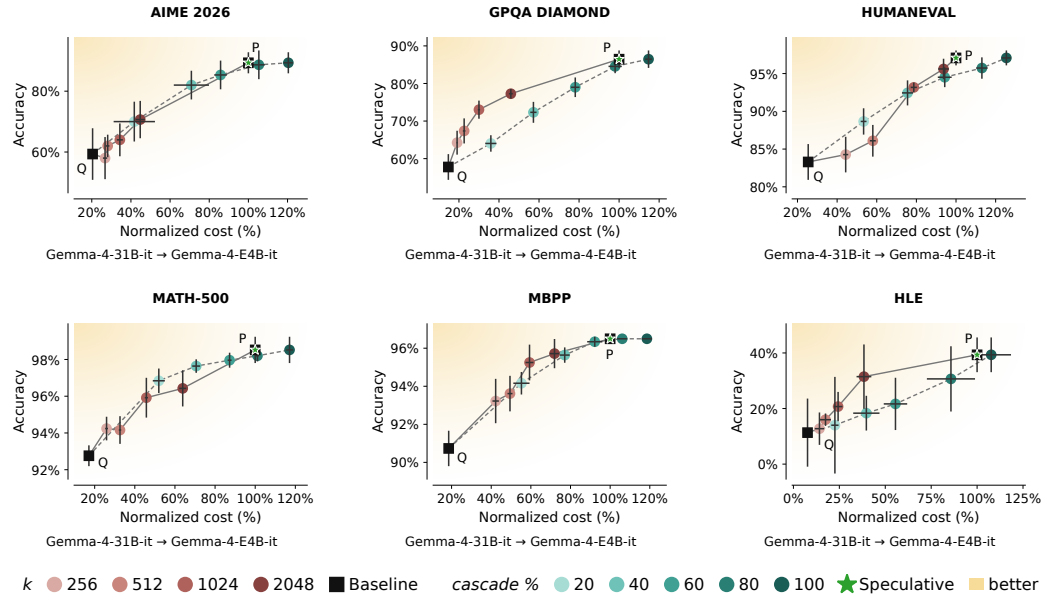

Figure 17: Accuracy vs normalized cost, `Gemma-4-31B-it` → `Gemma-4-E4B-it`.

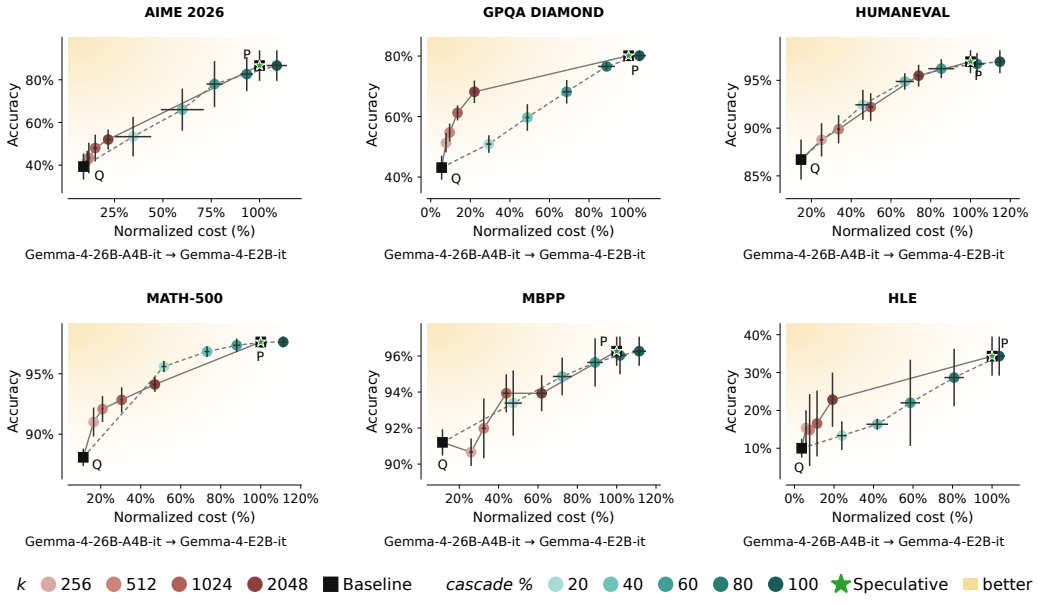

Figure 18: Accuracy vs normalized cost, `Gemma-4-26B-A4B-it` → `Gemma-4-E2B-it`.

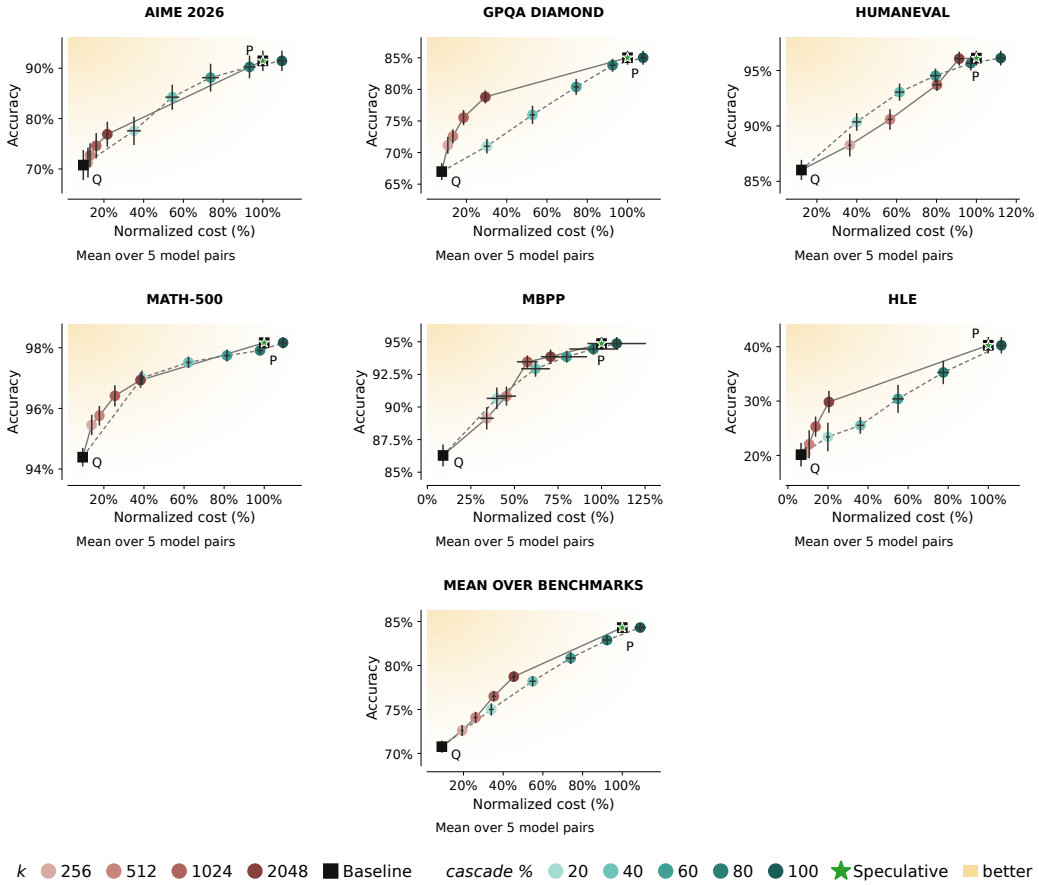

Figure 19: Accuracy vs normalized cost, mean over the five model pairs.

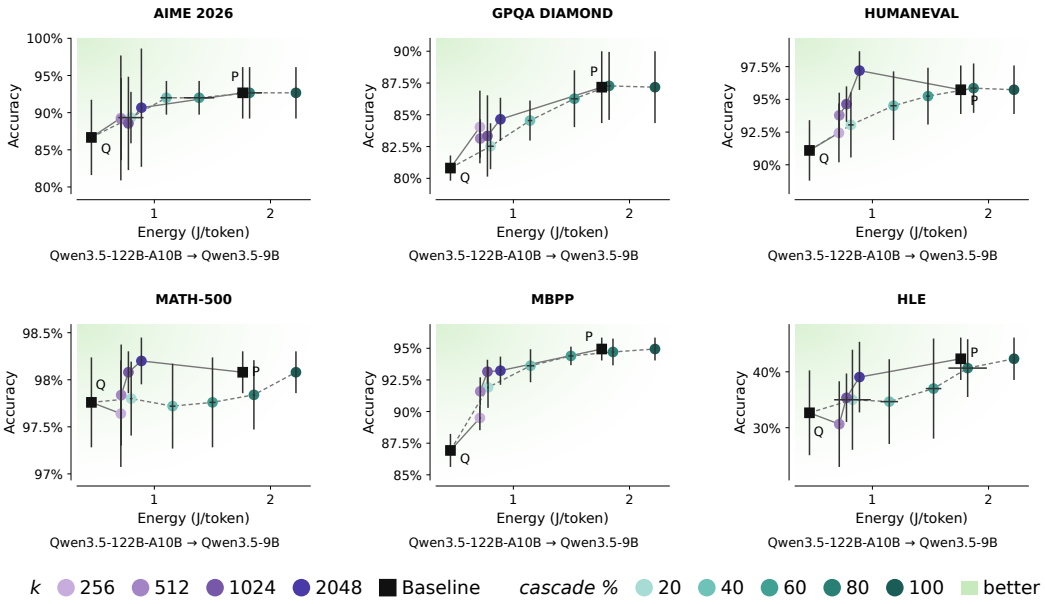

Figure 20: Accuracy vs energy per token, `Qwen3.5-122B-A10B` → `Qwen3.5-9B`.

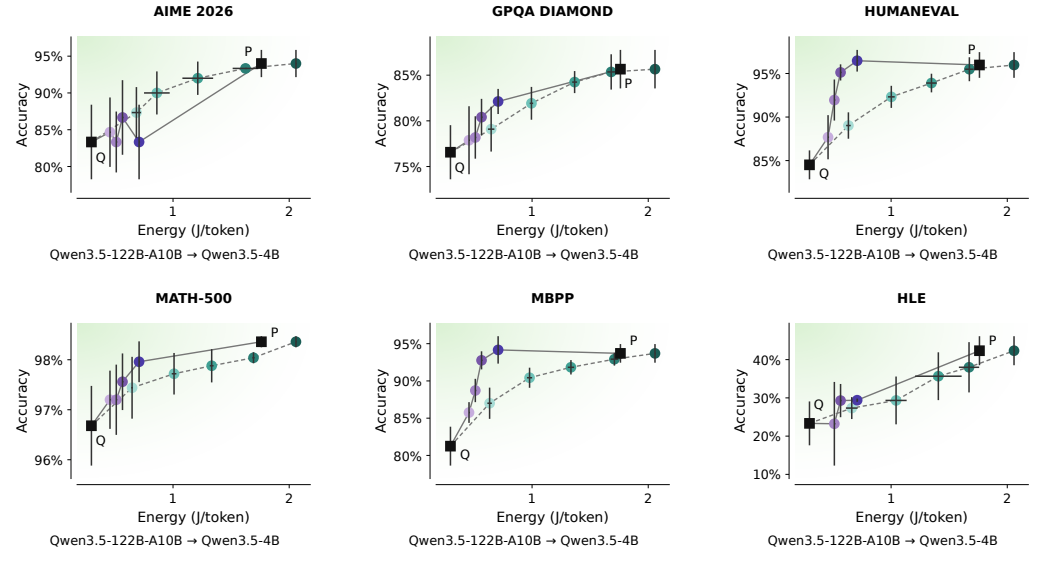

Figure 21: Accuracy vs energy per token, `Qwen3.5-122B-A10B` → `Qwen3.5-4B`.

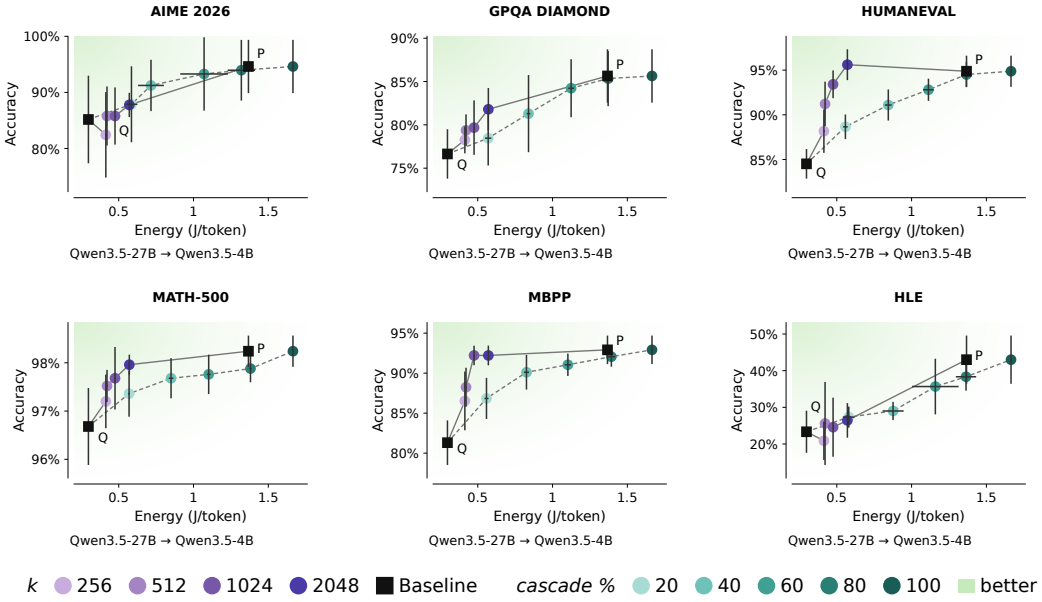

Figure 22: Accuracy vs energy per token, `Qwen3.5-27B` → `Qwen3.5-4B`.

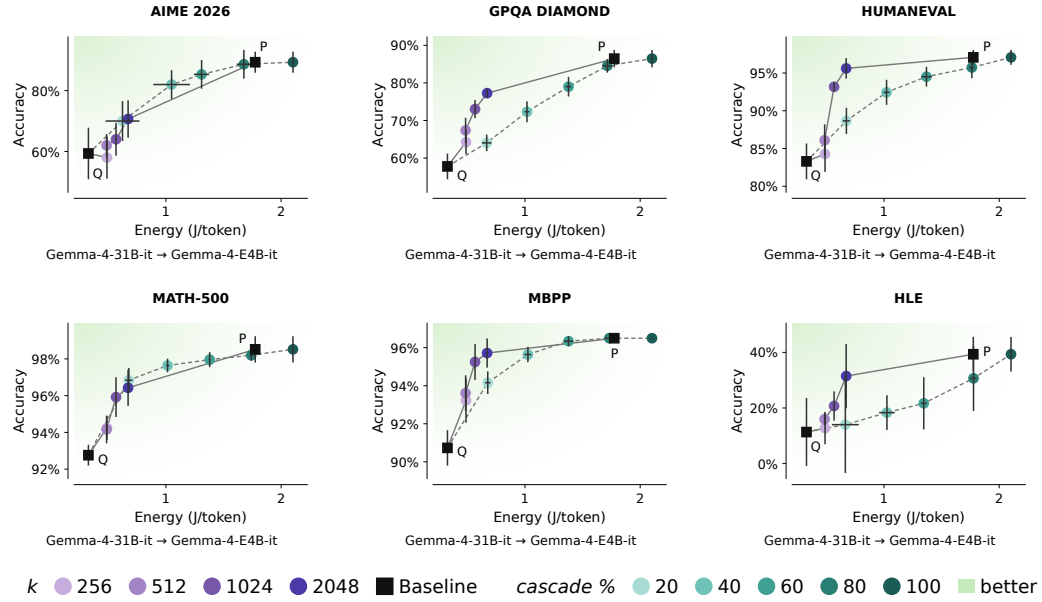

Figure 23: Accuracy vs energy per token, `Gemma-4-31B-it` → `Gemma-4-E4B-it`.

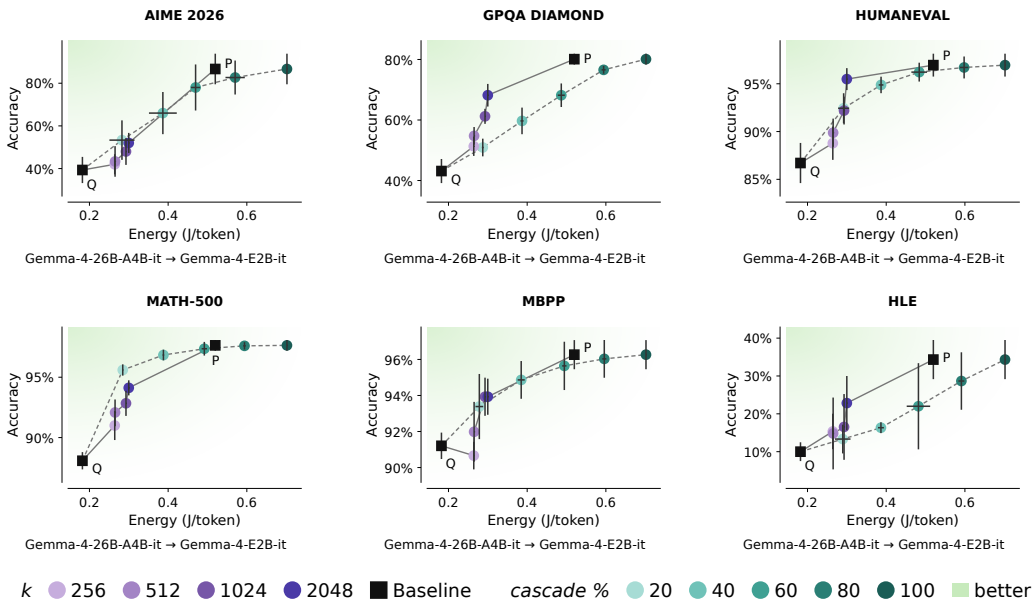

Figure 24: Accuracy vs energy per token, `Gemma-4-26B-A4B-it` → `Gemma-4-E2B-it`.

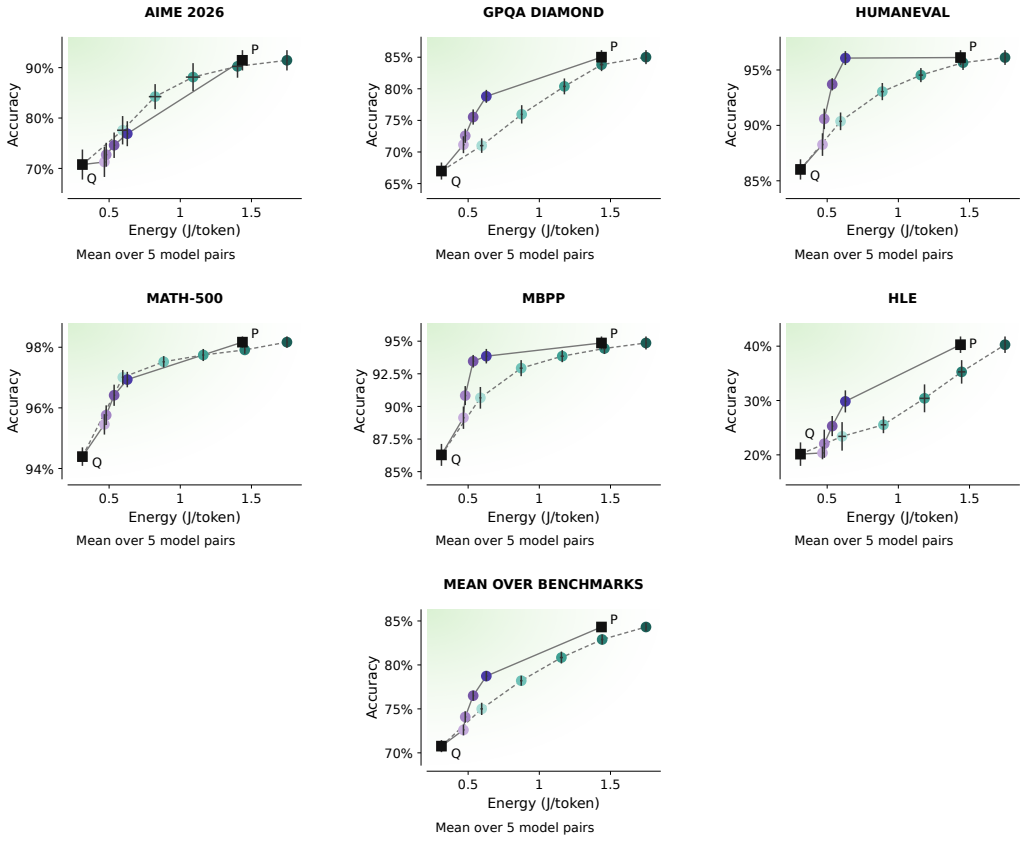

Figure 25: Accuracy vs energy per token, mean over the five model pairs.

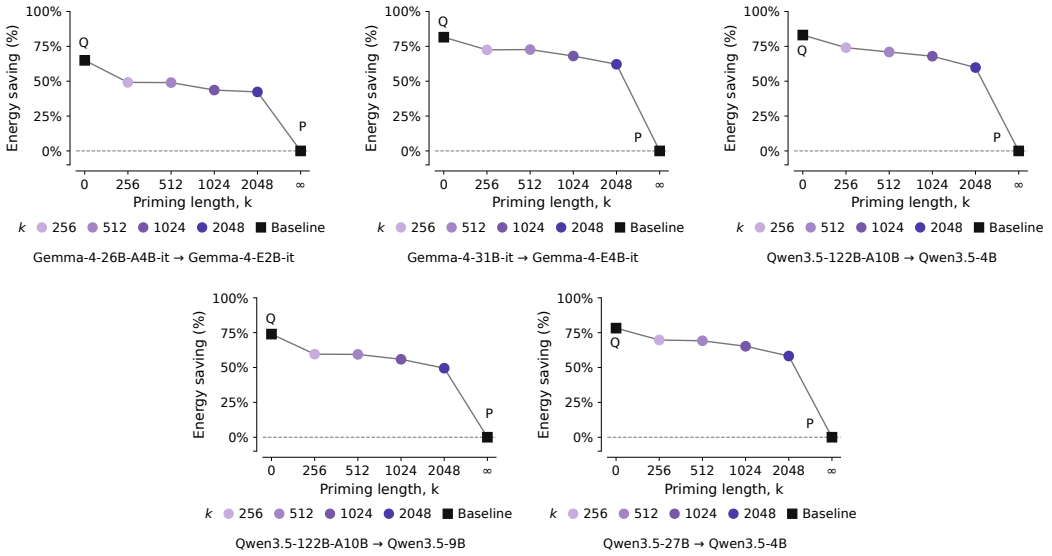

Figure 26: Energy saving as a function of priming length for all model pairs. The x-axis reports priming length k, while the y-axis reports energy saving (%). From left to right, top to bottom: `Gemma-4-26B-A4B-it` → `Gemma-4-E2B-it`, `Gemma-4-31B-it` → `Gemma-4-E4B-it`, `Qwen3.5-122B-A10B` → `Qwen3.5-4B`, `Qwen3.5-122B-A10B` → `Qwen3.5-9B`, and `Qwen3.5-27B` → `Qwen3.5-4B`.

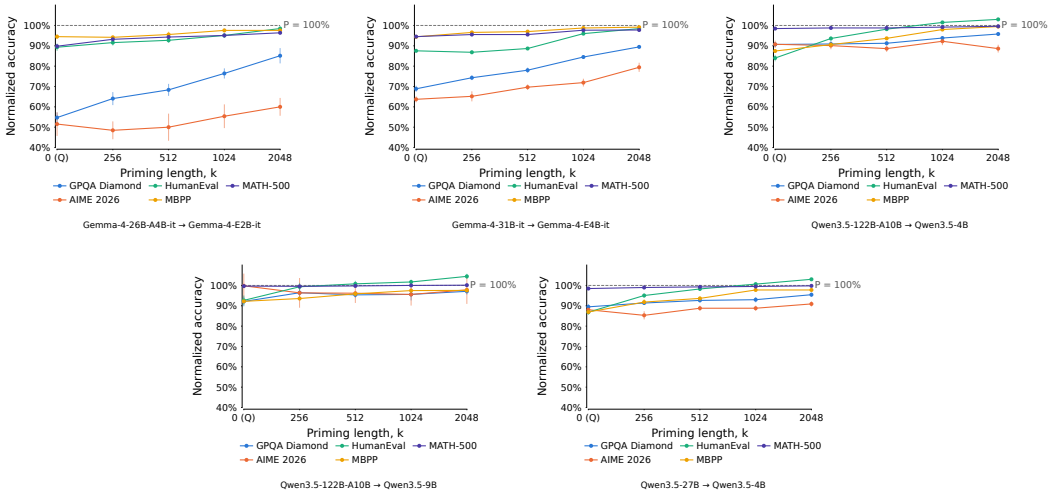

Figure 27: Normalized accuracy as a function of priming length for all model pairs. The x-axis reports priming length k, while the y-axis reports normalized accuracy. From left to right, top to bottom: `Gemma-4-26B-A4B-it` → `Gemma-4-E2B-it`, `Gemma-4-31B-it` → `Gemma-4-E4B-it`, `Qwen3.5-122B-A10B` → `Qwen3.5-4B`, `Qwen3.5-122B-A10B` → `Qwen3.5-9B`, and `Qwen3.5-27B` → `Qwen3.5-4B`.