ASYNCHRONOUS LARGE-TO-SMALL HANDOFF FOR EFFICIENT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long chain-of-thought (CoT) reasoning improves the performance of large language models on complex tasks, but also incurs substantial inference cost and latency. We investigate whether a smaller model can help complete an ongoing large-model reasoning trajectory faster while preserving the larger model's answer quality. We propose **Asynchronous Large-to-Small Reasoning Handoff (ALSH)**, a framework that launches small-model continuations from intermediate large-model reasoning states, creating an online race between continued large-model reasoning and candidate small-model completions. We further train a prefix-conditioned Reasoning Finisher with reinforcement learning to improve the accuracy–latency trade-off. Across ten benchmarks spanning mathematics, science, and code generation, ALSH reduces latency by 52.7% compared with standalone large-model inference in our main Qwen 35B–2B setting, with an 11.9% relative improvement in dataset-macro accuracy, demonstrating a stronger accuracy–latency balance than prior large–small collaboration methods. Experiments across different model scales and families further support the generality of ALSH.

1 INTRODUCTION

Recent large reasoning models (LRMs) achieve strong performance on complex tasks (Wei et al., 2022; Guo et al., 2025), yet longer reasoning does not necessarily yield commensurate gains: additional computation can exhibit diminishing returns (Zhou et al., 2026). The practical goal is therefore not maximal reasoning, but the *minimum sufficient reasoning* for each problem.

Existing work improves reasoning efficiency through training-time optimization and inference-time control (Sui et al., 2025; Zhu & Li, 2025; Aggarwal & Welleck, 2025; Hou et al., 2025; Luo et al., 2026; Han et al., 2025; Fu et al., 2025b; Yang et al., 2026). A complementary direction reallocates computation across models, from request-level routing to within-trajectory collaboration (Chen et al., 2024; Ong et al., 2025; Fu et al., 2025a; Huang et al., 2026; Song et al., 2026; Fu et al., 2026; Park et al., 2026). For within-trajectory collaboration, however, model-allocation decisions are typically binding: once a switch or handoff is executed, the active reasoning process changes before the system can observe how the source trajectory would have continued.

To address this limitation, we introduce **Asynchronous Large-to-Small Reasoning Handoff (ALSH)**, which decouples handoff attempts from handoff commitment. As the large model continues reasoning, ALSH asynchronously launches small-model completion probes from successive intermediate reasoning states, creating an online race among multiple candidate continuations and the advancing large-model trajectory. We further train a prefix-conditioned Reasoning Finisher with reinforcement learning to improve small-model completion from inherited reasoning states.

Across ten mathematical, scientific, and code-generation benchmarks, ALSH reduces request-to-commit latency by 52.7% while improving dataset-macro accuracy by 11.9% relative to standalone large-model inference. Matched controls and design analyses confirm the importance of prefix-conditioned finisher training, while experiments across different model scales and families further support the generality of ALSH.

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.

Our contributions are threefold:

- We propose **ALSH**, an efficient large–small model collaboration framework that decouples handoff attempts from commitment. By allowing repeated handoff attempts, ALSH creates opportunities for early completion without interrupting the large model or prematurely committing to a handoff.
- We introduce handoff-aware training for a small-model *Reasoning Finisher*. Through reinforcement learning, the finisher acquires the *collaborative reasoning capability* to take over intermediate large-model reasoning states and drive them toward correct, compact completions.
- We evaluate ALSH across ten benchmarks, demonstrating improved accuracy–latency trade-offs. Matched controls and design analyses isolate the contributions of the framework and handoff-aware training, while cross-scale and cross-family experiments further support the generality of ALSH.

2 Related Work

Training-time reasoning efficiency. Training-based methods directly optimize reasoning models for shorter or more concise trajectories. L1, ThinkPrune, and O1-Pruner reduce unnecessary reasoning through length-aware or constrained post-training (Aggarwal & Welleck, 2025; Hou et al., 2025; Luo et al., 2026). Such approaches optimize the reasoning backbone itself, making post-training particularly costly as the backbone scales.

Inference-time reasoning efficiency. At the token level, speculative decoding accelerates autoregressive generation through lightweight drafting and target-model verification (Leviathan et al., 2023; Chen et al., 2023; Cheng et al., 2026). This is complementary to reasoning-level control, acting at the token-generation level rather than changing how reasoning computation is allocated or terminated.

Beyond token-level acceleration, runtime methods control reasoning computation at different granularities. At the request level, routing and cascading assign or escalate complete requests across models with different capabilities and costs (Chen et al., 2024; Ong et al., 2025), whereas FFS runs multiple independent full trajectories for the same request using instances of the same reasoning model and returns the first completed trajectory (Agarwal et al., 2025).

Moving inside an ongoing reasoning trajectory, finer-grained methods adapt computation based on the evolving reasoning process. TALE and s1 regulate generation through adaptive or explicit token budgets (Han et al., 2025; Muennighoff et al., 2025), while Dynasor, DEER, ThoughtTerminator, REFRAIN, and PUMA use signals such as answer stability, confidence, difficulty, or redundancy to determine when reasoning should stop (Fu et al., 2025b; Yang et al., 2026; Pu et al., 2025; Sun et al., 2026b; Min et al., 2026). Whether through token budgets or runtime signals, these methods ultimately determine a stopping point for the current trajectory. Once the trajectory stops, potentially useful future reasoning is no longer available.

A softer alternative is to transfer the remaining computation to another model rather than terminate the trajectory. Hawkeye communicates concise reasoning instructions (She et al., 2025), Tandem estimates when accumulated guidance is sufficient for takeover (Fu et al., 2026), DUET transfers a compressed reasoning signal (Chen et al., 2026), PyroDash learns when to trigger a handoff (Lyu et al., 2026), and SAIL adaptively transfers an informative prefix (Park et al., 2026). Such methods still face a similar one-shot decision bottleneck: once the handoff occurs, later source-model states are no longer available as alternative takeover opportunities. An early transfer may provide insufficient reasoning context for successful completion, whereas a late transfer leaves less opportunity for efficiency gains.

A more interactive design revisits model allocation as reasoning unfolds. R2R, RelayLLM, TrigReason, and RelayGen invoke or switch models at token, span, trigger-point, or reasoning-segment granularity (Fu et al., 2025a; Huang et al., 2026; Zhao et al., 2026; Song et al., 2026). Related methods such as SpecReason, SCoT, and Speculative Thinking use lightweight models for reasoning drafts or intermediate steps while stronger models provide assessment, correction, selection, or guidance (Pan et al., 2025; Wang et al., 2026; Yang et al., 2025). Such interaction provides multiple

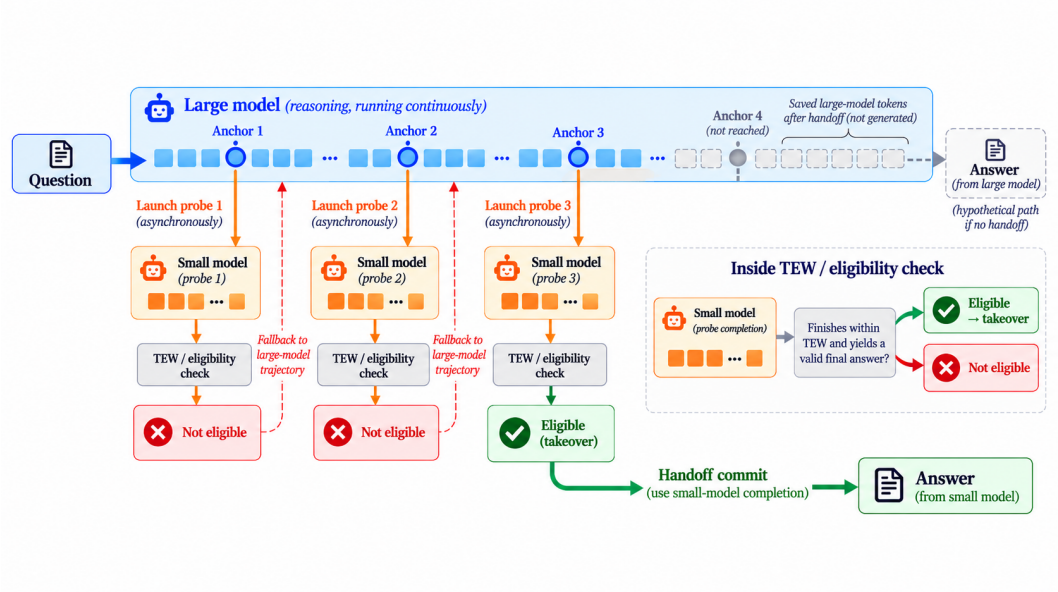


Figure 1: Overview of ALSH.

opportunities to adapt computation, but each intervention is based on the currently observed reasoning state and can immediately alter the active trajectory. A poorly timed intervention may therefore change subsequent reasoning before more informative states become available.

ALSH occupies a different point in this design space by decoupling handoff attempts from commitment: it launches multiple small-model completion attempts along the advancing large-model trajectory while allowing the large-model trajectory to continue until a final handoff is committed.

3 ALSH: Asynchronous Large-to-Small Reasoning Handoff

3.1 Overview

ALSH is motivated by three empirical observations, with supporting analysis provided in Appendix D.1.

First, multiple handoff attempts provide more takeover opportunities than a single fixed attempt. As illustrated in Figure 1, ALSH asynchronously launches *Small-Model Completion Probes* (SCPs) from *Large-Model Reasoning Prefixes* (LRPs) at successive Handoff Anchors while the large model continues reasoning, allowing unsuccessful attempts without interrupting the main trajectory. Second, shorter small-model reasoning trajectories are associated with higher accuracy, whereas longer trajectories are less reliable. ALSH therefore constrains each SCP with a *Takeover Eligibility Window* (TEW). Only an SCP that successfully completes within this window forms a *Takeover-Eligible Completion* (TEC), providing an operational criterion for whether the inherited LRP is sufficient for takeover. Third, handoff alone does not guarantee substantial latency reduction, since the original small model may still require a long continuation after receiving an LRP. ALSH therefore trains a prefix-conditioned *Reasoning Finisher* with reinforcement learning to complete inherited LRPs correctly and with less additional generation.

These three design principles define ALSH as an asynchronous race-based handoff framework that explores multiple takeover opportunities while preserving the advancing large-model trajectory until commitment. We next formalize the asynchronous handoff mechanism and then describe Reasoning Finisher training.

3.2 ASYNCHRONOUS HANDOFF FORMULATION

Given a problem x, a large model L autoregressively generates a reasoning trajectory $\mathbf{z}^L = (z_1^L, \ldots, z_T^L)$. The small model S serves as the Reasoning Finisher: instead of reasoning from scratch, it receives an intermediate state from L and completes the remaining reasoning. Our goal is to minimize the amount of large-model reasoning required before handoff while preserving the quality of full large-model inference.

We define candidate Handoff Anchors within the large model's reasoning region:

$$\mathcal{H}(x) = \{b_k = k\Delta \mid b_k < T_{\text{reason}}(x)\}, \tag{1}$$

where b_k is the position of the k-th anchor, Δ is the probe interval, and $T_{\text{reason}}(x)$ is the number of tokens generated before the reasoning-end marker. The constraint ensures that every candidate anchor lies inside the reasoning region.

At anchor b_k, the system snapshots the current LRP:

$$\mathbf{P}_k^L \triangleq \mathbf{z}_{1:b_k}^L. \tag{2}$$

Conditioned on this LRP, the small model launches the k-th SCP:

$$\hat{\mathbf{y}}_k \sim \pi_S\big(\cdot \mid \Phi_S(x, \mathbf{P}_k^L)\big), \qquad |\hat{\mathbf{y}}_k| \le W, \tag{3}$$

where $\Phi_S(x, \mathbf{P}_k^L)$ denotes the small-model input constructed from the problem and the textual LRP using the small model's native chat template, reasoning-start protocol, and tokenizer. The small model therefore continues reasoning from an input representation that incorporates the inherited LRP.

The TEW, parameterized by the inference continuation window W, requires each SCP to terminate naturally and produce a valid final-answer structure within W new tokens. Larger windows permit longer probes and may increase their chance of producing an eligible completion, but may also increase computation and probe occupancy. Smaller windows reduce probe cost but may truncate otherwise successful continuations.

A TEC is defined as

$$E_k(W) = \begin{cases} 1, & f_k = \text{stop} \ \wedge \ g_{\text{struct}}(\hat{\mathbf{y}}_k) = 1, \\ 0, & \text{otherwise}, \end{cases} \tag{4}$$

where f_k is the SCP termination reason and g_{struct} checks output validity. Thus, only SCPs that terminate naturally within W and produce a valid output are eligible for takeover.

At runtime, the large model and all launched SCPs execute asynchronously. Let τ_L and τ_k denote the completion times of the large model and the k-th SCP, respectively. A runtime policy Γ observes the current state $\mathcal{O}_t$ and selects

$$a_t = \Gamma(\mathcal{O}_t) \in \{\text{wait}, \text{commit}_S(k), \text{commit}_L\}, \tag{5}$$

where $\text{commit}_S(k)$ is available only when $E_k(W) = 1$ and $\tau_k \le t$, and commit_L only when $\tau_L \le t$.

ALSH commits the first available completion. If an SCP becomes a TEC before the large model finishes, Γ selects $\text{commit}_S(k)$ at $t = \tau_k$; if the large model finishes first, it selects commit_L at $t = \tau_L$, even if other SCPs remain in flight. The first commit is final and cancels all remaining requests.

The final output is

$$y_{\text{collab}} = \begin{cases} \Psi(\mathbf{P}_k^L, \hat{\mathbf{y}}_k), & a^\star = \text{commit}_S(k), \\ y_L, & a^\star = \text{commit}_L, \end{cases} \tag{6}$$

where $\Psi(\mathbf{P}_k^L, \hat{\mathbf{y}}_k)$ denotes the complete reasoning trace formed from the inherited LRP and the committed small-model completion.

3.3 Reasoning Finisher Training

We train the Reasoning Finisher with reinforcement learning to complete intermediate large-model reasoning states efficiently. Each training input follows the same prefix-conditioned input representation $\Phi_S(x, \mathbf{P}_k^L)$ used for SCP generation in Equation 3. The Reasoning Finisher samples continuations with budget W_{train}, set independently of the inference-time TEW W_{test}.

Following truncation-style reward shaping for efficient reasoning (Wu et al., 2026), a continuation receives positive reward only if it terminates within W_{train}, has a valid output structure, and yields the correct final answer; truncated, invalid, or incorrect continuations receive zero reward. We optimize the Reasoning Finisher with Group Relative Policy Optimization (GRPO) (Shao et al., 2024), normalizing rewards within each sampled group to form relative advantages. The LRP is treated as fixed conditioning context, and the policy loss is applied only to Reasoning Finisher continuation tokens.

4 Experiments

We organize our experiments around four questions: (1) Does ALSH improve the accuracy–latency trade-off over standalone inference and existing large–small collaboration methods? (2) Does the learned finishing capability generalize across model scales and model families? (3) How do Reasoning Finisher training choices affect the accuracy–latency trade-off? (4) How sensitive is ALSH to runtime configurations such as the probe interval and continuation budget W_{test}? We first describe the common setup, followed by the main results and an analysis of training and runtime design choices.

4.1 Experimental Setup

Models and training. Unless otherwise stated, we use Qwen3.5-35B-A3B as the large model and Qwen3.5-2B as the Reasoning Finisher. For generalization, we additionally evaluate Qwen3.5-27B, Qwen3.5-397B-A17B, Gemma-4-31B-it, and Gemma-4-E2B-it (Qwen Team, 2026; Gemma Team, 2026). For brevity, we refer to these models as Qwen 35B, Qwen 2B, Qwen 27B, Qwen 397B, Gemma 31B, and Gemma E2B, respectively. The Reasoning Finisher is trained on reasoning trajectories constructed from the MATH training split (Hendrycks et al., 2021), using leakage-free LRPs by default. Data construction, LRP variants, W_{train}, reward design, and training hyperparameters are detailed in Appendix A.

Benchmarks and metrics. We evaluate on 3,595 examples from ten benchmarks spanning three domains:

- **Mathematical reasoning:** MATH-500, GSM8K, AIME 2025, LiveBench-Math, OlymMATH, AIME 2026, AMC 2024, and AMC 2025 (Lightman et al., 2024; Cobbe et al., 2021; White et al., 2025; Sun et al., 2026a).
- **Scientific reasoning:** GPQA-Diamond (Rein et al., 2024).
- **Code generation:** LiveCodeBench release-v5 (Jain et al., 2025).

Our primary metrics are final-answer accuracy and request-to-commit latency. We additionally report large-model decoding reduction, analytical FLOPs, and handoff commit rate where relevant. Metric definitions are provided in Appendix B.

Baselines and inference configuration. We compare ALSH with standalone large-model inference and two large–small collaboration methods: Tandem (Fu et al., 2026) and RelayGen (Song et al., 2026). Unless otherwise stated, ALSH uses a 1K probe interval and $W_{\text{test}} = 2$K. All methods are evaluated under the same live concurrent-serving workload, with latency measured independently for each request including scheduling and queueing delays. Implementation and baseline configurations are provided in Appendix C.

4.2 Main Results

4.2.1 Overall Effectiveness

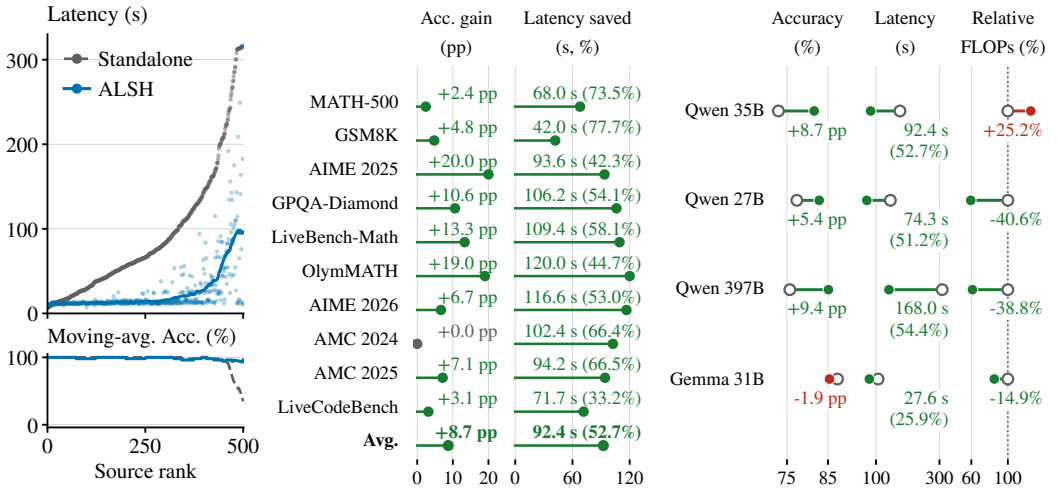
We evaluate ALSH using Qwen 35B as the large model and Qwen 2B as the Reasoning Finisher, with $W_{\text{test}} = 2\text{K}$ and a 1K probe interval. Table 1 compares standalone inference, existing large–small collaboration baselines, and ALSH with original, question-only-trained, and prefix-conditioned 2B finishers. The comparison is designed to separate the benefit of asynchronous handoff from that of finisher training.

For the matched training comparison, the question-only and prefix-conditioned finishers use the same RL objective and training budget and differ only in whether the Large-Model Reasoning Prefix (LRP) is provided as conditioning, with $W_{\text{train}} = W_{\text{test}} = 2\text{K}$. We additionally evaluate the original 2B finisher with a larger continuation budget of $W_{\text{test}} = 16\text{K}$ to ensure that its performance is not artificially limited by the shorter budget used for the trained finishers.

Standalone Qwen 35B achieves 72.92% dataset-macro accuracy at 175.31 s, whereas ALSH with the prefix-conditioned Reasoning Finisher reaches 81.62% at 82.91 s, improving accuracy by 8.70 percentage points (11.9% relative) while reducing request-to-commit latency by 52.7%.

The matched controls clarify the sources of this gain. Using the original 2B model within ALSH yields 74.15% accuracy at 169.34 s, and increasing its continuation window from 2K to 16K does not improve the aggregate result. Training the finisher substantially strengthens the handoff: the question-only variant reaches 79.19% accuracy at 123.57 s, while prefix-conditioned training further improves accuracy to 81.62% and reduces latency to 82.91 s. The trained 2B models remain substantially weaker as standalone solvers, indicating that the gain cannot be explained by finisher training alone. We analyze the training design further in Section 4.3.

ALSH also compares favorably with existing collaboration methods. Tandem is faster (25.29 s) but substantially less accurate (46.94%), whereas RelayGen reaches 66.23% accuracy at 244.07 s. Thus, ALSH operates in a substantially higher-accuracy regime than Tandem while improving both



(a) MATH-500 (b) Across benchmarks (c) Model pairings

Figure 2: **ALSH effectiveness across sample, benchmark, and model levels. (a)** MATH-500 samples ordered by standalone latency; dots show per-request latency and curves show moving-average latency and accuracy. **(b)** Per-benchmark accuracy gain and latency savings relative to standalone Qwen 35B; **Avg.** is the dataset-macro average. **(c)** Standalone inference and ALSH across four model pairings in accuracy, request-to-commit latency, and total analytical FLOPs, with standalone FLOPs normalized to 100%. Full sample-level results are in Appendix D.5.

Table 1: Main results across ten benchmarks. Each cell reports accuracy (%) above mean request-to-commit latency (s); **Avg.** is the dataset-macro average. ALSH uses a 1K probe interval and $W_{\text{test}} =$ 2K unless noted. Question-only and prefix-conditioned finishers share the same RL objective and $W_{\text{train}} =$ 2K, differing only in LRP conditioning. External methods follow their original policies and budgets (Appendix C.2). Bold marks the highest accuracy and lowest latency in each column.

Method	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
Standalone models											
Qwen 35B	96.00 92.51	91.51 54.05	63.33 221.10	70.20 196.09	49.46 188.24	47.50 268.73	76.67 220.14	**80.00** 154.32	85.71 141.67	68.83 216.26	72.92 175.31
Qwen 2B (original)	58.00 91.44	68.92 71.65	13.33 135.76	14.14 135.90	21.74 123.80	0.50 142.66	10.00 138.02	17.78 128.47	23.81 124.25	8.34 130.65	23.66 122.26
Qwen 2B (question-only)	86.40 28.73	87.87 6.41	36.67 103.88	47.47 87.84	45.92 64.07	9.50 135.80	36.67 106.03	48.89 64.14	61.90 64.17	28.62 98.20	48.99 75.93
Qwen 2B (prefix-conditioned)	87.20 18.40	87.19 **4.85**	26.67 84.75	55.05 47.28	44.02 41.05	17.50 95.80	30.00 72.24	60.00 37.44	61.90 36.04	32.33 72.83	50.19 51.07
External collaborative methods											
Tandem	88.80 **17.79**	96.13 14.70	13.33 **32.02**	56.57 **23.18**	36.41 **25.56**	7.00 **33.12**	16.67 **33.70**	53.33 **23.70**	61.90 **22.79**	39.28 **26.32**	46.94 **25.29**
RelayGen	90.40 135.32	92.19 60.43	60.00 256.68	64.65 271.96	41.30 326.68	43.50 364.68	66.67 312.43	71.11 216.87	76.19 180.00	56.32 315.61	66.23 244.07
ALSH											
Original ($W_{\text{test}} =$ 2K)	97.00 82.81	95.45 36.28	70.00 218.99	68.69 196.27	51.90 181.52	51.00 269.30	73.33 216.86	77.78 149.35	88.10 129.68	68.25 212.32	74.15 169.34
Original ($W_{\text{test}} =$ 16K)	97.00 91.84	**96.89** 36.15	63.33 223.42	67.17 200.37	53.26 185.40	49.50 270.74	66.67 230.89	77.78 159.28	85.71 140.82	68.71 215.74	72.60 175.47
Question-only	**98.60** 38.56	96.66 13.63	76.67 173.37	74.24 138.12	58.97 115.91	65.00 221.74	76.67 181.19	**80.00** 96.21	**92.86** 81.62	**72.19** 175.32	79.19 123.57
Prefix-conditioned	98.40 24.55	96.29 12.07	**83.33** 127.49	**80.81** 89.94	**62.77** 78.84	**66.50** 148.73	**83.33** 103.52	**80.00** 51.92	**92.86** 47.50	71.96 144.56	**81.62** 82.91

accuracy and latency over RelayGen. A concrete comparison of Tandem and ALSH is shown in Appendix Figure 4.

Figure 2 summarizes ALSH at three granularities. At the sample level on MATH-500, ALSH shows a broad reduction in latency while maintaining high local accuracy. Across all ten benchmarks, the default Qwen 35B–2B configuration matches or improves standalone accuracy while reducing latency on every benchmark. The same trend extends across model scales: ALSH substantially reduces latency while improving accuracy with Qwen 27B and Qwen 397B, and retains similar accuracy with lower latency for Gemma 31B. ALSH additionally reduces total analytical FLOPs by 40.6%, 38.8%, and 14.9% with Qwen 27B, Qwen 397B, and Gemma 31B, respectively. The Qwen 35B setting incurs a 25.2% overhead, consistent with the smaller gap in active parameter count between its MoE backbone and the 2B finisher.

4.2.2 Generalization across Model Scales and Families

We next test whether the learned finishing capability transfers beyond the large–small pair used to construct its training data. For *within-family scale transfer*, a fixed Qwen 2B Reasoning Finisher trained on Qwen 35B prefixes is paired, without retraining, with Qwen 27B and Qwen 397B. For *cross-family transfer*, we pair the same Qwen finisher with Gemma 31B and systematically evaluate Gemma E2B finishers trained on either Qwen 35B or Gemma 31B prefixes with large models from both families.

The fixed Qwen 2B finisher transfers effectively across Qwen model scales without retraining. With Qwen 27B, ALSH improves dataset-macro accuracy from 77.40% to 82.85% while reducing latency from 145.13 s to 70.88 s; with Qwen 397B, accuracy increases from 75.70% to 85.10% while latency decreases from 308.64 s to 140.61 s. These results show that a finisher trained on Qwen 35B prefixes is not tied to a specific large-model scale.

Transfer also extends across model families and finisher families. The same Qwen 2B finisher paired with Gemma 31B retains similar accuracy (85.35% versus 87.30% standalone) while reduc-

Table 2: Generalization across model scales and families. Each cell reports final-answer accuracy (%) above mean request-to-commit latency (s). The first panel uses a fixed Qwen 2B finisher trained on Qwen 35B prefixes. Gemma E2B^{Q} and Gemma E2B^{G} denote finishers trained on Qwen 35B and Gemma 31B prefixes, respectively. Latency should be compared within each large-model configuration. Bold marks the highest accuracy and lowest latency among rows sharing the same large model, across both panels; ties are included.

Method	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
Transfer of a fixed Qwen 2B Reasoning Finisher											
Qwen 35B	96.00	91.51	63.33	70.20	49.46	47.50	76.67	80.00	85.71	68.83	72.92
(standalone)	92.51	54.05	221.10	196.09	188.24	268.73	220.14	154.32	141.67	216.26	175.31
Qwen 35B	**98.40**	**96.29**	**83.33**	**80.81**	**62.77**	**66.50**	83.33	80.00	**92.86**	71.96	**81.62**
→ Qwen 2B	24.55	**12.07**	127.49	89.94	**78.84**	**148.73**	**103.52**	51.92	47.50	144.56	82.91
Qwen 27B	93.80	92.87	**76.67**	74.24	58.97	57.00	73.33	80.00	**92.86**	74.28	77.40
(standalone)	78.07	40.80	189.06	163.11	150.77	220.50	196.59	126.11	107.56	178.75	145.13
Qwen 27B	**97.80**	**96.74**	**76.67**	**80.30**	**64.13**	**70.50**	**93.33**	**84.44**	88.10	**76.48**	**82.85**
→ Qwen 2B	**20.34**	**10.27**	**106.40**	**77.06**	**67.53**	**126.35**	**94.58**	**41.30**	**41.22**	**123.76**	**70.88**
Qwen 397B	98.20	**97.19**	63.33	80.81	52.17	61.50	66.67	73.33	83.33	**80.42**	75.70
(standalone)	145.23	56.90	407.58	320.51	310.35	505.04	411.61	279.84	246.32	403.03	308.64
Qwen 397B	**98.40**	96.51	**83.33**	**86.36**	**69.57**	**73.00**	**86.67**	**84.44**	**92.86**	79.84	**85.10**
→ Qwen 2B	**40.35**	**23.59**	**233.71**	**138.06**	**131.57**	**255.78**	**173.82**	**84.85**	**72.29**	**252.12**	**140.61**
Gemma 31B	**98.80**	97.19	**86.67**	**85.35**	64.95	77.50	**90.00**	**84.44**	**95.24**	**92.82**	**87.30**
(standalone)	48.42	19.74	154.04	109.24	108.67	185.53	140.40	93.95	85.49	118.67	106.42
Gemma 31B	**98.80**	96.89	**86.67**	83.33	**66.30**	**79.00**	83.33	82.22	92.86	84.13	85.35
→ Qwen 2B	**30.69**	13.68	**119.50**	63.36	**76.78**	163.33	**120.13**	54.61	**47.78**	98.55	78.84
Transfer with Gemma E2B Reasoning Finishers											
Gemma 31B	98.60	97.19	80.00	75.76	61.68	75.50	83.33	**84.44**	92.86	88.88	83.82
→ Gemma E2B^{G}	33.10	**13.04**	133.55	**40.86**	83.03	**158.44**	125.19	**53.90**	53.98	88.19	**78.33**
Qwen 35B	98.00	95.83	63.33	71.21	60.33	61.00	**86.67**	80.00	85.71	79.37	78.15
→ Gemma E2B^{G}	24.35	12.12	**112.30**	48.41	98.43	154.12	113.20	51.15	**39.64**	128.06	**78.18**
Qwen 35B	97.80	95.98	70.00	73.74	61.96	63.00	83.33	**82.22**	88.10	**79.61**	79.57
→ Gemma E2B^{Q}	**24.20**	12.21	135.49	**44.27**	101.51	157.00	120.64	**50.21**	46.47	**127.59**	81.96
Gemma 31B	98.60	**97.50**	**86.67**	76.77	64.13	73.00	83.33	**84.44**	**95.24**	90.38	85.01
→ Gemma E2B^{Q}	33.02	13.36	139.80	46.10	92.09	163.27	124.23	64.68	55.66	**86.52**	81.87

ing latency from 106.42 s to 78.84 s. Conversely, a Gemma E2B finisher trained on Gemma 31B prefixes reaches 78.15% accuracy at 78.18 s with Qwen 35B, compared with 72.92% at 175.31 s for standalone Qwen 35B. Results across the full cross-family grid further show that Gemma finishers trained on either Qwen- or Gemma-generated prefixes remain effective with large models from both families, supporting the generality of ALSH across model scales and model families.

4.3 Analysis of Design Choices

4.3.1 Finisher Training

We vary two training-side design choices while keeping the inference configuration fixed: the LRP conditioning scheme and the training continuation budget W_{train}. All configurations in Table 3 use $W_{\text{test}} = 2\text{K}$ and a 1K probe interval.

At $W_{\text{train}} = 2\text{K}$, question-only training reaches 79.19% dataset-macro accuracy at 123.57 s, whereas Leakage-Free LRP reaches 81.62% at 82.91 s. Thus, conditioning on the inherited LRP improves accuracy by 2.43 percentage points while reducing latency by 32.9%. Short Leakage-Free LRP and Unfiltered LRP achieve similar operating points (79.48%/83.38 s and 79.95%/83.04 s, respectively); we treat the Unfiltered result as descriptive because it does not enforce the leakage boundary.

The training continuation budget also matters. A 1K budget is substantially faster but much less accurate (65.07% at 49.90 s), whereas increasing the budget from 2K to 4K provides no additional accuracy benefit and increases latency to 107.66 s. We therefore use $W_{\text{train}} = 2\text{K}$ by default. The matched training–inference window comparison in Appendix D.3.3 further supports this choice.

Table 3: Analysis of Reasoning Finisher training design. Each cell reports final-answer accuracy (%) above mean request-to-commit latency (s). The first panel compares prefix conditioning and LRP construction at $W_{\text{train}} = 2\text{K}$; the second varies W_{train} under Leakage-Free LRP. All configurations use $W_{\text{test}} = 2\text{K}$ and a 1K probe interval.

Method	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
Prefix conditioning and LRP construction ($W_{\text{train}} = 2K$)											
Original	97.00 82.81	95.45 36.28	70.00 218.99	68.69 196.27	51.90 181.52	51.00 269.30	73.33 216.86	77.78 149.35	88.10 129.68	68.25 212.32	74.15 169.34
Question-only	98.60 38.56	96.66 13.63	76.67 173.37	74.24 138.12	58.97 115.91	65.00 221.74	76.67 181.19	80.00 96.21	92.86 81.62	72.19 175.32	79.19 123.57
Leakage-Free LRP	98.40 24.55	96.29 12.07	83.33 127.49	80.81 89.94	62.77 78.84	66.50 148.73	83.33 103.52	80.00 51.92	92.86 47.50	71.96 144.56	81.62 82.91
Short Leakage-Free LRP	97.80 25.07	95.98 11.91	73.33 125.85	77.27 96.79	60.60 80.15	68.00 155.36	76.67 100.73	84.44 46.38	90.48 44.17	70.22 147.44	79.48 83.38
Unfiltered LRP	98.40 23.74	96.59 12.30	73.33 125.74	80.81 91.70	62.50 81.78	68.00 150.68	76.67 110.26	77.78 48.75	92.86 41.11	72.54 144.35	79.95 83.04
Training continuation budget (Leakage-Free LRP)											
Leakage-Free LRP ($W_{\text{train}} = 1\text{K}$)	91.20 16.07	95.91 11.38	56.67 77.97	74.24 53.07	56.79 46.02	30.50 68.94	40.00 57.71	71.11 24.28	69.05 22.76	65.24 120.82	65.07 49.90
Leakage-Free LRP ($W_{\text{train}} = 4\text{K}$)	99.00 34.54	96.82 13.77	80.00 158.77	78.79 120.35	60.05 101.98	69.00 190.90	80.00 150.34	82.22 75.54	92.86 61.08	72.65 169.35	81.14 107.66

4.3.2 RUNTIME CONFIGURATION

We vary the inference continuation window W_{test} and probe interval Δ independently while keeping the prefix-conditioned Reasoning Finisher fixed.

For W_{test}, the 2K setting achieves both the highest dataset-macro accuracy (81.62%) and the lowest latency (82.91 s) among the evaluated windows. For the probe interval, 1K likewise provides the best observed operating point, achieving 81.62% accuracy at 82.91 s. We therefore use $W_{\text{test}} = 2\text{K}$ and a 1K probe interval by default. Full per-benchmark results and complementary efficiency statistics are reported in Appendix D.3.

5 CONCLUSION

We introduced Asynchronous Large-to-Small Reasoning Handoff (ALSH), which decouples handoff attempts from commitment by asynchronously probing a small-model Reasoning Finisher while preserving the advancing large-model trajectory as a fallback. In the main Qwen 35B–2B setting, ALSH reduces request-to-commit latency by 52.7% while improving dataset-macro accuracy from 72.92% to 81.62% (11.9% relative). The learned finishing capability transfers without retraining across Qwen model scales and remains effective across model families, while design analyses show that both prefix-conditioned training and runtime configuration materially affect the resulting accuracy–latency trade-off. Together, these results demonstrate that asynchronous handoff can reallocate reasoning across models without requiring an early irreversible switch.

AI USE STATEMENT

In this work, we used generative AI tools to assist with research execution by drafting code for parts of the experimental pipeline. We additionally used generative AI tools to support literature retrieval and the discovery of relevant prior work, and to edit and polish the manuscript for clarity and readability. All AI-assisted code was reviewed and tested by the authors before use, and literature identified with the assistance of generative AI was verified against the original sources. All AI-assisted written content was reviewed and revised by the authors. We take full responsibility for the final content of this work, including its methodology, implementation, experiments, claims, and text.

REFERENCES

Aradhye Agarwal, Ayan Sengupta, and Tanmoy Chakraborty. First finish search: Efficient test-time scaling in large language models. *arXiv preprint arXiv:2505.18149*, 2025.

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=4jdIxXBNve.

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*, 2024. URL https://openreview.net/forum?id=cSimKw5p6R.

Yilei Chen, Sharut Gupta, Yannis Paschalidis, Ayush Sekhari, and Aldo Pacchiano. When less is enough: Efficient inference via collaborative reasoning. *arXiv preprint arXiv:2605.01111*, 2026.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

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. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

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 *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-4135.

Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. Efficiently scaling LLM reasoning programs with certaindex. In *Advances in Neural Information Processing Systems*, volume 38, 2025b.

Zichuan Fu, Xian Wu, Guojing Li, Yejing Wang, Yijun Chen, Zihao Zhao, Yixuan Luo, Hanyu Yan, Yefeng Zheng, and Xiangyu Zhao. Tandem: Riding together with large and small language models for efficient reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42286–42302, 2026. doi: 10.18653/v1/2026.findings-acl.2098.

Gemma Team. Gemma 4 Technical Report. *arXiv preprint arXiv:2607.02770*, 2026. doi: 10.48550/arXiv.2607.02770. URL https://arxiv.org/abs/2607.02770.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, 2025. doi: 10.18653/v1/2025.findings-acl.1274.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://arxiv.org/abs/2103.03874.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. ThinkPrune: Pruning long chain-of-thought of LLMs via reinforcement learning. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=V51gPu1uQD.

Chengsong Huang, Tong Zheng, Langlin Huang, Jinyuan Li, Haolin Liu, and Jiaxin Huang. RelayLLM: Efficient reasoning via collaborative decoding. *arXiv preprint arXiv:2601.05167*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2403.07974.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Haotian Luo, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Xiaochun Cao, Dacheng Tao, Naiqiang Tan, and Li Shen. O1-Pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. doi: 10.18653/v1/2026.findings-acl.697.

Niqi Lyu, Pengtao Shi, Wei Qiu, Jianlin Zhong, Sicong Xia, Jianyao Ma, and Yicheng Ding. PyroDash: Cost-efficient token-level small-large language model collaborative inference. *arXiv preprint arXiv:2607.20327*, 2026.

Dehai Min, Giovanni Vaccarino, Huiyi Chen, Yongliang Wu, Gal Yona, and Lu Cheng. Stop when reasoning converges: Semantic-preserving early exit for reasoning models. *arXiv preprint arXiv:2605.17672*, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

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*, 2025.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. SpecReason: Fast and accurate inference-time compute via speculative reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 14291–14310, 2025. doi: 10.52202/085713-0428.

Gibum Park, Sanghyun Han, Yonghwa Cho, Chanjeong Park, and Kyunghan Lee. SAIL: Redesigning collaborative language inference with a single server-to-mobile handoff. In *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services*, pp. 580–592. ACM, 2026. doi: 10.1145/3745756.3809224.

Xiao Pu, Michael Saxon, Wenyue Hua, and William Yang Wang. ThoughtTerminator: Benchmarking, calibrating, and mitigating overthinking in reasoning models. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=oHR862dpMC.

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

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. doi: 10.48550/arXiv.2402.03300. URL https://arxiv.org/abs/2402.03300.

Jianshu She, Zhuohao Li, Zhemin Huang, Qi Li, Peiran Xu, Haonan Li, and Qirong Ho. Hawkeye: Model collaboration for efficient reasoning. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=bdCWK4NkK7.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297. ACM, 2025. doi: 10.1145/3689031.3696075.

Jiwon Song, Yoongon Kim, and Jae-Joon Kim. RelayGen: Intra-generation model switching for efficient reasoning. *arXiv preprint arXiv:2602.06454*, 2026. doi: 10.48550/arXiv.2602.06454. URL https://arxiv.org/abs/2602.06454.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Xin Zhao, Liu Zheng, Zhongyuan Wang, Lei Fang, and Ji-Rong Wen. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026a. doi: 10.18653/v1/2026.acl-long.792. URL https://aclanthology.org/2026.acl-long.792/.

Renliang Sun, Wei Cheng, Dawei Li, Haifeng Chen, and Wei Wang. Stop when enough: Adaptive early-stopping for chain-of-thought reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27250–27268. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.acl-long.1256. URL https://aclanthology.org/2026.acl-long.1256/.

Jikai Wang, Juntao Li, Jianye Hou, Yan Bowen, Lijun Wu, and Min Zhang. Efficient reasoning for LLMs through speculative chain-of-thought. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 1541–1553, 2026. doi: 10.18653/v1/2026.findings-acl.76.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh Agrawal, Sandeep Singh Sandha, Siddartha Venkat Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. LiveBench: A challenging, contamination-free LLM benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.19314.

Taiqiang Wu, Zenan Xu, Bo Zhou, and Ngai Wong. The art of efficient reasoning: Data, reward, and optimization. *arXiv preprint arXiv:2602.20945*, 2026. URL https://arxiv.org/abs/2602.20945.

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 *The Fourteenth International Conference on Learning Representations*, 2026.

Wang Yang, Xiang Yue, Vipin Chaudhary, and Xiaotian Han. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time. *arXiv preprint arXiv:2504.12329*, 2025.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proceedings of the VLDB Endowment*, 16(12): 3848–3860, 2023. doi: 10.14778/3611540.3611569.

Yi Zhao, Yajuan Peng, Cam-Tu Nguyen, Zuchao Li, Xiaoliang Wang, Xiaoming Fu, and Hai Zhao. TrigReason: Trigger-based collaboration between small and large reasoning models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6700–6718, 2026. doi: 10.18653/v1/2026.findings-acl.333.

Shu Zhou, Rui Ling, Junan Chen, Xin Wang, Tao Fan, and Hao Wang. When more thinking hurts: Overthinking in LLM test-time compute scaling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23967–23977, 2026. doi: 10.18653/v1/2026.findings-acl.1199.

Jason Zhu and Hongyu Li. Towards concise and adaptive thinking in large reasoning models: A survey. *arXiv preprint arXiv:2507.09662*, 2025.

APPENDIX

A TRAINING DETAILS

We construct Reasoning Finisher training data from the official training split of MATH, which contains 7,500 competition-level mathematics problems (Hendrycks et al., 2021). For each problem, the corresponding large model generates a reasoning trajectory of up to 32K tokens. After filtering generations with invalid request status or malformed output structure, 7,240 examples remain. We split them into 7,040 examples for training, 100 for validation, and 100 for internal testing.

Each retained trajectory is paired with a training input constructed from either the original question alone or an intermediate Large-Model Reasoning Prefix (LRP), depending on the training configuration described below.

A.1 TRAINING CONFIGURATIONS

We study two aspects of Reasoning Finisher training: the conditioning context and the training continuation budget W_{train}.

The *question-only* configuration receives the original problem without an LRP and serves as a matched control for prefix-conditioned training. Our default *Leakage-Free LRP* configuration conditions the finisher on an intermediate large-model reasoning prefix from which detected answer leakage has been removed. Unless otherwise stated, both use $W_{\text{train}} = 2\text{K}$.

To study the training continuation budget, we additionally train Leakage-Free LRP finishers with

$$W_{\text{train}} \in \{1\text{K}, 2\text{K}, 4\text{K}\}.$$

To study LRP construction independently of this budget, we compare Leakage-Free LRP with Short Leakage-Free LRP and Unfiltered LRP at $W_{\text{train}} = 2\text{K}$.

All Qwen3.5-2B training variants use the same training, validation, and internal test splits; only the conditioning construction or W_{train} is changed. The default Qwen3.5-2B Reasoning Finisher is trained on LRPs generated by Qwen3.5-35B-A3B. For the cross-family experiments, Gemma-4-E2B-it finishers are trained separately on LRPs generated by Qwen3.5-35B-A3B or Gemma-4-31B-it, as indicated in Table 2.

The original released Qwen3.5-2B checkpoint is used as an untrained control and does not undergo Reasoning Finisher training.

A.2 LRP CONSTRUCTION

For prefix-conditioned training, we construct an LRP from the corresponding large-model reasoning trajectory and tokenize the resulting text with the Reasoning Finisher tokenizer. We consider three constructions.

Leakage-Free LRP. We prevent explicit final-answer leakage by identifying answer-revealing structures in the large-model trajectory, such as final-answer phrases, `\boxed{}`, `\fbox{}`, or other detectable answer indicators. If such a structure is present, the LRP is truncated before its first occurrence. The resulting prefix therefore contains only the reasoning context preceding the detected final answer.

Short Leakage-Free LRP. We first apply the same leakage-removal procedure and then truncate the LRP at an earlier position within the remaining leakage-free reasoning region. This variant provides less inherited reasoning context and tests whether the finisher can complete the trajectory from an earlier reasoning state.

Unfiltered LRP. This variant does not apply the answer-leakage boundary and may therefore contain text that directly or indirectly reveals the final answer. We include it only as a diagnostic construction; because it originates from the historical unfiltered setting, we do not interpret it as an isolated answer-leakage ablation.

A.3 Reward and Optimization

We use the verifiable outcome reward described in Section 3.3. A sampled continuation receives reward 1 only if it terminates naturally within W_{train}, has a structurally valid output, and produces the correct final answer:

$$r(\hat{\mathbf{y}}) = \begin{cases} 1, & f = \text{stop} \wedge g_{\text{struct}}(\hat{\mathbf{y}}) = 1 \wedge g_{\text{correct}}(\hat{\mathbf{y}}) = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Continuations that reach W_{train} before satisfying these conditions are treated as truncated and receive zero reward. Thus, W_{train} controls the amount of continuation reasoning available during training without changing the correctness criterion.

For each training input, rewards from the sampled continuations are normalized within the group to construct relative advantages.

A.4 Training Procedure

All Reasoning Finisher variants are trained using veRL (Sheng et al., 2025) with PyTorch FSDP2 (Zhao et al., 2023) on 32 NVIDIA L20Z GPUs across four nodes. Each configuration is trained for two epochs over the 7,040-example training set, corresponding to 440 optimizer updates. Each update contains 32 prompt groups with eight sampled continuations per prompt, yielding 256 trajectories per update. With one trajectory per GPU microbatch, gradients are accumulated for eight microsteps.

We optimize with AdamW using a constant learning rate of 2×10^{-7}, $\beta = (0.9, 0.999)$, weight decay 0.01, no warmup, and gradient clipping at 1.0. We use symmetric importance-ratio clipping with $\epsilon = 0.2$ and KL regularization with coefficient 0.01 via `low_var_kl` in the actor loss; no KL penalty is added to the reward.

For rollout sampling, we use temperature 1.0, top-$p = 0.95$, and top-$k = 20$. The maximum prompt length is 12,288 tokens; this limit is not reached by any training example.

B Evaluation Protocol and Metrics

B.1 Evaluation Protocol

We evaluate all methods on the same common set of 3,595 examples from the ten benchmarks described in Section 4.1. Unless otherwise stated, reported averages are first computed within each benchmark and then macro-averaged across datasets, so that each benchmark contributes equally regardless of its number of examples.

Latency experiments use the live concurrent-serving protocol described in Appendix C. Each source request is timed independently from dispatch until its final output is committed; scheduling and queueing delays are therefore included in the reported latency. Accuracy and latency are computed on the same evaluated examples for each configuration.

We use final-answer accuracy and request-to-commit latency as the primary metrics. We additionally report large-model decoding reduction, analytical FLOPs, and handoff commit rate to characterize computational savings and handoff behavior.

B.2 Metrics

Final-answer accuracy. For benchmark d, final-answer accuracy is

$$\text{ACC}_d = \frac{c_d}{n_d}, \tag{8}$$

where c_d is the number of examples whose committed final answer is judged correct and n_d is the number of evaluated examples in benchmark d. Across the $D = 10$ benchmarks, we report the dataset-macro average

$$\text{ACC}_{\text{avg}} = \frac{1}{D} \sum_{d=1}^{D} \text{ACC}_d. \tag{9}$$

Request-to-commit latency. For example i, request-to-commit latency is

$$T_i = t_i^{\text{commit}} - t_i^{\text{dispatch}}, \tag{10}$$

where t_i^{dispatch} is the time at which the request is dispatched and large-model inference begins, and t_i^{commit} is the earliest time at which the committed final output can be returned. For each benchmark, we average T_i over its evaluated examples and report the dataset-macro average across benchmarks.

Large-model decoding reduction. We measure the fraction of expensive large-model autoregressive decoding avoided by ALSH relative to standalone large-model inference. For benchmark d,

$$S_{L,d} = 1 - \frac{\sum_{i \in d} N_{L,i}^{\text{ALSH}}}{\sum_{i \in d} N_{L,i}^{\text{standalone}}}, \tag{11}$$

where $N_{L,i}^{\text{ALSH}}$ is the number of tokens decoded by the large model before the final output is committed, and $N_{L,i}^{\text{standalone}}$ is the number decoded by standalone large-model inference under the same generation horizon. We report the dataset-macro average of $S_{L,d}$ across benchmarks. Large- and small-model tokens are not combined into a single token count, since their per-token computational costs differ.

Analytical FLOPs. To account for computation performed by both models, we analytically compute the FLOPs associated with prefill and autoregressive decoding. For example i,

$$F_i = F_i^{\text{prefill}} + F_i^{\text{decode}}. \tag{12}$$

For ALSH, the total model computation is

$$F_i^{\text{ALSH}} = F_{L,i} + \sum_{k \in \mathcal{P}_i} F_{S,i,k}, \tag{13}$$

where $F_{L,i}$ is the computation performed by the large model and $\mathcal{P}_i$ is the set of Small-Model Completion Probes launched for example i. We count computation performed by the large model and all launched probes up to the final commit and subsequent cancellation of in-flight requests; work not executed after cancellation is excluded. These FLOPs characterize algorithmic model computation and exclude hardware-dependent overheads such as communication, scheduling, and kernel launch costs. We report benchmark-level mean prefill, decode, and total computation in TFLOPs and macro-average them across benchmarks. Detailed FLOPs accounting is provided in Appendix C.

Handoff commit rate. For benchmark d, handoff commit rate is the fraction of evaluated examples whose final output is committed from a Small-Model Completion Probe:

$$R_{\text{commit},d} = \frac{1}{n_d} \sum_{i \in d} \mathbb{I}\left[\exists k : \, a_i^\star = \text{commit}_S(k)\right], \tag{14}$$

where $a_i^\star$ is the final commit action for example i. We report the dataset-macro average of $R_{\text{commit},d}$ across benchmarks. This metric is defined over source examples rather than individual probes: multiple probes may be launched for one example, but at most one determines the committed final output.

C Inference and System Configuration

C.1 Live Serving Configuration

All inference experiments use vLLM 0.23.0 in bfloat16 precision. Our main evaluation uses live concurrent serving with up to 16 source questions active at a time. Each source request progresses independently, and Small-Model Completion Probes are submitted dynamically as individual requests reach their handoff points. Probe requests are scheduled by the finisher service as they arrive rather than being accumulated into fixed batches.

The 2B Reasoning Finishers use tensor parallelism of 1 with `max_num_seqs=16`; each probe requests a single continuation. Model-specific tensor-parallel configurations for the large models are reported in Appendix C.3.

Unless otherwise stated, ALSH uses a 1K probe interval and $W_{\text{test}} = 2\text{K}$. Generation uses temperature 1.0, top-$p = 0.95$, top-$k = 20$, and random seed 42. Latency is measured independently for each source request and includes scheduling and queueing delays, as defined in Appendix B.1.

C.2 Baseline Implementations

Tandem. We adapt Tandem (Fu et al., 2026) to the same Qwen model family used in our main ALSH configuration. Tandem uses Qwen3.5-35B-A3B as its large model and the original Qwen3.5-2B checkpoint as its answer model, rather than our trained Reasoning Finisher.

The sufficiency classifier is a 32-unit ReLU MLP with a sigmoid output. Its 17 standardized input features consist of seven summary statistics each for token-level perplexity and entropy, one temporal-trend feature for each, and the input length, extracted through a separate Qwen3.5-2B teacher-forcing pass over the complete question–insight context. Training and validation are source-disjoint, feature standardization is fitted only on the training split, and classifier thresholds are frozen before evaluation. Gold answers and evaluation-time correctness labels are not provided to the classifier.

At inference time, Tandem incrementally generates 100, 400, and 500 mentor tokens and applies thresholds $(1.0, 1.0, 0.0)$ using the original strict-greater-than stopping rule. The first two thresholds therefore cannot trigger transfer, so normal executions hand off after the cumulative 1K-token mentor stage. This preserves Tandem's classifier-based transfer mechanism while matching the large-model checkpoint used in the main ALSH comparison. Also, for Tandem, total FLOPs additionally include feature extraction and sufficiency-classifier inference.

RelayGen. We implement RelayGen (Song et al., 2026) using the same Qwen3.5-35B-A3B and original Qwen3.5-2B checkpoints used in the main large–small comparison. RelayGen is training-free and performs segment-level model switching using model-specific switch cues obtained from offline uncertainty profiling.

We preserve RelayGen's method-specific prefix-reuse mechanism during live inference. When generation switches between models, previously processed context is reused rather than recomputed from scratch.

All reported RelayGen accuracy, latency, and auxiliary compute statistics use this prefix-reuse-enabled implementation.

C.3 Hardware and Model Placement

All inference experiments use NVIDIA L20Z GPUs with 82 GB of memory per GPU. Qwen3.5-35B-A3B and Gemma-4-31B-it use tensor parallelism of 2, Qwen3.5-27B uses tensor parallelism of 8, and Qwen3.5-397B-A17B uses tensor parallelism of 16. All 2B models and their trained variants use tensor parallelism of 1, with one GPU per inference replica.

Because the large models use different tensor-parallel allocations, latency comparisons in the generalization experiments are made within each large-model configuration, i.e., between standalone inference and ALSH using the same large model and hardware allocation.

C.4 Analytical FLOPs Accounting

We report analytical model FLOPs rather than hardware utilization or GPU-time measures. We count one multiply–accumulate as two FLOPs. For each model request, prefill and decode computation are calculated from the model architecture and the actual prompt and generated-token lengths, rather than from maximum sequence lengths or a parameter-count approximation.

Prefill computation accounts for all uncached input tokens. ALSH uses finisher-side prefix reuse. Cache-off is the controlled comparison in Appendix D.4, which isolates the effect of prefix reuse.

For generation, we use causal accounting: prefill produces the hidden state used to predict the first output token, after which generating G tokens requires $G - 1$ cached decode forward passes and G output projections. For ALSH, we sum computation from the large model and all launched probes, including probes whose outputs are ultimately discarded and computation performed before cancellation.

The accounting is architecture-aware. For Qwen mixture-of-experts layers, we count the activated routed experts, the shared expert, and router computation rather than using the nominal model parameter count. Full-attention layers use the actual causal context length, while Qwen Gated DeltaNet layers are accounted for using their corresponding per-token operations. For Gemma, global-attention layers use the full available context and sliding-attention layers use the effective sliding-window context.

Tensor parallelism and GPU count do not change the reported aggregate FLOPs. We exclude hardware-dependent effects such as communication, memory traffic, kernel-launch overhead, scheduling, and serving utilization; these effects are instead reflected in measured request-to-commit latency.

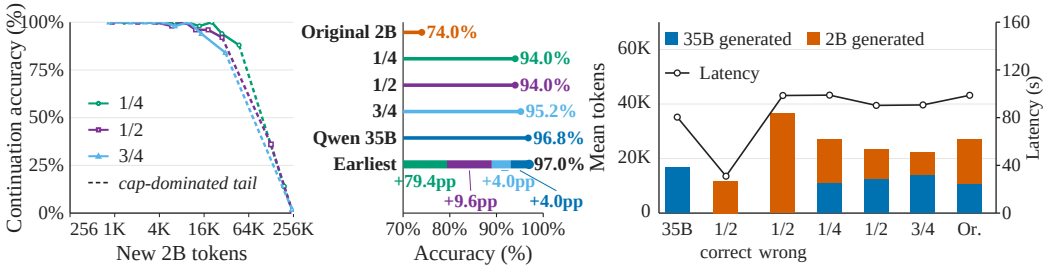
D Additional Experimental Results

D.1 Motivating Handoff Analysis

Figure 3 examines handoff behavior on 500 MATH-500 examples using Qwen 35B and the original Qwen 2B under a native 256K context. Alongside standalone inference, we evaluate fixed-point handoff: Qwen 2B independently continues from prefixes containing $1/4$, $1/2$, or $3/4$ of each pre-generated Qwen 35B trace by token count. Length-capped continuations fall back to the full 35B answer. The post-hoc earliest-correct oracle (labeled *Earliest* in panel (b) and *Or.* in panel (c)) selects the first correct uncapped continuation in the order $1/4 \to 1/2 \to 3/4$, otherwise using the full 35B answer; it is a diagnostic rather than an online policy.

Panel (a) shows that longer continuations are associated with lower pre-fallback accuracy; dashed tails mark bins in which at least half of the attempts are length-capped. The oracle's higher accuracy in panel (b) reveals complementary takeover opportunities across handoff positions, while panel (c) shows that long 2B continuations can offset their per-token speed advantage. These observations motivate repeated probing, a bounded takeover window, and dedicated Reasoning Finisher training (Section 3.1).

In panel (c), *1/2 correct* and *1/2 wrong* separate correct and incorrect uncapped continuations and report continuation-only tokens and latency; other categories report selected-path totals. Handoff latencies use replay estimates with proportional large-model prefix time, excluding discarded 2B work on fallback and failed-probe and verification costs for the oracle.



(a) New 2B tokens vs. accuracy (b) Final-answer accuracy (c) Mean tokens and latency

Figure 3: Fixed-point handoff and repeated-probing analysis on MATH-500. **(a)** Continuation accuracy versus newly generated 2B tokens. **(b)** Final-answer accuracy. **(c)** Generated tokens (bars) and latency (line).

D.2 Complementary Efficiency and Handoff Results

This section complements the accuracy–latency results in the main paper with large-model decoding reduction, analytical computation, and handoff behavior. Metric definitions and FLOPs accounting are provided in Appendices B.2 and C.4, respectively.

For the tables below, we report each auxiliary metric in a separate panel to preserve the full per-benchmark results without stacking quantities with different units in the same cell.

D.2.1 Main Comparison

Table 4: Complementary efficiency and routing results for the main comparison in Table 1. The three metrics are separated into panels so that quantities with different units are not stacked in one cell. For Tandem and RelayGen, the final panel reports terminal-small rate; for ALSH rows, it reports handoff commit rate. Avg. weights datasets equally. Bold marks the largest decoding reduction and lowest FLOPs in each column.

Method	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
Large-model decoding reduction (%)											
Tandem	**89.7**	**82.4**	**95.9**	**95.1**	**94.9**	**96.4**	**95.9**	**94.1**	**93.6**	**95.6**	**93.4**
RelayGen	-2.0	16.8	4.1	5.9	5.6	3.5	2.6	-8.6	0.6	4.2	3.3
Original ($W_{\text{test}} = 2$K)	10.5	32.6	1.0	0.7	4.1	0.4	1.6	3.1	7.6	2.6	6.4
Original ($W_{\text{test}} = 16$K)	0.8	32.1	-1.3	-1.3	2.0	-0.2	-3.5	-1.7	0.8	1.0	2.9
Question-only	58.2	75.7	20.1	28.8	38.2	16.8	16.4	36.7	39.6	18.8	34.9
Prefix-conditioned	73.7	78.8	40.3	53.0	57.9	43.5	50.3	64.1	62.8	32.9	55.7
Total analytical FLOPs (TFLOPs/source)											
Qwen 35B (standalone)	72.1	41.6	201.1	167.0	161.3	236.9	198.1	133.0	121.0	192.4	152.4
Tandem	**28.3**	23.5	**49.7**	**42.4**	**48.5**	**50.2**	**50.5**	**36.3**	**35.4**	**55.4**	**42.0**
RelayGen	221.5	90.8	406.2	550.9	849.8	714.2	647.2	342.9	301.0	625.4	475.0
Original ($W_{\text{test}} = 2$K)	163.0	62.9	562.4	461.1	420.8	678.0	534.3	340.5	298.8	519.1	404.1
Original ($W_{\text{test}} = 16$K)	379.9	101.9	1597.4	1208.6	1078.7	1837.2	1559.4	885.7	771.1	1375.0	1079.5
Question-only	70.9	15.4	442.0	318.2	273.1	544.0	455.6	221.9	192.5	426.1	296.0
Prefix-conditioned	38.7	**12.1**	312.4	195.3	178.6	337.8	252.3	117.1	115.9	348.0	190.8
Small-model final-output / handoff commit rate (%)											
Tandem	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
RelayGen	93.2	95.8	70.0	76.3	78.5	56.5	73.3	80.0	88.1	71.0	78.3
Original ($W_{\text{test}} = 2$K)	41.2	43.5	10.0	20.7	31.8	8.5	13.3	28.9	35.7	20.2	25.4
Original ($W_{\text{test}} = 16$K)	5.4	27.8	3.3	1.5	4.6	0.5	3.3	6.7	7.1	2.3	6.3
Question-only	94.6	79.9	73.3	85.4	85.1	68.0	76.7	91.1	95.2	61.9	81.1
Prefix-conditioned	93.6	78.1	90.0	94.4	91.0	87.0	93.3	97.8	95.2	75.3	89.6

ALSH uses substantially less total analytical computation than RelayGen (190.8 versus 475.0 TFLOPs/source) while reducing large-model decoding much more strongly (55.7% versus 3.3%).

Figure 4 shows that Tandem transfers before three valid cases are derived and returns an incorrect answer, whereas ALSH's prefix covers all six cases and its finisher returns the correct answer.

D.2.2 Generalization

Table 5 reports the same three complementary metrics for the model-scale and model-family transfer settings in Table 2.

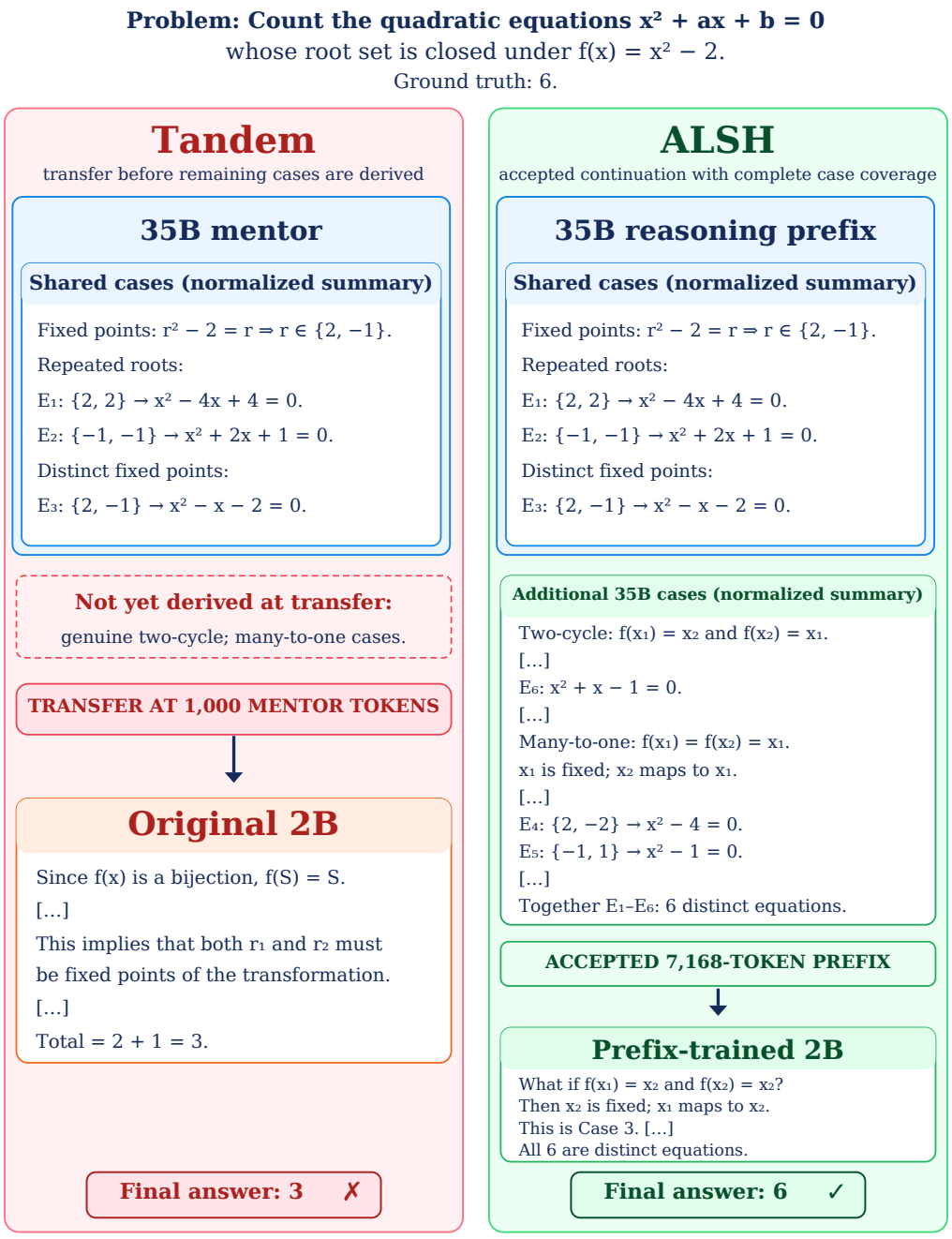


Figure 4: Tandem vs. ALSH on a MATH-500 algebra example.

Table 5: Complementary efficiency and handoff results for the generalization study in Table 2. Standalone models appear only in the FLOPs panel because decoding reduction and handoff commit rate are not defined for them. Avg. weights datasets equally. Bold marks the largest decoding reduction and lowest FLOPs among configurations sharing the same large model. Groups with only one eligible configuration are not highlighted.

Method	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
Large-model decoding reduction (%)											
Qwen 35B → Qwen 2B	73.7	**78.8**	40.3	53.0	**57.9**	**43.5**	**50.3**	64.1	62.8	32.9	55.7
Qwen 27B → Qwen 2B	74.2	76.5	43.1	53.0	56.4	42.3	51.2	66.7	61.1	32.0	55.7
Qwen 397B → Qwen 2B	72.4	60.5	40.2	56.2	57.6	48.4	54.8	67.3	67.1	37.0	56.1
Gemma 31B → Qwen 2B	**35.0**	26.3	**22.2**	41.0	**30.9**	12.7	**14.6**	**39.5**	**41.9**	18.2	**28.2**
Gemma 31B → Gemma E2B^{G}	30.3	27.0	14.4	**59.4**	23.5	**15.4**	10.5	39.0	34.1	**27.3**	28.1
Qwen 35B → Gemma E2B^{G}	73.8	78.5	**46.4**	74.7	47.3	41.1	46.9	63.8	**69.6**	39.7	**58.2**
Qwen 35B → Gemma E2B^{Q}	**74.0**	78.5	37.7	**76.6**	45.8	40.2	43.3	**64.3**	64.4	**40.2**	56.5
Gemma 31B → Gemma E2B^{Q}	29.6	**27.2**	11.1	55.3	19.7	12.8	13.1	29.7	33.8	27.1	25.9
Total analytical FLOPs (TFLOPs/source)											
Qwen 35B (standalone)	72.1	41.6	**201.1**	167.0	**161.3**	**236.9**	**198.1**	133.0	121.0	**192.4**	**152.4**
Qwen 35B → Qwen 2B	**38.7**	**12.1**	312.4	195.3	178.6	337.8	252.3	**117.1**	**115.9**	348.0	190.8
Qwen 27B (standalone)	549.7	286.8	1370.3	1145.0	1065.7	1534.8	1423.3	916.4	793.4	1262.2	1034.8
Qwen 27B → Qwen 2B	**160.1**	**72.1**	**950.9**	**662.5**	**581.8**	**1087.7**	**825.6**	**352.4**	**367.9**	**1082.5**	**614.4**
Qwen 397B (standalone)	267.7	101.4	844.7	625.4	609.8	1004.9	844.5	551.7	493.1	807.5	615.1
Qwen 397B → Qwen 2B	**89.5**	**44.4**	**663.8**	**354.2**	**341.6**	**661.9**	**499.0**	**227.2**	**211.4**	**672.9**	**376.6**
Gemma 31B (standalone)	176.5	77.0	575.2	386.5	395.7	**642.4**	525.0	349.2	323.6	439.9	389.1
Gemma 31B → Qwen 2B	**125.9**	58.8	**515.2**	265.4	**317.2**	647.1	**518.7**	**240.7**	**212.3**	411.8	**331.3**
Gemma 31B → Gemma E2B^{G}	137.2	58.3	584.3	**187.8**	357.1	648.3	562.0	247.4	249.2	**383.0**	341.4
Qwen 35B → Gemma E2B^{G}	44.5	13.4	396.3	121.5	402.8	646.4	372.3	162.0	116.1	586.1	286.2
Qwen 35B → Gemma E2B^{Q}	43.9	13.3	545.8	**104.7**	426.5	658.0	435.6	159.4	143.6	573.5	310.4
Gemma 31B → Gemma E2B^{Q}	138.8	**58.2**	610.2	206.4	375.1	666.5	542.8	287.5	251.2	383.7	352.1
Small-model final-output / handoff commit rate (%)											
Qwen 35B → Qwen 2B	93.6	78.1	90.0	94.4	91.0	87.0	93.3	97.8	95.2	75.3	89.6
Qwen 27B → Qwen 2B	93.2	79.5	93.3	93.9	93.5	90.5	96.7	100.0	95.2	78.3	91.4
Qwen 397B → Qwen 2B	95.6	68.8	90.0	97.5	95.9	93.0	90.0	97.8	97.6	83.4	91.0
Gemma 31B → Qwen 2B	47.0	13.6	66.7	86.4	67.4	52.0	66.7	82.2	76.2	47.3	60.5
Gemma 31B → Gemma E2B^{G}	43.6	14.2	53.3	96.0	51.6	61.0	70.0	80.0	83.3	73.2	62.6
Qwen 35B → Gemma E2B^{G}	95.8	81.8	96.7	99.5	82.3	91.5	96.7	97.8	97.6	87.3	92.7
Qwen 35B → Gemma E2B^{Q}	96.2	79.6	86.7	100.0	82.9	92.5	93.3	97.8	97.6	87.3	91.4
Gemma 31B → Gemma E2B^{Q}	45.6	14.4	56.7	96.0	52.2	52.5	73.3	75.6	83.3	73.0	62.3

D.2.3 FINISHER TRAINING DESIGN

Table 6 complements the training-design analysis in Section 4.3. It preserves the full ten-benchmark breakdown for each finisher-training variant.

Table 6: Complementary efficiency and handoff results for the finisher-training study in Table 3. The full ten-benchmark results are retained, with one panel per metric. Avg. weights datasets equally.

Method	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
Large-model decoding reduction (%)											
Original	10.5	32.6	1.0	0.7	4.1	0.4	1.6	3.1	7.6	2.6	6.4
Question-only	58.2	75.7	20.1	28.8	38.2	16.8	16.4	36.7	39.6	18.8	34.9
Leakage-Free LRP	73.7	78.8	40.3	53.0	57.9	43.5	50.3	64.1	62.8	32.9	55.7
Leakage-Free LRP ($W_{\text{train}} = 1$K)	83.1	80.0	61.3	72.1	75.3	73.0	70.8	83.1	82.4	43.7	72.5
Leakage-Free LRP ($W_{\text{train}} = 4$K)	62.2	74.7	26.7	37.6	45.2	27.8	30.0	48.3	53.0	21.2	42.7
Short Leakage-Free LRP	72.8	78.4	41.6	49.6	57.0	40.9	51.1	67.5	65.7	31.5	55.6
Unfiltered LRP	74.6	78.5	41.4	52.1	56.3	42.6	47.4	65.6	67.6	32.9	55.9
Total analytical FLOPs (TFLOPs/source)											
Original	163.0	62.9	562.4	461.1	420.8	678.0	534.3	340.5	298.8	519.1	404.1
Question-only	70.9	15.4	442.0	318.2	273.1	544.0	455.6	221.9	192.5	426.1	296.0
Leakage-Free LRP	38.7	12.1	312.4	195.3	178.6	337.8	252.3	117.1	115.9	348.0	190.8
Leakage-Free LRP ($W_{\text{train}} = 1$K)	21.0	11.0	195.0	107.1	90.1	136.5	129.0	44.6	43.6	283.4	106.1
Leakage-Free LRP ($W_{\text{train}} = 4$K)	61.3	16.5	394.4	270.9	234.0	446.7	380.6	175.3	144.4	408.1	253.2
Short Leakage-Free LRP	41.1	12.6	330.6	213.4	180.6	358.3	254.7	99.0	101.1	348.7	194.0
Unfiltered LRP	36.6	12.7	312.9	199.5	184.8	340.9	267.3	107.4	92.1	347.7	190.2
Small-model final-output / handoff commit rate (%)											
Original	41.2	43.5	10.0	20.7	31.8	8.5	13.3	28.9	35.7	20.2	25.4
Question-only	94.6	79.9	73.3	85.4	85.1	68.0	76.7	91.1	95.2	61.9	81.1
Leakage-Free LRP	93.6	78.1	90.0	94.4	91.0	87.0	93.3	97.8	95.2	75.3	89.6
Leakage-Free LRP ($W_{\text{train}} = 1$K)	95.0	80.2	93.3	98.5	97.8	99.5	100.0	100.0	100.0	81.8	94.6
Leakage-Free LRP ($W_{\text{train}} = 4$K)	94.2	77.6	80.0	87.9	89.1	79.0	83.3	95.6	97.6	65.8	85.0
Short Leakage-Free LRP	95.2	79.3	80.0	91.4	91.6	85.5	90.0	100.0	97.6	74.4	88.5
Unfiltered LRP	95.4	80.1	86.7	93.9	91.6	88.5	93.3	100.0	100.0	77.5	90.7

D.3 RUNTIME SENSITIVITY

We vary the inference continuation window W_{test} and probe interval Δ independently while keeping the prefix-conditioned Reasoning Finisher fixed. Table 7 summarizes dataset-macro accuracy, latency, and efficiency; the following tables provide per-benchmark results.

Table 7: Dataset-macro runtime-sensitivity results. Full per-benchmark accuracy and latency appear in Tables 8 and 9. Bold marks the highest accuracy and decoding reduction and the lowest latency and FLOPs within each sweep.

Sweep	Setting	Acc. (%) ↑	Lat. (s) ↓	LM decoding reduction (%) ↑	Total FLOPs ↓ (TFLOPs/source)	Handoff commit (%)
W_{test}	0.5K	75.27	166.25	10.2	271.8	18.7
	1K	80.41	115.01	39.4	214.1	76.9
	2K	**81.62**	**82.91**	**55.7**	**190.8**	89.6
	4K	75.88	87.93	54.5	360.0	87.5
	8K	71.75	99.46	49.8	534.0	81.5
Δ	0.5K	76.61	104.99	47.0	679.6	84.9
	1K	**81.62**	**82.91**	**55.7**	190.8	89.6
	2K	80.83	91.61	49.8	**161.2**	84.0
	4K	81.12	107.70	39.7	162.6	73.7
	8K	78.54	129.16	26.2	171.7	54.1

D.3.1 Inference Continuation Window

Table 8: Inference continuation-window sweep with the Leakage-Free LRP Qwen 2B finisher and Qwen 35B, at fixed $\Delta = 1$K under the live concurrent-serving setting. Each cell gives accuracy (%) above mean latency (s). W_{test} varies independently of the fixed 2K training budget. Bold marks the highest accuracy and lowest latency in each column.

W_{test}	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
0.5K	96.40	**96.89**	73.33	70.71	49.73	57.50	76.67	77.78	83.33	70.34	75.27
	79.63	17.96	219.00	192.45	177.80	257.32	214.49	151.92	137.66	214.26	166.25
1K	**98.40**	**96.89**	73.33	77.78	62.50	65.50	**83.33**	**82.22**	**92.86**	71.26	80.41
	38.57	12.96	179.57	121.47	105.74	207.24	143.85	85.92	72.35	182.39	115.01
2K	**98.40**	96.29	**83.33**	**80.81**	**62.77**	**66.50**	**83.33**	80.00	**92.86**	**71.96**	**81.62**
	24.55	12.07	**127.49**	**89.94**	**78.84**	**148.73**	**103.52**	51.92	47.50	**144.56**	**82.91**
4K	97.00	96.21	66.67	78.28	62.50	54.50	70.00	77.78	88.10	67.79	75.88
	20.41	11.85	136.34	96.98	82.88	173.46	114.17	42.64	**35.93**	164.63	87.93
8K	96.20	96.59	60.00	74.75	58.70	46.00	60.00	71.11	88.10	66.05	71.75
	22.21	**11.76**	169.30	103.08	90.13	206.87	132.17	**32.92**	42.17	183.95	99.46

D.3.2 Probe Interval

Table 9: Probe-interval sweep with the Leakage-Free LRP Qwen 2B finisher and Qwen 35B, at fixed $W_{\text{test}} = 2$K under the live concurrent-serving setting. Each cell gives accuracy (%) above mean latency (s). The 1K row is the same headline run used throughout the paper. Bold marks the highest accuracy and lowest latency in each column.

Δ	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
0.5K	96.20	95.38	63.33	78.79	57.07	60.50	76.67	80.00	90.48	67.67	76.61
	23.58	**7.99**	169.48	122.09	101.41	211.90	130.28	**49.79**	48.75	184.67	104.99
1K	98.40	96.29	**83.33**	**80.81**	**62.77**	66.50	83.33	80.00	92.86	**71.96**	**81.62**
	24.55	12.07	**127.49**	**89.94**	**78.84**	**148.73**	**103.52**	51.92	**47.50**	**144.56**	**82.91**
2K	**99.20**	96.44	73.33	76.77	62.23	69.00	**86.67**	**82.22**	90.48	**71.96**	80.83
	31.81	18.28	129.76	97.05	87.21	167.23	112.79	58.49	54.75	158.69	91.61
4K	98.60	96.51	**83.33**	75.76	61.14	**69.50**	80.00	80.00	**95.24**	71.15	81.12
	47.64	27.03	145.99	120.97	101.49	179.50	136.13	71.01	68.84	178.39	107.70
8K	98.80	**96.82**	73.33	75.76	57.34	65.50	76.67	80.00	90.48	70.68	78.54
	66.73	38.50	170.47	139.72	131.27	203.29	156.24	99.16	88.89	197.34	129.16

D.3.3 Matched Training and Inference Windows

To examine the effect of a mismatch between training and inference budgets, we evaluate $W_{\text{train}} = W_{\text{test}} \in \{1\text{K}, 2\text{K}, 4\text{K}\}$, with the remaining settings unchanged. Matching the 1K finisher's infer-

ence window increases accuracy from 65.07% to 72.95%, with latency increasing from 49.90 s to 67.26 s. The matched 2K configuration achieves the highest accuracy and outperforms the matched 4K configuration in both accuracy and latency, supporting our default choice. Table 10 reports the per-benchmark accuracy and latency.

Table 10: Matched training and inference windows with $W_{\text{train}} = W_{\text{test}}$. All other settings follow the default configuration. Each cell reports accuracy (%) above latency (s); Avg. weights the ten benchmarks equally. Bold marks the highest accuracy and lowest latency in each column.

Window	MATH 500	GSM8K	AIME 2025	GPQA-Diamond	LiveBench-Math	Olym MATH	AIME 2026	AMC 2024	AMC 2025	LiveCode Bench	**Avg.**
1K	95.60	96.06	66.67	77.27	58.42	46.50	63.33	75.56	80.95	69.18	72.95
	19.88	**11.42**	**94.87**	**82.67**	**67.52**	**98.13**	**80.84**	**36.41**	**28.10**	152.76	**67.26**
2K	98.40	96.29	**83.33**	**80.81**	**62.77**	**66.50**	**83.33**	**80.00**	**92.86**	**71.96**	**81.62**
	24.55	12.07	127.49	89.94	78.84	148.73	103.52	51.92	47.50	**144.56**	82.91
4K	**99.20**	**96.51**	73.33	79.29	57.34	61.50	**83.33**	**80.00**	**92.86**	71.84	79.52
	29.23	13.12	174.23	135.83	108.75	218.66	133.21	52.60	54.95	191.30	111.19

D.4 Effect of Prefix Reuse

The main experiments enable finisher-side prefix reuse across compatible Reasoning Finisher probes. To isolate its contribution, we additionally evaluate the same default ALSH configuration with prefix reuse disabled.

Prefix reuse reduces prefill computation from 323.4 to 49.8 TFLOPs/source and total analytical computation from 470.0 to 190.8 TFLOPs/source, while latency decreases from 85.70 s to 82.91 s.

Table 11: Effect of finisher-side prefix reuse. Accuracy and latency are dataset-macro averages; decoding reduction and handoff commit rate are percentages; FLOPs are TFLOPs/source. The reuse-on run keeps the large-model configuration unchanged. Missing finisher cache telemetry is conservatively charged as a full prefill. Bold marks higher accuracy and decoding reduction or lower latency and FLOPs between the two ALSH settings.

Setting	Acc.	Lat.	LM red.	Prefill TF	Decode TF	Total TF	Handoff commit (%)
Qwen 35B (standalone)	72.92	175.31	–	1.1	151.4	152.4	–
ALSH, prefix reuse off	79.93	85.70	54.6	323.4	146.6	470.0	87.3
ALSH, prefix reuse on	**81.62**	**82.91**	**55.7**	**49.8**	**141.0**	**190.8**	89.6

D.5 Sample-Level Comparisons

Figures 5 and 6 extend the sample-level view in Figure 2 to all ten benchmarks, under the default 32K generation budget and a native 256K context, respectively. Both compare standalone Qwen 35B with ALSH using the Qwen 2B finisher and finisher-side prefix reuse. Within each setting, the two methods evaluate the same questions using independently generated trajectories.

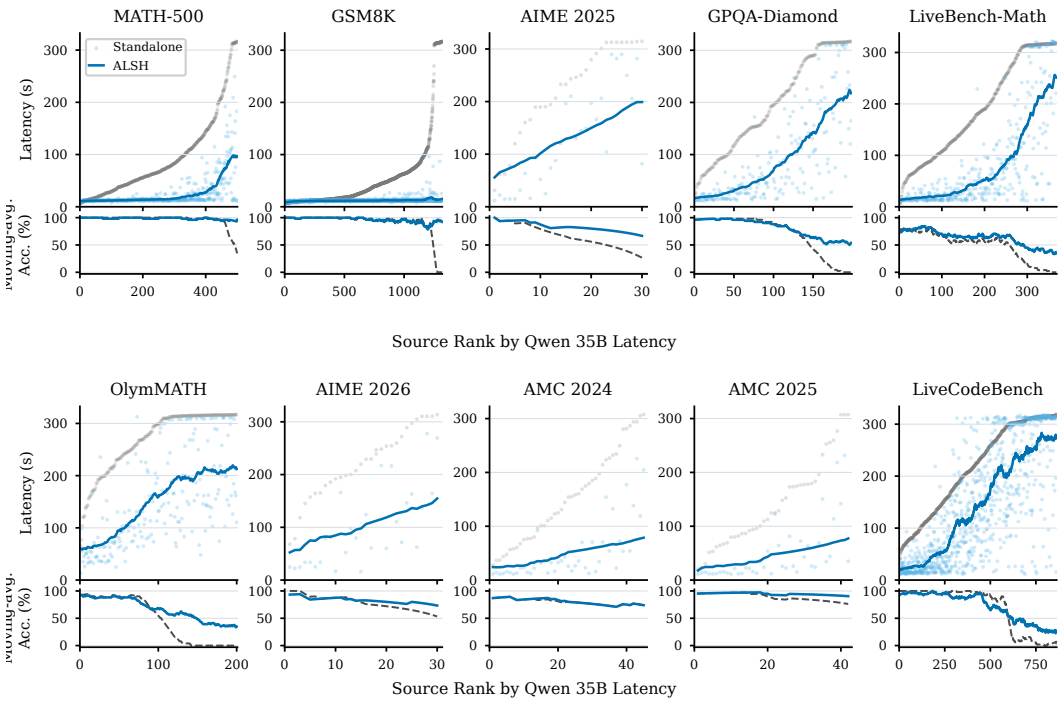


Figure 5: Sample-level results across ten benchmarks under the default 32K generation budget. Within each benchmark, examples are ordered by standalone latency. Dots show individual request latencies; curves show centered moving averages of latency and accuracy over up to 51 examples.

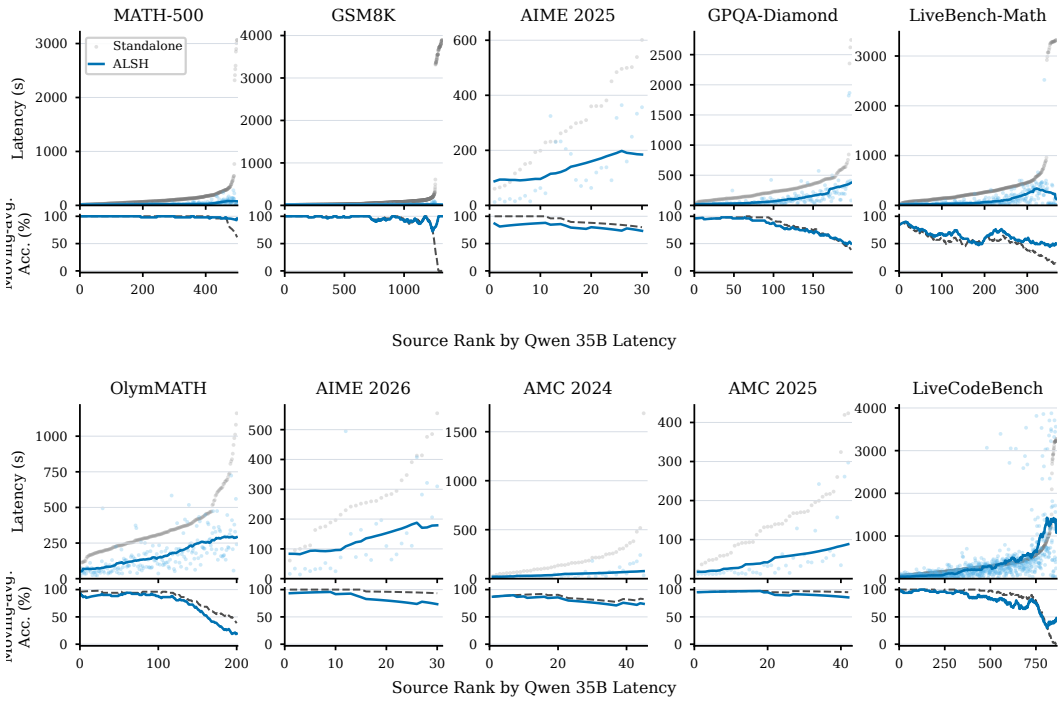


Figure 6: Sample-level results with a native 262,144-token context. Ordering and smoothing follow Figure 5, using standalone latencies from this setting. These results assess behavior under an extended context; trajectories are not held fixed across the two settings.