中文信息处理实验室-让机器理解语言
Chinese Information Processing Laboratory

Microsoft

ISCAS

FlashAgents

Accelerating Multi-Agent LLM Systems via Streaming Prefill Overlap

Taosong Fang, Zhen Zheng, Zhengzhao Ma, Yaojie Lu, Hongyu Lin, Xianpei Han, Le Sun

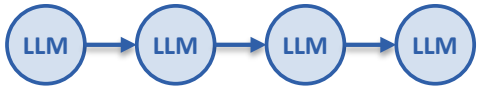
Ninth Annual Conference on Machine Learning and Systems

May 19, 2026

Multi-Agent LLM Workflows Are Now Production

A single user request now triggers **many dependent LLM calls**. What users feel is **workflow latency**, not single-call latency.

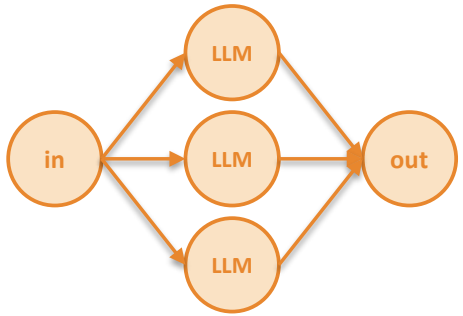
Chain



AutoGen travel planner

→ N – 1 handoffs

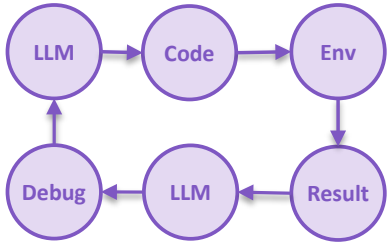
Fan-out / Fan-in



Slides Generation

→ N – 1 handoffs

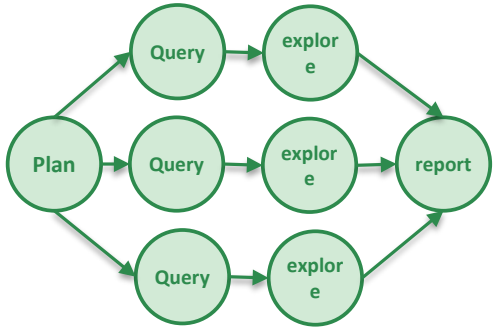
Code Agent



Claude Code w/ subagents

→ N – 1 handoffs

Deep Research

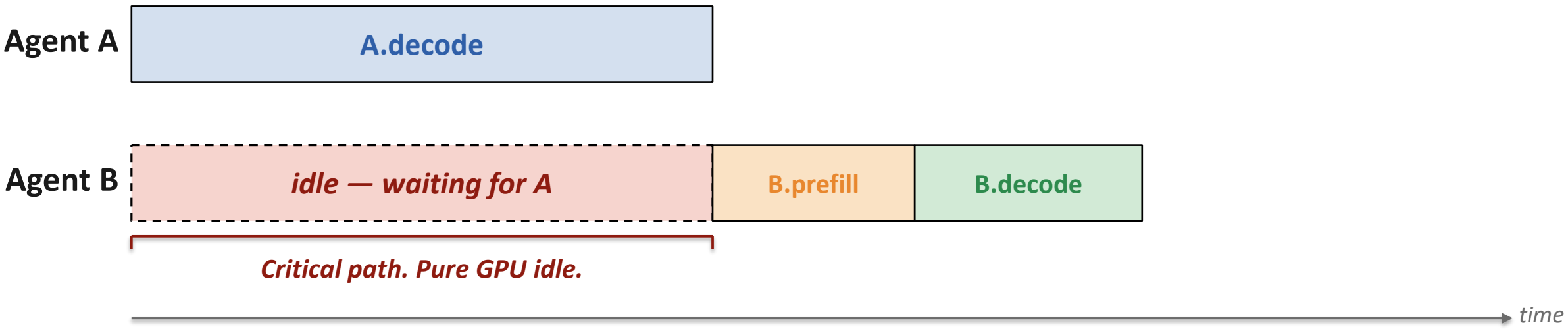


DeepResearch

→ N – 1 handoffs

Every handoff today costs a full prefill on the critical path.

The Hidden Bottleneck: Inter-Agent Idle Time



- Today, B starts prefill **only after A is fully decoded**.
- Existing serving sees **individual requests**, never the **dependency** between them.

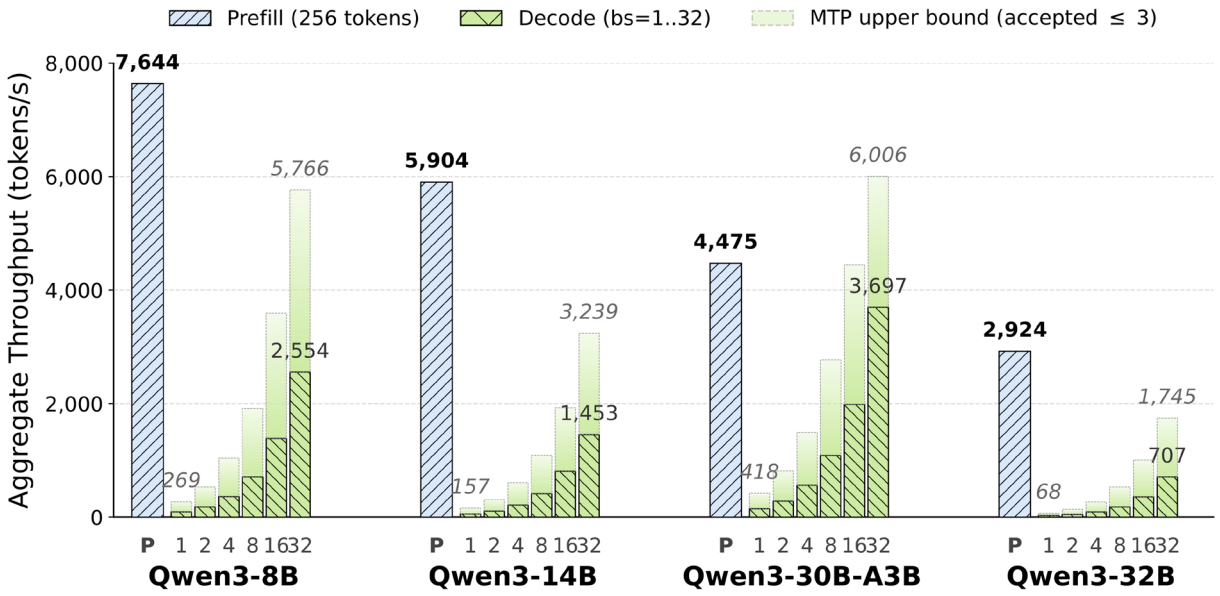
Existing Serving Misses the Inter-Agent Layer

Intra-Request	Cross-Request	Inter-Agent
Within one LLM call	*Across independent calls*	*Between dependent calls*
— FlashAttention — Paged KV cache — Prefill / decode disagg. — Chunked prefill	— Continuous batching — RadixAttention prefix cache — Speculative decoding(n-gram lookup)	— A.decode → B.prefill handoff — Concurrent shared-prefix reuse — Position-stable prefetch
Well solved	**Well solved**	**Unaddressed**

Sequentially dependent requests are nobody's job today.

Decode is Slow, Prefill is Fast?

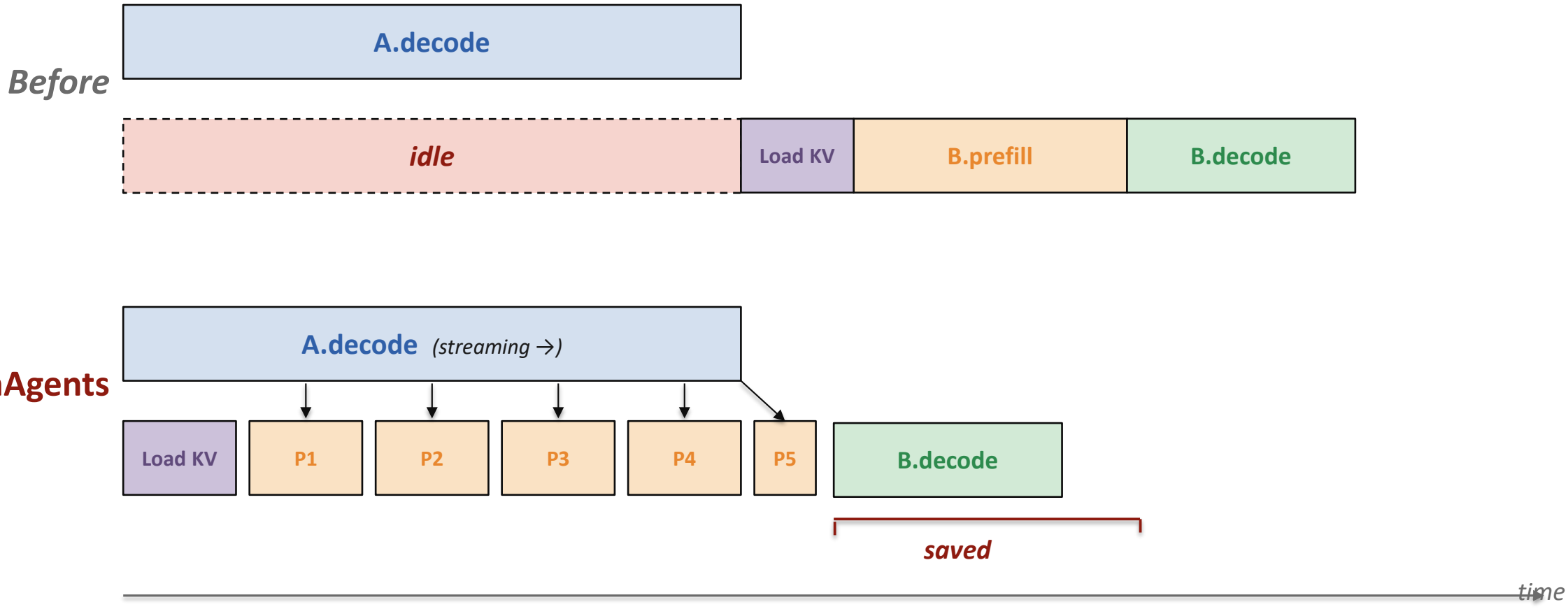
- **Gap is Shrinking:** Advanced decoding (MTP, SpecDec) and heterogeneous setups (SLM as subagent) speed up upstream generation.
- **Prefill isn't "Instant":** Concurrency stretches downstream compute time linearly.
- **Extra Downstream Workload:** B must prefill not just A's output, but extra system prompts, tools, and handle memory-bound KV cache loads.



Prefill vs. decode throughput (Qwen3 family, A100-80GB)

There is plenty of slack to overlap.

FlashAgents at a Glance



$$L = T_{decA} + T_{prefB} - T_{overlap} + T_{decB}$$

with $T_{overlap} \leq \min(T_{decA}, T_{prefB})$

Mechanism 1 — Inter-Agent Streaming and Incremental Prefill

- **Stream** A yields each token → per-pair buffer.
- **Trigger** buffer ≥ chunk size OR end-of-stream.
- **Incremental prefill** append-only KV growth, causal mask preserved.
- **Zero kernel change** — thin layer over SGLang's existing prefill path.

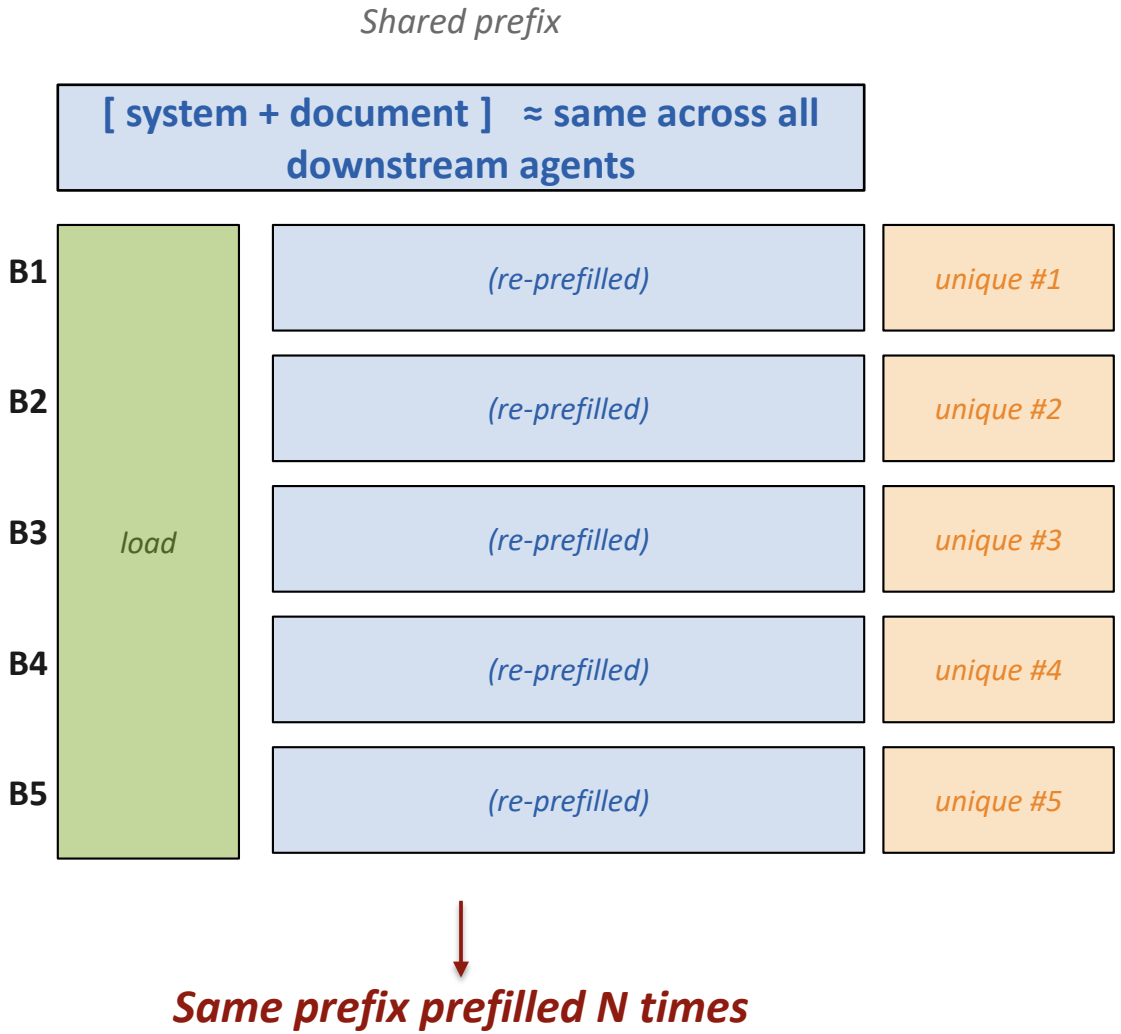
Core loop (per A→B pair)

```
buf, KV_B = [], {}

for tok in A.decode():
    buf.append(tok)
    if len(buf) >= chunk or tok.eos:
        KV_B = incPrefill(buf, KV_B)
        buf = []

return B.decode(KV_B)
```

Concurrency Reveals a Second Problem

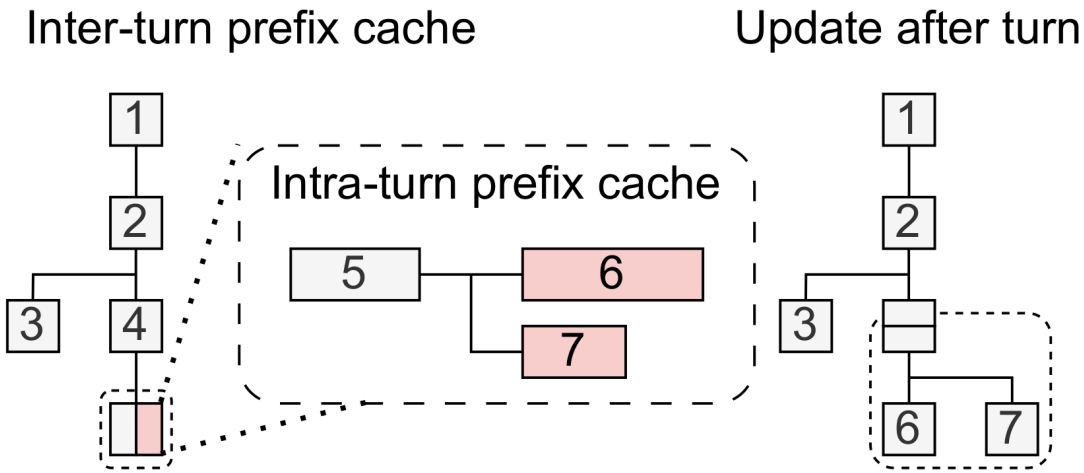


- Many downstream agents share the **same instruction + document**.
- Persistent prefix caches (RadixAttention) update ***after*** engine.step() completes.
- In-flight prefills **cannot reuse each other** within an engine.step() .
  - → *same shared prefix re-computed up to N times.*

Mechanism 2 — Intra-Turn Prefix Cache

- At each trigger, build a **temporary radix tree** over buffered tokens.
- Compute each **unique node once**; share KV across all paths.
- Persistent RadixAttention is **untouched** — we fill its blind spot for *in-flight* requests.

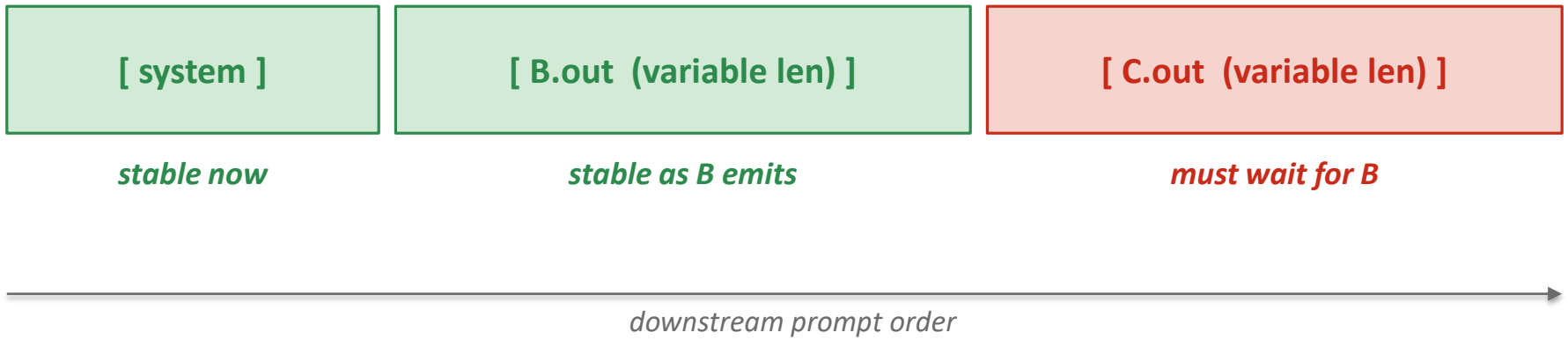
— *Linear time in total unbuffered tokens; bounded memory.*



Radix tree built at the trigger — unique nodes prefilled once, shared by all paths.

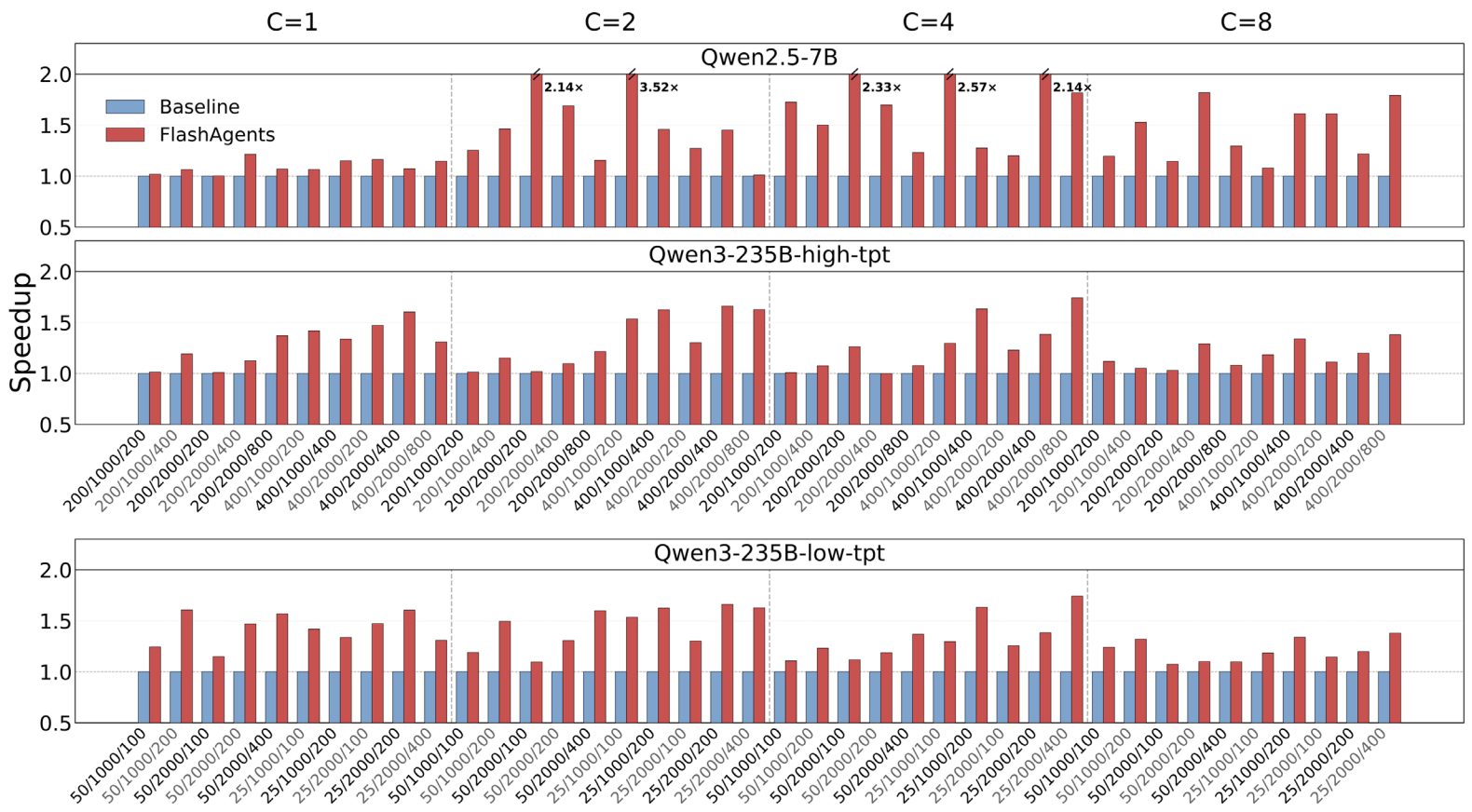
Reuse before requests finish — not after.

Correctness — Only Prefill Position-Stable Tokens



- RoPE / absolute positions: a token's KV depends on its **final index**.
- In fan-in, the tail of variable-length B-output **must wait** until B finishes — *then* we prefill it.
- FlashAgents is **conservative** — overlap is opportunistic, correctness is not.

Microbenchmark — Up to 3.52× Speedup



Sequential baseline vs. FlashAgents across 240 configurations.

3.52×

Peak speedup
(Qwen2.5-7B, C=2)

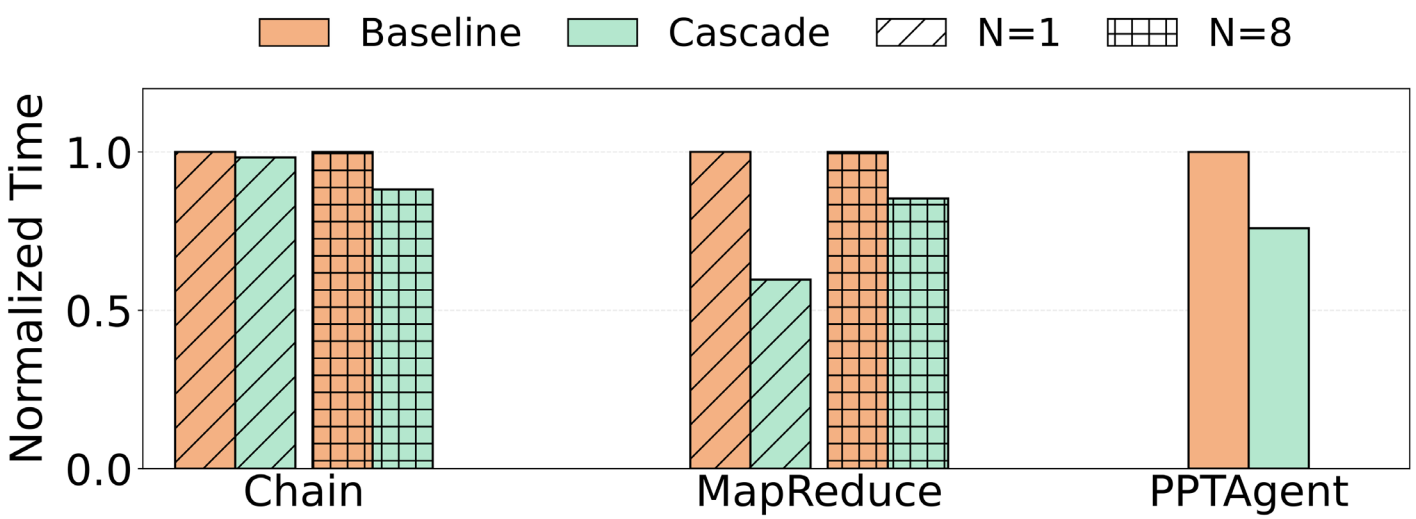
240/240

Configurations where
FlashAgents wins

1.05–3.5×

Speedup range across
models and concurrency

Real Workflows — Up to 40% Latency Reduction



Normalized latency on three end-to-end workflows.

Chain

4 agents · AutoGen

11.9% at N=8

MapReduce

3 reviewers + meta

40.3% (N=1) · 14.7% (N=8)

PPTAgent

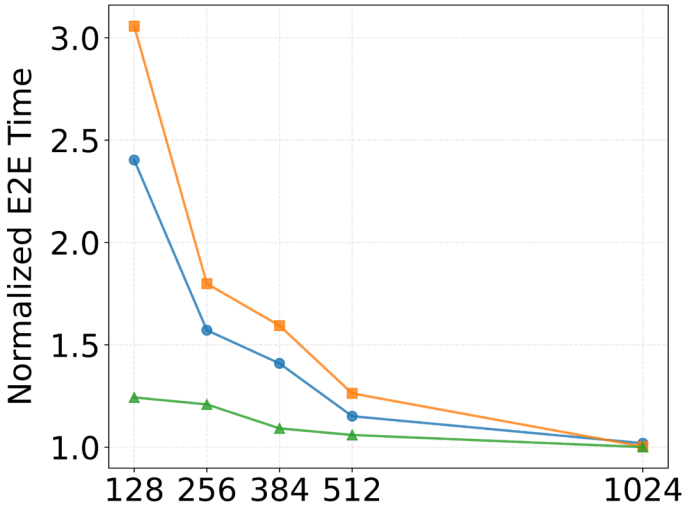
VL-7B → 14B · 2 GPUs

24.1% (1.32×)

Mechanism: pre-warm shared system+document prefix during upstream decode → hide most downstream prefill before the user-facing decode starts.

Take-aways

1. Multi-agent serving has a **new bottleneck — inter-agent idle time**.
2. **Streaming + incremental prefill** moves downstream state preparation (prefilling, KV cache loading) into upstream decode.
3. **Intra-turn prefix cache** scales the idea under concurrency.



Overhead vs. chunk size (1024-token context)

1.2× at chunk=512
KV-history alone: 4.6 – 7.9%

Schedule across agent boundaries — not just within requests.

THANKS