ACCORDION: BREAKING DEPENDENCY BARRIERS IN MULTI-AGENT LLM WORKFLOWS VIA CONTEXT SPECULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Hierarchical multi-agent workflows typically enforce strict parent–child dependency barriers: a child starts decoding only after its parent finishes, making end-to-end latency dominated by the longest dependency chain. We identify a key source of slack: *children can begin decoding under partial parent context, and revise only when new parent content arrives*. The core challenge lies in controlling speculative waste from context updates. To address this, we propose ACCORDION, a **context-level speculative decoding framework for dependency-constrained workflows**. The parent streams its output as incremental prompt commits, while each child decodes in parallel under its current partial context. After each commit, the child verifies drafted tokens against the updated context, preserves the longest accepted prefix, and rolls back from the first rejection. We further give an information-theoretic view of prefix reuse: under an ideal maximal-reuse verifier, expected acceptance after an update is lower-bounded by a decreasing function of the update's conditional entropy, while the parent self-surprisal provides an online proxy under a stated calibration assumption, yielding a surprisal-triggered commit rule that trades off reuse and rollback. Across diverse workloads, ACCORDION reduces end-to-end latency by up to **46.6**% relative to sequential execution and achieves a more stable speed–accuracy trade-off than heuristic streaming baselines while maintaining comparable task performance. It further composes with model-level speculative decoding.

1 INTRODUCTION

Large language model (LLM) agents are increasingly composed into multi-agent systems (MAS), where specialized agents exchange natural-language messages to solve complex tasks such as planning, critique, retrieval, tool use, and execution (Wu et al., 2024; Li et al., 2023; Guo et al., 2024; OpenAI, 2025; Significant Gravitas, 2023). A common abstraction for such systems is a directed acyclic workflow graph (DAG): each node is an agent invocation, and each edge denotes a dependency in which a *parent* agent's message becomes part of a *child* agent's prompt. This graph structure improves modularity, controllability, and interpretability, but it also changes the efficiency problem. The dominant bottleneck is not only how fast a single model decodes, but also how quickly the entire dependency system executes, especially how parent–child interactions are orchestrated.

In a standard execution model, a child agent begins decoding only after all required parent messages have been fully generated and assembled into its prompt. This strict *dependency barrier* serializes dependent calls and makes end-to-end latency dominated by the critical path of the DAG. In our experiments, increasing *full*-mode MMLU workflows from 3 to 6 agents increases mean end-to-end latency from 15.2s to 37.9s, a 2.49$\times$ growth as the dependency path lengthens. Our target is therefore the serialization-induced inefficiency in dependency-constrained MAS.

Prior work improves MAS efficiency from two complementary directions. One line searches for or learns more efficient workflows, agent organizations, or communication pipelines, such as AgentFlow, AgentPrune, and ADAS (Li et al., 2026; Zhang et al., 2025a; Hu et al., 2025). Another line reduces the cost of repeated context processing or text-mediated communication, for example by transferring cache or latent representations across models or agents (Fu et al., 2026; Zou et al., 2026; Ye et al.,

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.

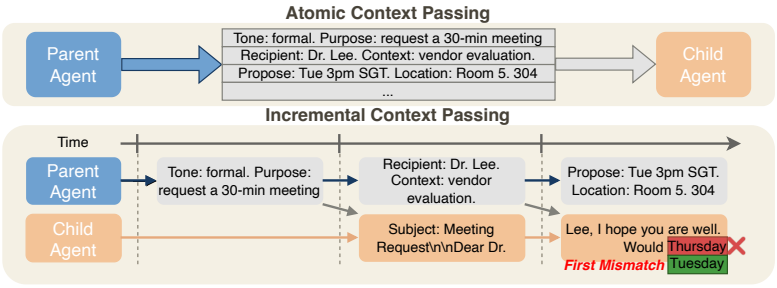


Figure 1: **Atomic vs. incremental context passing. Top:** With atomic context passing, the child begins decoding after the parent completes decoding. **Bottom:** Incremental context passing allows parallel decoding, where the first mismatch appears late, enabling prefix reuse under parent updates.

2025). These methods optimize *what* workflow to execute, *what* representation to communicate, or *how* to reuse already-processed context, rather than *when* a child may start decoding; with an atomic interface, the critical path stays serialized. Streaming systems let downstream agents consume partial outputs without reconciling them against later content (Yang et al., 2026; Wang et al., 2025b), and input-time speculation verifies drafts within a single model rather than across a dependency edge (Li & Grover, 2025). This raises a complementary question: **Can a child begin useful work from partial parent context and minimally revise its draft as additional information arrives?**

Our key observation is that parent messages are often incrementally useful. A child may not need the complete upstream message to begin productive work; early fragments can already reveal the intent, role, plan, output format, or action schema needed to start decoding. As illustrated in Figure 1, after the parent specifies the tone, recipient, and purpose of a meeting request, the child can already draft the subject and opening sentence. Later details, e.g., the proposed time, may only affect later phrases of the draft. This creates an opportunity to overlap parent and child decoding: the child starts from partial parent context and retains the prefix that remains valid after later updates.

However, this opportunity cannot be realized by naively streaming every parent token. Each parent update changes the child's prompt, so tokens drafted under the previous partial context should be checked against the updated context. If updates are too frequent, the system pays high synchronization and verification overhead. If updates are too sparse, the child may decode many tokens under stale context and discard them after rollback. **The challenge is to choose update boundaries that reveal useful parent information early while preserving most child draft work.**

This challenge motivates an information-aware commit policy. The downstream effect of a parent update is not determined simply by its token length. A long but predictable continuation may have little effect on the child's next-token distribution, while a short but informative update can change the child's plan and invalidate many drafted tokens. Our theoretical analysis in Section 4 formalizes this intuition: prefix reuse after a context update is controlled by the information content of the update, and the parent's self-surprisal provides a practical online proxy for this quantity.

In this paper, we introduce ACCORDION, a context-level speculative decoding framework for dependency-constrained multi-agent workflows. The key idea is to reinterpret parent–child execution as *speculation under an evolving prompt*: the child model itself is unchanged, but its conditioning context is gradually updated as upstream parents reveal new information. This adds a new level to speculative decoding: model-level speculation drafts and verifies within one request under a fixed prompt, whereas context-level speculation spans dependent requests, and its mismatch comes from *context drift* caused by newly committed parent content. To control this drift, ACCORDION uses a **surprisal-budgeted commit policy**. Each parent accumulates token-level self-surprisal during generation and publishes a new prompt increment only when the accumulated information exceeds a threshold or a maximum interval is reached. Each child decodes in parallel under the latest committed context; whenever any upstream parent commits an update, the child verifies its current draft under the updated prompt, preserves the longest still-valid prefix, and rolls back from the first mismatch. The same protocol naturally extends to general workflow DAGs: a child reconciles its draft upon any

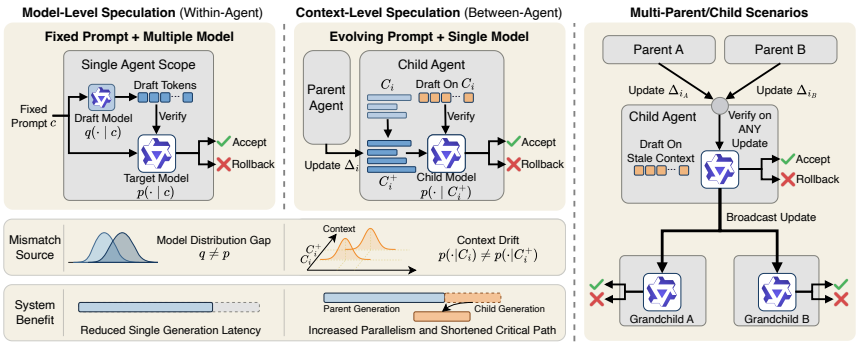


Figure 2: **Model-level vs. context-level speculative decoding. Left and middle:** Comparison between model-level speculation and context-level speculation. Model-level speculation drafts and verifies tokens within a single agent under a fixed context, whereas ACCORDION performs speculation across dependency edges by allowing children to draft under evolving contexts with verification and rollback. **Right:** Extension of context-level speculation to general multi-agent DAGs.

incoming parent update and, once its own output is committed, propagates the update to its children. In summary, our contributions are as follows:

- We identify *barrier-synchronized parent–child execution* as a workflow-level bottleneck in LLM-based multi-agent systems and characterize the *context drift* that arises when speculation spans dependent agents.
- We propose ACCORDION, which performs verified *context-level* speculation over LLM-generated context without changing how agents communicate: children decode under partial parent context and revalidate drafted prefixes as the context evolves. Such speculation cascades through multi-parent DAGs.
- We provide an information-theoretic characterization of prefix reuse under context updates and derive an online surrogate based on the parent's self-surprisal, yielding a principled *surprisal-budgeted* commit policy.
- Across diverse multi-agent workloads, ACCORDION reduces end-to-end latency by up to 46.6% relative to sequential execution and achieves a more stable speed–accuracy trade-off than heuristic streaming baselines, while maintaining comparable task performance.

2 PRELIMINARIES

Autoregressive agents. We consider a multi-agent workflow whose nodes invoke LLM agents and whose edges pass natural-language messages downstream. Let $\mathcal{V}$ be the vocabulary and x the external workflow input, e.g., a user query, task instruction, or tool request. For $Y = (y_1, \ldots, y_T) \in \mathcal{V}^*$, let $y_{<t} = (y_1, \ldots, y_{t-1})$. Invoked with context C, agent a generates Y with probability

$$p_a(Y \mid C) = \prod_{t=1}^{T} p_a(y_t \mid C, y_{<t}). \tag{1}$$

The generated message of agent a is denoted by $M_a \in \mathcal{V}^*$.

Directed message-passing workflows. We model an LLM-based multi-agent workflow as a directed acyclic graph $\mathcal{G} = (\mathcal{A}, \mathcal{E})$, where each node $a \in \mathcal{A}$ is an agent invocation and each edge $(u \to v) \in \mathcal{E}$ incorporates parent message M_u into the prompt of child agent v. Let $\mathrm{pa}(v)$ denote the set of parents of v. Given all required parent messages, C_v is assembled by a workflow-specific context-assembly rule:

$$C_v = \Phi_v\Big(x, \{M_u\}_{u \in \mathrm{pa}(v)}\Big). \tag{2}$$

In the standard *barrier* execution model, v decodes only after every M_u for $u \in \mathrm{pa}(v)$ is complete, so each parent message is an atomic dependency. If ℓ_a is the decoding latency of agent a after its

prompt is available, the barrier makespan is governed by the longest serialized dependency chain, $T_{\text{barrier}} \geq \max_{\pi \in \mathcal{P}(\mathcal{G})} \sum_{a \in \pi} \ell_a$, where $\mathcal{P}(\mathcal{G})$ is the set of directed paths in the workflow. This is the critical-path dominance targeted in this work.

Incremental context availability. ACCORDION keeps the workflow graph and prompt templates fixed, changing only when parent messages become visible to children. For each edge $(u \to v)$, let $M_u^{(\leq i)}$ be the prefix of parent u's message committed after its i-th update, with $M_u^{(\leq 0)} = \emptyset$ and $M_u^{(\leq I_u)} = M_u$ after its last update I_u. For multi-parent v, let i_u denote the latest commit index of parent u observed by v. The child sees a time-varying context

$$C_v^{(i)} = \Phi_v\Big(x, \{M_u^{(\leq i_u)}\}_{u \in \text{pa}(v)}\Big), \tag{3}$$

where i is the child's current synchronization step; any upstream commit increments some i_u and changes the effective child context.

Single-edge notation. Since the same protocol applies to every incoming edge of a DAG, we present the method and analysis for a single edge $(u \to v)$. Parent updates are indexed by $i \in \{1, 2, \ldots\}$. Let C_i denote the child context immediately before the i-th update on this edge. The parent commits a new increment $\Delta_i = (z_{i,1}, \ldots, z_{i,m_i})$, where $z_{i,j} \in \mathcal{V}$ is the j-th new parent token in the i-th update, and the updated child context is

$$C_i^+ := \Phi(C_i, \Delta_i), \tag{4}$$

where Φ abbreviates the single-edge context assembly operation in Eq. (2); equivalently, C_i^+ is the prompt obtained by inserting the newly committed parent segment Δ_i into the appropriate parent-message block of C_i. Let P_i denote the parent decoding state before generating Δ_i, i.e., its prompt and already generated prefix.

From model-level to context-level speculation. Classical model-level speculative decoding drafts tokens with an approximate model $q(\cdot \mid c)$ and verifies them with a target model $p(\cdot \mid c)$ under the same fixed prompt c (Leviathan et al., 2023; Chen et al., 2023; Cai et al., 2024; Xia et al., 2025b). Given a drafted block $\hat{Y}_{1:T}$, the verifier keeps the longest accepted prefix and rolls back from the first rejected token. Here the context is fixed and the mismatch is the model gap between q and p.

ACCORDION keeps the prefix-reuse interface but moves speculation from model level to context level. The child v uses its own model p_v to draft and verify tokens; the mismatch comes from a prompt update. Between two consecutive parent commits, the child drafts $\hat{Y}_{1:T_i}^{(i)} \sim p_v(\cdot \mid C_i)$. After receiving Δ_i, the child verifies the draft under the updated distribution $p_v(\cdot \mid C_i^+)$. Let $R_i \in \{1, \ldots, T_i + 1\}$ be the first rejected index, with $R_i = T_i + 1$ meaning that the whole draft is accepted. The accepted prefix length is $L_i := R_i - 1$, and the prefix acceptance rate is

$$\text{Acc}_i := \frac{L_i}{T_i}, \tag{5}$$

with the convention $\text{Acc}_i = 1$ when $T_i = 0$. The accepted prefix $\hat{Y}_{1:L_i}^{(i)}$ is retained, while the remaining $T_i - L_i$ tokens are discarded and regenerated under C_i^+. Thus, $T_i - L_i$ measures rollback-induced speculative waste, and Acc_i measures how much child-side work survives the parent update.

3 METHOD

3.1 CONTEXT-LEVEL SPECULATIVE DECODING AS A PARENT–CHILD PROTOCOL

We present ACCORDION on a single parent–child edge $(u \to v)$. Applied independently to every dependency edge in a workflow DAG, the protocol lets a child draft under its latest committed upstream context and reconcile whenever any parent commits new content.

Incremental context passing. Using the notation of Section 2, the parent publishes Δ_i and the child context moves from C_i to $C_i^+ = \Phi(C_i, \Delta_i)$. Between commits, the child decodes under C_i and stores a tentative draft $\hat{Y}_{1:T_i}^{(i)}$, which is useful only if it remains valid after the next update.

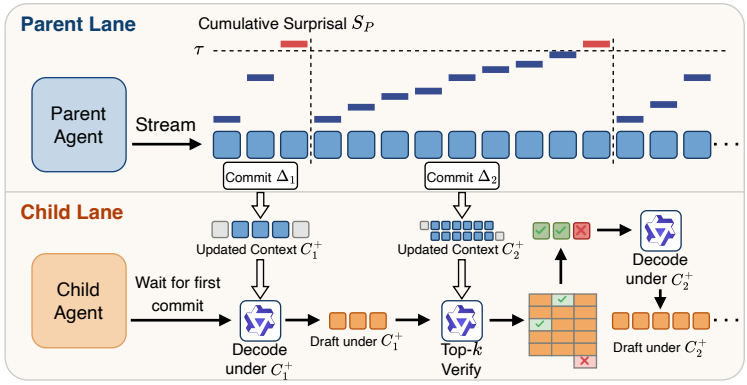


Figure 3: **Protocol of ACCORDION. Top:** The parent commits an increment when the cumulative surprisal reaches threshold τ. **Bottom:** The child drafts under the latest committed context and, upon a new commit, verifies the draft via top-k validation, accepting tokens until the first rejection.

Verification and rollback. When Δ_i arrives, the child verifies its draft under C_i^+ from left to right, retains the accepted prefix $\hat{Y}^{(i)}_{1:L_i}$, and resumes decoding from the first rejected position. Each update thus overlaps parent and child decoding, while rollback limits the cost of using stale context.

3.2 PARENT SIDE: SURPRISAL-TRIGGERED COMMIT

The parent must place commit boundaries in its output stream. Fixed-length or fixed-time commits treat all tokens equally, although parent-message information varies sharply. ACCORDION instead uses the cumulative information revealed since the previous commit.

Parent surprisal. Let P be the parent state before token z, including the parent prompt and generated prefix. Define self-surprisal as

$$s(z) \;:=\; -\log p_P(z \mid P). \tag{6}$$

For a contiguous parent segment $\Delta = (z_1, \ldots, z_m)$, cumulative surprisal is $S_P(\Delta) \;:=\; \sum_{j=1}^{m} s(z_j)$, where each $s(z_j)$ is evaluated at the parent state before generating z_j.

Commit rule. Starting from the previous boundary, the parent extends segment Δ token by token. ACCORDION commits it as update Δ_i once its cumulative surprisal reaches threshold τ or its length reaches the maximum interval $M_{\max}$:

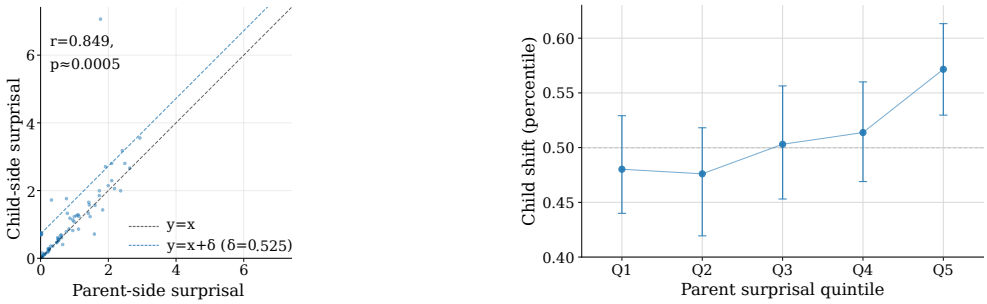
$$\textsc{Commit}(\Delta) \quad \text{if} \quad S_P(\Delta) \geq \tau \quad \text{or} \quad |\Delta| \geq M_{\max}. \tag{7}$$

After committing $\Delta_i = \Delta$, the next segment starts from the following parent token. The threshold τ controls the trade-off: smaller τ reveals parent information earlier but causes more child verification, while larger τ reduces verification frequency but may leave the child drafting longer under stale context. The cap $M_{\max}$ prevents long low-surprisal continuations from being delayed indefinitely.

3.3 CHILD SIDE: SPECULATION AND VERIFICATION

The child drafts under the latest context, and each update triggers VERIFY to keep valid tokens. (Appendix C discusses distribution-preserving verification and KV-cache repair overhead.)

Top-k prefix validation. We adopt a lightweight top-k rule. Given updated context C_i^+, the verifier replays the draft left to right. At position t with accepted prefix $\hat{y}_{<t}$, drafted token $\hat{y}_t$ is accepted if $\hat{y}_t \in \mathrm{TopK}\big(p_v(\cdot \mid C_i^+, \hat{y}_{<t}),\, k\big)$. Verification stops at the first token that fails this test. We use $k = 5$ by default throughout the experiments. Top-k validation keeps a drafted token as long as it stays among the child's k most likely next tokens under the updated context: $k = 1$ enforces exact greedy agreement, and larger k retains more drafted work, permitting mild deviations from greedy sequential decoding (Table 3).



(a) Child vs. parent surprisal

(b) Child distribution shift vs. surprisal

Figure 4: **Empirical evidence for Assumption 4.2 and Eq.** (10)**.** Parent and child surprisal are correlated, and higher-surprisal parent commits shift the child's distribution more (App. B.6).

4 Theory

We justify ACCORDION with a compact information-theoretic chain: controlling the *information content* of each streamed parent increment controls the child's *prefix reuse*, and thus rollback waste (Eq. (C.1)). Full proofs are deferred to Appendix B.

4.1 Acceptance vs. Increment Entropy

Suppose that before receiving Δ_i, the child drafts T_i tokens under C_i; it then verifies them under $C_i^+ = \Phi(C_i, \Delta_i)$ and keeps the longest accepted prefix, yielding $\mathrm{Acc}_i = L_i/T_i$.

Theorem 4.1 (Entropy-controlled acceptance)**.** *Assume the prefix-maximal verification idealization stated in Appendix B. There exist a context-conditioned coefficient $\eta_i \in [0,1]$ and a mismatch term $\varepsilon_i^{\mathrm{mc}} \geq 0$ such that*

$$\mathbb{E}[\mathrm{Acc}_i \mid C_i] \;\geq\; \left[1 - \sqrt{\frac{1}{2}\Big(\eta_i\, H(\Delta_i \mid C_i) + \varepsilon_i^{\mathrm{mc}}\Big)}\right]_+ . \tag{8}$$

Theorem 4.1 formalizes a stability principle: an update with larger conditional entropy $H(\Delta_i \mid C_i)$ induces a larger distribution shift for the child, reducing the guaranteed reuse fraction. The coefficient η_i, the KL contraction coefficient of the increment-to-child channel in the sense of the strong data-processing inequality (Polyanskiy & Wu, 2017), captures how strongly the increment can influence the child's drafted prefix (Definition B.1).

4.2 Parent Surprisal: Online Proxy for Increment Entropy

We now relate the generally unobservable increment entropy to an online-computable quantity. Let $S_P(\Delta_i \mid P_i) := -\log p_P(\Delta_i \mid P_i)$ be the parent's self-surprisal of the increment, as in Section 3.2.

Assumption 4.2 (Surprisal calibration under a length cap)**.** For increments produced with $|\Delta_i| \leq M_{\max}$, we have

$$H(\Delta_i \mid C_i) = \mathbb{E}[S_C(\Delta_i \mid C_i) \mid C_i] \leq \mathbb{E}[S_P(\Delta_i \mid P_i) \mid C_i] + c(M_{\max}), \tag{9}$$

where $S_C(\Delta_i \mid C_i) := -\log p(\Delta_i \mid C_i)$ is the child-side surprisal and $c(M_{\max})$ is a slack term that, for a fixed child–parent pairing, depends only on the cap.

Assumption 4.2 is empirically supported in Figure 4(a), which shows that the parent self-surprisal effectively bounds the child-side surprisal in most cases.

4.3 A Single Knob: Surprisal Threshold τ

Combining Theorem 4.1 and Assumption 4.2 yields

$$\mathbb{E}[\mathrm{Acc}_i \mid C_i] \;\geq\; \left[1 - \sqrt{\tfrac{1}{2}\big(\eta_i\, b_i + \varepsilon_i\big)}\right]_+ , b_i \;:=\; \mathbb{E}\big[S_P(\Delta_i \mid P_i) \mid C_i\big], \varepsilon_i \;:=\; \varepsilon_i^{\mathrm{mc}} + \eta_i\, c(M_{\max}). \tag{10}$$

Table 1: Performance of different commit policies under different numbers of agents in three multi-agent *full*-mode workflows.

Dataset	Commit Policy	3 Agents				4 Agents				5 Agents				6 Agents			
		Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)	Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)	Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)	Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)
MMLU	Sequential	72.5	15.2	+0.0	1.0	75.5	25.1	+0.0	1.0	77.5	32.5	+0.0	1.0	73.5	37.9	+0.0	1.0
	Chunk-16	75.5	12.0	+3.0	1.27	75.0	16.7	-0.5	1.51	73.5	21.5	-4.0	1.51	71.5	26.0	-2.0	1.46
	Chunk-32	75.0	12.4	+2.5	1.22	72.5	16.9	-3.0	1.49	74.0	20.8	-3.5	1.57	73.5	25.3	+0.0	1.50
	Chunk-48	74.5	12.5	+2.0	1.21	76.5	17.0	+1.0	1.47	75.0	20.4	-2.5	1.59	75.5	24.0	+2.0	1.58
	Chunk-64	76.0	12.4	+3.5	1.22	76.5	17.5	+1.0	1.44	76.5	19.8	-1.0	1.64	73.0	23.6	-0.5	1.61
	Entropy	75.5	14.1	+3.0	1.07	75.5	24.9	+0.0	1.01	75.0	29.2	-2.5	1.11	74.0	35.6	+0.5	1.06
	Delimiter	77.5	12.2	+5.0	1.25	73.5	17.5	-2.0	1.44	75.5	20.8	-2.0	1.56	77.5	24.4	+4.0	1.55
	ACCORDION	**79.0**	**12.3**	**+6.5**	**1.24**	**76.0**	**17.1**	**+0.5**	**1.47**	**75.5**	**20.4**	**-2.0**	**1.59**	**77.0**	**24.3**	**+3.5**	**1.56**
GSM8K	Sequential	93.0	13.9	+0.0	1.0	92.5	23.8	+0.0	1.0	92.5	28.5	+0.0	1.0	93.0	34.8	+0.0	1.0
	Chunk-16	90.0	9.68	-3.0	1.44	92.0	15.9	-0.5	1.49	91.0	19.3	-1.5	1.48	93.0	30.3	+0.0	1.15
	Chunk-32	92.0	9.05	-1.0	1.54	91.5	16.3	-1.0	1.46	91.5	19.0	-1.0	1.50	91.5	26.1	-1.5	1.33
	Chunk-48	91.5	9.45	-1.5	1.47	90.5	15.8	-2.0	1.50	91.5	18.4	-1.0	1.55	91.0	22.7	-2.0	1.53
	Chunk-64	90.5	9.37	-2.5	1.48	89.5	15.0	-3.0	1.58	91.5	18.3	-1.0	1.55	91.0	22.9	-2.0	1.52
	Entropy	89.0	16.7	-4.0	0.83	92.0	30.4	-0.5	0.78	90.5	43.8	-2.0	0.65	92.5	95.4	-0.5	0.36
	Delimiter	91.5	9.26	-1.5	1.50	92.0	14.7	-0.5	1.61	92.0	18.3	-0.5	1.56	93.0	29.5	+0.0	1.18
	ACCORDION	**91.0**	**9.41**	**-2.0**	**1.48**	**91.5**	**15.5**	**-1.0**	**1.53**	**92.0**	**17.8**	**-0.5**	**1.60**	**92.0**	**22.4**	**-1.0**	**1.56**
HumanEval	Sequential	80.1	5.52	+0.0	1.0	77.6	9.0	+0.0	1.0	78.3	10.9	+0.0	1.0	77.6	15.2	+0.0	1.0
	Chunk-16	82.0	4.17	+1.9	1.32	81.4	5.43	+3.8	1.66	82.6	6.36	+4.3	1.71	82.6	8.15	+5.0	1.87
	Chunk-32	82.6	4.55	+2.5	1.21	81.4	5.93	+3.8	1.52	80.8	7.13	+2.5	1.53	80.1	9.10	+2.5	1.67
	Chunk-48	81.4	4.92	+1.3	1.12	77.6	7.72	+0.0	1.17	81.4	8.35	+3.1	1.31	79.5	10.2	+1.9	1.49
	Chunk-64	80.1	5.08	+0.0	1.09	76.4	7.60	-1.2	1.18	77.0	9.10	-1.3	1.20	80.1	11.2	+2.5	1.36
	Entropy	79.5	4.53	-0.6	1.22	75.8	8.67	-1.8	1.04	78.9	11.2	+0.6	0.97	77.0	15.9	-0.6	0.96
	Delimiter	76.4	5.09	-3.7	1.08	72.7	7.56	-4.9	1.19	77.6	7.76	-0.7	1.40	78.3	9.34	+0.7	1.63
	ACCORDION	**82.0**	**4.29**	**+1.9**	**1.29**	**83.2**	**5.59**	**+5.6**	**1.61**	**82.0**	**6.56**	**+3.7**	**1.66**	**82.6**	**8.11**	**+5.0**	**1.87**

Eq. (10) implies that, up to workload-dependent factors η_i and $c(M_{\max})$, larger surprisal in a committed increment lowers the child's guaranteed prefix acceptance, consistent with the larger child distribution shift in Figure 4(b); surprisal is thus a natural budget for scheduling context commits.

A single surprisal threshold τ therefore serves as the commit trigger, trading off update frequency against expected rollback.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Tasks and workflows. We evaluate ACCORDION on knowledge-intensive reasoning MMLU (Hendrycks et al., 2021), mathematical reasoning GSM8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2024), programming HumanEval (Chen et al., 2021), and open-ended research workflows ResearchBench (Yu et al., 2025). Following GPTSwarm (Zhuge et al., 2024) and Agent-Prune (Zhang et al., 2025a), we instantiate multi-agent workflows with few-shot prompting (Brown et al., 2020), chain-of-thought (Wei et al., 2022), tool calling (Qin et al., 2024), and structured outputs (Shorten et al., 2024). We consider five canonical DAG structures (chain, tree, star, grouped, and full; see Figure E.1); unless stated otherwise, the main experiments use *full*-mode workflows.

Models, baselines, and metrics. We use Qwen2.5-14B-Instruct (Yang et al., 2024) for MMLU/GSM8K and Qwen2.5-Coder-14B-Instruct (Hui et al., 2024) for HumanEval, with greedy decoding for all agents. Tables 1–4 use fixed 200-question subsets of the MMLU validation split and the GSM8K test set, and the complete 161 test samples of HumanEval (Cassano et al., 2023; pass@1 of the final agent's program); Appendix E.8 reports the mean and standard deviation over six paired runs. Baselines include sequential barrier execution; fixed-length commits (Chunk-16/32/48/64), which commit every 16/32/48/64 parent tokens; delimiter commits at each newline; and entropy commits, which commit once the entropy of the parent's top-5 next-token distribution reaches 1.5 nats. We report task accuracy/pass@1, end-to-end (E2E) latency, accuracy change relative to sequential execution, and speedup. Experiments are implemented on vLLM (Kwon et al., 2023) and run on NVIDIA A100 GPUs unless otherwise specified; default ACCORDION hyperparameters are $\tau = 10$ and $M_{\max} = 64$.

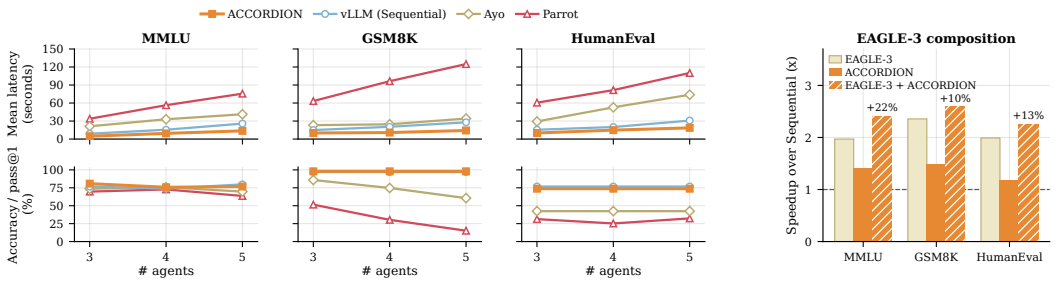


Figure 5: **Comparison with systems baselines and composition with EAGLE-3.** Left: Mean E2E latency and accuracy/pass@1 across three to five agents. Right: Speedup over Sequential in four-agent workflows, labeled with the gain over EAGLE-3 alone.

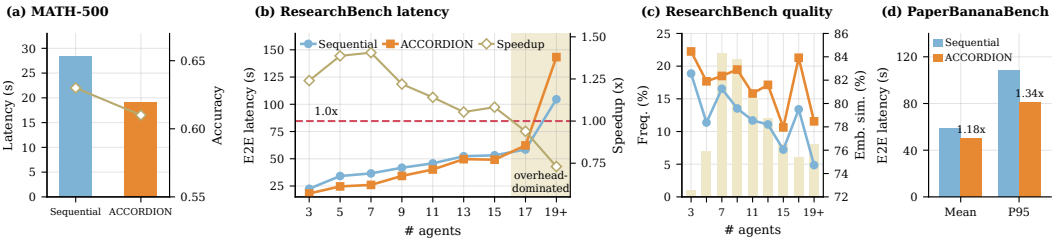


Figure 6: **Harder, open-ended, and dynamic workflows:** MATH-500, ResearchBench grouped by workflow size, and PaperBananaBench.

5.2 Main Results

Table 1 reports results on three primary benchmarks as the number of agents increases from three to six in *full*-mode workflows. ACCORDION consistently reduces latency while maintaining accuracy comparable to sequential execution. Its speedup grows with workflow size, peaking at $1.59\times$ on MMLU, $1.60\times$ on GSM8K, and $1.87\times$ on HumanEval.

Comparison with commit-policy baselines. Fixed-length streaming is sensitive to chunk size; delimiter triggers are brittle across task formats, especially for code generation; and entropy triggers are unstable on reasoning-heavy workloads. In contrast, ACCORDION uses surprisal as an information budget and therefore achieves a more stable latency–accuracy trade-off without hand-tuning a chunk length for each benchmark.

Comparison with systems baselines. Figure 5 (left) compares ACCORDION with Ayo (Tan et al., 2025) and Parrot (Lin et al., 2024a), which optimize scheduling, batching, asynchronous execution, or prefix sharing while preserving the parent–child barrier. ACCORDION is fastest in all nine settings, with $1.69\times$, $3.15\times$, and $6.66\times$ lower mean E2E latency than vLLM, Ayo, and Parrot, respectively, while preserving accuracy comparable to vLLM. This suggests that systems-level scheduling is complementary to, but not a substitute for, breaking generation-time dependency barriers. Our re-implementation of FlashAgents-style prefill overlap (Fang et al., 2026) stays within $1.03\times$ of Sequential, whereas ACCORDION reaches 1.08–$1.37\times$ (Appendix E.2). Applied on top of EAGLE-3 (Li et al., 2025), ACCORDION yields a further 1.10–$1.22\times$ speedup (Figure 5, right).

Generality across workflows and models. ACCORDION also improves latency across different dependency graphs (Table 4) and model sizes (Table 2). The gains are largest for deeper graphs such as full and chain, where children can exploit earlier partial context, and smaller for star-like graphs where the sequential baseline is already dominated by the longest parent branch. Scaling Qwen2.5 models from 7B to 32B preserves the same trend: larger models increase sequential latency, but ACCORDION continues to recover substantial overlap, e.g., a $1.54\times$ speedup at 32B on MMLU. Figure 6 further shows that the method extends to MATH-500 and to ResearchBench, using its 100-workflow HighImpactPaperBench subset (Yu et al., 2025); the latter also reveals a limitation, as

Table 2: Performance of ACCORDION under different model sizes in three four-agent *full*-mode workflows.

Dataset	Method	7B		14B		32B	
		Acc (%)	E2E (s)	Acc (%)	E2E (s)	Acc (%)	E2E (s)
MMLU	Sequential	68.0	14.0	75.5	25.1	81.5	53.5
	ACCORDION	**69.0**	**10.0**	**76.0**	**17.1**	**79.5**	**34.8**
GSM8K	Sequential	90.5	7.95	92.5	23.8	97.5	38.5
	ACCORDION	**90.0**	**5.88**	**91.5**	**15.5**	**97.0**	**26.9**
HumanEval	Sequential	80.8	7.17	77.6	9.00	83.9	20.2
	ACCORDION	**80.8**	**5.67**	**83.2**	**5.59**	**83.9**	**15.4**

Table 3: Hyperparameter analysis of the trigger policy and verification strategy.

Parent-side	**Threshold τ ($M_{\max} = 64$)**			
	5	10	15	20
Accuracy (%)	75.0	76.0	72.0	74.0
E2E Speedup	1.36	1.38	1.38	1.45
Parent-side	**Maximum Interval $M_{\max}$ ($\tau = 10$)**			
	16	32	48	64
Accuracy (%)	74.0	76.5	78.0	76.0
E2E Speedup	1.43	1.44	1.39	1.38
Child-side	**Verification Strategy**			
	Top-1		Top-5	Top-∞
Accuracy (%)	78.0		76.0	72.5
E2E Speedup	1.00		1.38	1.86

Table 4: Performance of ACCORDION across dependency-graph structures in four-agent workflows.

Dataset	Method	Chain				Full				Tree				Star				Grouped			
		Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)	Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)	Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)	Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)	Acc (%)	E2E (s)	ΔAcc (pp)	Speedup (×)
MMLU	Sequential	74.0	25.9	+0.0	1.00	75.5	25.1	+0.0	1.00	77.0	16.2	+0.0	1.00	75.5	16.0	+0.0	1.00	75.5	17.7	+0.0	1.00
	ACCORDION	**74.5**	**18.7**	**+0.5**	**1.39**	**76.0**	**17.1**	**+0.5**	**1.47**	**76.5**	**13.5**	**-0.5**	**1.20**	**76.0**	**12.8**	**+0.5**	**1.25**	**76.5**	**14.5**	**+1.0**	**1.22**
GSM8K	Sequential	92.5	22.8	+0.0	1.00	92.5	23.8	+0.0	1.00	93.0	11.6	+0.0	1.00	93.0	10.2	+0.0	1.00	93.5	14.7	+0.0	1.00
	ACCORDION	**93.0**	**15.3**	**+0.5**	**1.49**	**91.5**	**15.5**	**-1.0**	**1.53**	**94.0**	**10.7**	**+1.0**	**1.08**	**93.5**	**10.1**	**+0.5**	**1.01**	**92.5**	**11.3**	**-1.0**	**1.30**
HumanEval	Sequential	81.4	8.97	+0.0	1.00	77.6	9.0	+0.0	1.00	84.5	7.04	+0.0	1.00	82.0	6.21	+0.0	1.00	88.2	6.71	+0.0	1.00
	ACCORDION	**81.4**	**7.05**	**+0.0**	**1.27**	**83.2**	**5.59**	**+5.6**	**1.61**	**81.4**	**6.36**	**-3.1**	**1.11**	**81.4**	**4.84**	**-0.6**	**1.28**	**83.2**	**5.44**	**-5.0**	**1.23**

workflows with very high fan-in can become dominated by synchronization and KV-cache recovery. On PaperBananaBench (Zhu et al., 2026) with OpenManus (Liang et al., 2025), where a planner builds heterogeneous LLM, VLM, and plotting-tool workflows at runtime, ACCORDION reduces mean and P95 latency by 15.2% and 25.4% (Appendix E.3).

5.3 ABLATION AND BEHAVIORAL ANALYSIS

We ablate the surprisal threshold, maximum interval, and verification rule on four-agent MMLU workflows in Table 3. Overall, $\tau = 10$ offers a strong reuse–update trade-off: lower thresholds over-synchronize, while higher thresholds risk stale-context rollbacks. On the child side, Top-5 verification is the most practical setting, balancing accuracy and speed better than conservative Top-1 and overly permissive Top-∞.

Effect of topology. Gains grow with dependency depth (Table 4): chain and full graphs serialize all agents and reach 1.27–1.61×, while tree, star, and grouped graphs run parents in parallel and reach 1.01–1.30×, as multi-parent sinks add KV-cache rebuilds (Appendix C.4).

6 CONCLUSION

Hierarchical multi-agent workflows are often bottlenecked by dependency barriers that serialize computation along parent–child chains. ACCORDION alleviates this bottleneck with context-level speculative decoding: parents stream incremental prompt commits while children decode under partial context, verifying and rolling back only when needed. Our information-theoretic analysis shows that prefix reuse under evolving context is governed by conditional entropy rather than token count, motivating a surprisal-budgeted commit policy that uses parent self-surprisal to control reuse and rollback with a single threshold. Overall, these results show the value of treating parent messages as incrementally useful rather than atomic. ACCORDION targets latency-critical serving at low-to-moderate load (Appendix E.4); it runs inside the serving stack, where logits and KV caches are accessible, and executes side-effecting tool calls after their context is committed and verified. Future directions include learned commit policies and richer reconciliation mechanisms for general multi-agent dependency graphs.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, to design and refine experiments, and to interpret results: LLM-based coding assistants helped write and debug parts of the experiment and analysis code, provided feedback on the experimental design, and helped analyze experiment logs, compute summary statistics, and explain observed behaviors. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate or prove mathematical claims, propose or refine research hypotheses, or clean or reformat datasets. Additionally, we used generative AI tools to aid and polish writing (editing for readability, condensing, and restructuring text), for retrieval and discovery (identifying related work and verifying and formatting bibliographic metadata), to create or edit figures from our experimental data, and to fix LaTeX build issues. The large language models evaluated in our experiments (the Qwen2.5 family) are objects of study rather than research or writing assistants. The authors reviewed and tested all AI-written code, checked every AI-suggested edit, verified every AI-suggested reference against its published version, and checked the reported results against the raw experiment outputs. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies inference-time scheduling for multi-agent LLM workflows. All experiments use publicly available benchmarks (MMLU, GSM8K, MATH-500, the MultiPL-E version of HumanEval, ResearchBench, PaperBananaBench, and CriticBench) and open-weight models from the Qwen2.5 family; the study involves no human subjects, crowdsourcing, or personal data, and it introduces no new datasets. ACCORDION changes when downstream agents begin decoding, while verification keeps their outputs consistent with the updated context, so the capabilities and risks of the underlying models, including biased or incorrect outputs, carry over unchanged. Side-effecting tool calls execute only after their context has been committed and verified, so speculative drafts cannot trigger external actions.

REPRODUCIBILITY STATEMENT

Section 3 and Algorithms 1 and 2 (Appendix C) specify the surprisal-triggered commit rule and the verification procedure, and Appendix C.3 describes KV-cache handling under prompt updates. Appendix B states the assumptions behind Theorem 4.1 and gives complete proofs. Section 5 lists the models, datasets and their sizes, baselines, decoding settings, and default hyperparameters; Appendix C.5 lists the hardware used for each study; Appendix D reproduces the chat template and every agent prompt verbatim; and Appendix E describes the setups of the additional-baseline, PaperBananaBench, throughput, and threshold-adaptation studies.

REFERENCES

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. In *International Conference on Learning Representations*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

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 *International Conference on Machine Learning*, pp. 5209–5235, 2024.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023. doi: 10.1109/TSE.2023.3267446.

Harrison Chase. LangChain. https://github.com/langchain-ai/langchain, October 2022. Accessed: 2025-12-11.

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.

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. *arXiv preprint arXiv:2107.03374*, 2021.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R. Glass, and Pengcheng He. DoLa: Decoding by contrasting layers improves factuality in large language models. In *International Conference on Learning Representations*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Taosong Fang, Zhen Zheng, Zhengzhao Ma, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. FlashAgents: Accelerating multi-agent LLM systems via streaming prefill overlap. In *Proceedings of Machine Learning and Systems*, volume 8, pp. 318–331, 2026.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-cache: Direct semantic communication between large language models. In *International Conference on Learning Representations*, 2026.

Yingqiang Ge, Wenyue Hua, Kai Mei, Jianchao Ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. OpenAGI: When LLM meets domain experts. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2023.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 8048–8057, 2024. doi: 10.24963/ijcai.2024/890.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

LangChain. LangGraph. https://github.com/langchain-ai/langgraph, 2023. Accessed: 2025-12-11.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286, 2023.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, 2023.

Shufan Li and Aditya Grover. PredGen: Accelerated inference of large language models through input-time speculation for real-time speech interaction. In *Second Conference on Language Modeling (COLM)*, 2025.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. In *International Conference on Learning Representations*, 2026.

Xinbing Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, Xiao Tang, Bang Liu, Yuyu Luo, and Chenglin Wu. OpenManus: An open-source framework for building general AI agents. Zenodo, 2025.

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.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, 2024a.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. CriticBench: Benchmarking LLMs for critique-correct reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1552–1587, 2024b.

OpenAI. Deep research system card. System card, OpenAI, February 2025. URL `https://cdn.openai.com/deep-research-system-card.pdf`. Accessed: 2025-12-11.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. doi: 10.1145/3586183.3606763.

Yury Polyanskiy and Yihong Wu. Strong data-processing inequalities for channels and Bayesian networks. In *Convexity and Concentration*, volume 161 of *The IMA Volumes in Mathematics and its Applications*, pp. 211–249. Springer, 2017.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024.

Keshav Santhanam, Deepti Raghavan, Muhammad Shahir Rahman, Thejas Venkatesh, Neha Kunjal, Pratiksha Thaker, Philip Levis, and Matei Zaharia. ALTO: An efficient network orchestrator for compound AI systems. In *Proceedings of the 4th Workshop on Machine Learning and Systems*, pp. 117–125, 2024. doi: 10.1145/3642970.3655844.

Connor Shorten, Charles Pierse, Thomas Benjamin Smith, Erika Cardenas, Akanksha Sharma, John Trengrove, and Bob van Luijt. StructuredRAG: JSON response formatting with large language models. *arXiv preprint arXiv:2408.11061*, 2024.

Significant Gravitas. AutoGPT. `https://github.com/Significant-Gravitas/AutoGPT`, 2023. Accessed: 2025-12-11.

Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. Towards end-to-end optimization of LLM-based applications with Ayo. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1302–1316, 2025. doi: 10.1145/3676641.3716278.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of LLMs. *arXiv preprint arXiv:2501.06322*, 2025.

Jikai Wang, Zhenxu Tian, Juntao Li, Qingrong Xia, Xinyu Duan, Zhefeng Wang, Baoxing Huai, and Min Zhang. Alignment-augmented speculative decoding with alignment sampling and conditional verification. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6751–6763, 2025a. doi: 10.18653/v1/2025.emnlp-main.343.

Junlin Wang, Jue Wang, Zhen Xu, Ben Athiwaratkun, Bhuwan Dhingra, Ce Zhang, and James Zou. Staircase streaming for low-latency multi-agent inference. *arXiv preprint arXiv:2510.05059*, 2025b.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025a. doi: 10.1145/3715754.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *International Conference on Learning Representations*, 2025b.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Zhen Yang, Xiaogang Xu, Wen Wang, Cong Chen, Xander Xu, and Ying-Cong Chen. Streaming communication in multi-agent reasoning. *arXiv preprint arXiv:2606.05158*, 2026.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, et al. KVCOMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems*, 2025.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster AI agents. In *International Conference on Learning Representations*, 2026.

Haofei Yu, Zhaochen Hong, Zirui Cheng, Kunlun Zhu, Keyang Xuan, Jinwei Yao, Tao Feng, and Jiaxuan You. ResearchTown: Simulator of human research community. In *International Conference on Machine Learning*, pp. 73051–73096, 2025.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *International Conference on Learning Representations*, 2025a.

Ziyin Zhang, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Rui Wang, and Zhaopeng Tu. Draft model knows when to stop: Self-verification speculative decoding for long-form generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 16685–16697, 2025b. doi: 10.18653/v1/2025.emnlp-main.844.

Dawei Zhu, Rui Meng, Yale Song, Xiyu Wei, Sujian Li, Tomas Pfister, and Jinsung Yoon. PaperBanana: Automating academic illustration for AI scientists. In *International Conference on Machine Learning*, 2026.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *International Conference on Machine Learning*, 2024.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems. In *International Conference on Machine Learning*, 2026.

APPENDIX

The appendix is organized as follows.

- Sec. A: Related Work.
- Sec. B: Theoretical Analysis, including the definition of η_i, the proofs for Section 4, and the surprisal experiment behind Figure 4(b).
- Sec. C: Method and System Details, including the algorithms, verification correctness, KV-cache reuse, and overhead analysis.
- Sec. D: Agent Prompts for MMLU, GSM8K, and HumanEval.
- Sec. E: Additional Experimental Results on dependency-graph topologies, additional baselines, dynamic workflows, throughput under load, threshold adaptation, verification and output-length statistics, and execution dynamics.

A RELATED WORK

Multi-agent LLM systems and workflow scheduling. Recent multi-agent LLM systems operationalize complex reasoning by composing specialized agents (e.g., planner–executor, manager–worker, critique–revise) into explicit dependency graphs, where intermediate natural-language messages are passed along edges and assembled into downstream prompts (Wu et al., 2024; Li et al., 2023; Chen et al., 2024; Ge et al., 2023; Park et al., 2023; Yao et al., 2023; Guo et al., 2024; Tran et al., 2025; Zhuge et al., 2024; Zeng et al., 2024; Xia et al., 2025a). While this modularity improves controllability and often yields better task performance, most deployed stacks enforce *dependency barriers*: a child agent starts only after the relevant parent messages are fully materialized. A complementary line of work optimizes *what* is communicated or *how* agents are connected by searching, learning, or pruning the workflow graph (Zhuge et al., 2024; Zhang et al., 2025a; Li et al., 2026; Santhanam et al., 2024). ACCORDION targets a different axis and composes with these graph-level methods: given a fixed dependency graph and fixed agent prompts, we ask *when* information should be revealed across an edge to overlap decoding without incurring excessive rollback waste.

Speculative decoding and verification. Speculative decoding accelerates autoregressive generation by drafting tokens under an approximate process and verifying them under a target process, rolling back at the first mismatch (Leviathan et al., 2023; Chen et al., 2023; Cai et al., 2024; Xia et al., 2025b). Subsequent work explores practical verification rules and stability–efficiency trade-offs, including judge-based verification and alignment-augmented verification (Bachmann et al., 2025; Wang et al., 2025a; Zhang et al., 2025b). These methods primarily operate in the *model-level* regime: the prompt is fixed while the draft and target distributions differ. DoLa (Chuang et al., 2024) contrasts next-token distributions from different layers of one model to improve factuality, whereas ACCORDION's verifier compares the child's distributions under stale and updated contexts to decide which drafted tokens to reuse. ACCORDION reinterprets speculation through a *context-level* lens tailored to workflows: the child model is fixed, but its conditioning prompt evolves as the parent streams incremental commits. This shift introduces a distinct control problem—prompt updates can invalidate already-computed child tokens—and calls for scheduling policies that explicitly regulate acceptance and rollback under time-varying context, rather than under a fixed prompt.

Streaming and speculative execution across agents. Several recent systems also start downstream work before upstream generation completes, but they target different workloads. StreamMA forwards each reasoning step to downstream agents (Yang et al., 2026), and Staircase Streaming lets a mixture-of-agents aggregator generate from fixed-size chunks of proposer outputs (Wang et al., 2025b); both redesign the communication protocol of a specific pipeline so that downstream agents act on partial content by design. PredGen drafts a response while a user is still speaking and verifies it as more speech arrives, targeting streaming user input to a single model (Li & Grover, 2025), and Speculative Actions predicts upcoming agent actions to hide environment latency in sequential agent trajectories (Ye et al., 2026). FlashAgents shares our setting and overlaps the child's prefill with parent decoding (Fang et al., 2026); Appendix E.2 compares against it. ACCORDION targets existing dependency-constrained workflows and leaves their communication protocol unchanged: the workflow graph, prompt templates, and message formats stay fixed, and only the time at which parent content becomes visible changes. Within this protocol, ACCORDION makes the dependency edge the

unit of speculation: a child drafts under partial parent context and reconciles each draft with later parent content, which helps speculate the full-context decoding accurately, while the parent times each commit by the self-surprisal that our analysis links to the child's expected prefix reuse. The resulting parallelism is semantics-aware and adapts to each context dependency, from a single edge to multi-parent DAGs in which a child reconciles its draft whenever any of its parents commits.

Streaming prompting, incremental generation, and uncertainty-driven synchronization. Streaming partial generations is ubiquitous in interactive LLM applications and agent frameworks (Chase, 2022; LangChain, 2023; Wu et al., 2024), but deciding *update boundaries* is typically handled by fixed cadences (e.g., every k tokens), surface-form delimiters (e.g., punctuation/newlines), or heuristic uncertainty triggers (e.g., entropy thresholds). Such rules conflate *length* with *information*: equally long segments can induce vastly different distribution shifts downstream, leading to either unnecessary synchronization overhead or large rollbacks. ACCORDION makes this connection explicit: we characterize how *prefix reuse* after a commit is controlled by the update's conditional entropy via an information-theoretic acceptance lower bound (Polyanskiy & Wu, 2017), and then derive an online proxy in terms of *parent self-surprisal* under a length cap. This yields a surprisal-budgeted commit rule with a single interpretable knob that directly governs the expected acceptance–overhead trade-off.

B THEORETICAL ANALYSIS

This section defines the coefficient η_i, proves the results of Section 4, and reports the experiment behind Figure 4(b).

B.1 INCREMENT-TO-CHILD CHANNEL

Fix an update index i. Let C_i be the child context *before* the parent increment, and let the parent reveal Δ_i, yielding the updated context $C_i^+ = \Phi(C_i, \Delta_i)$. Before receiving Δ_i, the child drafts a length-T_i prefix under C_i; after receiving Δ_i, it verifies under C_i^+ and keeps the longest accepted prefix, with acceptance $\mathrm{Acc}_i := L_i/T_i$.

For a fixed context value c, define the *increment-to-child* channel by the updated-prefix law:

$$W_{i,c}(y_{1:T_i} \mid \Delta) \;:=\; p(y_{1:T_i} \mid C_i = c, \Delta), \tag{B.1}$$

i.e., the child's length-T_i prefix law after incorporating Δ. We also write the stale (pre-update) drafted-prefix law as

$$Q_{i,c}(y_{1:T_i}) \;:=\; p(y_{1:T_i} \mid C_i = c).$$

When conditioning on $(C_i = c, \Delta_i = \Delta)$, the post-update target law is $P_{i,c,\Delta}(\cdot) := W_{i,c}(\cdot \mid \Delta)$.

B.2 KL CONTRACTION COEFFICIENT AND THE STRONG DATA-PROCESSING INEQUALITY

Definition B.1 (KL contraction coefficient)**.** The KL contraction coefficient of a channel $W_{i,c}$ is

$$\eta_{\mathrm{KL}}(W_{i,c}) \;:=\; \sup_{P \neq Q} \frac{D_{\mathrm{KL}}(PW_{i,c} \,\|\, QW_{i,c})}{D_{\mathrm{KL}}(P \,\|\, Q)} \;\in\; [0,1]. \tag{B.2}$$

We define the context-conditioned coefficient $\eta_i(c) := \eta_{\mathrm{KL}}(W_{i,c}) \in [0,1]$ and write $\eta_i := \eta_i(C_i)$ when conditioning on the random context C_i.

We use the following standard consequence of the strong data-processing inequality (SDPI) for KL-contractive channels (e.g., Polyanskiy & Wu, 2017): for each c,

$$I(Y_{1:T_i}; \Delta_i \mid C_i = c) \;\le\; \eta_{\mathrm{KL}}(W_{i,c})\, H(\Delta_i \mid C_i = c), \tag{B.3}$$

where $Y_{1:T_i} \sim W_{i,c}(\cdot \mid \Delta_i)$ is the child's updated T_i-prefix random variable.

B.3 PROOF OF THEOREM 4.1

We state and prove the acceptance guarantee underlying Theorem 4.1.

Verification idealization (prefix-maximal coupling). Conditioned on $(C_i = c, \Delta_i = \Delta)$, the verifier induces a coupling between a stale draft prefix $\widetilde{Y}_{1:T_i} \sim Q_{i,c}$ and a target prefix $Y_{1:T_i} \sim P_{i,c,\Delta}$ that maximizes prefix agreement: for every $t \le T_i$,

$$\mathbb{P}\Big(Y_{1:t} = \widetilde{Y}_{1:t} \mid C_i = c, \Delta_i = \Delta\Big) = \sum_{y_{1:t}} \min\Big\{P^{(t)}_{i,c,\Delta}(y_{1:t}),\, Q^{(t)}_{i,c}(y_{1:t})\Big\}, \tag{B.4}$$

where $P^{(t)}_{i,c,\Delta}$ and $Q^{(t)}_{i,c}$ are the length-t prefix marginals.

From prefix overlap to a KL bound. Let L_i be the accepted prefix length, and define survival indicators $A_{i,t} := \mathbf{1}\{L_i \ge t\}$ so that $\mathrm{Acc}_i = \frac{1}{T_i}\sum_{t=1}^{T_i} A_{i,t}$. Under the prefix-maximal coupling in Eq. (B.4),

$$\mathbb{P}(L_i \ge t \mid c, \Delta) = 1 - d_{\mathrm{TV}}\Big(P^{(t)}_{i,c,\Delta}, Q^{(t)}_{i,c}\Big).$$

Pinsker's inequality gives

$$1 - d_{\mathrm{TV}}(P, Q) \;\ge\; \Big[1 - \sqrt{\tfrac{1}{2} D_{\mathrm{KL}}(P\|Q)}\Big]_+.$$

Using (i) this inequality for each t, (ii) the identity $\mathbb{E}[\mathrm{Acc}_i \mid c, \Delta] = \frac{1}{T_i}\sum_{t=1}^{T_i} \mathbb{P}(L_i \ge t \mid c, \Delta)$, and (iii) KL monotonicity under marginalization $D_{\mathrm{KL}}(P^{(t)}\|Q^{(t)}) \le D_{\mathrm{KL}}(P^{(T_i)}\|Q^{(T_i)})$, we obtain the bound

$$\mathbb{E}[\mathrm{Acc}_i \mid C_i = c, \Delta_i = \Delta] \;\ge\; \Big[1 - \sqrt{\tfrac{1}{2} D_{\mathrm{KL}}(P_{i,c,\Delta} \,\|\, Q_{i,c})}\Big]_+. \tag{B.5}$$

Averaging over increments and the mutual-information-plus-mismatch decomposition. Let $\mu(\cdot \mid c)$ denote the conditional law of Δ_i given $C_i = c$, and define the mixture marginal

$$\bar{Q}_{i,c}(\cdot) \;:=\; \mathbb{E}_{\Delta\sim\mu(\cdot|c)}[P_{i,c,\Delta}(\cdot)]\,.$$

Taking $\mathbb{E}_{\Delta|c}$ of Eq. (B.5) and applying Jensen's inequality (to the convex function $x \mapsto [1 - \sqrt{x/2}]_+$) yields

$$\mathbb{E}[\mathrm{Acc}_i \mid C_i = c] \;\ge\; \Big[1 - \sqrt{\tfrac{1}{2}\,\mathbb{E}_{\Delta|c}\, D_{\mathrm{KL}}(P_{i,c,\Delta} \,\|\, Q_{i,c})}\Big]_+. \tag{B.6}$$

A standard decomposition gives

$$\mathbb{E}_{\Delta|c}\, D_{\mathrm{KL}}(P_{i,c,\Delta} \,\|\, Q_{i,c}) = I(Y_{1:T_i}; \Delta_i \mid C_i = c) + D_{\mathrm{KL}}(\bar{Q}_{i,c} \,\|\, Q_{i,c}), \tag{B.7}$$

where $Y_{1:T_i} \sim P_{i,c,\Delta_i}$.

Define the marginal mismatch term

$$\varepsilon^{\mathrm{mc}}_i(c) \;:=\; D_{\mathrm{KL}}(\bar{Q}_{i,c} \,\|\, Q_{i,c}) \;\ge\; 0, \tag{B.8}$$

and write $\varepsilon^{\mathrm{mc}}_i := \varepsilon^{\mathrm{mc}}_i(C_i)$.

SDPI closes the chain. Applying the SDPI in Eq. (B.3) with $\eta_i(c) = \eta_{\mathrm{KL}}(W_{i,c})$ and combining Eqs. (B.6)–(B.8) yields

$$\mathbb{E}[\mathrm{Acc}_i \mid C_i = c] \;\ge\; \left[1 - \sqrt{\tfrac{1}{2}\big(\eta_i(c)\, H(\Delta_i \mid C_i = c) + \varepsilon^{\mathrm{mc}}_i(c)\big)}\right]_+,$$

which is Eq. (8) (and hence Theorem 4.1).

B.4 Surprisal Calibration

Recall the child-side surprisal $S_C(\Delta_i \mid C_i) := -\log p(\Delta_i \mid C_i)$ and the parent self-surprisal $S_P(\Delta_i \mid P_i) := -\log p_P(\Delta_i \mid P_i)$.

Justification of Assumption 4.2. For any reals a, b, $a \le b + (a-b)_+$. Applying this with $a = S_C(\Delta_i \mid C_i)$ and $b = S_P(\Delta_i \mid P_i)$ and taking $\mathbb{E}[\cdot \mid C_i]$ gives

$$\mathbb{E}[S_C(\Delta_i \mid C_i) \mid C_i] \le \mathbb{E}[S_P(\Delta_i \mid P_i) \mid C_i] + \mathbb{E}\Big[\big(S_C(\Delta_i \mid C_i) - S_P(\Delta_i \mid P_i)\big)_+ \mid C_i\Big].$$

Under a fixed streaming interface with a length cap $|\Delta_i| \le M_{\max}$, we upper bound the last term by a cap-dependent slack $c(M_{\max})$, yielding Eq. (9).

B.5 Surprisal-Based Acceptance Bound

Starting from Eq. (8) and substituting the calibration bound in Eq. (9) into $H(\Delta_i \mid C_i)$ gives

$$\mathbb{E}[\mathrm{Acc}_i \mid C_i] \;\ge\; \Big[1 - \sqrt{\tfrac{1}{2}\big(\eta_i\, b_i + \varepsilon_i\big)}\,\Big]_+,$$
$$b_i := \mathbb{E}\big[S_P(\Delta_i \mid P_i) \mid C_i\big], \qquad \varepsilon_i := \varepsilon_i^{\mathrm{mc}} + \eta_i\, c(M_{\max}),$$

which is exactly Eq. (10).

B.6 Parent Surprisal and Child Distribution Shift

Figure 4(b) measures how strongly a parent commit changes the child's next-token distribution. We use Qwen2.5-14B-Instruct as both the parent (Critic) and the child (Reviser) on 800 CriticBench (Lin et al., 2024b) questions spanning commonsense, mathematical, symbolic, and coding tasks, with greedy decoding on NVIDIA H200 NVL GPUs. For each question, the Critic commits a 16-token correction, and the Reviser drafts a continuation under its stale context. We insert the same correction into two matched Reviser prompts: in the *relevant* prompt the correction supplies new information, and in the *redundant* prompt that information is already present. For each prompt, we compute the full-vocabulary $D_{\mathrm{KL}}(P_{\text{updated}} \,\|\, P_{\text{stale}})$ between the Reviser's updated and stale next-token distributions, averaged over the first 32 positions of the stale draft; this is the stale-versus-updated divergence that the acceptance bound of Theorem 4.1 controls. The child distribution shift is the relevant-minus-redundant difference, which isolates the effect of the new information from the effect of a longer prompt. Because task sources differ in their typical surprisal and shift levels, both quantities are converted to percentiles within each question source. Figure 4(b) groups the 800 commits into surprisal quintiles and reports the mean shift percentile with 95% intervals from 10,000 question-source-stratified bootstrap resamples; the dashed line at 0.5 marks no association. The shift rises by 9.1 percentile points from the lowest to the highest quintile (95% CI $[2.1, 15.3]$), and a rank regression that controls for question source yields a positive coefficient of 0.123 (95% CI $[0.051, 0.194]$).

C Method and System Details

C.1 Algorithms

Algorithm 1 describes parent-side surprisal-triggered commit. Algorithm 2 describes child-side speculative decoding with verification and rollback.

Algorithm 2 ACCORDION Child Agent: Context-Level Speculative Decoding

Require: initial context C; top-k parameter k (we use k=5)
$Y \leftarrow [\,]$ {drafted child tokens}
while parent or child decoding not finished **do**
  if NEWCOMMITARRIVES(Δ) **then**
    $C \leftarrow \Phi(C, \Delta)$
    $L \leftarrow \text{VERIFY}(C, Y, k)$
    $Y \leftarrow Y_{1:L}$ {rollback}
  else
    if EOS $\notin Y$ **then**
      decode one token; append it to Y
    end if
  end if
end while
return Y

Algorithm 1 ACCORDION Parent Agent: Surprisal-Triggered Commit and Streaming

Require: surprisal threshold τ; maximum interval $M_{\max}$
$S \leftarrow 0$; $\Delta \leftarrow [\,]$; $Z \leftarrow [\,]$ {S: cumulative surprisal, Δ: uncommitted segment, Z: generated tokens}
while parent decoding not finished **do**
  generate next parent token z and probability $p_P(z \mid P)$
  append z to Z and to Δ
  $S \leftarrow S - \log p_P(z \mid P)$
  if $S \geq \tau$ **or** $|\Delta| \geq M_{\max}$ **then**
    COMMIT(Δ)
    $S \leftarrow 0$; $\Delta \leftarrow [\,]$
  end if
end while
COMMIT(Δ) {final commit}
return Z

C.2 CORRECTNESS OF VERIFICATION

Distribution-preserving verifier. In principle, ACCORDION can preserve the exact output distribution of barrier execution by adopting the same draft–verify construction as model-level speculative decoding, but with a different interpretation of *draft* versus *target* distributions.

Fix an update index i. The child drafts tokens under the stale context C_i, which induces a next-token distribution $q_t(\cdot) := p(\cdot \mid C_i, y_{<t})$, while verification is performed under the updated context C_i^+ with $p_t(\cdot) := p(\cdot \mid C_i^+, y_{<t})$. A maximal-coupling or rejection-sampling verifier, as used in classical speculative decoding (Leviathan et al., 2023; Chen et al., 2023), guarantees that the accepted prefix together with the resampled suffix is exactly distributed as decoding directly under C_i^+.

Relation to prior verification heuristics. Recent extensions of speculative decoding have explored a variety of *practical* verification strategies that relax exact distributional equivalence in exchange for improved stability or efficiency, including top-k or rank-based acceptance rules. Such heuristics have been adopted as baselines or ablations in multiple prior works, including Judge Decoding (Bachmann et al., 2025) and alignment-augmented speculative decoding (Wang et al., 2025a). These studies empirically demonstrate that top-k verification provides a robust and low-overhead approximation to exact verification, motivating its use as a practical baseline in our setting as well.

The key distinction is that ACCORDION applies verification under a *changing conditioning context* (from C_i to C_i^+), rather than under a fixed prompt with different draft and target models.

C.3 KV-CACHE REUSE UNDER PROMPT UPDATES

Unlike standard speculative decoding, where the context is fixed, ACCORDION must handle prompt edits caused by incremental parent commits. In many multi-agent prompt layouts, each parent writes into a dedicated block in the child's prompt; thus a new increment may be inserted at a position that is *not* the end of the prompt. As a result, cached KV states for already drafted child tokens may become invalid.

We therefore reuse the KV cache *only* for the shared (unchanged) prefix of the child's full prompt across updates. During VERIFY, we re-score the drafted prefix $\hat{Y}^{(i)}_{1:T_i}$ under the updated context C_i^+, which rebuilds the KV cache for the correct post-update prompt; decoding then continues from the cache entries of the accepted prefix $\hat{Y}^{(i)}_{1:L_i}$. This makes verification the natural synchronization point for correctness and cache repair.

C.4 OVERHEADS AND SCALING WITH THE TRIGGER THRESHOLD

Rollback waste (child). Let updates occur for $i = 1, \ldots, U$. Let T_i be the drafted prefix length immediately before update i, and L_i the accepted prefix length after verification. The number of discarded draft tokens at update i is $T_i - L_i$. We measure rollback-induced wasted computation by the total discarded tokens:

$$\begin{aligned} N_{\text{discard}} &:= \sum_{i=1}^{U} (T_i - L_i) \\ &= \sum_{i=1}^{U} (1 - \text{Acc}_i)\, T_i, \end{aligned} \tag{C.1}$$

where Acc_i is defined in Eq. (5). Higher prefix acceptance directly reduces rollback waste.

Update frequency (parent). Let the parent stream have N_P tokens and total cumulative surprisal $S_{\text{tot}} := \sum_j -\log p_P(z_j \mid P)$. Each committed increment except possibly the last either has surprisal at least τ or reaches the maximum interval $M_{\max}$. Therefore the number of commits satisfies

$$U \;\le\; 1 + \frac{S_{\text{tot}}}{\tau} + \frac{N_P}{M_{\max}}. \tag{C.2}$$

Smaller τ increases the number of verification events (and cache rebuilds) but typically increases acceptance, reducing N_{discard}; larger τ reduces verification frequency at the risk of larger rollbacks.

End-to-end latency decomposition. Let T_{seq} and T_{par} denote the sequential and ACCORDION end-to-end latencies. We decompose the realized ACCORDION latency as

$$T_{\text{par}} = T_{\text{seq}} - G_{\parallel} + O_{\text{ver}} + O_{\text{KV}} + O_{\text{rb}} + \xi, \tag{C.3}$$

where $G_{\parallel}$ is recovered parallelism from overlapping parent and child decoding, O_{ver} is verification compute, O_{KV} is the KV-cache rebuild stall after prompt refreshes, O_{rb} is rollback-induced wasted computation, and ξ captures small runtime noise. Equivalently, ACCORDION yields speedup whenever

$$G_{\parallel} > O_{\text{ver}} + O_{\text{KV}} + O_{\text{rb}} + \xi. \tag{C.4}$$

Measured overhead. Figure C.1 measures these terms in MMLU star-mode workflows, where more parents feed into one sink agent as the number of agents grows. Verification compute remains small, but both verification and KV-cache rebuild increase with the number of parents, and the rebuild term becomes the dominant overhead at larger agent counts. This explains why high-fan-in open-ended workflows such as ResearchBench can preserve output quality while seeing marginal or negative speedup: repeated prompt refreshes and cache recovery can absorb the overlap unlocked by context-level speculation.

C.5 HARDWARE

Unless noted otherwise, experiments use NVIDIA A100-SXM4-80GB GPUs. MATH-500 and ResearchBench use NVIDIA H100 NVL GPUs, and PaperBananaBench, the CriticBench analysis in Appendix B.6, the throughput study in Appendix E.4, and the threshold-adaptation study in Appendix E.5 use NVIDIA H200 NVL GPUs.

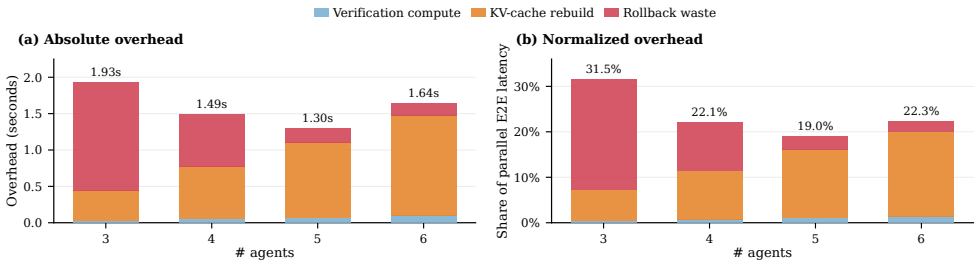


Figure C.1: **Latency overhead in MMLU star-mode workflows.**

D AGENT PROMPTS

We follow the exact prompt template proposed in Qwen2.5 (Yang et al., 2024). The template is as follows:

Qwen2.5 Chat Template

```
<|im_start|>system
{system_prompt}
```

 # Tools

You may call one or more functions to assist with the user query.
You are provided with function signatures within `<tools></tools>` XML tags:

```
<tools>
{tool_1_json}
{tool_2_json}
...
</tools>
```

For each function call, return a JSON object with function name and arguments within `<tool_call></tool_call>` XML tags:

```
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
```

```
<|im_end|>
<|im_start|>{parent_1_name}
{parent_1_message}
<|im_end|>
<|im_start|>{parent_2_name}
{parent_2_message}
<|im_end|>
...
<|im_start|>user
{user_message}
<|im_end|>
<|im_start|>assistant
<tool_call>
{"name":  "{tool_name}", "arguments":  {...}}
</tool_call>
<|im_end|>
<|im_start|>user
<tool_response>
{tool_output}
</tool_response>
<|im_end|>
<|im_start|>assistant
{final_assistant_response}
<|im_end|>
```

We present the detailed system prompts and tool registries for each agent deployed in our experiments, following the settings of GPTSwarm (Zhuge et al., 2024) and AgentPrune (Zhang et al., 2025a). The prompts are reproduced verbatim from our implementation, including their original wording.

D.1 MMLU

We cycle through the following roles for multi-agent reasoning:

- Knowledgeable Expert
- Wiki Searcher (with access to the wiki search tool)
- Critic
- Mathematician
- FinalRefer

Below, we show the system prompt for each role:

Knowledgeable Expert
You are a knowledgeable expert in question answering.
Please give at most six key entities that need to be searched in wikipedia to solve the problem.
Key entities that need to be searched are included between two '@' when output.
For example: @catfish effect@, @broken window effect@, @Shakespeare@.
Search every entity separately.
If there is no entity in the question that needs to be searched in Wikipedia, you don't have to provide it.

Wiki Searcher
You will be given a question and a wikipedia overview of the key entities within it.
Please refer to them step by step to give your answer.

Critic
You are an excellent critic.
Please point out potential issues in other agent's analysis point by point.
Start your answer with "To evaluate the potential issues in the provided analysis, let's break down each option and the reasoning behind it:"

Mathematician
You are a mathematician who is good at math games, arithmetic calculation, and long-term planning.

FinalRefer
You are the top decision-maker and are good at analyzing and summarizing other people's opinions, finding errors and giving final answers.

D.2 GSM8K

We cycle through the following roles:

- Math Solver
- Mathematical Analyst
- Programming Expert (with access to the python execution tool)
- Inspector
- FinalRefer

Below are the prompt templates for each agent:

Math Solver
You are a math expert.
You will be given a math problem and hints from other agents.
Give your own solving process step by step based on hints.
The last line of your output contains only the final result without any units, for example: The answer is 140
You will be given some examples you may refer to.
Q: Angelo and Melanie want to plan how many hours over the next week they should study together for their test next week.
They have 2 chapters of their textbook to study and 4 worksheets to memorize.
They figure out that they should dedicate 3 hours to each chapter of their textbook and 1.5 hours for each worksheet.
If they plan to study no more than 4 hours each day, how many days should they plan to study total over the next week if they take a 10-minute break every hour, include 3 10-minute snack breaks each day, and 30 minutes for lunch each day?.

A: Let's think step by step.
Angelo and Melanie think they should dedicate 3 hours to each of the 2 chapters, 3 hours x 2 chapters = 6 hours total.
For the worksheets they plan to dedicate 1.5 hours for each worksheet, 1.5 hours x 4 worksheets = 6 hours total. Angelo and Melanie need to start with planning 12 hours to study, at 4 hours a day, 12 / 4 = 3 days.
However, they need to include time for breaks and lunch. Every hour they want to include a 10-minute break, so 12 total hours x 10 minutes = 120 extra minutes for breaks.
They also want to include 3 10-minute snack breaks, 3 x 10 minutes = 30 minutes.
And they want to include 30 minutes for lunch each day, so 120 minutes for breaks + 30 minutes for snack breaks + 30 minutes for lunch = 180 minutes, or 180 / 60 minutes per hour = 3 extra hours.
So Angelo and Melanie want to plan 12 hours to study + 3 hours of breaks = 15 hours total.
They want to study no more than 4 hours each day, 15 hours / 4 hours each day = 3.75
They will need to plan to study 4 days to allow for all the time they need.
The answer is 4

Q: Bella has two times as many marbles as frisbees. She also has 20 more frisbees than deck cards. If she buys 2/5 times more of each item, what would be the total number of the items she will have if she currently has 60 marbles?
A: Let's think step by step
When Bella buys 2/5 times more marbles, she'll have increased the number of marbles by 2/5*60 = 24
The total number of marbles she'll have is 60+24 = 84
If Bella currently has 60 marbles, and she has two times as many marbles as frisbees, she has 60/2 = 30 frisbees.
If Bella buys 2/5 times more frisbees, she'll have 2/5*30 = 12 more frisbees.
The total number of frisbees she'll have will increase to 30+12 = 42
Bella also has 20 more frisbees than deck cards, meaning she has 30-20 = 10 deck cards
If she buys 2/5 times more deck cards, she'll have 2/5*10 = 4 more deck cards.
The total number of deck cards she'll have is 10+4 = 14
Together, Bella will have a total of 14+42+84 = 140 items
The answer is 140

Q: Susy goes to a large school with 800 students, while Sarah goes to a smaller school with only 300 students. At the start of the school year, Susy had 100 social media followers. She gained 40 new followers in the first week of the school year, half that in the second week, and half of that in the third week. Sarah only had 50 social media followers at the start of the year, but she gained 90 new followers the first week, a third of that in the second week, and a third of that in the third week. After three weeks, how many social media followers did the girl with the most total followers have?
A: Let's think step by step
After one week, Susy has 100+40 = 140 followers.
In the second week, Susy gains 40/2 = 20 new followers.
In the third week, Susy gains 20/2 = 10 new followers.
In total, Susy finishes the three weeks with 140+20+10 = 170 total followers.
After one week, Sarah has 50+90 = 140 followers.
After the second week, Sarah gains 90/3 = 30 followers.
After the third week, Sarah gains 30/3 = 10 followers.
So, Sarah finishes the three weeks with 140+30+10 = 180 total followers.

Thus, Sarah is the girl with the most total followers with a total of 180.
The answer is 180

Mathematical Analyst
You are a mathematical analyst. You will be given a math problem, analysis and code from other agents. You need to first analyze the problem-solving process step by step, where the variables are represented by letters. Then you substitute the values into the analysis process to perform calculations and get the results. The last line of your output contains only the final result without any units, for example: The answer is 140
You will be given some examples you may refer to.
Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?
A: ## Problem solving process analysis
There are {ori_tree_num} trees originally.
Then there were {after_planted_tree_num} trees after some more were planted.
So the number of trees planted today {today_planted_num} is the number of trees after planting {after_planted_tree_num} minus the number of trees before planting {ori_tree_num}.
The answer is {today_planted_num} = {after_planted_tree_num} - {ori_tree_num}.
Actual analysis and solution process
In this question, {ori_tree_num} = 15 and {after_planted_tree_num} = 21.
There are 15 trees originally.
Then there were 21 trees after some more were planted.
So the number of trees planted today must have been 21 - 15 = 6.
The answer is 6

Q: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
A:## Problem solving process analysis
Originally, Leah had {Leah_num} Leah_num chocolates.
Her sister had {sister_num} chocolates.
So in total they had {all_num} = {Leah_num} + {sister_num} chocolates.
After eating {eating_num} chocolates, the number of chocolates they have left {remain_num} is {all_num} minus {eating_num}.
The answer is {remain_num} = {all_num} - {eating_num}.
Actual analysis and solution process
In this question, {Leah_num} = 32, {sister_num} = 42 and {all_num} = 35.
So, in total they had 32 + 42 = 74 chocolates originally.
After eating 35 chocolates, they had 74 - 35 = 39 chocolates.
The answer is 39

Programming Expert
You are a programming expert. You will be given a math problem, analysis and code from other agents. Integrate step-by-step reasoning and Python code to solve math problems. Analyze the question and write functions to solve the problem. The function should not take any arguments and use the final result as the return value. The last line of code calls the function you wrote and assigns the return value to the `answer` variable. Use a Python code block to write your response. For example:

```
```python
def fun():
    x = 10
    y = 20
    return x + y
answer = fun()
```
```

Do not include anything other than Python code blocks in your response. You will be given some examples you may refer to.
Q: Olivia has $23. She bought five bagels for $3 each. How much money does she have left?
A:

```
```python
def money_left():
    money_initial = 23
```
```

````
    bagels = 5
    bagel_cost = 3
    money_spent = bagels * bagel_cost
    remaining_money = money_initial - money_spent
    return remaining_money
answer = money_left()
```
````

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?
A:

````
```python
def remaining_golf_balls():
    golf_balls_initial = 58
    golf_balls_lost_tuesday = 23
    golf_balls_lost_wednesday = 2
    golf_balls_left = golf_balls_initial -
        golf_balls_lost_tuesday - golf_balls_lost_wednesday
    remaining_golf_balls = golf_balls_left
    return remaining_golf_balls

answer = remaining_golf_balls()
```
````

Inspector
You are an Inspector. You will be given a math problem, analysis and code from other agents. Check whether the logic/calculation of the problem solving and analysis process is correct (if present). Check whether the code corresponds to the solution analysis (if present). Give your own solving process step by step based on hints. The last line of your output contains only the final result without any units, for example: The answer is 140
You will be given some examples you may refer to.

FinalRefer
You are the top decision-maker. Good at analyzing and summarizing mathematical problems, judging and summarizing other people's solutions, and giving final answers to math problems. You will be given a math problem, analysis and code from other agents. Please find the most reliable answer based on the analysis and results of other agents. Give reasons for making decisions. The last line of your output contains only the final result without any units, for example: The answer is 140

D.3 HUMANEVAL

For HumanEval, the following roles are cycled for collaborative code generation and debugging:

- Project Manager
- Algorithm Designer
- Programming Expert (with access to the python execution tool)
- Test Analyst
- FinalRefer

Below are the prompt templates for each agent:

Project Manager
You are a project manager. You will be given a function signature and its docstring by the user. You are responsible for overseeing the overall structure of the code, ensuring that the code is structured to complete the task. Implement code concisely and correctly without pursuing over-engineering. You need to suggest optimal design patterns to ensure that the code follows best practices for maintainability and flexibility. You can specify the overall design of the code, including the classes that need to be
````

defined (maybe none) and the functions used (maybe only one function). I hope your reply will be more concise. Preferably within fifty words. Do not list too many points.

Algorithm Designer
You are an algorithm designer. You will be given a function signature and its docstring by the user. You need to specify the specific design of the algorithm, including the classes that may be defined and the functions used. You need to generate the detailed documentation, including explanations of the algorithm, usage instructions, and API references. When the implementation logic is complex, you can give the pseudocode logic of the main algorithm. I hope your reply will be more concise. Preferably within fifty words. Do not list too many points.

Programming Expert
You are a programming expert. You will be given a function signature and its docstring by the user. You may be able to get the output results of other agents. They may have passed internal tests, but they may not be completely correct. Write your full implementation (restate the function signature). Use a Python code block to write your response. For example:

```
```python
print('Hello world!')
```
```

Do not include anything other than Python code blocks in your response. Do not change function names and input variable types in tasks.

Test Analyst
You are a test analyst. You will be given a function signature and its docstring by the user. You need to provide problems in the current code or solution based on the test data and possible test feedback in the question. You need to provide additional special use cases, boundary conditions, etc. that should be paid attention to when writing code. You can point out any potential errors in the code. I hope your reply will be more concise. Preferably within fifty words. Do not list too many points.

FinalRefer
You are the top decision-maker and are good at analyzing and summarizing other people's opinions, finding errors, and giving final answers. And you are an AI that only responds with only Python code. You will be given a function signature and its docstring by the user. You may be given the overall code design, algorithm framework, code implementation or test problems. Write your full implementation (restate the function signature). If the prompt given to you contains code that passed internal testing, you can choose the most reliable reply. If there is no code that has passed internal testing in the prompt, you can change it yourself according to the prompt. Use a Python code block to write your response. For example:

```
```python
print('Hello world!')
```
```

Do not include anything other than Python code blocks in your response

E Additional Experimental Results

This section complements Section 5 with results on dependency-graph topologies, additional baselines, dynamic workflows, throughput under load, edge-specific threshold adaptation, verification and output-length statistics, and execution dynamics. Each study runs in its own harness and reports speedups relative to the Sequential run of that harness, so absolute latencies differ across studies.

E.1 Dependency-Graph Topologies

Figure E.1 illustrates the five dependency graphs evaluated in Table 4; Section 5.3 analyzes how their depth and fan-in shape the speedup.

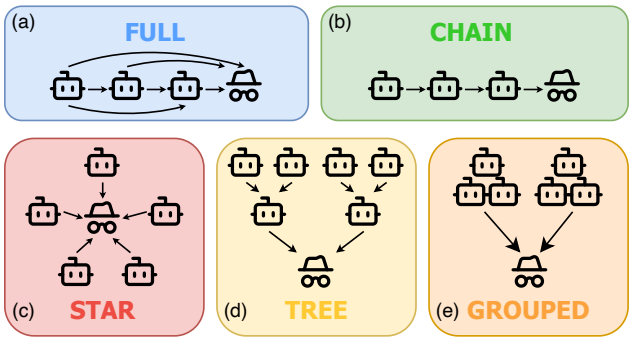


Figure E.1: **Five canonical dependency-graph structures.** (a) Full, (b) chain, (c) star, (d) tree, and (e) grouped workflows; arrows point from parent to child agents.

E.2 Additional Baselines

FlashAgents-style streaming prefill. FlashAgents (Fang et al., 2026) streams parent tokens to the child so that the child's prefill overlaps with parent decoding, and the child starts decoding once the parent finishes. We re-implement this mechanism in our runtime and evaluate it on *full*-mode workflows with 200 MMLU questions, 200 GSM8K problems, and the 161 HumanEval problems on one A100. Sequential and FlashAgents-style runs replay ACCORDION's per-agent tokens and tool results, so all three modes produce identical outputs and the speedups in Table E.1 isolate latency. FlashAgents-style prefill overlap stays within $1.03\times$ of Sequential, while ACCORDION reaches 1.08–$1.37\times$ by overlapping child decoding.

Table E.1: Speedup over Sequential against FlashAgents-style streaming prefill.

Dataset	Agents	FlashAgents-style	ACCORDION
MMLU	3	$1.02\times$	$\mathbf{1.18}\times$
	4	$1.03\times$	$\mathbf{1.33}\times$
GSM8K	3	$1.02\times$	$\mathbf{1.33}\times$
	4	$1.02\times$	$\mathbf{1.37}\times$
HumanEval	3	$0.99\times$	$\mathbf{1.08}\times$
	4	$1.01\times$	$\mathbf{1.28}\times$

Composition with EAGLE-3. The EAGLE-3 comparison in Figure 5 (right) uses four-agent *full*-mode workflows with 153 MMLU questions, the 1,319 GSM8K problems, and the 161 HumanEval problems.

E.3 Dynamic Workflows on PaperBananaBench

PaperBananaBench (Zhu et al., 2026) evaluates automated generation of academic plots. We run its plot-generation test set on OpenManus (Liang et al., 2025), where a Qwen2.5-VL-7B-Instruct planner decides the workflow at runtime: Qwen2.5-14B-Instruct stylist and visualizer agents write plotting code, a matplotlib tool renders it, and a Qwen2.5-VL-7B-Instruct critic inspects the rendered plot and requests up to three revision rounds, each of which adds new visualizer, tool, and critic nodes to the graph. We evaluate the 240 test plots with three seeds on NVIDIA H200 NVL GPUs, giving 720 runs per method. ACCORDION reduces mean E2E latency from 58.7s to 49.8s (paired 95% CI of speedup: 1.14–$1.21\times$), median latency from 54.2s to 45.1s, and P95 latency from 108.3s to 80.8s, and is faster on 64.4% of runs. ACCORDION produces a valid plot during the run in 74.9% of cases (vs. 71.8% for Sequential). Among runs that end with a valid plot, the mean judge score is 8.73 for ACCORDION and 8.82 for Sequential.

E.4 THROUGHPUT UNDER LOAD

Context-level speculation adds verification, prompt refresh, and KV-cache repair work, so its benefit depends on the available GPU headroom. We stress-test this trade-off with Qwen2.5-14B-Instruct on one H200, a 2,048-token prompt per agent, and fixed 16-token commits with controlled acceptance rates, submitting batches of 1 to 64 concurrent workflows and averaging three measured repeats per setting. We consider four DAGs: a two-agent chain (P→C), a three-agent chain (P→C→G), a fan-out (P→{C_1,C_2}), and a fan-in (P_1,P_2→C). Table E.2 reports the workflow-throughput ratio over Sequential for acceptance rate α and B concurrent workflows, reaching up to $1.92\times$ at low load; Table E.3 reports the largest concurrency at which context-level speculation keeps a throughput advantage, by output length per agent ("–": no gain at the tested loads). This range is widest for short outputs, high acceptance, and chains, matching latency-critical serving at low-to-moderate load.

Table E.2: Throughput ratio over Sequential at 128 output tokens per agent.

	$\alpha = 0.9$			$\alpha = 1.0$		
DAG	$B = 1$	$B = 8$	$B = 64$	$B = 1$	$B = 8$	$B = 64$
P→C	1.32	1.10	0.77	1.55	1.22	0.79
P→C→G	1.59	1.14	0.64	1.92	1.32	0.69
P→{C_1,C_2}	1.27	0.96	0.62	1.46	1.04	0.65
P_1,P_2→C	1.24	1.00	0.67	1.40	1.08	0.72

Table E.3: Largest concurrency with a throughput gain over Sequential.

		Output length (tokens)			
Acceptance	**DAG**	**128**	**256**	**512**	**1,024**
0.9	P→C	8	4	2	1
	P→C→G	8	4	2	1
	P→{C_1,C_2}	4	2	1	–
	P_1,P_2→C	4	2	1	–
1.0	P→C	16	8	8	4
	P→C→G	16	8	8	4
	P→{C_1,C_2}	8	4	4	2
	P_1,P_2→C	8	4	4	2

E.5 EDGE-SPECIFIC AND ONLINE THRESHOLD ADAPTATION

The main experiments use a single shared surprisal threshold for simplicity and comparability. We further examine whether this threshold should adapt across dependency edges and workloads. Different edges can prefer different commit granularities: a child whose prompt is highly sensitive to a particular parent may benefit from earlier, lower-threshold commits, while a less sensitive edge can tolerate coarser updates. Figure E.2 sweeps edge-specific thresholds on a three-agent GSM8K full DAG with edges $A \to B$, $A \to C$, and $B \to C$. The best shared setting reaches 91.9% accuracy at 4.653s mean E2E latency, while the best edge-specific setting reaches 92.9% accuracy at 4.441s. Thus edge-specific thresholds improve both latency and accuracy in this setting, showing that threshold adaptation is a useful direction rather than merely parameter tuning.

Motivated by this sweep, we evaluate a lightweight online edge-threshold policy. The policy initializes each edge threshold using the parent's prompt position relative to the child response, then adjusts the threshold after verification according to smoothed latency gain and prefix-reuse statistics. We use one shared online-policy configuration across all evaluated benchmarks and topologies. Figure E.3 compares this online policy with the fixed shared default $\tau = 10$ on GSM8K chain/full and MMLU star workflows. The online policy is faster in 9 of 12 settings, with the largest gain of 0.750s on six-agent GSM8K full mode; the few regressions are small, at most 0.123s. Taken together with the edge-specific sweep, these results show a clear improvement signal: adaptive edge thresholds are worth pursuing as a practical extension of the fixed-threshold rule.

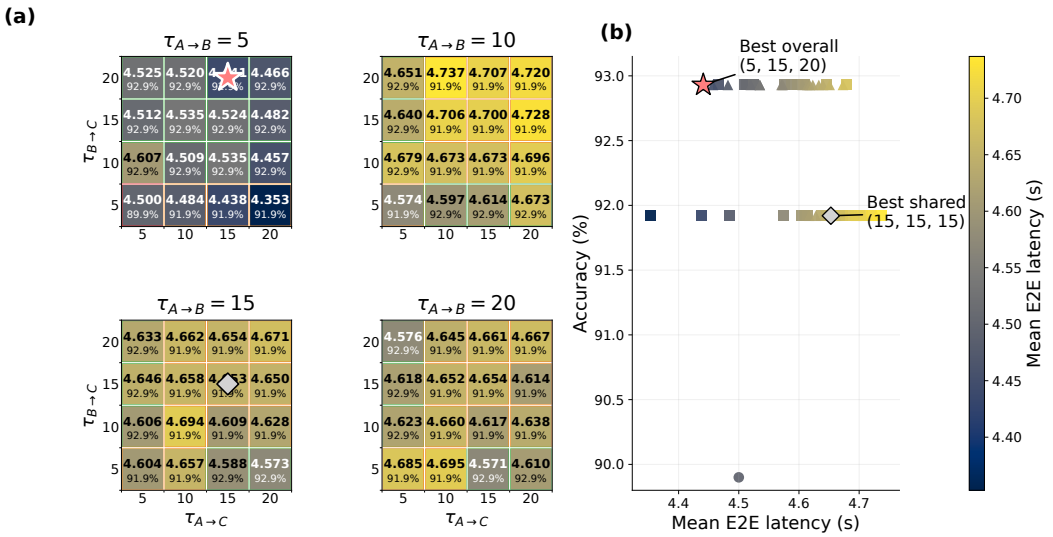


Figure E.2: **Sensitivity to edge-specific surprisal thresholds.** On a three-agent GSM8K full DAG, each cell reports mean E2E latency (top) and accuracy (bottom) for one choice of $(\tau_{A\to B}, \tau_{A\to C}, \tau_{B\to C})$. The best edge-specific threshold improves both accuracy and latency over the best shared threshold, supporting edge-aware threshold selection.

E.6 Verification Counts and Output Lengths

Verification counts. Figure E.4 reports the number of verification operations. The overhead is modest: even with six agents, each sample requires only a few hundred verifications, far fewer than the thousands of generated tokens shown in Figure E.5. Moreover, ACCORDION consistently uses fewer verification operations than frequent fixed-length policies, reducing synchronization and verification cost.

Response length distribution. Figure E.5 complements the verification-count analysis by comparing finalized response lengths across agents, tasks, and commit policies. The surprisal-triggered policy produces response-length distributions close to the sequential baseline on MMLU, GSM8K, and HumanEval, indicating that the observed latency gains do not come from prematurely shortening generations. Instead, most variation is explained by task and agent role: reasoning-heavy agents naturally produce longer responses, while aggregation or answer-format agents tend to be shorter. This indicates that ACCORDION preserves the effective amount of generated content while reducing latency through overlap and prefix reuse.

E.7 Execution Dynamics

Overview. We next inspect time-resolved behavior: per-agent timelines show the recovered overlap, acceptance traces show how much speculative work is retained, and trigger traces show how the surprisal budget schedules updates. Overall, surprisal-triggered commits yield longer retained-prefix plateaus than fixed-length chunking, and KV-cache rebuild, rather than verification, accounts for most of the overhead at high fan-in (Appendix C.4).

Response dynamics. Figure E.6 shows the finalized response length, i.e., the retained prefix, after each parent commit on MMLU. The fixed-length policy (16-token chunks; left) and surprisal-based commits (right) both reach around 400 tokens. The fixed-length policy exhibits abrupt drops, whereas ACCORDION adaptively adjusts commit boundaries in semantically challenging regions, resulting in higher late-stage acceptance rates and fewer execution steps.

Per-agent timing. Figure E.7 compares per-agent execution time under sequential and parallel modes on GSM8K. Under sequential execution (upper), each child waits idly for its parent to finish,

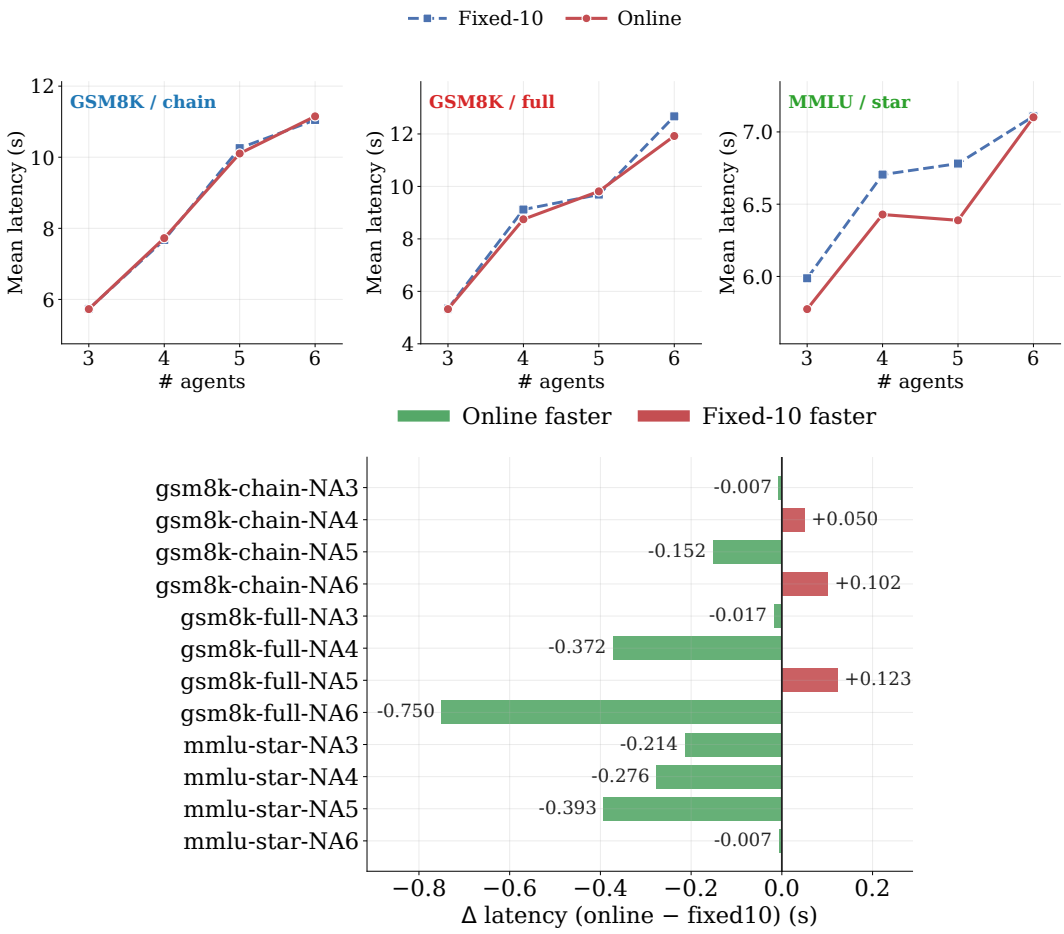


Figure E.3: **Online edge-threshold adaptation.** Top: Mean E2E latency of the online edge-threshold policy versus a fixed shared $\tau = 10$ baseline across GSM8K chain/full and MMLU star workflows. Bottom: Paired latency difference, where negative values indicate that the online policy is faster. With one shared online-policy configuration, edge adaptation improves latency in most settings, giving evidence that online threshold adaptation is beneficial and worth further development.

leading to staggered execution. Under parallel execution (lower), children begin drafting as soon as partial context is available, overlapping compute across agents and reducing the makespan. The figure shows that most time savings stem from overlapping parent–child decoding rather than from faster generation by the LLMs themselves.

Prefix acceptance. Figure E.8 reports the acceptance rate of speculative prefixes across agents. Across all tasks and agents, the acceptance rate remains high under the surprisal trigger, indicating that most drafted tokens are retained. This behavior keeps rollback waste controlled and complements the overhead decomposition in Appendix C.4.

Trigger traces. Figure E.9 compares trigger traces over time on GSM8K, where the surprisal-based policy keeps the trigger frequency low overall and raises it only in semantically challenging regions.

E.8 Statistical Reliability

To quantify run-to-run variation, we repeat the Sequential and ACCORDION runs of Table 1 on MMLU and GSM8K, giving six paired runs per setting: the run reported in Table 1 and five additional seeds, all on the same 200 questions and on NVIDIA A100-SXM4-80GB GPUs. Within each run, both methods process the same examples. Table E.4 reports the mean and standard deviation across the six

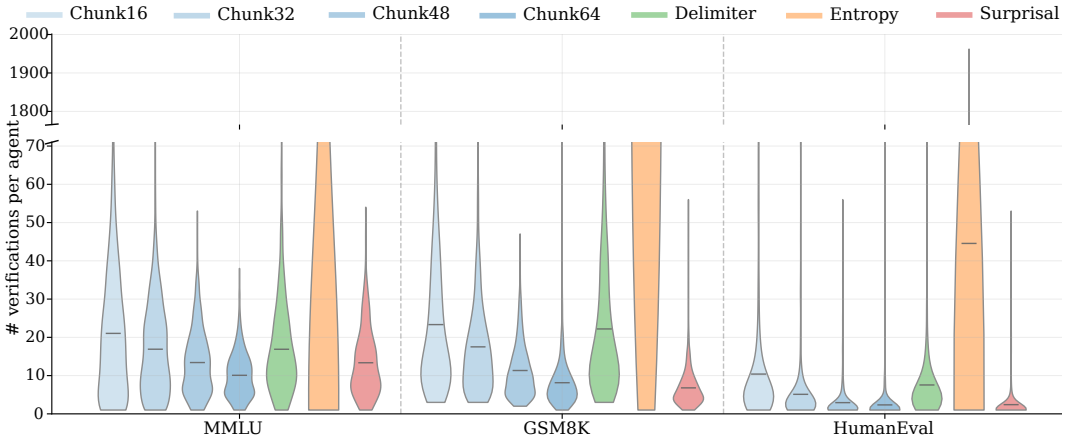


Figure E.4: **Number of verification operations per agent.** Verification overhead is modest relative to the total number of generated tokens.

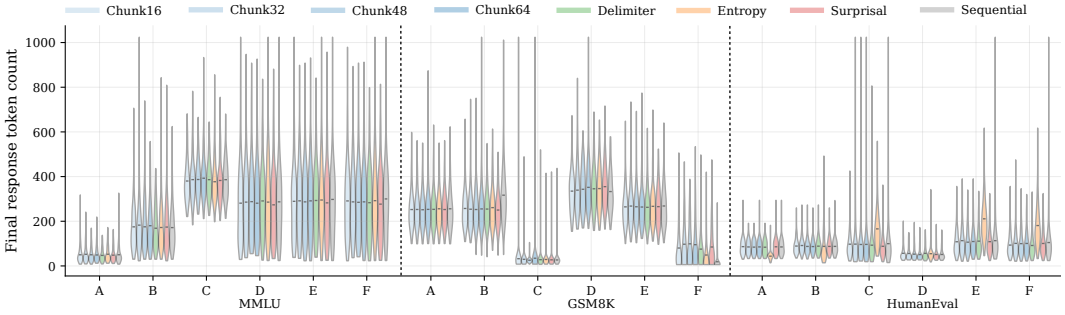


Figure E.5: **Distribution of response token lengths across agents.** The surprisal trigger results in similar output lengths compared with the sequential baseline.

runs. ACCORDION is faster than Sequential in every run, with mean speedups of 1.21–1.46× and standard deviations of at most 0.11×, while its mean accuracy differs from Sequential by −1.17 to +0.92 pp. HumanEval uses the complete 161-problem set with greedy decoding and is not resampled.

Table E.4: Six paired runs on MMLU and GSM8K in full-mode workflows (mean ± standard deviation).

		E2E latency (s)			Accuracy (%)		
Dataset	**Agents**	**Sequential**	**ACCORDION**	**Speedup (×)**	**Sequential**	**ACCORDION**	**ΔAcc (pp)**
MMLU	3	15.87 ± 0.45	13.18 ± 0.59	1.21 ± 0.02	74.58 ± 2.27	75.50 ± 2.61	$+0.92 \pm 3.93$
	4	25.72 ± 0.70	18.38 ± 1.14	1.40 ± 0.05	75.17 ± 1.83	74.00 ± 1.92	-1.17 ± 2.46
	5	32.40 ± 0.81	22.30 ± 1.60	1.46 ± 0.08	74.58 ± 2.87	73.42 ± 2.69	-1.17 ± 3.24
	6	36.32 ± 1.29	26.32 ± 1.80	1.38 ± 0.10	74.58 ± 3.40	73.75 ± 1.92	-0.83 ± 3.53
GSM8K	3	13.63 ± 0.43	10.03 ± 0.56	1.36 ± 0.06	93.00 ± 0.95	92.50 ± 1.18	-0.50 ± 1.05
	4	17.06 ± 3.32	11.81 ± 1.89	1.44 ± 0.06	93.25 ± 0.94	92.75 ± 1.25	-0.50 ± 0.45
	5	20.78 ± 3.81	14.66 ± 1.74	1.41 ± 0.10	92.92 ± 1.11	92.67 ± 1.33	-0.25 ± 0.94
	6	26.51 ± 4.16	18.17 ± 2.40	1.46 ± 0.07	93.00 ± 1.05	92.75 ± 0.76	-0.25 ± 0.82

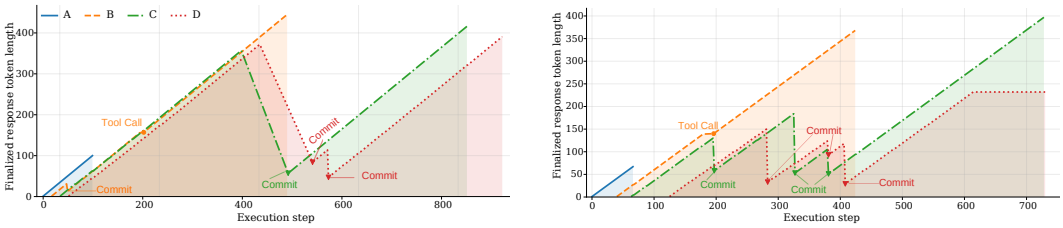


Figure E.6: **Step-wise finalized response length on MMLU.** Left: Chunk-16. Right: ACCORDION. The surprisal trigger yields longer plateaus and fewer resets than fixed-length chunking.

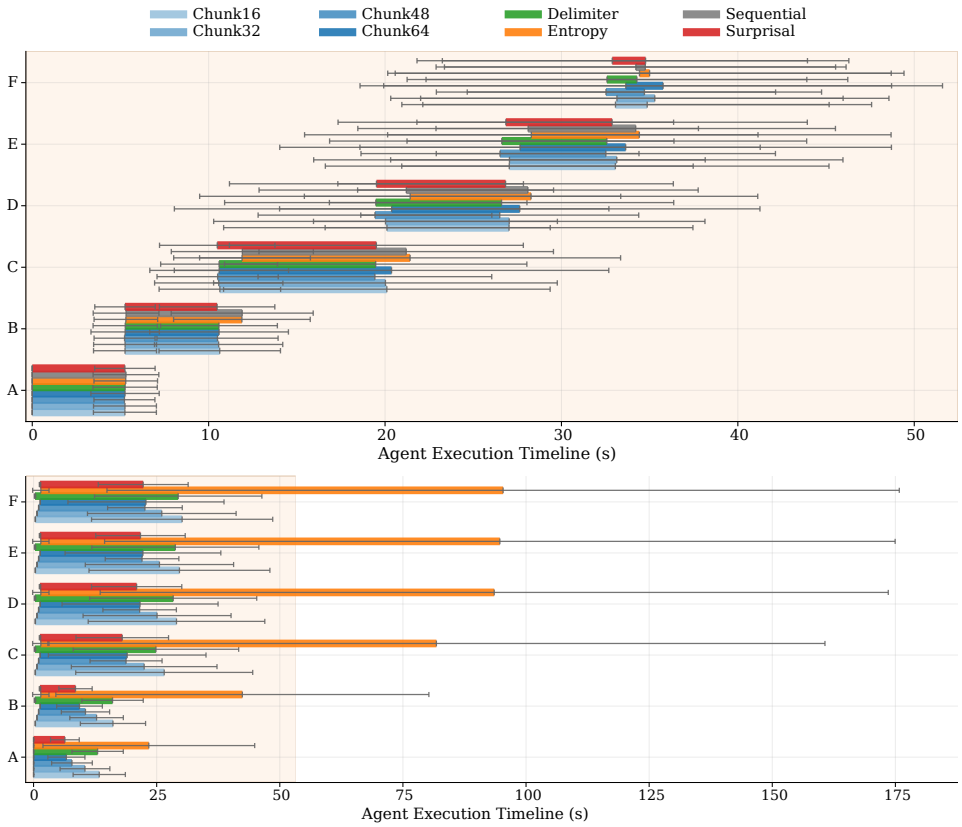


Figure E.7: **Per-agent execution timelines on GSM8K.** Top: Sequential execution. Bottom: Parallel execution. Each panel shows every commit policy; overlapping parent and child decoding can significantly shorten the makespan.

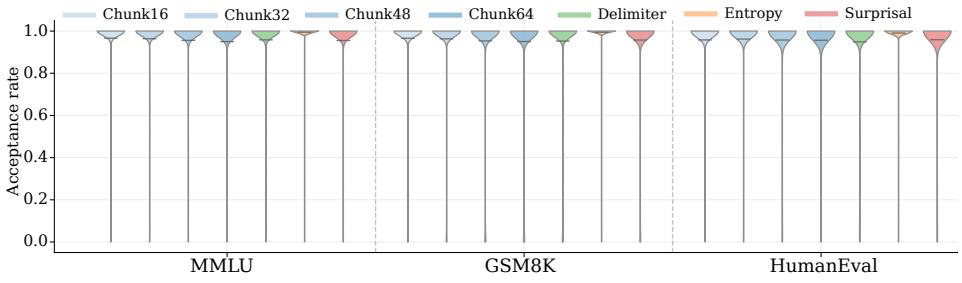


Figure E.8: **Prefix acceptance rate in six-agent workflows.** Acceptance remains high under ACCORDION.

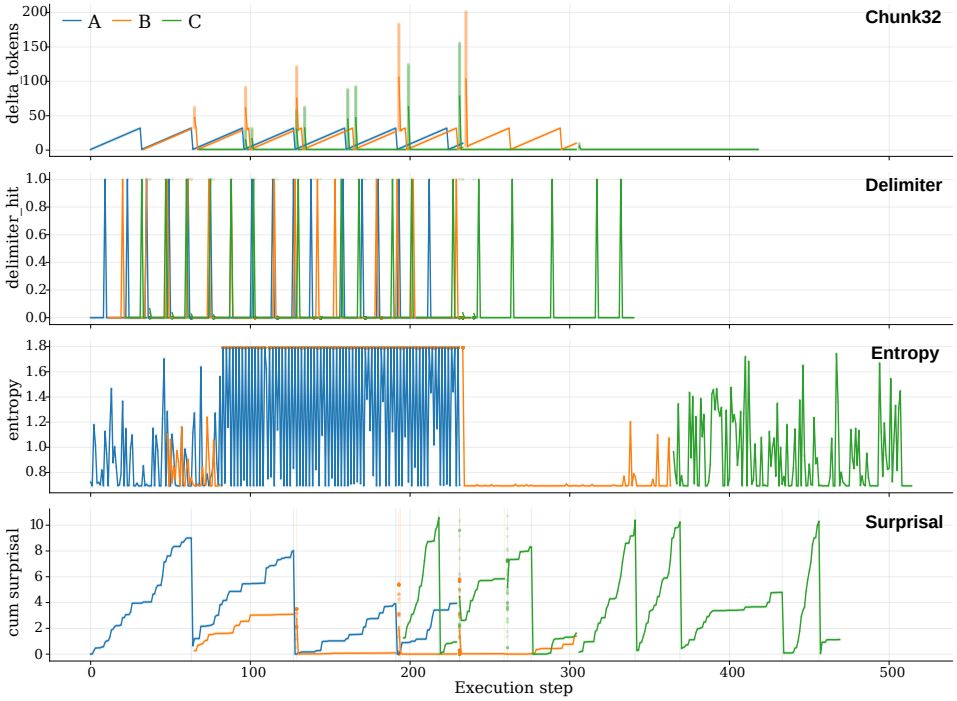


Figure E.9: **Trigger traces on GSM8K.** The surprisal-budgeted policy maintains a stable trade-off over time.