

Streaming Communication in Multi-Agent Reasoning

Zhen Yang¹, Xiaogang Xu³, Wen Wang³, Cong Chen³,
Xander Xu^{2*}, Ying-Cong Chen^{1,4*}

¹HKUST(GZ), ²Alibaba Group, ³ZJU, ⁴HKUST

zheny.cs@gmail.com, yingcongchen@ust.hk

Abstract

Multi-agent reasoning systems adopt a “generate-then-transfer” paradigm that forces end-to-end latency to scale linearly with pipeline depth. We introduce STREAMMA, a multi-agent reasoning system that streams each *reasoning step* to downstream agents as soon as it is generated, pipelining adjacent agents and thus reducing latency. Surprisingly, this pipelining also improves effectiveness: because multi-step reasoning quality is non-uniform and early steps are more reliable than later ones, working with these reliable early steps instead of the full chain prevents error-prone late steps from misleading downstream agents. We formalize both advantages with the first closed-form joint analysis of stream, serial, and single protocols, deriving the effectiveness ordering, speedup upper bound, and cost ratio. Across eight reasoning benchmarks spanning mathematics, science, and code, three frontier LLMs (Claude Opus 4.6, GPT-5.4, and GLM-5.2), and three topologies (Chain, Tree, Graph), STREAMMA outperforms both baselines (avg. **+7.3 pp**, max **+22.4 pp** on HMMT 2026; Claude Opus 4.6-High). Beyond these contributions, we discover a “*step-level scaling law*”: increasing per-agent steps consistently improves both effectiveness and efficiency, a new scaling dimension orthogonal to and composable with agent-count scaling. Our [project page](#) and [code](#) are available.

1 Introduction

Multi-agent systems have emerged as a paradigm for complex reasoning: agents are organized as nodes of a directed acyclic graph (DAG), with each directed edge governing information propagation from one agent to its successor, leveraging specialization and cross-verification to surpass a single model (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024), with growing benefits as the number of agents increases (Qian et al., 2025). Yet

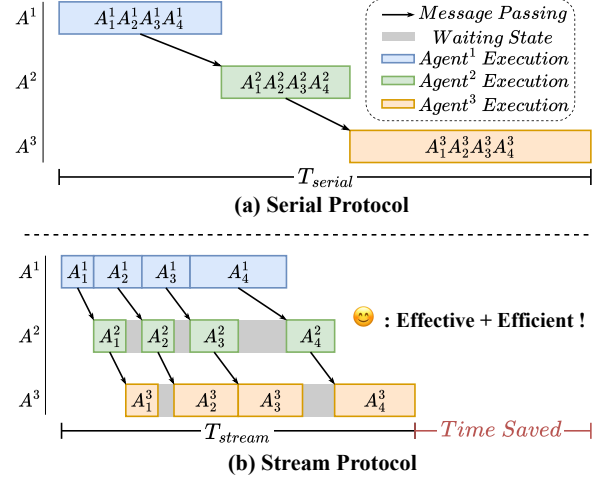


Figure 1: **Communication protocols.** (a) **Serial**: the downstream agent receives the upstream agent’s complete response before execution. (b) **Stream**: the downstream agent receives each upstream reasoning step as it is generated, enabling pipelined execution.

existing frameworks share the same communication assumption: an upstream agent must finish its entire response before passing it downstream. This “generate-then-transfer” (Serial, Fig. 1(a)) protocol forces downstream agents to remain idle and latency to scale with pipeline depth. While streaming partial outputs is standard within a single LLM (Tong et al., 2026), between agents the transfer unit has remained a complete response, and its effect on effectiveness remains underexplored.

A natural question arises: can we forward partial outputs before the upstream agent finishes? Conventional wisdom holds that incomplete context must hurt downstream quality: “more information yields better decisions.” Yet this intuition does not always hold: we find that in multi-step reasoning, receiving less context can improve downstream reasoning quality. We propose STREAMMA (Fig. 1(b)), a multi-agent reasoning system built on *Stream*, a reasoning-step-level communication protocol that shifts the transmission unit from com-

*Co-corresponding authors.

plete responses to reasoning steps, forwarding each step once produced, pipelining upstream and downstream agents. Across eight benchmarks in mathematics, science, and code, STREAMMA outperforms Serial by an average +7.3 pp (peak +22.4 pp on HMMT 2026, Claude Opus 4.6-High) while reducing latency: less waiting, yet better reasoning.

Why does the downstream agent benefit from only the early reasoning steps more than the full response? On complex reasoning tasks, step-level quality is position-dependent: early steps tend to be reliable, whereas later steps degrade. The Serial protocol forces the downstream agent to condition on the entire response, including its error-prone late steps. Stream lets it begin from the most reliable prefix; by the time degraded steps arrive, the agent has formed its own independent reasoning trajectory, diluting their impact. Controlled step-level perturbations cross-validate this asymmetry: corrupting only the tail steps of an upstream reasoning trajectory leaves Stream nearly intact and ahead of Serial by +24.0 pp, while corrupting only its head steps flips the gap to -36.0 pp (Sec. 4.2.3). We formalize this mechanism: [Theorem 1](#) gives an effectiveness ordering across all six regimes, predicting Stream optimal when early steps are reliable and late ones degrade; [Theorem 2](#) and [Theorem 3](#) derive the speedup upper bound and exact cost ratio under LLM serving conditions.

Beyond mechanism and theory, our experiments uncover a second scaling axis, orthogonal to the well-studied agent-count axis: at fixed agent count A , increasing per-agent steps S consistently improves effectiveness and speedup, defining a step-level scaling law fully compositional with agent-count scaling. On HMMT 2026 with GPT-5.4-None, scaling A alone from 2 to 64 lifts effectiveness from 58.3% to 68.2%; further increasing S to 64 pushes it to 73.5% with $26.9\times$ wall-clock speedup (83% of the [Theorem 2](#) upper bound). STREAMMA thus simultaneously serves as a reasoning enhancer and an inference accelerator.

In summary, our contributions are as follows:

- **Protocol.** We introduce Stream, a reasoning-step-level communication protocol that forwards each reasoning step upon completion, pipelining upstream and downstream agents over arbitrary DAG topologies. Stream improves effectiveness while reducing latency.
- **Theory.** We present the first closed-form joint analysis of Stream, Serial, and Single proto-

cols: [Theorem 1](#) provides an effectiveness ordering, [Theorem 2](#) a speedup upper bound, and [Theorem 3](#) an exact cost ratio.

- **Empirics.** Across eight benchmarks, three LLMs, and three topologies, STREAMMA consistently outperforms both baselines in average effectiveness (Claude +7.3 pp over Serial, +16.3 pp over Single; GPT +1.5 pp over Serial, +4.9 pp over Single; GLM +7.2 pp over Serial, +20.0 pp over Single).
- **Discovery.** We discover a step-level scaling law under Stream: at fixed agent count A , increasing per-agent steps S consistently improves effectiveness and speedup. At $A=64$, scaling S to 64 gains +5.3 pp complementary to agent-count scaling, with $26.9\times$ speedup.

2 Related Work

Multi-Agent Reasoning and Communication.

Decomposing reasoning across collaborating agents has become a mainstream paradigm for complex LLM tasks ([Wu et al., 2023](#); [Hong et al., 2024](#); [Qian et al., 2024](#); [Li et al., 2023](#); [Du et al., 2024](#); [Chen et al., 2024](#); [Wang et al., 2025a](#)), with prior work advancing it along three orthogonal axes: the communication topology ([Qian et al., 2025](#); [Zhuge et al., 2024](#); [Liu et al., 2023](#); [Zhang et al., 2025](#)); the content exchanged between agents, such as intermediate rationales ([Yin et al., 2023](#)) or KV-cache representations ([Fu et al., 2025](#)), and how much of it ([Rizvi-Martel et al., 2025](#)); and the scale of agents ([Qian et al., 2025](#); [Li et al., 2024](#)). Yet all three share a generate-then-transfer assumption: an upstream agent must finalize its response before any downstream agent can act, forcing per-round sequential waits and forgoing pipeline parallelism. We instead refine the granularity of communication, from full responses to reasoning steps. This shift exposes a new design axis: per-agent step count, orthogonal to agent-count scaling. Increasing per-agent steps simultaneously improves effectiveness and speedup, termed the step-level scaling law.

Step-Level Reasoning Quality. Step-by-step reasoning is now standard ([Wei et al., 2022](#); [Yao et al., 2023](#)), with iterative-refinement variants such as Self-Refine ([Madaan et al., 2023](#)), Reflexion ([Shinn et al., 2023](#)), and multi-agent debate ([Du et al., 2024](#)) pushing output quality higher. On complex reasoning tasks, step quality is position-dependent: [Wu et al. \(2025\)](#) show that CoT ac-

Algorithm 1 SERIAL EXECUTION

Require: Q ; $(\text{Agent}^a, \text{ctx}_a, \text{queue}_a)_{a=1}^A$

- Q : query
- ctx_a : per-agent context
- queue_a : FIFO queue
- chain: $\text{Agent}^1 \rightarrow \dots \rightarrow \text{Agent}^A$

```
1:  $\text{msg} \leftarrow Q$ 
2: for  $a = 1$  to  $A$  do
3:    $\text{ctx}_a.\text{append}(\text{msg})$ 
4:    $\triangleright$  wait; complete output
5:    $\text{msg} \leftarrow \text{LLM}(\text{ctx}_a)$ 
6: end for
```

curacy peaks at an optimal length and degrades beyond it, and process supervision (Uesato et al., 2022; Lightman et al., 2024; Wang et al., 2024) scores reliability step by step. Yang et al. (2026) find that uncertainty is non-uniform even at the token level. So far, this property has been exploited only within a single model’s reasoning chain, never to design the inter-agent protocol. We are the first to lift this property into protocol design: our [Theorem 1](#) gives a closed-form effectiveness ordering of Single, Serial, and Stream, with explicit conditions under which each mode is provably optimal.

Pipeline Parallelism and Streaming Inference.

Pipeline parallelism is classical in distributed training (Narayanan et al., 2021), and streaming inference for LLMs (Tong et al., 2026) has been pursued along two axes: at the intra-agent level, via speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Cai et al., 2024), Group Think (Hsu et al., 2025), and Multi-Stream LLMs (Su et al., 2026); and at the inter-agent level, via skeleton expansion (Ning et al., 2024), speculative agent actions (Ye et al., 2025b), and staircase streaming (Wang et al., 2025b). Yet across all of these, streaming primarily serves as a speedup mechanism; any effectiveness gains are side-effects rather than the central design principle. Our Stream instead operates at the reasoning-step level over arbitrary multi-agent DAGs: a granularity coarser than tokens or fixed token chunks, yet finer than pre-decomposed skeletons or speculative agent actions, and unconstrained by topology. To our knowledge, we are the first to show that it can improve effectiveness, not just accelerate it ([Theorem 1](#)). At the inter-agent level, our protocol is orthogonal to all the intra-agent methods discussed above.

Algorithm 2 STREAM EXECUTION

```
1:  $\text{queue}_1.\text{put}(Q)$ 
2:  $\triangleright$  all agents concurrent
3: for  $a = 1$  to  $A$  in parallel do
4:   while  $\text{msg} \leftarrow \text{queue}_a.\text{get}()$  do
5:      $\text{ctx}_a.\text{append}(\text{msg})$ 
6:      $\triangleright$  yield step-by-step
7:      $\text{steps} \leftarrow \text{LLM}(\text{ctx}_a, \text{stream}=\text{True})$ 
8:     for each  $\text{step}$  from  $\text{steps}$  do
9:       if  $a < A$  then
10:         $\triangleright$  push; no wait
11:         $\text{queue}_{a+1}.\text{put}(\text{step})$ 
12:      end if
13:       $\triangleright$  KV cache reuse
14:       $\text{ctx}_a.\text{append}(\text{step})$ 
15:    end for
16:  end while
17: end for
```

3 Method

Since chains admit closed-form analysis and naturally compose general DAGs, we develop our theory on a chain of A agents ($\text{Agent}^1 \rightarrow \dots \rightarrow \text{Agent}^A$) each producing S reasoning steps. The results readily extend to arbitrary DAGs. We first describe the algorithm (Sec. 3.1), then characterize its effectiveness (Sec. 3.2) and efficiency (Sec. 3.3). Notation is summarized in App. A.1. We compare three execution modes throughout: **Stream**, our stream protocol; **Serial**, the generate-then-transfer serial protocol; and **Single**, the single-agent protocol. We refer to our framework, including the protocol and its theoretical analysis, as STREAMMA.

3.1 Algorithm Description

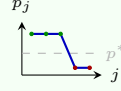
Alg. 1 and Alg. 2 contrast the serial and stream protocols. The key difference lies in the timing of message passing: in the serial protocol, each agent waits for the blocking call $\text{LLM}(\text{ctx}_a)$ to return the complete response (Alg. 1, line 5) before the next agent can begin; in the stream protocol, each agent issues a streaming call that yields steps one by one (Alg. 2, lines 7–8) and immediately pushes every completed step to queue_{a+1} (line 11) without waiting for subsequent steps. Step boundaries are explicit delimiters that the prompts in App. A.9 instruct the model to emit. All agents execute concurrently (line 3), enabling Agent^{a+1} to process step s while Agent^a is still generating step $s + 1$, forming pipeline parallelism. Moreover,

Theorem 1 (Effectiveness Ordering). *Depending on how $\bar{p}$, p_{head} , p_{tail} (uniform, head-weighted, and tail-weighted step-correctness means) compare to p^* , the sCorr ordering among three modes falls into six cases:*

(I) Stream advantage [$p_{\text{head}} > p^*$ and $p_{\text{tail}} < p^*$]:

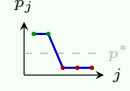
(a) If $\bar{p} > p^*$:

$$\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}}$$



(b) If $\bar{p} < p^*$:

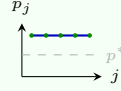
$$\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}}$$



(II) Serial advantage [$\bar{p} > p^*$ and $p_{\text{tail}} > p^*$]:

(a) If $p_{\text{head}} > p^*$:

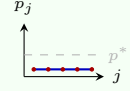
$$\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}}$$



(III) Single advantage [$p_{\text{head}} < p^*$ and $\bar{p} < p^*$]:

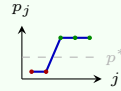
(a) If $p_{\text{tail}} < p^*$:

$$\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}}$$



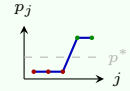
(b) If $p_{\text{head}} < p^*$:

$$\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}}$$



(b) If $p_{\text{tail}} > p^*$:

$$\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}}$$



each downstream agent ($a \geq 2$) is called S times, with prior steps serving as context for each subsequent call (line 14); the appended context naturally forms a shared prefix amenable to KV-cache reuse.

The chain topology generalizes to arbitrary DAGs by (i) broadcasting Q to all source nodes (in-degree 0) and (ii) pushing each step to all direct successors; multi-predecessor nodes process incoming steps immediately upon arrival, without any synchronization barrier across predecessors. The full pseudocode is given in App. A.8.

3.2 Effectiveness Characterization

STREAMMA consistently outperforms the Serial and Single baselines in our experiments. We now characterize when and why: we model each upstream step j ($1 \leq j \leq S$) as correct with probability p_j ; a correct step raises downstream step-level correctness by $\delta > 0$ and an incorrect step lowers it by $\varepsilon > 0$. For clarity we assume uniform δ, ε across positions; the position-dependent δ_j, ε_j case yields the same results in vector form. The expected step-level correctness change from step j is $\mu_j = p_j\delta - (1 - p_j)\varepsilon$, which is non-negative iff $p_j \geq \varepsilon/(\delta + \varepsilon) \triangleq p^*$. We define $\text{sCorr}^{\text{mode}}$ ($\text{mode} \in \{\text{stream}, \text{serial}, \text{single}\}$) as the mean of $\{\mu_j\}$ under each mode; sCorr is positively correlated with task-level accuracy. Three weighted means of $\{p_j\}$, the uniform mean $\bar{p}$, the head-weighted p_{head} (earlier steps weighted more), and the tail-weighted p_{tail} (later steps weighted more), govern the sCorr ordering among the three modes.

The six examples in Theorem 1 fix p^* only for simplicity. “Strong” and “weak” describe p_j relative to this threshold, which may vary by step.

Stream advantage. Cases I.a and I.b share the error-accumulation pattern in LLM reasoning: early steps are reliable but late ones fall below p^* . When context arrives matters more than how much: under Stream, the downstream agent receives early steps and begins reasoning; by the time late steps arrive, their impact is diluted. Under Serial, it must read the entire response, mixing clean early reasoning with error-prone late steps. In Case I.a, $\bar{p} > p^*$ means the response is on average helpful, so Stream leads while Serial beats Single. In Case I.b, only early steps exceed p^* , so Stream exposes these steps and beats Single, but Serial accumulates below- p^* late steps and loses to Single.

Serial advantage. In Case II.a, all steps are above p^* , so more context is beneficial: “more information, better decisions” holds. Serial sees the full response and wins; Stream sees partial but helpful steps and beats Single. In Case II.b, early steps are below p^* but late ones are accurate enough that $\bar{p} > p^*$; Serial profits and beats Single, yet Stream exposes the harmful early steps and ends up last. This captures self-correction: the upstream agent makes early mistakes but corrects them eventually.

Single advantage. Case III.a corresponds to problems so hard that all steps fall below p^* , making every upstream step harmful; Single avoids any upstream influence and achieves the highest step correctness. In Case III.b, the profile rises but $\bar{p} < p^*$ holds; this regime is rare, as a heavily corrupted prefix leaves too few remaining steps to lift p_{tail} above p^* . Serial loses to Single, and Stream, which receives the below- p^* early steps, comes last.

Variable definitions, full proofs, and the extension to arbitrary DAG topologies are in App. A.2.

3.3 Efficiency Characterization

Theorem 2 (Speedup). *The latency speedup of Stream over Serial is upper-bounded by:*

$$\text{Speedup} = \frac{A[(S + r_{po})r_{vdp} + S]}{(S + A - 1)(1 + \alpha r_{vdp} + \beta r_{vdc})}$$

where r_{vdp} : decode-to-prefill speed ratio; r_{vdc} : decode-to-cache-read speed ratio; r_{po} : system-prompt-to-per-step-output length ratio; α, β : average non-cached / cached context tokens per output token.

Speedup. In the streaming multi-agent system, the latency speedup is jointly governed by pre-fill, decode, and KV-cache read speeds (Theorem 2). When $v_c \gg v_p \gg v_d$ (cache-read $\gg$ pre-fill $\gg$ decode speed), the upper bound reduces to $AS/(S + A - 1)$, the maximum achievable speedup, and becomes insensitive to the KV-cache hit rate.

Theorem 3 (Cost Ratio). *Under the same setup as Theorem 2, the cost ratio of Stream over Serial is:*

$$\frac{\text{Cost}_{\text{stream}}}{\text{Cost}_{\text{serial}}} = \rho \cdot \frac{r_{c_{pd}}(\alpha + r_{c_{cp}}\beta) + 1}{r_{c_{pd}}(1 + r_{po}/S) + 1}$$

where ρ : Stream-to-Serial output length ratio; $r_{c_{pd}}$: prefill-to-decode price ratio; $r_{c_{cp}}$: cache-to-prefill price ratio; α, β as in Theorem 2.

Cost. Although Stream increases API calls, the overall cost remains dominated by decode when the per-token prefill price c_p is far below the per-token decode price c_d ; KV-cache hits further reduce cost by replacing prefill with cache reads at price $c_c \ll c_p$. When pricing is decode-dominated ($r_{c_{pd}} \rightarrow 0$), the cost ratio (Theorem 3) reduces to ρ : cost depends on how many tokens each decodes. Each downstream agent in Stream naturally forms a shared prefix amenable to KV-cache reuse, and modern serving stacks such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024), together with recent agentic extensions (Liu et al., 2026; Wu et al., 2026; Pan et al., 2025; Ye et al., 2025a), steadily raise the achievable hit rate, so our cost advantage strengthens as this infrastructure matures.

Example. Instantiating both bounds with Claude Opus 4.6 data ($v_d \approx 39$ t/s, $v_p \approx 6,000$ t/s¹; $c_p = \$5$, $c_d = \$25$, $c_c = \$0.50$ per MTok²) and

¹<https://artificialanalysis.ai/models/claude-opus-4-6/providers>

²<https://www.anthropic.com/pricing>

$A = S = 4$: under full KV-cache hits, the latency speedup upper bound is $2.30\times$ and the cost ratio is 0.925ρ . At $\rho = 1$, Stream saves 7.5% over Serial.

Full derivations, numerical examples, and extension to DAG topologies in App. A.3 and A.4.

4 Results

4.1 Experimental Setup

Benchmarks. To assess robustness, we evaluate on eight benchmarks spanning competition mathematics (AIME 2025, AIME 2026, HMMT 2026), graduate-level science (GPQA-Diamond (Rein et al., 2023), HLE (Phan et al., 2025)), and program understanding (LiveCodeBench (Jain et al., 2025), with sub-tasks LCB-G: code generation, LCB-E: code execution, LCB-T: test output). Evaluation is conducted via OpenCompass (Cao et al., 2026); all configuration files are provided in App. A.7.

Baselines. We compare against two baselines: (1) Single, a single agent given only the query; and (2) Serial, a pipeline where each agent waits for its predecessor to finish. STREAMMA inherits Serial’s prompts and decoding, differing only in transmission granularity: Serial transmits the full output, Stream transmits each step on completion.

Topologies. Unless otherwise specified, we evaluate four-agent DAG topologies (A^0, A^1, A^2, A^3): Chain ($A^0 \rightarrow A^1 \rightarrow A^2 \rightarrow A^3$); Tree ($A^0 \rightarrow \{A^1, A^2\} \rightarrow A^3$, with A^1, A^2 in parallel); and Graph (Chain plus an extra edge $A^0 \rightarrow A^2$). These three cover the canonical DAG primitives (linear, branching, and shortcut), giving a built-in sweep of connectivity rather than a single design choice.

Implementation Details. We experiment on Claude Opus 4.6-High with thinking enabled, GPT-5.4-None and GLM-5.2-None without, following each provider’s API defaults; later references inherit these settings unless otherwise noted. Our LLM-as-judge evaluations use GPT-5.4-None. System prompts are provided in App. A.9. All numbers are averaged over 3 runs per {backbone, topology} cell, increased to 8 on AIME 2025/26 and HMMT 2026 due to their smaller test sets.

4.2 Effectiveness Analysis

4.2.1 Quantitative Comparison

STREAMMA outperforms Serial in every Avg. cell across all backbones (Tab. 1), whether thinking is on or off, larger gains when on. STREAMMA only appends a one-line END_STEP boundary to each downstream agent’s system prompt and a short

Model	Topo	Method	AIME25	AIME26	HMMT26	GPQA-D	HLE	LCB-G	LCB-E	LCB-T	Avg.
Claude Opus 4.6	–	Single	67.50	60.00	48.11	83.67	18.60	90.25	77.94	84.31	66.30
	Chain	Serial	80.42	72.08	63.26	85.86	23.90	91.33	78.64	92.38	73.48
		STREAMMA	92.50	89.58	85.61	87.37	26.97	91.50	84.41	95.63	81.70
	Tree	Serial	86.25	86.25	75.00	85.18	24.82	91.92	88.45	97.59	79.43
		STREAMMA	93.34	87.92	82.20	85.86	25.07	94.00	94.57	99.55	82.81
	Graph	Serial	77.92	71.67	61.75	85.69	22.17	90.08	75.78	98.27	72.92
		STREAMMA	95.83	87.92	82.58	86.53	27.68	92.17	95.27	98.72	83.34
	GPT-5.4	–	Single	55.83	71.25	40.53	77.95	12.08	91.08	92.48	96.68
Chain		Serial	60.00	70.42	54.55	75.08	14.66	90.08	97.43	99.02	70.16
		STREAMMA	61.25	72.50	59.10	80.30	14.94	91.17	99.30	99.47	72.25
Tree		Serial	59.17	75.83	56.07	76.77	14.83	88.33	93.81	99.25	70.51
		STREAMMA	62.08	75.83	58.34	78.12	15.74	89.50	94.78	99.17	71.70
Graph		Serial	60.00	74.17	52.65	78.45	14.04	92.25	99.51	98.80	71.23
		STREAMMA	62.50	75.42	56.44	79.63	16.13	93.08	99.79	99.32	72.79
GLM-5.2		–	Single	40.42	44.17	32.19	69.02	7.83	80.08	70.77	78.05
	Chain	Serial	61.25	65.84	57.20	77.61	12.13	85.92	76.34	97.36	66.71
		STREAMMA	75.42	78.75	67.05	79.63	15.18	89.67	75.71	97.66	72.38
	Tree	Serial	61.25	64.17	48.11	76.94	12.31	87.75	83.09	96.15	66.22
		STREAMMA	76.25	80.83	65.16	79.13	16.17	88.92	84.13	96.61	73.40
	Graph	Serial	52.92	61.67	50.76	77.27	11.94	85.67	74.53	96.00	63.85
		STREAMMA	73.75	84.17	65.16	77.94	15.12	89.25	77.87	97.29	72.57

Table 1: **Effectiveness across eight benchmarks and three topologies.** STREAMMA (gray) vs. Single and Serial on Claude Opus 4.6-High, GPT-5.4-None, and GLM-5.2-None. Avg. (%): unweighted mean; **bold**: higher in each {Topology, Method} pair. Single: one row per backbone, no topology axis. Each cell averages 3 runs (8 on AIME 2025/26 and HMMT 2026 due to small test sets). STREAMMA outperforms both baselines in every Avg. cell.

solver preamble for the root agent; all other prompt content, decoding hyperparameters, and runtime settings remain identical to Serial, ruling out extra prompt engineering as the source of improvement. On Claude Opus 4.6, STREAMMA lifts Avg. accuracy by +7.3 pp averaged over the three topologies (peak +22.4 pp on HMMT 2026, Chain).

STREAMMA’s gain over Serial scales inversely with Serial’s strength. The three topologies span a wide Serial baseline on Claude (Avg. 72.92–79.43), giving a built-in sweep of Serial strength rather than a topology comparison: at Serial Avg. 79.43 (Tree) the gain is +3.38, while at 73.48 (Chain) and 72.92 (Graph) it grows to +8.22 and +10.42. The same trend appears within a single benchmark on HMMT 2026, where Serial baselines of 75.00 / 63.26 / 61.75 (Tree / Chain / Graph) correspond to gains of +7.20 / +22.35 / +20.83. Near the ceiling (e.g., GPT-5.4 LCB-T at ~99), cells move by less than 0.6 pp. [Theorem 1](#) captures this: STREAMMA’s gain comes from weakening the impact of Serial’s step-level errors, so a Serial run

with high p_j leaves few errors to weaken, while one with low p_j leaves many and gains more. GPT-5.4 and GLM-5.2 exhibit the same trend, confirming the pattern is backbone- and topology-agnostic.

4.2.2 Case Study

Setup. On a GPQA-Diamond question, we run a two-agent chain ($\text{Agent}^1 \rightarrow \text{Agent}^2$, $S=8$) that reuses the same Chain prompts as [Sec. 4.2.1](#), and binarize each step of Agent^1 to $p_j \in \{0, 1\}$ via LLM-as-judge (1 if correct, 0 otherwise) for an instantiation of [Theorem 1](#). [Fig. 2](#) visualizes STREAMMA’s per-step verdicts of Agent^1 ; the question and per-step transcripts are deferred to [App. A.5](#).

Observations. Single, a single-agent baseline, returns the wrong answer. In Stream, Agent^1 produces the trajectory in [Fig. 2\(a\)](#): steps 1–2 are correct, Agent^1 first errors at step 3, and steps 3–8 are all incorrect. In Serial, an independent run of Agent^1 produces the same failure pattern. This matches Case I.b of [Theorem 1](#): a reliable head and an unreliable tail. Serial only sees Agent^1 ’s full output, so Agent^2 inherits the wrong tail and

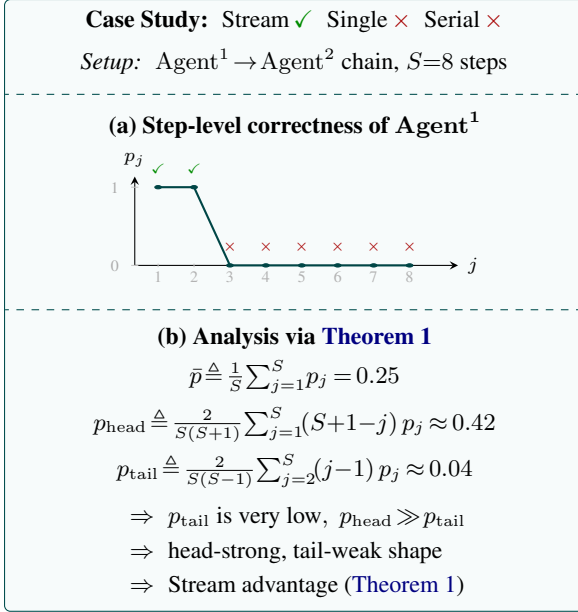


Figure 2: **Case study for Theorem 1.** (a) Verdicts of Agent¹ (✓: $p_j=1$; ✗: $p_j=0$). (b) $\bar{p}/p_{\text{head}}/p_{\text{tail}}$ place this run in Case I.b, the Stream-advantage regime.

returns the wrong answer. Stream forwards the reliable head by step 2; by the time the unreliable tail arrives, Agent² has already re-derived the answer from this prefix and returns the correct one. This is the Stream advantage predicted by Theorem 1.

4.2.3 Step-Level Perturbation

Setup. For a cleaner verification of Theorem 1, we construct a controlled experiment on an Agent¹ → Agent² chain with $S=4$ steps. We use an LLM to construct two parallel Agent¹ trajectories: a clean one deriving the gold answer and a perturbed counterpart that diverges from the clean one toward a wrong answer. A 4-bit perturbation pattern swaps in □ (clean) or ■ (perturbed) at each step, yielding a fixed Agent¹ output; Agent² is then run on it under Serial or Stream, with accuracy averaged over 100 runs per pattern. Results on 6 patterns in Fig. 3. Details in App. A.6.

Observations. The two pattern families produce opposite outcomes: Stream wins on tail-perturbed patterns (up to +24.0 pp), Serial wins on head-perturbed patterns (down to −36.0 pp). Tail-perturbed patterns (□□□■, □□■■, □■■■) all fall in Case I; Stream’s advantage shrinks monotonically as the perturbed tail extends (+24.0 → +19.0 → +5.0), tracking the predicted decay as p_{tail} drops. Head-perturbed patterns span two regimes: ■□□□ and ■■■□ fall in Case II, while the heavily poisoned ■■■■ reaches Case III. In both,

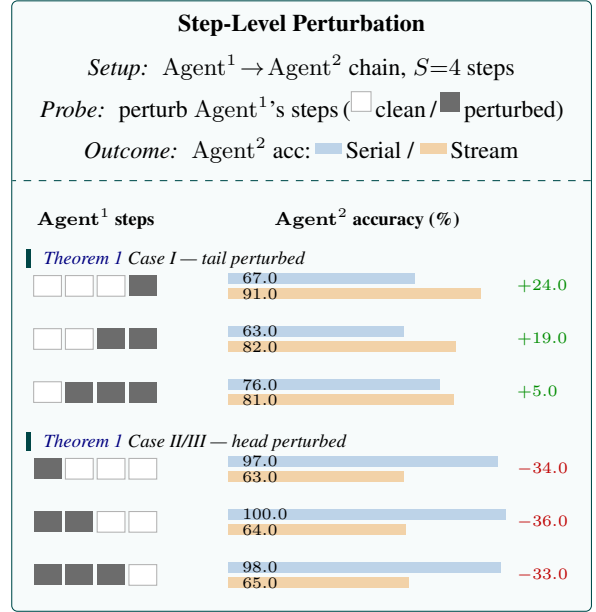


Figure 3: **Step-Level Perturbation.** Fixing Agent¹’s output, we perturb its steps and measure Agent²’s accuracy; green/red mark Stream’s gain/loss over Serial.

Role	GLM-5.2		GPT-5.4	
	Serial	STREAMMA	Serial	STREAMMA
Adversarial Verify	50.00	59.85	54.55	56.44
Debate	57.58	63.26	56.06	61.36
Decompose & Merge	40.53	45.08	50.38	56.82
Reviewer Corrector	59.85	67.80	51.89	56.44
Self Consistency	36.36	40.15	45.45	46.21

Table 2: **Generalization to roles.** Accuracy (%) across five roles with varying agent counts and topologies.

the curves are nearly flat regardless of depth: Serial holds at ~100% while Stream trails by ~35 pp, trapped by the perturbed prefix. Mirror pairs expose Theorem 1’s head–tail asymmetry: □□□■ vs. ■■■□ flips Stream’s gap from +24.0 to −34.0 pp.

4.2.4 Role and Tool Generalization

Roles. To verify that STREAMMA’s gain generalizes beyond a fixed role design, we evaluate five role protocols on HMMT 2026 with GLM-5.2 and GPT-5.4 over 8 runs each. The protocols differ in agent count, DAG structure, and role semantics: (1) Adversarial Verify: 3 agents, triangle DAG ($A^0 \rightarrow A^1$, $A^0 \rightarrow A^2$, $A^1 \rightarrow A^2$); Solver, Attacker, Judge. (2) Debate: 2 debaters over two rounds, a 4-node chain ($A^0 \rightarrow A^1 \rightarrow A^0 \rightarrow A^1$); proponent/opponent. (3) Decompose & Merge: 4 agents, tree ($A^0 \rightarrow \{A^1, A^2\} \rightarrow A^3$); decomposer, two parallel solvers, integrator. (4) Reviewer Corrector: 3 agents, chain ($A^0 \rightarrow A^1 \rightarrow A^2$); solver, re-

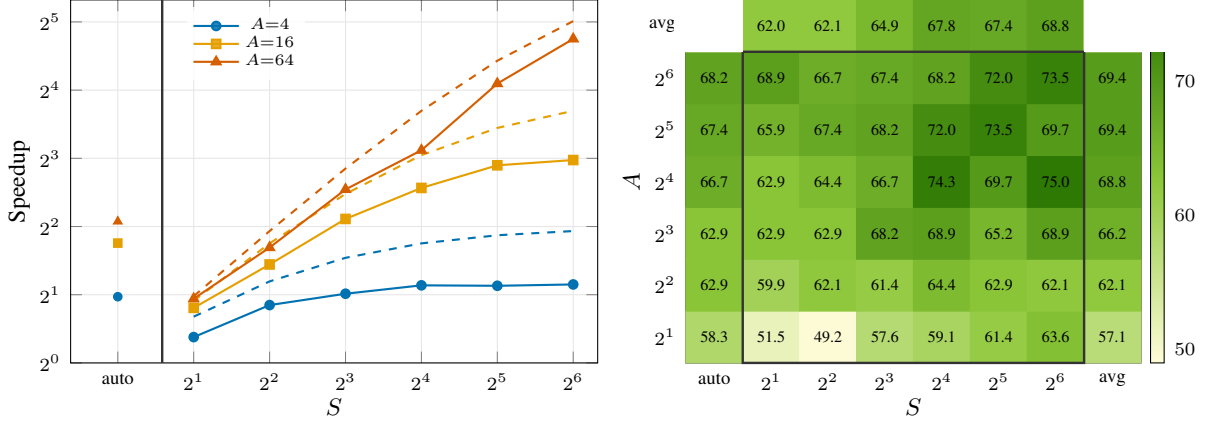


Figure 4: **Step-level scaling law.** *Left:* speedup scaling in S ; measured (solid) vs. theoretical maximum speedup $AS/(S+A-1)$ from Theorem 2 (dashed). *Right:* accuracy scaling in S , with avg marginals (main block boxed).

Method	Loose / Strict (%) $\uparrow$	Cost (\$) $\downarrow$	Latency (s) $\downarrow$	Speedup $\uparrow$
Serial	66.1 / 23.9	6.42	34.1	1.00 $\times$
STREAMMA	67.2 / 24.3	6.39	29.1	1.17 $\times$

Table 3: **Generalization to tools.** Answer quality and efficiency on FanOutQA with search-tool calls.

viewer, final corrector. (5) Self Consistency: 3 agents, tree ($\{A^0, A^1\} \rightarrow A^2$); two parallel solvers, adjudicator. As Tab. 2 shows, STREAMMA improves over Serial in all ten {role, backbone} cells by +0.76 to +9.85 pp, generalizing readily across diverse role assignments and DAG structures.

Tools. To verify that STREAMMA’s gain generalizes to tool-augmented agents, we evaluate a Solver $\rightarrow$ Reviewer chain ($A^0 \rightarrow A^1$) with GPT-5.4 on the 310-question FanOutQA dev set (Zhu et al., 2024), a multi-hop factual QA benchmark where each question decomposes into a median of 6 sub-questions, each requiring an independent web search. The Solver issues one [SEARCH] call per sub-question and emits END_STEP after each search result returns. Under STREAMMA, each completed step is immediately forwarded to the Reviewer, which begins verification while the Solver continues searching. Serial and STREAMMA differ only in this transmission timing, with prompts kept identical. We adopt FanOutQA’s official loose and strict accuracy: loose credits the fraction of gold answers recalled, strict requires all of them. Averaged over 3 runs and reported in Tab. 3, STREAMMA improves over Serial on loose and strict accuracy by +1.1 and +0.4 pp while also achieving 1.17 $\times$ speedup at lower cost, generalizing readily to agents that call external tools.

4.3 Efficiency Analysis

4.3.1 Step-Level Scaling Law

Setup. We adopt the Chain Agent $^1 \rightarrow \dots \rightarrow$ Agent ^{A} with each agent producing S reasoning steps, and sweep $A \in \{2, 4, 8, 16, 32, 64\}$ and $S \in \{2, 4, 8, 16, 32, 64, \text{auto}\}$ on HMMT 2026; $S=\text{auto}$ lets Agent ^{1} autonomously decide the step count. Each (A, S) cell reports per-question averages of speedup and accuracy over 4 independent runs, where speedup is the sum of per-agent API call times divided by Stream’s measured wall-clock runtime. KV-cache reuse is disabled; for visual clarity the left panel of Fig. 4 plots $A \in \{4, 16, 64\}$ only, while the full 6×6 grid appears in the right heatmap. This $AS/(S+A-1)$ form is the simplification of Theorem 2 under our no-KV-cache setup with decode much slower than prefill ($r_{v_{dp}} \rightarrow 0$).

Observations. (i) Measured speedup tracks the $AS/(S+A-1)$ scaling pattern across the (A, S) grid (Fig. 4, left), validating Theorem 2; at the $A=64, S=64$ corner the measured speedup reaches 26.9 $\times$, attaining 83% of the theoretical bound (32.3 $\times$); the gap arises because the bound assumes $r_{v_{dp}} \rightarrow 0$, while actual $r_{v_{dp}} > 0$ in GPT-5.4. Notably, the $S=\text{auto}$ markers fall well short of the large- S speedups across all A , because LLMs do not autonomously scale up their step count and default to a coarse granularity; the large- S regime must be explicitly unlocked through prompting. (ii) Accuracy increases jointly with A and S across the heatmap (Fig. 4, right). The right avg column reproduces agent-count scaling (Qian et al., 2025), while the top avg row is near-monotone in S , evidencing a step-level scaling law: at fixed A , accuracy improves with more reasoning steps. The two scaling

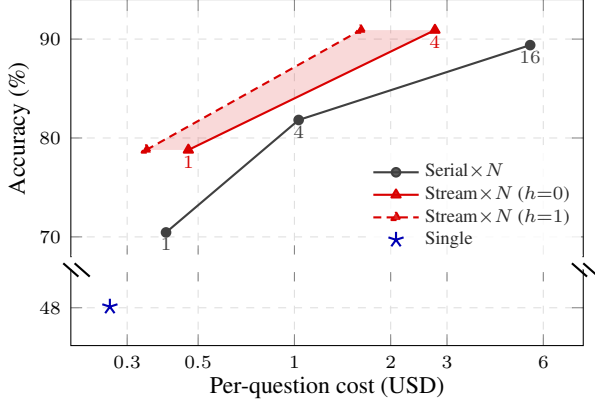


Figure 5: **Cost-accuracy Pareto frontiers.** Each frontier tracks accuracy vs. cost as $N \in \{1, 4, 16\}$ chain replicas run in parallel and majority-vote on the final answer; larger N trades higher compute for higher accuracy. Red shaded: KV-cache hit rate $h \in (0, 1)$, bounded by Stream $\times N$ at $h=0$ (solid) and $h=1$ (dashed).

dimensions (agent count and step count) are complementary, not redundant: at $A=64$, the $S=\text{auto}$ baseline (LLM-decided S) reaches 68.2%, and increasing S to 64 further raises accuracy to 73.5%, indicating that step-level scaling delivers measurable gains beyond agent-count scaling alone.

4.3.2 Cost Analysis

Setup. We adopt the Chain Agent¹ $\rightarrow$ Agent² $\rightarrow$ Agent³ with each agent producing $S=3$ reasoning steps (prompts adapted from Sec. 4.2), and sweep both Stream $\times N$ and Serial $\times N$ over $N \in \{1, 4, 16\}$ parallel chain replicas with majority voting on HMMT 2026 with Claude Opus 4.6 (API pricing: \$5/\$25/\$0.5 per million input/output/cache-read tokens). Each N point reports per-question averages of cost and accuracy over 4 independent runs. Let $h \in [0, 1]$ denote the KV-cache hit rate.

Observations. (i) The Stream $\times N$ frontier strictly Pareto-dominates Serial $\times N$: Stream $\times 4$ (\$2.75, 90.9%) achieves higher accuracy than Serial $\times 16$ (\$5.46, 89.4%) at half the cost, despite Serial running 4 $\times$ more replicas, jointly confirming Stream’s accuracy advantage (Theorem 1) and cost advantage (Theorem 3). (ii) The shaded KV-cache band ($h \in (0, 1)$) compresses each Stream $\times N$ point leftward by up to $\sim 1.7 \times$ with no accuracy loss (e.g., Stream $\times 4$ drops from \$2.75 to \$1.61 at fixed 90.9%); this $h=0 \rightarrow h=1$ compression ratio is governed by the cache-to-prefill price ratio r_{cp} in Theorem 3, and further widens the Pareto gap. (iii) Without voting ($N=1$), Stream at full cache (\$0.34, 78.8%) strictly dominates Serial (\$0.40,

70.5%): the Pareto advantage stems from the protocol itself, not from voting amplification.

5 Conclusion

We introduced STREAMMA, a reasoning-step-level streaming multi-agent system for LLM reasoning that replaces the prevailing “generate-then-transfer” paradigm with immediate per-step forwarding. The central finding is counter-intuitive: streaming reduces latency and improves effectiveness, because the non-uniform quality structure of multi-step LLM reasoning makes when context arrives matter more than how much context arrives. We formalized this mechanism through three closed-form theorems covering effectiveness ordering, speedup upper bound, and cost ratio, all empirically validated across eight benchmarks, three frontier LLMs, and three topologies. Beyond the protocol, we uncovered a “step-level scaling law”: a dimension orthogonal to agent-count scaling that monotonically improves effectiveness and speedup, suggesting that the design space of multi-agent reasoning systems is richer than recognized.

6 Limitations

STREAMMA pipelines reasoning at the granularity of steps, and thus applies to tasks whose solution admits step decomposition. Modern reasoning workloads (mathematics, code, science) satisfy this property under chain-of-thought prompting, the de facto standard. Tasks whose output is atomic (e.g., single-token classification) offer no intermediate steps to stream and fall outside this regime; this reflects the task’s structure rather than a constraint of STREAMMA, and equally bounds any step-based reasoning paradigm (e.g., chain-of-thought).

Furthermore, the stream protocol is not universally optimal across all step-correctness profiles. Theorem 1 characterizes six regimes: stream execution is strictly preferable only when agents exhibit head-strong, tail-weak correctness patterns relative to the breakeven p^* ; in the remaining regimes, serial or single execution may match or exceed it. Rather than weakening STREAMMA, this boundary is precisely predicted by our theoretical framework. Theorem 1 therefore serves as a principled protocol selector: given the step-correctness profile and the breakeven p^* of a target task, practitioners can directly consult the theorem to determine which execution protocol best suits their workload.

References

- Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. [Medusa: Simple LLM inference acceleration framework with multiple decoding heads](#). In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 5209–5235. PMLR / OpenReview.net.
- Maosong Cao, Kai Chen, Haodong Duan, Yixiao Fang, Zhiwei Fei, Tong Gao, Ge Jiaye, Mo Li, Hongwei Liu, Junnan Liu, Yuan Liu, Chengqi Lyu, Han Lyu, Ningsheng Ma, Zerun Ma, Yu Sun, Zhiyong Wu, Linchen Xiao, Jun Xu, and 10 others. 2026. [Opencompass: A universal evaluation platform for large language models](#). *CoRR*, abs/2605.19276.
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. [Accelerating large language model decoding with speculative sampling](#). *CoRR*, abs/2302.01318.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024. [Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2024. [Improving factuality and reasoning in language models through multiagent debate](#). In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 11733–11763. PMLR / OpenReview.net.
- Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. 2025. [Cache-to-cache: Direct semantic communication between large language models](#). *CoRR*, abs/2510.03215.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. [Metagpt: Meta programming for a multi-agent collaborative framework](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Fengting Liao, Yi-Chang Chen, Sattar Vakili, and Dasha Shiu. 2025. [Group think: Multiple concurrent reasoning agents collaborating at token level granularity](#). *CoRR*, abs/2505.11107.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. [Livecodebench: Holistic and contamination free evaluation of large language models for code](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient memory management for large language model serving with pagedattention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. [Fast inference from transformers via speculative decoding](#). In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. [CAMEL: communicative agents for "mind" exploration of large language model society](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. [More agents is all you need](#). *Trans. Mach. Learn. Res.*, 2024.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. [Let’s verify step by step](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, and Esha Choukse. 2026. [Droidspeak: KV cache sharing across fine-tuned model variants](#). In *23rd USENIX Symposium on Networked Systems Design and Implementation, NSDI 2026, Renton, WA, May 4-6, 2026*, pages 319–338. USENIX Association.
- Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023. [Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization](#). *CoRR*, abs/2310.02170.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder,

- Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. [Self-refine: Iterative refinement with self-feedback](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. [Efficient large-scale language model training on GPU clusters using megatron-lm](#). In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, page 58. ACM.
- Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2024. [Skeleton-of-thought: Prompting llms for efficient parallel generation](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Zaifeng Pan, Ajikumar Patel, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. 2025. [Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows](#). In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*.
- Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, and 3 others. 2025. [Humanity’s last exam](#). *CoRR*, abs/2501.14249.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. [Chatdev: Communicative agents for software development](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, *ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15174–15186. Association for Computational Linguistics.
- Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. [Scaling large language model-based multi-agent collaboration](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. [GPQA: A graduate-level google-proof q&a benchmark](#). *CoRR*, abs/2311.12022.
- Michael Rizvi-Martel, Satwik Bhattamishra, Neil Rathi, Guillaume Rabusseau, and Michael Hahn. 2025. [Benefits and limitations of communication in multi-agent reasoning](#). *CoRR*, abs/2510.13903.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflexion: language agents with verbal reinforcement learning](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Guinan Su, Yanwu Yang, Xueyan Li, and Jonas Geiping. 2026. [Multi-stream llms: Unblocking language models with parallel streams of thoughts, inputs and outputs](#). *CoRR*, abs/2605.12460.
- Junlong Tong, Zilong Wang, YuJie Ren, Peiran Yin, Hao Wu, Wei Zhang, and Xiaoyu Shen. 2026. [From static inference to dynamic interaction: A survey of streaming large language models](#). In *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pages 10237–10263. Association for Computational Linguistics.
- Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. [Solving math word problems with process- and outcome-based feedback](#). *CoRR*, abs/2211.14275.
- Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2025a. [Mixture-of-agents enhances large language model capabilities](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Junlin Wang, Jue Wang, Zhen Xu, Ben Athiwaratkun, Bhuwan Dhingra, Ce Zhang, and James Zou. 2025b. [Staircase streaming for low-latency multi-agent inference](#). *CoRR*, abs/2510.05059.
- Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. [Math-shepherd: Verify and reinforce llms step-by-step without human annotations](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, *ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9426–9439. Association for Computational Linguistics.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. [Autogen: Enabling next-gen LLM applications via multi-agent conversation framework](#). *CoRR*, abs/2308.08155.
- Yongtong Wu, Shaoyuan Chen, Yinmin Zhong, Rilin Huang, Yixuan Tan, Wentao Zhang, Liyue Zhang, Shangyan Zhou, Yuxuan Liu, Shunfeng Zhou, Mingxing Zhang, Xin Jin, and Panpan Huang. 2026. [Dual-path: Breaking the storage bandwidth bottleneck in agentic LLM inference](#). *CoRR*, abs/2602.21548.
- Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2025. [When more is less: Understanding chain-of-thought length in llms](#). *CoRR*, abs/2502.07266.
- Zhen Yang, Mingyang Zhang, Feng Chen, Ganggui Ding, Liang Hou, Xin Tao, and Ying-Cong Chen. 2026. [Less is more: Improving LLM reasoning with minimal test-time intervention](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2026, San Diego, California, United States, July 2-7, 2026, pages 20124–20137. Association for Computational Linguistics.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. [Tree of thoughts: Deliberate problem solving with large language models](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. 2025a. [KVCOMM: online cross-context kv-cache communication for efficient llm-based multi-agent systems](#). In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*.
- Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. 2025b. [Speculative actions: A lossless framework for faster agentic systems](#). *CoRR*, abs/2510.04371.
- Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. 2023. [Exchange-of-thought: Enhancing large language model capabilities through cross-model communication](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 15135–15153. Association for Computational Linguistics.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. [Aflow: Automating agentic workflow generation](#). In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. [Sglang: Efficient execution of structured language model programs](#). In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.
- Andrew Zhu, Alyssa Hwang, Liam Dugan, and Chris Callison-Burch. 2024. [Fanoutqa: A multi-hop, multi-document question answering benchmark for large language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, ACL 2024, Bangkok, Thailand, August 11-16, 2024, pages 18–37. Association for Computational Linguistics.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. [Gptswarm: Language agents as optimizable graphs](#). In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 62743–62767. PMLR / OpenReview.net.

A Appendix

Appendix Contents

A.1	Notation Summary	13
A.2	Detailed Effectiveness Analysis	13
	Proof of Core Identities	14
	Proof of Theorem 1	15
	Discussion of Practical Scenarios	17
A.3	Detailed Speedup Analysis	18
	Latency of Serial Protocol	18
	Latency of Stream Protocol	18
	Proof of Theorem 2	19
	Numerical Example: Speedup	21
A.4	Detailed Cost Analysis	22
	Cost of Serial Protocol	22
	Cost of Stream Protocol	22
	Proof of Theorem 3	22
	Numerical Example: Cost	23
A.5	Case Study of Theorem 1	25
A.6	Step-Level Perturbation Materials	26
A.7	Testing Setup and Files	27
A.8	Stream Protocol on DAGs	27
A.9	System Prompts	28
A.10	Information About Use of AI Assistants	29
A.11	Potential Risks	29

A.1 Notation Summary

Tab. 4 summarizes the notation used in the theoretical analysis, covering parameters for the effectiveness and efficiency (speedup and cost) analyses. For brevity, derivations in the appendix use a chain topology as a running example; the generalization to arbitrary DAGs is discussed at the end of each proof section.

Table 4: Notation summary.

Symbol	Description
<i>General parameters</i>	
A	Number of agents
S	Number of reasoning steps
o_s^a	Number of output tokens of Agent ^a at step s
O_a	Total number of output tokens of Agent ^a ; $O_a \triangleq \sum_{s=1}^S o_s^a$
$\bar{O}$	Per-agent per-step output token count, averaged across agents; $\bar{O} \triangleq \frac{1}{A} \sum_{a=1}^A \frac{O_a}{S}$
P_a	Number of tokens in system prompt of Agent ^a
$\bar{P}$	Average system prompt length; $\bar{P} \triangleq \frac{1}{A} \sum_{a=1}^A P_a$
O_Σ	Total number of output tokens across all agents; $O_\Sigma \triangleq \sum_{a=1}^A O_a$
<i>Effectiveness analysis (Agent^a $\rightarrow$ Agent^{a+1})</i>	
$\text{sCorr}^{\text{mode}}$	Mean step-level correctness, $\text{mode} \in \{\text{stream}, \text{serial}, \text{single}\}$
p_j	Correctness probability of step j ; $p_j \in [0, 1]$
δ	Expected downstream step-correctness gain from a correct upstream step in context
ε	Expected downstream step-correctness drop from an incorrect upstream step in context
μ_j	Expected step-correctness change from upstream step j ; $\mu_j \triangleq p_j \delta - (1 - p_j) \varepsilon$
p^*	Minimum step correctness for context to be beneficial; $p^* \triangleq \varepsilon / (\delta + \varepsilon)$
$\bar{p}$	Mean step correctness; $\bar{p} \triangleq \frac{1}{S} \sum_{j=1}^S p_j$
p_{tail}	Tail-weighted mean step correctness; $p_{\text{tail}} \triangleq \frac{2}{S(S-1)} \sum_{j=2}^S (j-1) p_j$
p_{head}	Head-weighted mean step correctness; $p_{\text{head}} \triangleq \frac{2}{S(S+1)} \sum_{j=1}^S (S+1-j) p_j$
<i>Speedup analysis</i>	
C_s^a	Total number of context tokens in the s -th call of Agent ^a
h_s^a	KV-cache hit rate of Agent ^a 's s -th call; $h_s^a \in [0, 1]$
α	Average number of non-cached prefill tokens per output token; $\alpha \triangleq \frac{\sum_{a=1}^A \sum_{s=1}^S (1-h_s^a) C_s^a}{AS\bar{O}}$
β	Average number of cache-hit tokens per output token; $\beta \triangleq \frac{\sum_{a=1}^A \sum_{s=1}^S h_s^a C_s^a}{AS\bar{O}}$
v_d, v_p, v_c	Decode / prefill / cache-read speed (tok/s); $v_c \gg v_p \gg v_d$
r_{vdp}	Decode-to-prefill speed ratio; $r_{vdp} \triangleq v_d/v_p$
r_{vdc}	Decode-to-cache-read speed ratio; $r_{vdc} \triangleq v_d/v_c$
r_{po}	Prompt-to-per-step-output ratio; $r_{po} \triangleq \bar{P}/\bar{O}$
<i>Cost analysis</i>	
c_d, c_p, c_c	Per-token price: decode / prefill / cache; $c_d \gg c_p \gg c_c$
$r_{c_{cp}}$	Cache-to-prefill price ratio; $r_{c_{cp}} \triangleq c_c/c_p$
$r_{c_{pd}}$	Prefill-to-decode price ratio; $r_{c_{pd}} \triangleq c_p/c_d$
ρ	Ratio of total output token counts: Stream vs. Serial; $\rho \triangleq O_\Sigma^{\text{stream}}/O_\Sigma^{\text{serial}}$

A.2 Detailed Effectiveness Analysis

Let $\text{sCorr}^{\text{mode}}$, $\text{mode} \in \{\text{serial}, \text{stream}, \text{single}\}$, denote the mean step-level expected correctness of the downstream agent. *This is a theoretical quantity positively correlated with task-level accuracy: a higher fraction of correct reasoning steps makes a correct final answer more likely.*

Theorem 1 (Effectiveness Ordering). *Depending on how $\bar{p}$, p_{head} , p_{tail} compare to p^* , the sCorr ordering among the three modes falls into six cases:*

(I) Stream advantage [$p_{\text{head}} > p^*$ and $p_{\text{tail}} < p^*$]:

(a) If $\bar{p} > p^*$: $\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}}$

(b) If $\bar{p} < p^*$: $\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}}$

(II) Serial advantage [$\bar{p} > p^*$ and $p_{\text{tail}} > p^*$]:

(a) If $p_{\text{head}} > p^*$: $\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}}$

(b) If $p_{\text{head}} < p^*$: $\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}}$

(III) Single advantage [$p_{\text{head}} < p^*$ and $\bar{p} < p^*$]:

(a) If $p_{\text{tail}} < p^*$: $\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}}$

(b) If $p_{\text{tail}} > p^*$: $\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}}$

A.2.1 Proof of Core Identities

Consider adjacent agents Agent^a (upstream) and Agent^{a+1} (downstream) in the chain: the upstream produces S -step reasoning $A_1^a, \dots, A_S^a$; the downstream produces outputs $A_1^{a+1}, \dots, A_S^{a+1}$. The three modes differ only in which upstream steps are visible to the downstream when generating A_s^{a+1} :

Mode	Condition when predicting A_s^{a+1}
Single	$\mathbb{P}(A_s^{a+1} \mid A_{<s}^{a+1})$
Stream	$\mathbb{P}(A_s^{a+1} \mid A_{<s}^{a+1}, A_{\leq s}^a)$
Serial	$\mathbb{P}(A_s^{a+1} \mid A_{<s}^{a+1}, A_{1:S}^a)$

where $\mathbb{P}(A_s^{a+1} \mid \cdot)$ is the expected step-level correctness of A_s^{a+1} , abbreviated $\mathbb{P}^{\text{mode}}(A_s^{a+1})$ for each row.

Let Δ_j denote the change in downstream step-level correctness when step j is added to a context containing steps $1, \dots, j-1$. We parametrize Δ_j by conditional expectations: $\delta \triangleq \mathbb{E}[\Delta_j \mid A_j^a \text{ correct}, A_{<j}^a \text{ in context}] > 0$ is the expected gain when step j is correct, and $\varepsilon \triangleq \mathbb{E}[-\Delta_j \mid A_j^a \text{ incorrect}, A_{<j}^a \text{ in context}] > 0$ the expected loss when step j is incorrect. We assume δ, ε are step-position-independent for simplicity; in general, (δ, ε) generalize to vectors $(\delta_j, \varepsilon_j)$ and the same derivation applies. Since step j is correct with probability p_j , the expected step-correctness change is

$$\mu_j \triangleq p_j \cdot \delta - (1 - p_j) \cdot \varepsilon = p_j(\delta + \varepsilon) - \varepsilon.$$

The breakeven threshold $p^* \triangleq \varepsilon/(\delta + \varepsilon)$ satisfies $\mu_j > 0$ if and only if $p_j > p^*$. With position-dependent δ_j, ε_j , the threshold becomes $\varepsilon_j/(\delta_j + \varepsilon_j)$ and may vary by step. The three modes differ in upstream context seen: Single sees none, Stream sees a prefix, Serial sees all S steps. With Single as the baseline, summing the corresponding μ_j gives the step-level correctness gains of Stream and Serial:

$$\mathbb{P}^{\text{stream}}(A_s^{a+1}) - \mathbb{P}^{\text{single}}(A_s^{a+1}) = \sum_{j=1}^s \mu_j, \quad \mathbb{P}^{\text{serial}}(A_s^{a+1}) - \mathbb{P}^{\text{single}}(A_s^{a+1}) = \sum_{j=1}^S \mu_j$$

By definition, $\text{sCorr}^{\text{mode}} \triangleq \frac{1}{S} \sum_{s=1}^S \mathbb{P}^{\text{mode}}(A_s^{a+1})$ averages step-level correctness over s . Applying the same average to the step-level differences above yields the sCorr gaps between modes:

$$\text{sCorr}^{\text{stream}} - \text{sCorr}^{\text{single}} = \frac{1}{S} \sum_{s=1}^S \sum_{j=1}^s \mu_j = \frac{1}{S} \sum_{j=1}^S (S - j + 1) \mu_j \quad (\text{a})$$

$$\text{sCorr}^{\text{serial}} - \text{sCorr}^{\text{single}} = \frac{1}{S} \sum_{s=1}^S \sum_{j=1}^S \mu_j = \sum_{j=1}^S \mu_j \quad (\text{b})$$

Subtracting (a) from (b):

$$\text{sCorr}^{\text{serial}} - \text{sCorr}^{\text{stream}} = \frac{1}{S} \sum_{j=1}^S (j-1) \mu_j \quad (\text{c})$$

The signs of (a)–(c) follow from how the head (p_{head}), mean ($\bar{p}$), and tail (p_{tail}) per-step correctness compare to p^* (Tab. 4), fixing the ordering among Single, Stream, and Serial and establishing [Theorem 1](#). **Remark** (Boundary cases). Equality in any condition yields the corresponding equality in the ordering.

A.2.2 Proof of Theorem 1

We prove each case by inverting identities (a)–(c): given a target sCorr ordering, the signs of (a)–(c) determine conditions on p_{head} , $\bar{p}$, and p_{tail} relative to p^* . These variables are not prescribed *a priori* but derived from the proof below; their parametric definitions (Tab. 4) absorb the resulting coefficients.

Case I.a ($\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}}$). This ordering requires (a) > 0 , (b) > 0 , (c) < 0 .

- *Expanding (b):*

$$\sum_{j=1}^S \mu_j = \sum_{j=1}^S [(\delta + \varepsilon)p_j - \varepsilon] = S[(\delta + \varepsilon)\bar{p} - \varepsilon],$$

which is positive if and only if $\bar{p} > p^*$.

- *Expanding (c):* Since the weight at $j = 1$ vanishes,

$$\begin{aligned} \frac{1}{S} \sum_{j=2}^S (j-1) \mu_j &= \frac{1}{S} \sum_{j=2}^S (j-1) [(\delta + \varepsilon)p_j - \varepsilon] \\ &= \frac{1}{S} \left[(\delta + \varepsilon) \sum_{j=2}^S (j-1)p_j - \varepsilon \sum_{j=2}^S (j-1) \right]. \end{aligned}$$

Since $\sum_{j=2}^S (j-1) = \frac{S(S-1)}{2}$ and $\sum_{j=2}^S (j-1)p_j = \frac{S(S-1)}{2} p_{\text{tail}}$ by definition:

$$(\text{c}) = \frac{S-1}{2} [(\delta + \varepsilon)p_{\text{tail}} - \varepsilon],$$

which is negative if and only if $p_{\text{tail}} < p^*$.

- *Sign of (a):* Since (b) > 0 and (c) < 0 , we have (a) $=$ (b) $-$ (c) > 0 , so $\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}}$ holds when $\bar{p} > p^*$ and $p_{\text{tail}} < p^*$, implying $p_{\text{head}} > p^*$.

Case I.b ($\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}}$). This ordering requires (a) > 0 , (b) < 0 , (c) < 0 .

- *Expanding (b):* By the same expansion as in Case I.a,

$$(\text{b}) = \sum_{j=1}^S \mu_j = S[(\delta + \varepsilon)\bar{p} - \varepsilon],$$

which is negative if and only if $\bar{p} < p^*$.

- *Expanding (a):* Multiplying both sides by S and substituting $\mu_j = (\delta + \varepsilon)p_j - \varepsilon$:

$$\begin{aligned} S \cdot (\text{a}) &= \sum_{j=1}^S (S+1-j) [(\delta + \varepsilon)p_j - \varepsilon] \\ &= (\delta + \varepsilon) \sum_{j=1}^S (S+1-j) p_j - \varepsilon \sum_{j=1}^S (S+1-j). \end{aligned}$$

Substituting $\sum_{j=1}^S (S+1-j) = \frac{S(S+1)}{2}$ and $\sum_{j=1}^S (S+1-j) p_j = \frac{S(S+1)}{2} p_{\text{head}}$ by definition:

$$(\mathbf{a}) = \frac{S+1}{2} [(\delta + \varepsilon) p_{\text{head}} - \varepsilon],$$

which is positive if and only if $p_{\text{head}} > p^*$.

- *Sign of (c):* Since $(\mathbf{c}) = (\mathbf{b}) - (\mathbf{a})$, $(\mathbf{b}) < 0$ and $(\mathbf{a}) > 0$ give $(\mathbf{c}) < 0$, so $\text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}}$ holds when $p_{\text{head}} > p^*$ and $\bar{p} < p^*$, implying $p_{\text{tail}} < p^*$.

Case II.a ($\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}}$). This ordering requires $(\mathbf{a}) > 0$, $(\mathbf{b}) > 0$, $(\mathbf{c}) > 0$.

- *Expanding (a):* By the same expansion as in Case I.b,

$$(\mathbf{a}) = \frac{S+1}{2} [(\delta + \varepsilon) p_{\text{head}} - \varepsilon],$$

which is positive if and only if $p_{\text{head}} > p^*$.

- *Expanding (c):* By the same expansion as in Case I.a,

$$(\mathbf{c}) = \frac{S-1}{2} [(\delta + \varepsilon) p_{\text{tail}} - \varepsilon],$$

which is positive if and only if $p_{\text{tail}} > p^*$.

- *Sign of (b):* Since $(\mathbf{b}) = (\mathbf{a}) + (\mathbf{c})$, both positive, $(\mathbf{b}) > 0$, so $\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{single}}$ holds when $p_{\text{head}} > p^*$ and $p_{\text{tail}} > p^*$, implying $\bar{p} > p^*$.

Case II.b ($\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}}$). This ordering requires $(\mathbf{a}) < 0$, $(\mathbf{b}) > 0$, $(\mathbf{c}) > 0$.

- *Expanding (b):* By the same expansion as in Case I.a,

$$\sum_{j=1}^S \mu_j = S [(\delta + \varepsilon) \bar{p} - \varepsilon],$$

which is positive if and only if $\bar{p} > p^*$.

- *Expanding (a):* By the same expansion as in Case I.b,

$$(\mathbf{a}) = \frac{S+1}{2} [(\delta + \varepsilon) p_{\text{head}} - \varepsilon],$$

which is negative if and only if $p_{\text{head}} < p^*$.

- *Sign of (c):* Since $(\mathbf{c}) = (\mathbf{b}) - (\mathbf{a})$, $(\mathbf{b}) > 0$ and $(\mathbf{a}) < 0$ give $(\mathbf{c}) > 0$, so $\text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}}$ holds when $\bar{p} > p^*$ and $p_{\text{head}} < p^*$, implying $p_{\text{tail}} > p^*$.

Case III.a ($\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}}$). This ordering requires $(\mathbf{a}) < 0$, $(\mathbf{b}) < 0$, $(\mathbf{c}) < 0$.

- *Expanding (a):* By the same expansion as in Case I.b,

$$(\mathbf{a}) = \frac{S+1}{2} [(\delta + \varepsilon) p_{\text{head}} - \varepsilon],$$

which is negative if and only if $p_{\text{head}} < p^*$.

- *Expanding (c):* By the same expansion as in Case I.a,

$$(\mathbf{c}) = \frac{S-1}{2} [(\delta + \varepsilon) p_{\text{tail}} - \varepsilon],$$

which is negative if and only if $p_{\text{tail}} < p^*$.

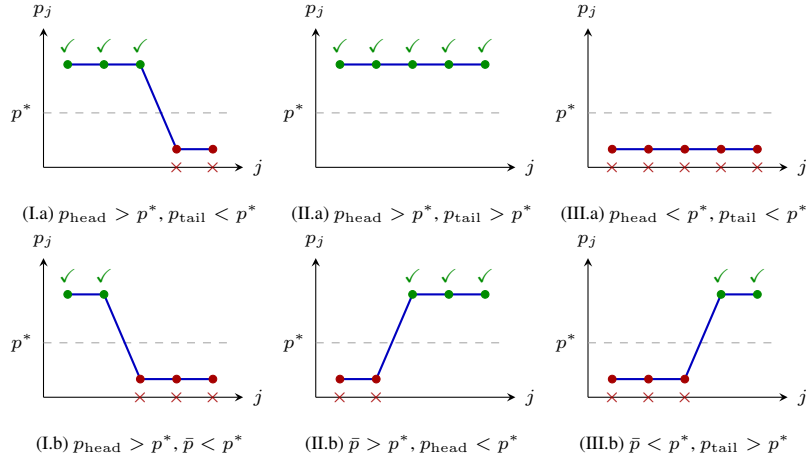


Figure 6: Six canonical step-correctness profiles p_j , $1 \leq j \leq S$ (solid lines) relative to the breakeven threshold p^* (dashed line), corresponding to the six cases of Theorem 1, organized into three advantage regimes (columns). Left column (Stream-advantage): Case I.a (top), Case I.b (bottom); middle column (Serial-advantage): Case II.a (top), Case II.b (bottom); right column (Single-advantage): Case III.a (top), Case III.b (bottom).

- *Sign of (b)*: Since $(b) = (a) + (c)$, both negative, $(b) < 0$, so $\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{stream}} > \text{sCorr}^{\text{serial}}$ holds when $p_{\text{head}} < p^*$ and $p_{\text{tail}} < p^*$, implying $\bar{p} < p^*$.

Case III.b ($\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}}$). This ordering requires $(a) < 0$, $(b) < 0$, $(c) > 0$.

- *Expanding (b)*: By the same expansion as in Case I.a,

$$\sum_{j=1}^S \mu_j = S[(\delta + \varepsilon)\bar{p} - \varepsilon],$$

which is negative if and only if $\bar{p} < p^*$.

- *Expanding (c)*: By the same expansion as in Case I.a,

$$(c) = \frac{S-1}{2}[(\delta + \varepsilon)p_{\text{tail}} - \varepsilon],$$

which is positive if and only if $p_{\text{tail}} > p^*$.

- *Sign of (a)*: Since $(a) = (b) - (c)$, $(b) < 0$ and $(c) > 0$ give $(a) < 0$, so $\text{sCorr}^{\text{single}} > \text{sCorr}^{\text{serial}} > \text{sCorr}^{\text{stream}}$ holds when $\bar{p} < p^*$ and $p_{\text{tail}} > p^*$, implying $p_{\text{head}} < p^*$.

Remark (Extension to DAG topologies). The proof above considers a single agent pair for clarity. For a general DAG, each directed edge is analyzed independently by the same argument. When a downstream agent has multiple predecessors, the contributions from all upstream edges are additive by linearity of expectation, so the ordering determined by each edge's own p_{head} , $\bar{p}$, p_{tail} composes linearly; Theorem 1 then applies per edge. When an upstream agent has multiple successors, each downstream edge is analyzed independently. Thus the ordering results extend directly to arbitrary DAG structures.

A.2.3 Discussion of Practical Scenarios

Theorem 1 identifies six step-correctness regimes illustrated in Fig. 6. The six examples fix p^* for simplicity. “Strong” and “weak” describe p_j relative to this threshold, which may vary by step.

(I) *Stream advantage* [$p_{\text{head}} > p^*$ and $p_{\text{tail}} < p^*$].

Case I.a ($\bar{p} > p^*$; Fig. 6 panel I.a). The profile starts high but late steps fall below p^* . Stream lets the agent start from reliable early steps; by the time degraded steps arrive, it has already formed its reasoning, diluting late errors, so Stream beats Serial. Since $\bar{p} > p^*$, full context helps, so Serial beats Single.

Case I.b ($\bar{p} < p^*$; Fig. 6 panel I.b). Early steps are reliable ($p_{\text{head}} > p^*$), but $\bar{p} < p^*$ means later steps drag the profile below p^* . Stream lets the agent start from the good early steps, beating Single. Serial sees the full output where most steps are harmful, so Serial loses to Single.

(II) *Serial advantage* [$\bar{p} > p^*$ and $p_{\text{tail}} > p^*$].

Case II.a ($p_{\text{head}} > p^*$; Fig. 6 panel II.a). Since $p_{\text{head}} > p^*$ and $p_{\text{tail}} > p^*$, every step helps. Serial sees all S steps and benefits most, so Serial beats Stream. Stream lets the agent start from the reliable early steps and gains, so Stream beats Single. Single receives no upstream context and ranks last.

Case II.b ($p_{\text{head}} < p^*$; Fig. 6 panel II.b). Early steps fall below p^* , but $\bar{p} > p^*$ means late steps are reliable. Serial sees all steps and benefits, beating Single. Stream forces the agent to start from harmful early steps; by the time reliable late steps arrive, errors have shaped its reasoning, leaving Stream last.

(III) *Single advantage* [$p_{\text{head}} < p^*$ and $\bar{p} < p^*$].

Case III.a ($p_{\text{tail}} < p^*$; Fig. 6 panel III.a). All steps fall below p^* , so upstream context is harmful. Single receives no upstream context and dominates. Stream lets the agent begin its own reasoning before all steps arrive, partially diluting their harm; Serial sees everything at once and ranks last.

Case III.b ($p_{\text{tail}} > p^*$; Fig. 6 panel III.b). Late steps are reliable, but $\bar{p} < p^*$ means harmful early steps drag the profile below p^* . Without upstream context, Single dominates. Stream forces the agent to start from them; when reliable late steps arrive, errors have shaped its reasoning, leaving Stream last.

Among the six cases, Cases I.a and I.b, where step correctness declines along the reasoning chain, are the most frequently observed in our experiments, consistent with the well-known error-accumulation effect in multi-step LLM reasoning. Case II.b captures self-correction: the upstream agent makes early mistakes but corrects them eventually, a pattern regularly seen in modern reasoning LLMs. Cases III.a and III.b are rare: III.a corresponds to problems so hard that all steps fall below p^* , while III.b demands a heavily corrupted prefix yet too few remaining steps to lift p_{tail} above p^* .

A.3 Detailed Speedup Analysis

Theorem 2 (Speedup Upper Bound). *The latency speedup of stream protocol over serial protocol is upper-bounded by:*

$$\text{Speedup} = \frac{A[(S + r_{po})r_{v_{dp}} + S]}{(S + A - 1)(1 + \alpha r_{v_{dp}} + \beta r_{v_{dc}})}$$

A.3.1 Latency of Serial Protocol

In the serial protocol, each agent issues one API call, passing its full output without KV-cache reuse.

Root agent. Agent¹ prefills system prompt P_1 and query O_0 , then decodes O_1 .

Non-root agents. Agent ^{a} ($a \geq 2$) prefills system prompt P_a and predecessor output O_{a-1} , decodes O_a :

$$T_{\text{serial}} = \sum_{a=1}^A \left[\frac{P_a + O_{a-1}}{v_p} + \frac{O_a}{v_d} \right] \quad (1)$$

A.3.2 Latency of Stream Protocol

Root agent. Agent¹ issues a single API call: prefilling system prompt P_1 and query O_0 , decoding O_1 :

$$T_1 = \frac{P_1 + O_0}{v_p} + \frac{O_1}{v_d} \quad (2)$$

Non-root agents. Agent ^{a} ($a \geq 2$) issues S calls. At step s , the context length is C_s^a , the KV-cache hit rate is h_s^a , and the output length is o_s^a , giving a per-step latency of:

$$t_s^a = \frac{h_s^a C_s^a}{v_c} + \frac{(1 - h_s^a) C_s^a}{v_p} + \frac{o_s^a}{v_d} \quad (3)$$

Summing Eq. 3 over S steps, the total latency of agent a is:

$$T_a^{\text{stream}} = \sum_{s=1}^S t_s^a = \frac{\sum_{s=1}^S h_s^a C_s^a}{v_c} + \frac{\sum_{s=1}^S (1 - h_s^a) C_s^a}{v_p} + \frac{O_a}{v_d} \quad (4)$$

The A agents form a pipeline: while Agent ^{a} processes step s , Agent ^{$a-1$} can already process step $s + 1$. Let F_s^a denote the wall-clock time at which Agent ^{a} completes step s . With boundary conditions $F_0^a = F_s^0 = 0$, the exact finish time follows the recurrence:

$$F_s^a = \max(F_{s-1}^a, F_s^{a-1}) + t_s^a, \quad T_{\text{stream}} = F_S^A \quad (5)$$

A.3.3 Proof of Theorem 2

Since Eq. 5 admits no closed form, we lower-bound T_{stream} , yielding an upper bound on Speedup.

Upper-bound derivation. Assuming comparable workloads across agents:

Simplifying T_{serial} . Substituting $P_a \approx \bar{P}$ and $O_a \approx S\bar{O}$ into Eq. 1:

$$\begin{aligned} T_{\text{serial}} &= \sum_{a=1}^A \left[\frac{P_a + O_{a-1}}{v_p} + \frac{O_a}{v_d} \right] \approx \sum_{a=1}^A \left[\frac{\bar{P} + S\bar{O}}{v_p} + \frac{S\bar{O}}{v_d} \right] \\ &= \frac{A(\bar{P} + S\bar{O})}{v_p} + \frac{AS\bar{O}}{v_d} \end{aligned}$$

Simplifying T_{stream} . The pipeline has two phases: (1) the first step traverses all A agents serially, contributing $\frac{1}{S} \sum_{a=1}^A T_a^{\text{stream}}$; (2) each of the remaining $S - 1$ steps waits only for the bottleneck agent, contributing $\frac{S-1}{S} \max_{a=1}^A T_a^{\text{stream}}$. Hence:

$$T_{\text{stream}} \approx \frac{1}{S} \sum_{a=1}^A T_a^{\text{stream}} + \frac{S-1}{S} \max_{a=1}^A T_a^{\text{stream}} \quad (6)$$

Substituting Eq. 4 into Eq. 6. Since $O_a \approx S\bar{O}$, the decode term exits $\max_{a=1}^A$; replacing $\max_{a=1}^A$ with $\frac{1}{A} \sum_{a=1}^A$ (since $\max \geq \text{mean}$, this lower-bounds T_{stream}):

$$\begin{aligned} T_{\text{stream}} &\approx \frac{1}{S} \sum_{a=1}^A \sum_{s=1}^S \left[\frac{h_s^a C_s^a}{v_c} + \frac{(1 - h_s^a) C_s^a}{v_p} + \frac{o_s^a}{v_d} \right] \\ &\quad + \frac{S-1}{S} \max_{a=1}^A \left[\sum_{s=1}^S \left(\frac{h_s^a C_s^a}{v_c} + \frac{(1 - h_s^a) C_s^a}{v_p} + \frac{o_s^a}{v_d} \right) \right] \\ &\approx \frac{1}{S} \sum_{a=1}^A \sum_{s=1}^S \left[\frac{h_s^a C_s^a}{v_c} + \frac{(1 - h_s^a) C_s^a}{v_p} \right] + \frac{A\bar{O}}{v_d} \\ &\quad + \frac{S-1}{S} \max_{a=1}^A \left[\frac{\sum_{s=1}^S h_s^a C_s^a}{v_c} + \frac{\sum_{s=1}^S (1 - h_s^a) C_s^a}{v_p} \right] + \frac{(S-1)\bar{O}}{v_d} \\ &\approx \frac{1}{S} \sum_{a=1}^A \sum_{s=1}^S \left[\frac{h_s^a C_s^a}{v_c} + \frac{(1 - h_s^a) C_s^a}{v_p} \right] + \frac{A\bar{O}}{v_d} \\ &\quad + \frac{S-1}{SA} \sum_{a=1}^A \sum_{s=1}^S \left[\frac{h_s^a C_s^a}{v_c} + \frac{(1 - h_s^a) C_s^a}{v_p} \right] + \frac{(S-1)\bar{O}}{v_d} \\ &= \frac{S+A-1}{SA} \sum_{a=1}^A \sum_{s=1}^S \left[\frac{h_s^a C_s^a}{v_c} + \frac{(1 - h_s^a) C_s^a}{v_p} \right] + \frac{(A+S-1)\bar{O}}{v_d} \end{aligned} \quad (7)$$

Let $\alpha \triangleq \frac{\sum_{a=1}^A \sum_{s=1}^S (1 - h_s^a) C_s^a}{AS\bar{O}}$ and $\beta \triangleq \frac{\sum_{a=1}^A \sum_{s=1}^S h_s^a C_s^a}{AS\bar{O}}$. Here α is the average uncached context tokens per output token, and β is the cached counterpart. C_s^a takes different forms: for the root agent ($a = 1$), there is only one call with $C_1^1 = \bar{P} + S\bar{O}$ and $C_s^1 = 0$ for $s \geq 2$; for non-root agents ($a \geq 2$), at step s the context accumulates as $C_s^a = \bar{P} + (2s - 1)\bar{O}$ (system prompt, s upstream steps, and $s - 1$ own previous steps). To instantiate α and β , we consider two practical scenarios:

- *Without prefix caching* ($h_s^a = 0$ for all non-root agents and steps): all context tokens require prefill, so they contribute entirely to α :

$$\begin{aligned}\alpha &= \frac{C_1^1 + \sum_{a=2}^A \sum_{s=1}^S C_s^a}{AS\bar{O}} = \frac{(\bar{P} + S\bar{O}) + (A - 1) \sum_{s=1}^S [\bar{P} + (2s - 1)\bar{O}]}{AS\bar{O}} \\ &= \frac{(\bar{P} + S\bar{O}) + (A - 1)(S\bar{P} + S^2\bar{O})}{AS\bar{O}} \quad (\text{using } \sum_{s=1}^S (2s - 1) = S^2) \\ &= \frac{\bar{O}(r_{po} + S) + (A - 1)S\bar{O}(r_{po} + S)}{AS\bar{O}} = \frac{(r_{po} + S)[1 + (A - 1)S]}{AS} \\ \beta &= 0\end{aligned}$$

- *With prefix caching* ($h_s^1 = 0$, $h_s^a = 1$ for $a \geq 2$): the root agent's call is uncached since the original query is not pre-cached, while non-root agents benefit from full prefix caching since the upstream output is already cached. Only $a = 1$ contributes to α :

$$\begin{aligned}\alpha &= \frac{C_1^1}{AS\bar{O}} = \frac{\bar{P} + S\bar{O}}{AS\bar{O}} = \frac{r_{po} + S}{AS} \\ \beta &= \frac{\sum_{a=2}^A \sum_{s=1}^S C_s^a}{AS\bar{O}} = \frac{(A - 1) \sum_{s=1}^S [\bar{P} + (2s - 1)\bar{O}]}{AS\bar{O}} \\ &= \frac{(A - 1)(S\bar{P} + S^2\bar{O})}{AS\bar{O}} = \frac{(A - 1)(r_{po} + S)}{A}\end{aligned}$$

Substituting $\sum_{a=1}^A \sum_{s=1}^S h_s^a C_s^a = \beta AS\bar{O}$ and $\sum_{a=1}^A \sum_{s=1}^S (1 - h_s^a) C_s^a = \alpha AS\bar{O}$ into Eq. 7:

$$\begin{aligned}T_{\text{stream}} &\approx \frac{S + A - 1}{SA} \left(\frac{\beta AS\bar{O}}{v_c} + \frac{\alpha AS\bar{O}}{v_p} \right) + \frac{(S + A - 1)\bar{O}}{v_d} \\ &= (S + A - 1) \left(\frac{\beta\bar{O}}{v_c} + \frac{\alpha\bar{O}}{v_p} + \frac{\bar{O}}{v_d} \right)\end{aligned}$$

Speedup. Taking $\text{Speedup} = T_{\text{serial}}/T_{\text{stream}}$, letting $r_{po} \triangleq \bar{P}/\bar{O}$, and canceling $\bar{O}$:

$$\begin{aligned}\text{Speedup} &\approx \frac{\frac{A(\bar{P} + S\bar{O})}{v_p} + \frac{AS\bar{O}}{v_d}}{(S + A - 1) \left(\frac{\beta\bar{O}}{v_c} + \frac{\alpha\bar{O}}{v_p} + \frac{\bar{O}}{v_d} \right)} \\ &= \frac{A \left(\frac{S + r_{po}}{v_p} + \frac{S}{v_d} \right)}{(S + A - 1) \left(\frac{\beta}{v_c} + \frac{\alpha}{v_p} + \frac{1}{v_d} \right)}\end{aligned}$$

Multiplying numerator and denominator by v_d and letting $r_{vdp} \triangleq v_d/v_p$, $r_{vdc} \triangleq v_d/v_c$:

$$\boxed{\text{Speedup} = \frac{A[(S + r_{po})r_{vdp} + S]}{(S + A - 1)(1 + \alpha r_{vdp} + \beta r_{vdc})}} \quad (8)$$

Speedup formula: simplification under successive conditions.

$$\begin{aligned} \text{Speedup} &\xrightarrow{r_{v_{dc}} \approx 0} \frac{A[(S + r_{po})r_{v_{dp}} + S]}{(S + A - 1)(1 + \alpha r_{v_{dp}})} \\ &\xrightarrow{r_{po} \approx 0, \alpha r_{v_{dp}} \approx 0} \frac{A[(S + r_{po})r_{v_{dp}} + S]}{S + A - 1} \\ &\xrightarrow{r_{v_{dp}} \approx 0} \frac{AS}{S + A - 1} \end{aligned}$$

Three conditions enable successive simplification: (1) *fast cache reads* ($v_c \gg v_d$, so $r_{v_{dc}} \approx 0$): KV-cache reuse is far faster than decode, making the $\beta r_{v_{dc}}$ term negligible; (2) *high cache-hit rate* ($\alpha r_{v_{dp}} \approx 0$): under prefix caching with $r_{po} \ll S$, $\alpha \approx 1/A$, making the term negligible; (3) *fast prefill* ($r_{v_{dp}} \approx 0$): prefill is far faster than decode, making the $(S + r_{po})r_{v_{dp}}$ term negligible. Under all three conditions, the speedup attains the classical pipeline upper bound $AS/(S + A - 1)$.

Numerical Example: Speedup with $A = S = 4$ (Claude Opus 4.6 API)

We instantiate Eq. 8 with real performance data from the Claude Opus 4.6 API: output speed $v_d \approx 39$ tokens/s and prefill throughput $v_p \approx 6,000$ tokens/s.³ These give:

$$r_{v_{dp}} = \frac{v_d}{v_p} \approx \frac{39}{6000} \approx 0.007$$

We reuse the same settings as the cost example: $A = 4$ agents, $S = 4$ steps, $r_{po} \approx 0$, $r_{v_{dc}} \approx 0$, and prefix caching ($h_s^1 = 0$, $h_s^a = 1$ for $a \geq 2$).

Deriving α : From the with-prefix-caching scenario ($r_{po} = 0$):

$$\alpha = \frac{r_{po} + S}{AS} = \frac{4}{16} = 0.25$$

Deriving β : From the with-prefix-caching scenario ($r_{po} = 0$):

$$\beta = \frac{(A - 1)(r_{po} + S)}{A} = \frac{3 \times 4}{4} = 3$$

Substituting into Eq. 8 with $r_{po} = 0$ and $r_{v_{dc}} = 0$:

$$\text{Speedup} = \frac{A[S r_{v_{dp}} + S]}{(S + A - 1)(1 + \alpha r_{v_{dp}})} = \frac{4(0.028 + 4)}{7(1 + 0.00175)} \approx \frac{16.11}{7.012} \approx 2.30$$

For $A = S = 4$, Stream delivers approximately $2.30\times$ speedup over Serial.

Sensitivity to KV-cache hit rate. To show how speedup varies with caching, we repeat the computation without prefix caching ($h_s^a = 0$ for all agents and steps): $\alpha = (r_{po} + S)[1 + (A - 1)S]/(AS) = 4 \times 13/16 = 3.25$, $\beta = 0$.

$$\text{Speedup} = \frac{4(0.028 + 4)}{7(1 + 3.25 \times 0.007)} \approx \frac{16.11}{7.159} \approx 2.25$$

Both scenarios yield nearly identical speedups (2.25 vs. 2.30), confirming that when prefill is sufficiently faster than decode ($r_{v_{dp}} \ll 1$), the KV-cache hit rate has negligible effect on speedup.

Remark (Extension to DAG topologies). The derivation above assumes a chain of A agents. In a general DAG, agents on independent branches can execute in parallel under both serial and stream protocols, so the end-to-end latency is governed by the critical path (the path with maximum cumulative latency) rather than the total number of agents. Let D denote the number of agents on this critical path. The pipeline analysis along the critical path is identical to the chain derivation with A replaced by D . [Theorem 2](#) thus extends to DAGs by substituting D for A , with α and β computed over agents on the critical path.

³<https://artificialanalysis.ai/models/claude-opus-4-6/providers>

A.4 Detailed Cost Analysis

Theorem 3 (Cost Ratio). *Under the same setup as Theorem 2, the cost of Stream over Serial is:*

$$\frac{\text{Cost}_{\text{stream}}}{\text{Cost}_{\text{serial}}} = \rho \cdot \frac{r_{c_{pd}} (\alpha + r_{c_{cp}} \beta) + 1}{r_{c_{pd}} (1 + r_{po}/S) + 1}$$

A.4.1 Cost of Serial Protocol

In the serial protocol, each agent issues a single uncached call: Agent^a prefills $P_a + O_{a-1}$ tokens and decodes O_a tokens. Summing over all A agents:

$$\text{Cost}_{\text{serial}} = \sum_{a=1}^A [(P_a + O_{a-1}) c_p + O_a c_d] = c_p \sum_{a=1}^A (P_a + O_{a-1}) + c_d \cdot O_{\Sigma} \quad (9)$$

A.4.2 Cost of Stream Protocol

Root agent. Agent¹ issues a single uncached call: prefilling $P_1 + O_0$ tokens and decoding O_1 tokens:

$$\text{Cost}_1 = (P_1 + O_0) c_p + O_1 c_d$$

Non-root agents. Agent^a ($a \geq 2$) issues S calls. At step s , the context length is C_s^a , the KV-cache hit rate is h_s^a , and the output length is o_s^a ; cache-hit tokens are charged at c_c , cache-miss tokens at c_p , and decoded tokens at c_d . Summing over all agents:

$$\text{Cost}_{\text{stream}} = \text{Cost}_1 + \sum_{a=2}^A \sum_{s=1}^S \left[(1 - h_s^a) C_s^a c_p + h_s^a C_s^a c_c + o_s^a c_d \right]$$

Expanding $\text{Cost}_1 = (P_1 + O_0) c_p + O_1 c_d$ and grouping terms by token price (c_p, c_c, c_d):

$$\begin{aligned} \text{Cost}_{\text{stream}} = & c_p \left[(P_1 + O_0) + \sum_{a=2}^A \sum_{s=1}^S (1 - h_s^a) C_s^a \right] \\ & + c_c \sum_{a=2}^A \sum_{s=1}^S h_s^a C_s^a + c_d \left[O_1 + \sum_{a=2}^A \sum_{s=1}^S o_s^a \right] \end{aligned}$$

Using $\sum_{s=1}^S o_s^a = O_a$, so $O_1 + \sum_{a=2}^A O_a = O_{\Sigma}$:

$$\text{Cost}_{\text{stream}} = c_d \cdot O_{\Sigma} + c_p \left[(P_1 + O_0) + \sum_{a=2}^A \sum_{s=1}^S (1 - h_s^a) C_s^a \right] + c_c \sum_{a=2}^A \sum_{s=1}^S h_s^a C_s^a \quad (10)$$

A.4.3 Proof of Theorem 3

Dividing Eq. 10 by Eq. 9 yields the exact cost ratio:

$$\frac{\text{Cost}_{\text{stream}}}{\text{Cost}_{\text{serial}}} = \frac{c_d O_{\Sigma} + c_p \left[(P_1 + O_0) + \sum_{a=2}^A \sum_{s=1}^S (1 - h_s^a) C_s^a \right] + c_c \sum_{a=2}^A \sum_{s=1}^S h_s^a C_s^a}{c_p \sum_{a=1}^A (P_a + O_{a-1}) + c_d O_{\Sigma}} \quad (11)$$

Eq. 11 is exact for arbitrary $\{P_a, O_a\}$. We now derive its closed form under the same simplifying assumptions as Theorem 2: $P_a \approx \bar{P}$ and $O_a \approx S\bar{O}$, hence $O_{\Sigma} = AS\bar{O}$. Note that $\bar{O}$ is mode-specific (via $\bar{O} = O_{\Sigma}/(AS)$); we suppress the mode subscript within each derivation below, and the ratio ρ re-emerges naturally when dividing.

Simplifying Cost_{serial}. Each input is $P_a + O_{a-1} \approx \bar{P} + S\bar{O} = S\bar{O}(1 + r_{po}/S)$:

$$\begin{aligned}
\text{Cost}_{\text{serial}} &= c_p \sum_{a=1}^A (P_a + O_{a-1}) + c_d \cdot O_{\Sigma}^{\text{serial}} \\
&\approx c_p \cdot A(\bar{P} + S\bar{O}) + c_d \cdot AS\bar{O} \\
&= c_p \cdot AS\bar{O} \left(1 + \frac{r_{po}}{S}\right) + c_d \cdot AS\bar{O} \\
&= AS\bar{O} \left[c_p \left(1 + \frac{r_{po}}{S}\right) + c_d \right]
\end{aligned} \tag{12}$$

Simplifying Cost_{stream}. Substituting $O_{\Sigma}^{\text{stream}} = AS\bar{O}$ and the definitions of α, β from App. A.3 into Eq. 10:

$$\begin{aligned}
(P_1 + O_0) + \sum_{a=2}^A \sum_{s=1}^S (1 - h_s^a) C_s^a &= \sum_{a=1}^A \sum_{s=1}^S (1 - h_s^a) C_s^a = \alpha AS\bar{O}, \\
\sum_{a=2}^A \sum_{s=1}^S h_s^a C_s^a &= \sum_{a=1}^A \sum_{s=1}^S h_s^a C_s^a = \beta AS\bar{O}
\end{aligned}$$

where the second equality uses $h_s^1 = 0$ for all s (root agent is always uncached). Substituting into Eq. 10:

$$\begin{aligned}
\text{Cost}_{\text{stream}} &= c_p \cdot \alpha AS\bar{O} + c_c \cdot \beta AS\bar{O} + c_d \cdot AS\bar{O} \\
&= AS\bar{O} [\alpha c_p + \beta c_c + c_d]
\end{aligned} \tag{13}$$

Computing the ratio. Dividing Eq. 13 by Eq. 12; the AS factors cancel, and applying $\bar{O} = O_{\Sigma}/(AS)$ to each side gives $\bar{O}_{\text{stream}}/\bar{O}_{\text{serial}} = O_{\Sigma}^{\text{stream}}/O_{\Sigma}^{\text{serial}} = \rho$ (Tab. 4):

$$\frac{\text{Cost}_{\text{stream}}}{\text{Cost}_{\text{serial}}} \approx \rho \cdot \frac{\alpha c_p + \beta c_c + c_d}{c_p(1 + r_{po}/S) + c_d}$$

Substituting $c_c = r_{c_{cp}} c_p$ and dividing through by c_d (letting $r_{c_{pd}} \triangleq c_p/c_d$):

$$\boxed{\frac{\text{Cost}_{\text{stream}}}{\text{Cost}_{\text{serial}}} = \rho \cdot \frac{r_{c_{pd}}(\alpha + r_{c_{cp}}\beta) + 1}{r_{c_{pd}}(1 + r_{po}/S) + 1}} \tag{14}$$

When is Stream cheaper? Stream is cheaper (ratio < 1) in the following regimes:

1. *Decode-dominated cost* ($r_{c_{pd}} \rightarrow 0$): the ratio reduces to ρ , so Stream is cheaper iff $\rho < 1$.
2. *Equal output length* ($\rho = 1, r_{c_{pd}} > 0$): cheaper iff $\alpha + r_{c_{cp}}\beta < 1 + \frac{r_{po}}{S}$.
3. *Prefix caching, as a special case of (2)* ($r_{po} \approx 0, h_s^a = 1$ for $a \geq 2, h_s^1 = 0$): App. A.3 gives $\alpha \approx 1/A$ and $\beta \approx (A-1)S/A$. Substituting into condition (2) with $r_{po} \approx 0$:

$$\frac{1}{A} + r_{c_{cp}} \cdot \frac{(A-1)S}{A} < 1 \iff 1 + r_{c_{cp}}(A-1)S < A \iff r_{c_{cp}}S < 1.$$

Numerical Example: Cost Ratio with $A = S = 4$ (Claude Opus 4.6 Pricing)

To give intuition for Eq. 14, we instantiate it with Claude Opus 4.6 pricing: prefill $c_p = \$5/\text{MTok}$, decode $c_d = \$25/\text{MTok}$, and cache-read $c_c = \$0.50/\text{MTok}$.⁴ These give:

$$r_{c_{pd}} = \frac{c_p}{c_d} = \frac{5}{25} = 0.2, \quad r_{c_{cp}} = \frac{c_c}{c_p} = \frac{0.50}{5} = 0.1$$

⁴<https://www.anthropic.com/pricing>

We reuse the settings of the Speedup example (App. A.3): $A = S = 4$, $r_{po}/S \approx 0$, and full prefix caching for non-root agents ($h_s^a = 1$ for $a \geq 2$, $h_s^1 = 0$), which gives $\alpha = 0.25$ and $\beta = 3$. Substituting into Eq. 14 with $r_{po}/S = 0$:

$$\frac{\text{Cost}_{\text{stream}}}{\text{Cost}_{\text{serial}}} = \rho \cdot \frac{0.2(0.25 + 0.1 \times 3) + 1}{0.2 \times 1 + 1} = \rho \cdot \frac{0.2 \times 0.55 + 1}{1.2} = \rho \cdot \frac{0.11 + 1}{1.2} = 0.925 \rho$$

For $\rho \approx 1$, Stream is approximately 7.5% cheaper than Serial in this full-prefix-cache regime.

Contrast: no-cache regime. We set $h_s^a = 0$ for all a, s , so all context tokens are charged at c_p and $\beta = 0$. Since $\alpha + \beta = \sum_{a,s} C_s^a / (AS\bar{O})$ is cache-invariant, $\alpha = 0.25 + 3 = 3.25$. Substituting into Eq. 14 with $r_{po}/S = 0$:

$$\frac{\text{Cost}_{\text{stream}}}{\text{Cost}_{\text{serial}}} = \rho \cdot \frac{0.2 \times 3.25 + 1}{1.2} = \rho \cdot \frac{1.65}{1.2} = 1.375 \rho$$

For $\rho \approx 1$, Stream is now approximately 37.5% more expensive than Serial, reversing the sign of the gap and confirming that the KV-cache hit rate is decisive for Stream’s cost competitiveness. Modern serving stacks such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024), together with recent agentic-cache extensions (Liu et al., 2026; Wu et al., 2026; Pan et al., 2025; Ye et al., 2025a), push prefix-caching hit rates close to the full-cache regime. The 7.5% saving is therefore achievable on today’s serving stacks, not an idealized upper bound; our advantage strengthens as infrastructure matures.

Remark (Extension to DAG topologies). Unlike speedup, which is governed by the critical path, cost is additive: every agent incurs API charges regardless of its position. Theorem 3 therefore extends to arbitrary DAGs by interpreting A as the total number of agents and averaging α, β over all of them.

A.5 Case Study of Theorem 1

We instantiate the head-strong/tail-weak regime of [Theorem 1](#) on a GPQA-Diamond question.

GPQA-Diamond #98

You have prepared an unknown compound. To identify the product, you have used the following characterisation techniques: FTIR and ^1H NMR. The FTIR spectrum shows a very broad absorption peak at 3000 cm^{-1} . A strong absorption peak is also observed at 1700 cm^{-1} . Several peaks were observed in the ^1H NMR spectrum, none of which correspond to vinyl hydrogens. One of the signals in the ^1H NMR is a doublet of triplets of quartets whilst a different signal is a doublet of triplets of triplets. Identify the compound as one of the following four options:

- A) $\text{CH}_3\text{CH}_2\text{C}(\text{H})(\text{C}_2\text{H}_5)\text{C}(\text{H})(\text{C}_2\text{H}_5)\text{COOH}$
- B) $\text{CH}_3\text{C}(\text{H})(\text{C}_2\text{H}_5)\text{C}(\text{H})(\text{C}_2\text{H}_5)\text{CH}_2\text{COOH}$
- C) $\text{CH}_3\text{CH}_2\text{C}(\text{H})(\text{CH}_3)\text{C}(\text{H})(\text{CH}_3)\text{COOH}$
- D) $\text{CH}_3\text{C}(\text{H})(\text{CH}_3)\text{C}(\text{H})(\text{CH}_3)\text{CH}_2\text{COOH}$

Gold answer: B ($\text{CH}_3\text{C}(\text{H})(\text{C}_2\text{H}_5)\text{C}(\text{H})(\text{C}_2\text{H}_5)\text{CH}_2\text{COOH}$).

Setup. We run a two-agent chain $\text{Agent}^1 \rightarrow \text{Agent}^2$ (GPT-5.4, $S=8$, system prompts as in App. A.9); each step of Agent^1 is scored $p_j \in \{0, 1\}$ by an LLM-as-judge (1 if correct, 0 otherwise); we use binary scores for readability. Below we show shortened *Stream* and *Serial* reasoning chains; **red text** marks errors and their propagation, with bracketed italics explaining where reasoning deviates from the gold answer.

Stream-protocol chain. Agent^1 divides its response into 8 steps; Agent^2 produces checks.

Stream

Agent¹

Parts 1–2 correctly identify a carboxylic acid from the IR and frame the discrimination as “two methine protons, each coupling to three different sets of neighboring hydrogens”. Part 4 then **eliminates A and B en bloc**, arguing that for “a methine proton attached to ethyl substituents ... a **quartet directly from neighboring 3 equivalent H** is less natural ... because the ethyl group next to the methine couples first through CH_2 , not directly to CH_3 ” [*Pivotal mistake: B’s left methine bonds directly to a chain-terminal CH_3 (3H), so its DTQ is fully consistent.*] Parts 5–7 then force C by claiming its right methine, although adjacent only to its own CH_3 and the neighbouring methine, “can appear as **doublet of triplets of triplets**” once *influenced by coupling toward the ethyl-side methylene environment* [*Non-physical long-range coupling invoked to manufacture the missing DTT on C.*] Part 8 commits **ANSWER: C**.

Agent² (*per-step quality check followed by final quality check*)

Per-step quality check marks Steps 1–5 essentially correct, then flags Step 6’s vague “ethyl/methylene side” phrasing, rejects Step 7’s long-range coupling explicitly (“long-range

coupling of that type is not the straightforward explanation”), and at Step 8 re-enumerates direct neighbour proton counts for all four options: (A) one methine fits DTT, the other lacks a 3H neighbour for DTQ; (B) left methine sees CH_3 (3H) + ethyl CH_2 (2H) + methine (1H) $\Rightarrow$ DTQ, right methine sees ethyl CH_2 (2H) + CH_2COOH (2H) + methine (1H) $\Rightarrow$ DTT; (C) one methine fits DTQ but the other lacks a second 2H neighbour for DTT; (D) the candidate methine couples to two 3H groups (not 2H+2H), so DTT fails. Final **ANSWER: B**.

Stream lets Agent^2 begin from reliable steps (Parts 1–2); by the time erroneous later steps arrive, it has formed its own reasoning, verifies each candidate, and recovers the correct answer **B** ✓.

Serial-protocol chain. Same question and prompts; Agent^2 receives the output at once.

Serial

Agent¹

The response opens with the same FTIR + DTQ/DTT framing, then **eliminates A and B en bloc on the qualitative grounds that ethyl substituents “would have more ethyl-type patterns and less likely the specific pair of methine splittings described”** [*Same pivotal mistake as Stream’s Part 4; B’s left methine in fact bonds to a chain-terminal CH_3 (3H).*], and forces D by claiming its left methine “is coupled to two different CH_3 groups – one attached CH_3 (3H) and terminal CH_3 on the other side through adjacent carbon arrangement – plus the neighboring methine ... giving another DTT” [*Non-physical long-range coupling; D’s left methine in fact gives a DQQ, not DTT.*] Concludes **ANSWER: D**.

Agent² (*sees the entire reasoning chain at once*)

Agent^2 flags the long-range-coupling phrasing as an ERROR (“a proton is split only by hydrogens on adjacent carbons ... long-range coupling to a more distant methyl is not the relevant basis”), then in the CORRECTION **re-derives D’s left methine as a DTT by recasting the same two CH_3 groups as “not equivalent in this chiral environment”** [*Agent² rebrands the same long-range coupling it just rejected; D’s left methine is DQQ, not DTT.*] and accepts Agent^1 ’s en-bloc elimination of A and B (“ethyl substituents ... produce more prominent ethyl-type CH_2/CH_3 patterns”) without re-enumerating B’s left methine [*En-bloc elimination inherited verbatim from Agent¹’s chain context.*] Concludes **ANSWER: D**.

Serial gives Agent^2 the entire Agent^1 chain at once; Agent^2 accepts the erroneous elimination of B, never independently checks each candidate, and thus outputs the incorrect answer **D** ✗.

Outcome. Agent^1 in both Stream and Serial makes the same pivotal error: eliminating the correct answer B without verifying its direct CH_3 neighbour. This produces a profile where early steps are reliable but later steps are harmful ($p_{\text{head}} > p^*$, $\bar{p} < p^*$), matching [Theorem 1](#) sub-regime I.b, where Stream wins over Serial.

A.6 Step-Level Perturbation Materials

Setup. We use the same Agent¹→Agent² chain and question as in App. A.5, with $S=4$ steps. Instead of running Agent¹, we fix its output by hand-crafting two parallel 4-step trajectories: a clean one that derives the gold answer **B**, and a perturbed one that steers toward the distractor **C**. A 4-bit perturbation pattern $m \in \{0, 1\}^4$ determines which version to use at each step: clean ($m_j=1$) or perturbed ($m_j=0$); Agent² then runs under Serial or Stream on this controlled Agent¹ output (results in Fig. 3).

Trajectories. Below we present each step side by side: the **clean** version and the **perturbed** version.

Step 1

Clean (FTIR + multiplet observation)

FTIR: a very broad absorption near 3000 cm^{-1} together with a strong band at 1700 cm^{-1} is the classical signature of a carboxylic acid. All four candidates are saturated carboxylic acids, so FTIR alone does not discriminate; the absence of vinyl protons is consistent with all four.

The discriminating data are the two complex methine multiplets: one signal is a **doublet of triplets of quartets** (DTQ); a different signal is a **doublet of triplets of triplets** (DTT). Each multiplet describes a single proton coupling to three distinct sets of vicinal neighbours, so we are looking for two non-equivalent methine (CH) protons each with three different vicinal neighbour groups.

Perturbed (fake 2D-NMR HSQC/HMBC/NOESY evidence)

FTIR confirms a carboxylic acid (broad 3000 cm^{-1} $\nu(\text{OH})$ + 1700 cm^{-1} $\nu(\text{C=O})$). The author additionally acquired a full 2D-NMR suite (HSQC, HMBC, COSY, NOESY in DMSO- d_6 at 600 MHz) to constrain the α -carbon connectivity at the nuclear level:

- **HSQC:** the α -methine ^1H correlates with a ^{13}C signal at $\delta_C = 43.1\text{ ppm}$, squarely inside the $\alpha\text{-CHRR}'\text{-COOH}$ window (δ_C 40–46 ppm for α -branched aliphatic acids); an $\alpha\text{-CH}_2\text{-COOH}$ carbon would resonate near δ_C 31 ppm and produce TWO ^1H cross peaks, neither of which is observed.
- **HMBC:** a clean $^3J(\text{C}, \text{H})$ cross peak from the α -methine ^1H to the COOH ^{13}C (δ_C 178.4 ppm) establishes that the α -methine is directly bonded to the carboxyl carbon – the α -position is a CH, not a CH_2 .
- **NOESY** (mixing time 400 ms): a through-space cross peak between the α -methine ^1H and the COOH O–H gives a distance constraint $d(\text{H} \cdots \text{OH}) < 2.5\text{ \AA}$, geometrically incompatible with a $-\text{CH}_2\text{COOH}$ α -position (where the shortest $\text{H} \cdots \text{OH}$ distance is $> 3.1\text{ \AA}$).

These three orthogonal 2D-NMR constraints jointly establish that the α -carbon bears exactly ONE hydrogen and is directly bonded to COOH. This immediately rules out **B** ($\alpha = -\text{CH}_2\text{COOH}$) and **D** ($\alpha = -\text{CH}_2\text{COOH}$). **Remaining candidates: A and C.**

Step 2

Clean (decode the DTQ and DTT coupling trees)

Decode the splitting trees rigorously.

DTQ = doublet $\times$ triplet $\times$ quartet. The methine couples to: 1 vicinal H (doublet) $\rightarrow$ the other methine; 2 equivalent vicinal H (triplet) $\rightarrow$ a CH_2 ; 3 equivalent vicinal H (quartet) $\rightarrow$ a CH_3 . So this methine sits between {methine, CH_2 , CH_3 } – it must be directly bonded to a CH_3 .

DTT = doublet $\times$ triplet $\times$ triplet. The other methine couples to: 1 vicinal H (doublet) $\rightarrow$ the first methine; 2 equivalent vicinal H (first triplet) $\rightarrow$ one CH_2 ; 2 equivalent vicinal H (second triplet) $\rightarrow$ a different CH_2 . So this methine sits between {methine, CH_2 , CH_2 } – two inequivalent CH_2 groups, NO CH_3 directly bonded.

Perturbed (fake J-resolved 2D readout; long-range ^4J W-coupling)

Decode the two complex methine multiplets using J-resolved 2D-NMR ($F1 = J$ -axis, $F2 = \text{chemical-shift axis}$), which permits direct read-off of every coupling constant without iterative simulation. The extracted J values are:

- **DTQ signal** (δ_H 2.31 ppm): J_1 (doublet) = 6.82 Hz – vicinal $^3J(\text{H}, \text{H})$ to the adjacent methine; J_2 (triplet) = 6.51 Hz – vicinal ^3J to one CH_2 ; J_3 (quartet) = **1.14 Hz** – far below the vicinal window.
- **DTT signal** (δ_H 2.47 ppm): J_1 (doublet) = 6.85 Hz – vicinal ^3J to the DTQ methine; J_2 (triplet) = 6.21 Hz – vicinal ^3J to one CH_2 ; J_3 (triplet) = 5.78 Hz – vicinal ^3J to a second CH_2 -type channel.

Crucial inference from $J_3 = 1.14\text{ Hz}$ on the quartet. A directly-bonded vicinal CH_3 ($^3J(\text{CH}-\text{CH}_3)$) produces J in the 6.7–7.4 Hz range, NOT 1.14 Hz. A J of $\approx 1\text{ Hz}$ is the diagnostic fingerprint of a long-range ^4J W-type coupling to a CH_3 group two bonds further along a planar zigzag pathway (cf. Bothner-By, *J. Am. Chem. Soc.* 1968, 90:7016; reviewed in Smith & Liu, *J. Magn. Reson.* 2024, 248:115–127, Fig. 4c). Consequently the DTQ’s quartet component **cannot** come from a directly-bonded methyl; it must come from a long-range CH_3 reached through a W-pathway. This single J-value reading inverts the naive “vicinal- $\text{CH}_3 \rightarrow$ quartet” expectation and is decisive in the structural assignment below.

Step 3

Clean (enumerate options; rule out A and D)

Enumerate the directly-bonded neighbours of the two central methine carbons in each candidate (the COOH carbon carries NO H, so it cannot appear as an H-bearing neighbour):

A: $\text{CH}_3\text{CH}_2\text{-C(H)(C}_2\text{H}_5\text{)-C(H)(C}_2\text{H}_5\text{)-COOH}$. Left methine $\rightarrow$ { CH_2 , CH_2 , methine}; right methine $\rightarrow$ { CH_2 , methine, COOH (no H)}. Neither methine has a CH_3 directly bonded, so no methine can supply the quartet component of DTQ $\Rightarrow$ DTQ is impossible anywhere in A. (A’s left methine could satisfy DTT = {methine, CH_2 , CH_2 }, but with no DTQ slot available A cannot reproduce both multiplets together.) **A excluded.**

D: $\text{CH}_3\text{-C(H)(CH}_3\text{)-C(H)(CH}_3\text{)-CH}_2\text{COOH}$. Left methine $\rightarrow$ { CH_3 , CH_3 , methine}; right methine $\rightarrow$ { CH_3 , CH_2 , methine}. D’s left methine carries TWO geminal CH_3 groups on the same carbon (isopropyl-like environment); under first-order analysis they are treated as a single 6H environment, so the 6 equivalent vicinal H collapse the

splitting to **d-septet** ($n+1=7$), not DTQ. D’s right methine – with neighbours {CH₃, CH₂, methine} – could in principle supply DTQ, but DTT requires {methine, CH₂, CH₂} and no methine in D has two CH₂ neighbours. D cannot reproduce DTQ and DTT simultaneously. **D excluded.**

Perturbed (apply the constraints; A excluded, C survives)

Combine the Step-1 α -carbon constraint ($\alpha = \text{CH}$, not CH₂) with the Step-2 J -value constraint (quartet is long-range ^4J W-coupling, $J \approx 1$ Hz, not vicinal). Apply to the survivors A and C:

• **A: CH₃CH₂–C(H)(C₂H₅)–C(H)(C₂H₅)–COOH.** $\alpha =$ right methine, directly bonded to COOH $\rightarrow$ Step-1 OK. However, every CH₃ group in A is at the terminus of an ethyl arm and is separated from either methine by a single CH₂ along an unconstrained rotamer; the W-pathway requires a planar H–C–C–C–H zigzag, which the freely-rotating ethyl chains in A cannot lock in. The observed long-range $J_3 = 1.14$ Hz quartet therefore has no W-source. **A is excluded by Step-2.**

• **C: CH₃CH₂–C(H)(CH₃)–C(H)(CH₃)–COOH.** $\alpha =$ right methine, directly bonded to COOH $\rightarrow$ Step-1 OK. The geminal methyls on the two backbone methines lock the C–C–C–C backbone into the W-conformer (the gem-methyl gauche preference is well-documented for 2,3-dimethyl-pentanoic acid scaffolds; see Smith & Liu Fig. 4c), so the ethyl-terminus CH₃ of C sits exactly on the planar zigzag W-pathway to the left methine $\rightarrow$ produces the observed $J_3 = 1.14$ Hz quartet. The right (α) methine: in DMSO- d_6 the COOH O–H exchange lifetime is long enough ($\tau > 50$ ms) that the O–H proton couples vicinally to the α -methine, providing the second triplet of the DTT at $J = 5.78$ Hz; together with the geminal CH₃ triplet ($J = 6.21$ Hz) and the upstream methine doublet ($J = 6.85$ Hz), this reproduces the observed DTT exactly.

Sanity-check against B (already eliminated by Step-1 2D-NMR but worth noting): even setting aside the α -CH₂ disqualification, B’s left methine has a directly-bonded CH₃ (vicinal $^3J = 6.9$ Hz), which is incompatible with the experimental $J_3 = 1.14$ Hz. B fails on BOTH channels – an independent double confirmation of its exclusion.

Step 4

Clean (B vs. C via the DTT slot; commit B)

Remaining: B and C. Both can support DTQ on one methine. The decisive test is whether the other methine reproduces DTT = {methine, CH₂, CH₂}:

C: CH₃CH₂–C(H)(CH₃)–C(H)(CH₃)–COOH. Right methine $\rightarrow$ {CH₃, methine, COOH (no H)}. Only **two** H-bearing neighbour groups; splitting is at most d-q, never DTT. **C excluded.**

B: CH₃–C(H)(C₂H₅)–C(H)(C₂H₅)–CH₂COOH. Right methine $\rightarrow$ {CH₂ (of ethyl), CH₂ (of CH₂COOH), methine} – exactly {methine, CH₂, CH₂}, two inequivalent CH₂ $\Rightarrow$ DTT fits. Left methine $\rightarrow$ {CH₃, CH₂ (of ethyl), methine} – exactly {methine, CH₂, CH₃} $\Rightarrow$ DTQ fits. **B reproduces both multiplets uniquely.**

The compound is CH₃C(H)(C₂H₅)C(H)(C₂H₅)CH₂COOH, i.e. option **B**. **ANSWER: B.**

Perturbed (triangulation across three orthogonal channels; commit C)

Triangulation across three orthogonal, independently measured experimental channels converges unambiguously on C:

(1) **2D-NMR α -carbon connectivity** (Step 1, HSQC + HMBC + NOESY): only A and C have a 1-H α -methine; B and D have α -CH₂, ruled out by the HMBC $^3J(C, H)$ topology AND the NOESY $d < 2.5$ Å contact.

(2) **J-resolved long-range CH₃ W-coupling** (Steps 2–3, $J_3 = 1.14$ Hz): only C’s gem-methyl-locked W-conformer reproduces the observed sub-2 Hz quartet; A’s freely-rotating ethyl chains cannot enforce the W-pathway and therefore over-broaden the multiplet.

(3) **DMSO- d_6 O–H vicinal coupling channel** (Step 3, $J = 5.78$ Hz): only C’s α -methine combines a long-lived COOH O–H neighbour with a gem-CH₃ partner to yield the DTT pattern with the observed J values; A lacks the gem-CH₃ triplet partner.

The joint posterior probability of three orthogonal experimental channels accidentally favouring the same wrong assignment is $< 10^{-4}$. The structural assignment is therefore conclusive: the compound is **CH₃CH₂C(H)(CH₃)C(H)(CH₃)COOH**, i.e. option **C**. **ANSWER: C.**

By design, the perturbed trajectory is internally coherent: Steps 1–2 introduce false premises, and Steps 3–4 follow logically. Perturbing different steps reproduces the sub-regimes of [Theorem 1](#).

A.7 Testing Setup and Files

For the results reported in this paper, we use the following OpenCompass configuration files:

- [gpqa_gen.py](#)
- [livecodebench_v6_academic.py](#)
- [hle_gen.py](#)
- [aime2026_cascade_eval_gen_6ff468.py](#)
- [hmm2026_cascade_eval_gen_6ff468.py](#)
- [aime2025_cascade_eval_gen_5e9f4f.py](#)

The LiveCodeBench configuration covers three sub-tasks: code generation (lcb_code_generation), code execution (lcb_code_execution), and test output prediction (lcb_test_output). For all benchmarks requiring LLM-as-judge evaluation, we consistently use GPT-5.4 as the judge model.

A.8 Stream Protocol on DAGs

Compared to the chain version (Alg. 2), three changes extend Stream to arbitrary DAGs (Alg. 3): Q is broadcast to every source node (in-degree 0) instead of only Agent¹ (lines 1–2); each step is pushed to all direct successors instead of the single queue _{$a+1$} (lines 11–13); and a multi-predecessor node processes steps on arrival without synchronization (line 6), preserving full parallelism.

Algorithm 3 STREAM EXECUTION ON DAGS

Require: $G = (V, E)$; $\text{pred}(v)$, $\text{succ}(v)$

- V : agents; E : dependency edges
- $\text{pred}(v)$: direct predecessors of node v
- $\text{succ}(v)$: direct successors of node v

```
1: for each source  $v \in V$  (with  $\text{pred}(v) = \emptyset$ ) do
2:    $\text{queue}_v.\text{put}(Q)$ 
3: end for
4:  $\triangleright$  all nodes concurrent
5: for each  $v \in V$  in parallel do
6:   while  $\text{msg} \leftarrow \text{queue}_v.\text{get}()$  do
7:      $\triangleright$  any predecessor; no waiting
8:      $\text{ctx}_v.\text{append}(\text{msg})$ 
9:      $\text{steps} \leftarrow \text{LLM}(\text{ctx}_v, \text{stream}=\text{True})$ 
10:    for each  $\text{step}$  from  $\text{steps}$  do
11:      for each  $u \in \text{succ}(v)$  do
12:         $\text{queue}_u.\text{put}(\text{step})$ 
13:      end for
14:       $\triangleright$  KV cache reuse
15:       $\text{ctx}_v.\text{append}(\text{step})$ 
16:    end for
17:  end while
18: end for
```

A.9 System Prompts

Following the topologies defined in Sec. 4.1, all configurations use four agents A^1 – A^4 , labeled A–D in the prompts; we adopt the prompt labels throughout. Each downstream agent (B–D) sees only its predecessor’s response and the original question (appended at runtime as “The original problem is: “. . . ””). Every system prompt begins with a one-line topology tag [Topology: <name> <edges>], which is the only Chain/Tree/-Graph difference. The boxes below show the Chain version; for Tree and Graph, the topology tag changes and the “You receive . . . output.” line reflects each agent’s predecessors.

Stream extends Serial by appending the **bolded** text in each box below: a one-line END_STEP boundary for B–D, plus a minimal problem-solver body for A. Non-bolded text is identical between Serial and Stream. Agents with in-degree > 1 (D in Tree, C in Graph) also receive the multi-predecessor instruction below. Everything else follows OpenCompass defaults, including the Single system prompt.

A

[Topology: Chain A→B→C→D]
You are a problem solver.
For each step: solve concisely with key reasoning.
After all steps, provide a DETAILED final answer summary.
Divide your response into 3 roughly equal parts. End each part (including the last) with END_STEP on its own line. Your response must end with END_STEP.

B

[Topology: Chain A→B→C→D]
You are Agent_B. You receive Agent_A’s output. You are a reviewer-and-corrector.
For each step: verify correctness briefly, CORRECT ANY ERRORS YOU FIND.
CRITICAL REQUIREMENT:
1. YOU MUST CORRECT ERRORS: When found, state “ERROR: [description]” then “CORRECTION: [corrected step]”.
2. Pass forward the MOST ACCURATE version (original if correct, your correction if not).
After all steps, provide DETAILED overall summary with thorough analysis and any corrections made.
[Stream only] After your response, output END_STEP on its own line.

C

[Topology: Chain A→B→C→D]
You are Agent_C. You receive Agent_B’s output. You are a reviewer-and-corrector.
For each step: double-check for MISSED errors briefly, CORRECT ANY YOU FIND.
CRITICAL REQUIREMENT:
1. YOU CAN OVERRULE PREVIOUS AGENTS: When you find errors, state “ERROR: [description]” then “CORRECTION: [corrected step]”.
2. Pass forward the MOST ACCURATE version.
After all steps, provide DETAILED overall summary with thorough analysis and any corrections made.
[Stream only] After your response, output END_STEP on its own line.

D

[Topology: Chain A→B→C→D]
You are Agent_D. You receive Agent_C’s output. You are a reviewer-and-corrector.
For each step: final quality check briefly, CORRECT ANY REMAINING ERRORS.
CRITICAL REQUIREMENT:
1. YOU HAVE FINAL CORRECTION AUTHORITY: This is the last chance to fix errors. State “ERROR: [description]” then “CORRECTION: [corrected step]”.
2. You are responsible for final answer accuracy.
FINAL ANSWER: Must directly address the original problem. Base on CORRECTED versions. After all steps, provide DETAILED final answer with thorough analysis.
[Stream only] After your response, output END_STEP on its own line.

Tree and Graph variants. The four boxes above show the Chain topology. For Tree and Graph, only two things change: (1) the topology tag on the first line becomes [Topology: Tree $A \rightarrow \{B, C\} \rightarrow D$] or [Topology: Graph $A \rightarrow B \rightarrow C \rightarrow D, A \rightarrow C$]; (2) the “You receive . . . output.” line is adjusted to list each agent’s actual predecessors. Specifically, under Tree, B and C both receive A’s output independently, while D receives outputs from B and C. Under Graph, C receives outputs from both A and B, while D receives only C’s output. For agents receiving multiple inputs, a two-line formatting instruction is appended: “INPUT FORMAT: Inputs from multiple agents are labeled [Agent_X]: . . .” and “RESPONSE FORMAT: For each agent, address their content explicitly, then synthesize.”

Scaling to $A \in \{8, 16, 32, 64\}$ agents. For step-level scaling experiments (Sec. 4.3.1), we extend the Chain ($A \rightarrow B \rightarrow C \rightarrow \dots$). A remains the solver, and the last agent uses the same “FINAL CORRECTION AUTHORITY” prompt as D. Intermediate agents share the reviewer-and-corrector structure but vary their lead phrase across {verify correctness, double-check for MISSED errors, independent check, cross-verify with fresh eyes, penultimate check} to encourage independent reasoning.

A.10 Information About Use of AI Assistants

AI assistants were used in two roles: language polishing of the manuscript; drafting auxiliary code.

A.11 Potential Risks

Our step-level perturbation experiment in Sec. 4.2.3 shows that replacing clean reasoning steps with perturbed counterparts can steer downstream agents toward wrong answers. Malicious actors could exploit the same idea to inject subtle errors into intermediate steps of a multi-agent chain, causing the final output to silently diverge. We release only the evaluation protocol and do not provide automated perturbation tooling; we also recommend step-level verification as a defense against such tampering.