# The FastPath Agentic Recipe: 50% More Agents and 60% More Throughput at Equal P90 Interactivity on Stock SGLang

Asghar Ali, Muthu Durai, Mohamed Hussain, Asif Ali

*FastPathLabs*

## Abstract

AI coding agents send their full conversation history on every turn. A serving engine can reuse the key-value (KV) cache of that history, but only while the history stays in GPU memory. We measured the published InferenceX recipe for Qwen3.8-Flash-Next FP8 on 2× H200 with SGLang 0.5.23, on the InferenceX AgentX agent traces. At 16 concurrent agents the running requests alone hold up to 99% of the KV pool, so the server evicts the cached prefixes of waiting agents. The prefix-cache hit rate is 89.3% against a limit of 96.1% for the trace, so the server computes about 26 million prompt tokens per hour, about 16 million more than the trace requires. We propose the FastPath Agentic Recipe: the same recipe with an FP8 KV cache and the SGLang host-memory cache tier (HiCache) turned on. It needs six server flags and no code, model or hardware change. The FP8 cache grows the pool 1.9 times, from 2.10 to 3.96 million tokens, and the host tier keeps evicted pages. In three pre-registered runs on two nodes, throughput at 16 agents rose by 11.18% (range 9.83% to 12.04%) over the same-node stock server, against a noise bound of 2.47%. P90 time to first token fell from 2.53 to 0.93 s, P90 interactivity rose by 33% and P90 end-to-end-normalized interactivity by 53%, with accuracy held on GSM8K. The recipe gives 37,995 tokens per second per GPU, 11.5% above the published InferenceX point (34,070). The gain follows cache pressure: it is zero at 8 agents, 5% at 14 and 11% at 16. At 24 agents the recipe gives 54,476 tokens per second per GPU at a P90 interactivity of 55 tokens per second per user, the same as stock at 16 agents: 50% more agents on the same GPUs and 60% more throughput than the published point at 16 agents (stock was not run at 24). At 16 agents the host-memory tier serves only 0.25% of prompt tokens; we did not measure how the gain splits between the two features. Neither feature is new. InferenceX already uses the same flags for this model on four other accelerators (with FP4 checkpoints), and the H200 cell is the one that leaves them off. Our contribution is the measurement.

## 1 Introduction

An AI coding agent works in turns. On each turn it sends the model the task, the files it read, the tool output and every earlier answer. The prompt grows each turn. In the InferenceX AgentX traces as replayed in our runs, the median request carries about 91,000 input tokens (mean 102,000) and asks for a few hundred output tokens (median 314) [1].

Most of each prompt is the same as on the last turn. A serving engine can keep the KV cache of those tokens and reuse it. If the prefix is still in GPU memory when the agent returns, the engine computes only the new tokens. If the prefix was evicted, the engine computes the full history again. That is not new work. It is the same work done twice.

InferenceX publishes a recipe and a result for this workload on 2× H200 [1, 2]. We ran that recipe and measured where the time goes. At 16 agents, the top of the published range, running requests held up to 99% of the KV pool. The prefix-cache hit rate was 89.3%, against a limit of 96.1% for the trace. The server computed 10.7% of all prompt tokens where 3.9% would do: about 26 million prompt tokens in one hour, about 16 million more than the trace requires. The published InferenceX point shows the same gap (hit rate 88.8%).

SGLang already has two features that make the cache larger: an FP8 KV cache and a host-memory tier (HiCache) [3]. The published recipe for this cell leaves both off. We turned them on, pre-registered the test, and measured three runs on two nodes. Figure 1 shows the result from 1 to 24 agents.

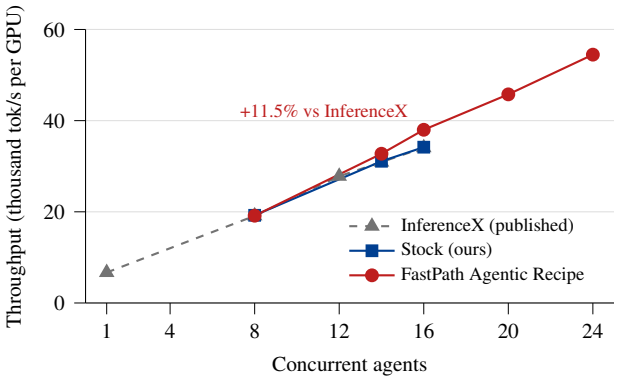


**Figure 1.** Throughput per GPU from 1 to 24 concurrent agents, Qwen3.8-Flash-Next FP8 on 2× H200, InferenceX AgentX traces. InferenceX: published points (9 November 2026). Stock and FastPath Agentic Recipe: means of three 1-hour runs on two nodes. At 16 agents the recipe gives 37,995 tok/s per GPU against 34,070 published.

This paper makes four contributions:

1. **A diagnosis.** On the public AgentX cell, the published recipe is limited by KV-cache capacity at 16 agents. We show this with the share of the pool held by running requests, the hit rate and the count of computed prompt tokens.
2. **A recipe.** The FastPath Agentic Recipe adds six existing SGLang server flags to the published recipe. It changes no code, model, image or hardware. InferenceX already

uses the same flags for this model on B200, B300, GB300 and MI355X, with FP4 checkpoints. We apply them to the H200 cell, which has none of them.

3. **A pre-registered measurement.** Three runs on two nodes pass every pre-registered result bar, with two deviations from the repeat rule (Section 7). The gain at 16 agents is 11.18% (9.83% to 12.04%), with accuracy held and zero request errors.
4. **A load map and a capacity result.** The gain is zero at 8 agents, about 5% at 14 and about 11% at 16, and it is largest for the longest prompts. At 24 agents the recipe serves 54,476 tokens per second per GPU (mean of three runs) at P90 interactivity 55 tokens per second per user, the same as stock and InferenceX at 16 agents. The same two GPUs serve 50% more agents at the same P90 speed per user, with 60% more throughput than the published InferenceX point at 16 agents. Stock was not run at 24 agents.

## 2 Background

### 2.1 KV cache and prefix reuse

A transformer stores a key vector and a value vector for each token in each attention layer. This is the KV cache. During decoding the model reads it for every new token. During prefill the model computes it for every prompt token, which is the expensive part of a long prompt. Qwen3.8-Flash-Next is a hybrid model: three of every four layers use linear attention and every fourth layer uses global attention [4]. The per-token KV cache belongs to the global-attention layers.

SGLang indexes cached prefixes in a radix tree (RadixAttention) [5]. A new request reuses the longest cached prefix and computes the rest. This is a prefix-cache hit. The KV pool takes the GPU memory left after weights and buffers. When the pool is full, SGLang evicts the least recently used prefixes, and the data is gone.

### 2.2 Agent traffic

Agent traffic is close to the worst case for this design. Each agent holds a long history that grows each turn. Many agents are active at the same time. A turn returns after a tool call, often seconds later. With 16 agents, each holding up to several hundred thousand tokens, the working set fills the pool, so least-recently-used eviction removes histories that are about to return.

### 2.3 InferenceX AgentX and the published recipe

InferenceX is a public benchmark of serving stacks [1]. Its AgentX workload replays recorded coding-agent sessions and counts load in concurrent agents. It reports throughput per GPU (all input tokens, cached or not, plus output tokens), time per output token, interactivity (1 ÷ P90 time per token) and time to first token (TTFT). It also reports end-to-end-normalized interactivity, which it names as its primary user-side latency metric for AgentX [6].

The cell was added in PR #2753 [7]. PR #4272 replaced its curve with the TP2 points used today, among them `tp2_c16` [2]. PR #4888 moved the image to SGLang 0.5.23 [8], and that run produced point #447792. The recipe runs SGLang 0.5.23 with FP8 weights, tensor parallel 2, expert parallel 2 and multi-token prediction (MTP). The InferenceX harness has a host-memory switch (`kv-offloading: dram`, with HiCache as the backend), but this cell lists only `kv-offloading: none` [6]. The recipe sets no `--kv-cache-dtype`, so the KV cache takes the data type of the model, BF16. On our nodes the pool holds 2,097,856 tokens (`max_total_num_tokens` in the server log).

Both features are in use elsewhere. InferenceX has HiCache arms for this same model on GB300, B300, B200 and MI355X (the MI355X arm is PR #4779 [9]), and those recipes also use FP8 KV caches. The B200 arms set the same six flags as our recipe, with a 200 GB tier [6]. The H200 cell is the only SGLang cell for this model without such an arm. The SGLang cookbook sets an FP8 KV cache in its verified H200 commands for this model. It has HiCache launches for this model on GB300, B300, B200 and MI355X, but it lists no verified H200 HiCache settings [4].

## 3 The FastPath Agentic Recipe

The FastPath Agentic Recipe (called C in tables) is the published recipe plus the six flags in Table 1. Everything else is unchanged. We call the published recipe *stock*. Two of the six values, `kernel` and `page_first`, are already the defaults in SGLang 0.5.23 [3]. The recipe sets them so that it does not depend on defaults. Figure 2 shows the two paths.

### 3.1 Why an FP8 KV cache helps

The pool size is fixed by memory, not by tokens. At half the bytes per element, the same memory holds about twice the tokens. Here the pool grows 1.89 times. More agent histories stay resident, fewer prefixes are evicted, and fewer prompts are computed twice. The GPU time saved goes to new tokens.

### 3.2 Why a host-memory tier helps

Without HiCache, an evicted page is lost, and a return costs a prefill through every layer. With HiCache, the page goes to host DRAM. A return costs a copy over PCIe, which is much cheaper than a long prefill.

### 3.3 What it can cost

FP8 elements carry less precision, so model output can change. Attention must convert FP8 elements back, which adds work. HiCache adds copy traffic. We therefore measured accuracy, time per token and TTFT, not only throughput. FP8 can also lower the MTP acceptance length; Section 7 gives what we know.

## 4 Experimental Setup

### 4.1 Cell, arms and runs

Table 2 gives the setup. The model, image, traces and recipe are the ones InferenceX uses for its published point.

**Table 1.** The six flags of the FastPath Agentic Recipe. Defaults are those of SGLang 0.5.23 [3].

| Flag | Recipe value | Default in 0.5.23 | Effect |
|---|---|---|---|
| `--kv-cache-dtype` | `fp8_e4m3` | `auto` (model data type; BF16 here) | 1 byte per KV element instead of 2. The pool grows from 2.10M to 3.96M tokens (1.9×). |
| `--enable-hierarchical-cache` | on | off | Adds a host-DRAM tier (HiCache) [3]. Evicted pages go to DRAM. |
| `--hicache-size` | 50 (GB per GPU) | 0 (`--hicache-ratio` applies) | Size of the DRAM tier, set before the runs from container memory. |
| `--hicache-io-backend` | `kernel` | `kernel` | GPU kernels copy pages between HBM and DRAM. |
| `--hicache-mem-layout` | `page_first` | `page_first` | Each page is contiguous in DRAM, so one page moves in one copy. |
| `--hicache-write-policy` | `write_back` | `write_through` | A page is written to DRAM only when it leaves the GPU. |

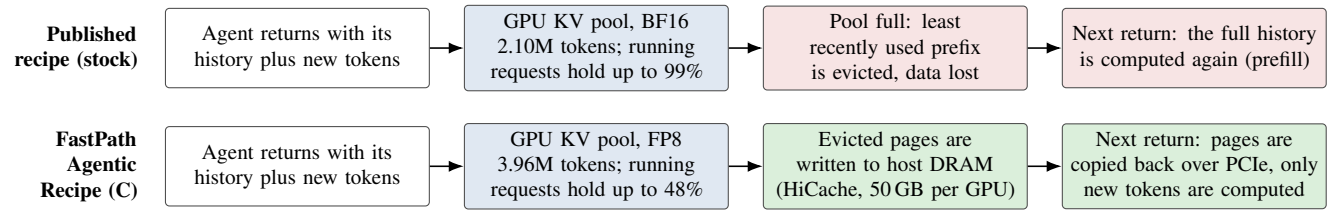


**Figure 2.** What happens to a cached history at 16 agents. Stock: running requests hold up to 99% of the pool, an evicted prefix is lost, and the returning agent pays a prefill of its full history. C: the FP8 pool is 1.89 times larger, and pages that still have to leave the GPU go to host DRAM and are copied back on return. Pool sizes and the peak share held by running requests are the measured values of Table 7.

**Table 2.** Setup.

| Item | Value |
|---|---|
| Model | Qwen3.8-Flash-Next-FP8, revision `7fb48d70` |
| Engine | SGLang 0.5.23, image `lmsysorg/sglang:v0.5.23-cu130`, digest `sha256:f832b5c3...` (the InferenceX image) |
| Recipe | InferenceX TP2 recipe [2]. At 16 agents (`tp2_c16`): TP2, EP2, MTP with 3 steps and 4 draft tokens, at most 32 running requests. C adds the six flags of Table 1. |
| Hardware | 2× H200 SXM, 141 GB each, per node. Runpod Secure Cloud, 377 GB container memory. Node A (runs 1 and 2): AMD EPYC 9554, 64 vCPU. Node B (run 3): 40 vCPU. |
| Workload | InferenceX AgentX agent traces (the 256k-capped corpus that InferenceX selects for this model [6]), seed 42, 1-hour measured window per point |
| Load | 8, 14 and 16 agents for both arms. C also at 20 and 24 agents. |
| Runs | 3 runs on 2 nodes. Stock first in runs 1 and 3, C first in run 2. Two stock points at 16 agents per run (S1 first, S2 last). |
| Accuracy | GSM8K, first 100 questions, both arms, once per node [10] |

### 4.2 Metrics

We report the three numbers InferenceX reports [1]: throughput in tokens per second per GPU, time per output token at p50 and p90, and TTFT at p50 and p90. Throughput counts every input token, cached or not, plus output tokens. Because most of it is cached input, we also report output tokens per second. Interactivity is 1 ÷ P90 time per token, in tokens per second per user. End-to-end-normalized (E2E-normalized) interactivity divides the full latency of each request, TTFT included, by its output tokens, and is 1 ÷ the P90 of that ratio [6]. A slow first token lowers it.

### 4.3 Pre-registration

We wrote the arms, the points and the pass rules into the plan and committed them before the first measured run. A 20-minute pilot over six candidate flag sets chose C. Each confirmation run passes only if:

1. At 16 agents, C beats the mean of the two stock points on throughput by more than the noise bound.
2. At 8 and 14 agents, C loses no more throughput than the noise bound.
3. At 8, 14 and 16 agents, C loses no more P90 interactivity or P90 E2E-normalized interactivity than their noise bounds.
4. There are zero request errors, and GSM8K of C is at most 0.5 points below stock on each node.

The noise bound of a metric is the larger of a fixed floor (1.0% for throughput and for output tokens, 1.5% for P90 interactivity, 3.0% for P90 E2E-normalized interactivity) and the largest spread between two stock points in the pilot or any run. For the two interactivity metrics the pilot spreads, 5.6% and 9.9%, set the bounds. For throughput, each run used the larger of the floor and its own stock spread (Table 3). The final bound is the largest of these, 2.47%, and every run also passes against it. The verdict is GO only if all three runs pass.

### 4.4 Checks

At every server start the kit reads `/server_info` and stops an arm if a flag did not take effect. It checks that the KV pool of C is larger than that of stock and that the DRAM tier receives pages. The size of the DRAM tier follows a pre-registered rule from free container memory: at the start of each run (170 to 184 GB free) it gave 50 GB per GPU. On the pilot node the same rule gave 100 GB per GPU.

**Table 3.** Throughput at 16 agents, tokens per second per GPU, one 1-hour window per point. S1 and S2 are the two stock points of a run. Bound: the larger of the 1.0% floor and the S1–S2 spread of the run; the last row gives the final bound.

| Run (node) | S1 | S2 | C | Gain | Bound | Result |
|---|---|---|---|---|---|---|
| 1 (A) | 34,376 | 34,803 | 37,989 | +9.83% | 1.23% | PASS |
| 2 (A) | 33,934 | 34,049 | 38,085 | +12.04% | 1.00% | PASS |
| 3 (B) | 33,534 | 34,371 | 37,911 | +11.66% | 2.47% | PASS |
| All | mean 34,178 | | 37,995 | +11.18% | 2.47% | GO |

**Table 4.** The three numbers at 16 agents, with the end-to-end metrics and the hit rate. InferenceX values are from its published point [1]; its error count is from the record of that run [8]. Ours are means of 1-hour windows over three runs on two nodes. The last column is the ratio of the two means; the pre-registered gains are in Table 3.

| 16 agents | InferenceX #447792 | Stock (6 points) | C (3 points) | C vs stock |
|---|---|---|---|---|
| Throughput, tok/s/GPU | 34,070 | 34,178 | 37,995 | +11.2% |
| Output, tok/s/GPU | 304.2 | 306.2 | 334.7 | +9.3% |
| Time per token p50, ms | 9.8 | 9.7 | 8.9 | −8.5% |
| Time per token p90, ms | 18.2 | 18.1 | 13.6 | −24.7% |
| P90 interactivity, tok/s/user | 55.0 | 55.3 | 73.3 | +32.6% |
| P90 E2E-norm. interactivity | 36.1 | 36.6 | 55.9 | +52.5% |
| TTFT p50, s | 0.48 | 0.45 | 0.45 | 0% |
| TTFT p90, s | 2.72 | 2.53 | 0.93 | −63.3% |
| E2E latency p50, s | 4.51 | 4.47 | 3.82 | −14.5% |
| E2E latency p90, s | 27.6 | 26.8 | 20.3 | −24.3% |
| Prefix-cache hit rate | 88.8% | 89.3% | 95.3% | +6.0 pts |
| Request errors | 0 | 0 | 0 | |

# 5 Results

## 5.1 Pre-registered verdict

All three runs passed every result bar. The verdict is GO, with two deviations from the repeat rule (Section 7). Table 3 gives each run. The gain of C over the mean of two same-node stock points was 9.83%, 12.04% and 11.66%, with a mean of 11.18% against a noise bound of 2.47%. Output tokens per second rose by 9.29% (8.39% to 9.75%). P90 interactivity rose by 32.65% (30.80% to 35.87%), and P90 E2E-normalized interactivity by 52.65% (47.64% to 59.52%).

## 5.2 The three numbers at 16 agents

Table 4 compares the published InferenceX point with our means. Our six stock points lie between −1.6% and +2.1% of the published point (34,070), so our setup reproduces the published baseline. The three C points lie 11.3% to 11.8% above it (mean 37,995, +11.5%). The machines differ, so we treat this as a reference. The claim is the same-node comparison in Table 3.

P90 TTFT fell from 2.53 s to 0.93 s, and P90 end-to-end latency from 26.8 s to 20.3 s. Request errors were zero. At the rental price of our nodes ($5.29 per GPU-hour) the cost per million total tokens falls from $0.0430 to $0.0387, which is 10.0% lower. Most of these tokens are cached input (Section 7).

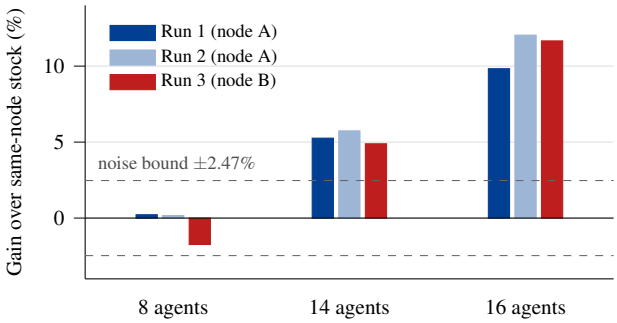


**Figure 3.** Throughput gain of the FastPath Agentic Recipe over same-node stock, by load and run. Dashed lines: the pre-registered noise bound (±2.47%).

## 5.3 From 1 to 24 agents

Table 5 places our points next to every published InferenceX point for this cell. InferenceX publishes 1 to 16 agents. We measured stock at 8, 14 and 16 agents and C from 8 to 24 agents. We did not measure 1 or 12 agents. At 24 agents the recipe gives 54,476 tokens per second per GPU at a P90 interactivity of 55 tokens per second per user, the same interactivity as stock and InferenceX at 16 agents. This is 50% more agents on the same GPUs (stock was not run at 24). Section 6.4 gives the per-run values.

## 5.4 Accuracy and errors

GSM8K (first 100 questions) gave 0.99 for both arms on node A, and 0.98 for stock and 0.99 for C on node B. Run 2 uses the node A result, as pre-registered. All 27 measured points had zero request errors.

# 6 Analysis

## 6.1 The gain follows cache pressure

Table 6 and Figure 3 show the gain at each load. At 8 agents running requests hold at most 43% of the stock pool, and there is nothing to gain. At 14 agents they hold up to 75% to 84%, and the gain is about 5%. At 16 agents they hold up to 97% to 99%, and the gain is about 11%. All three runs show the same order. The tail moves before running requests fill the pool: at 14 agents P90 TTFT already falls from 1.89 s to 0.92 s and P90 interactivity rises by 21%.

## 6.2 Mechanism

Table 7 gives the cache counters, and Figure 4 shows the share of prompt tokens that the server has to compute. At 16 agents the FP8 pool grew from 2.10M to 3.96M tokens, and the peak share held by running requests fell from 99% to 48%. The hit rate rose from 89.3% to 95.3%, close to the trace limit of 96.3%.

The stock server computed 26.0M prompt tokens in the hour, 2.7 times the 9.6M that the trace requires at its limit. C computed 12.7M against a requirement of 10.1M, while it completed 8% more requests (2,628 against 2,440). The excess over the limit fell from 16.4M to 2.5M tokens per hour, by 85%. At 8 agents, where running requests hold at most 43% of the stock pool, an excess of 1.2M to 2.1M remains.

**Table 5.** Throughput, P90 interactivity and P90 TTFT from 1 to 24 agents. InferenceX: published points of 9 November 2026 (#447793, #447794, #447790, #447791, #447792); the throughput values also appear in the record of that run in PR #4888 [8]. Stock and C: means of three runs on two nodes. A dash means not measured or not published.

| | Throughput, tok/s per GPU | | | C against | | P90 interactivity, tok/s per user | | | P90 TTFT, s | |
|---|---|---|---|---|---|---|---|---|---|---|
| Agents | InferenceX | Stock | C | stock | InferenceX | InferenceX | Stock | C | Stock | C |
| 1 | 6,685 | – | – | – | – | 189.4 | – | – | – | – |
| 8 | 19,112 | 19,233 | 19,146 | −0.5% | +0.2% | 125.8 | 131.2 | 126.6 | 0.76 | 0.75 |
| 12 | 27,802 | – | – | – | – | 89.4 | – | – | – | – |
| 14 | 30,916 | 31,091 | 32,738 | +5.3% | +5.9% | 70.1 | 72.0 | 87.4 | 1.89 | 0.92 |
| 16 | 34,070 | 34,178 | 37,995 | +11.2% | +11.5% | 55.0 | 55.3 | 73.3 | 2.53 | 0.93 |
| 20 | – | – | 45,763 | – | – | – | – | 70.1 | – | 1.02 |
| 24 | – | – | 54,476 | – | – | – | – | 55.4 | – | 1.23 |

**Table 6.** Throughput gain of C over same-node stock, by load. At 16 agents, stock is the mean of S1 and S2.

| Agents | Stock peak active use | Run 1, A | Run 2, A | Run 3, B |
|---|---|---|---|---|
| 8 | 37% to 43% | +0.22% | +0.17% | −1.74% |
| 14 | 75% to 84% | +5.26% | +5.74% | +4.89% |
| 16 | 97% to 99% | +9.83% | +12.04% | +11.66% |

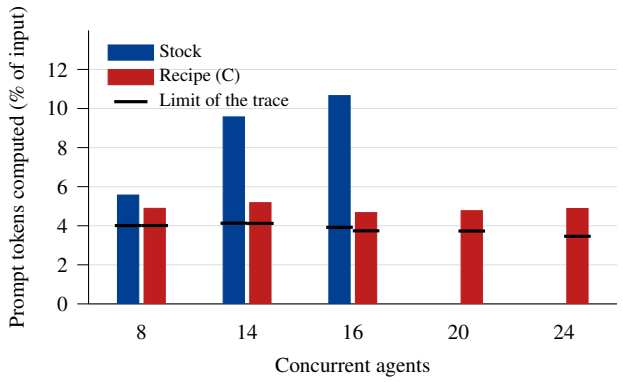


**Figure 4.** Share of prompt tokens that the server computes (100% minus the hit rate), by load. Black marks: the share that the trace requires at its limit. The stock bar leaves the limit as running requests take more of the pool. The recipe stays 1 to 1.6 points above it up to 24 agents.

So a gap of 1 to 1.6 points to the limit is not caused by cache pressure alone.

In the three runs the DRAM tier served only 0.25% to 0.26% of prompt tokens at 16 agents (1.3% to 1.5% at 20 agents and 2.4% to 2.7% at 24 agents). Thus at the headline load almost all cache hits came from the GPU pool. This describes how C works at this load. It does not show what each feature gives alone (Section 7).

### 6.3 Long prompts gain most

Figure 5 splits TTFT by prompt length, over all three runs. Below 32k tokens P90 TTFT fell from 1.12 s to 0.92 s (means). Above 128k tokens it fell from 5.65 s (5.09 to 6.32 s over six stock points) to 0.91 s (0.88 to 0.93 s over three C points), a factor of 6.2. With C, P90 TTFT stays between 0.86 and 1.17 s in all four groups. With stock it grows five times from the shortest group to the longest. The median TTFT is below 0.6 s in every group for both arms, so the loss is in

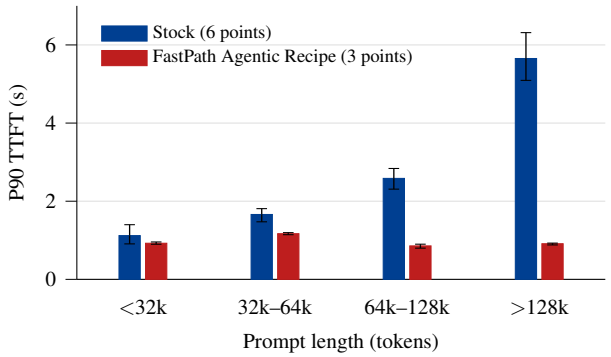


**Figure 5.** P90 time to first token at 16 agents by prompt length. Bars: means over three runs on two nodes (stock: S1 and S2 of each run; FastPath Agentic Recipe: one point per run). Whiskers: lowest and highest point. Long prompts gain most, because an eviction forces a prefill of the full history.

the tail: the requests whose history was evicted. An eviction hurts these requests most, because a miss means a prefill of the full history. C also completes more of them: 763 requests above 128k tokens per hour against 646.

### 6.4 More agents at the same interactivity

C also ran at 20 and 24 agents. These points are outside the published range. The only rule there was that C must not be dominated, and it was not in any run. At 24 agents C gave 54,507, 54,574 and 54,348 tokens per second per GPU, at a P90 interactivity of 56.2, 54.9 and 55.0. Stock at 16 agents gave 52.5 to 57.5. The mean is 54,476 tokens per second per GPU at P90 interactivity 55, the same as stock and InferenceX at 16 agents (55.3 and 55.0). So the same two GPUs serve 50% more agents at the same P90 speed per user, with 60% more throughput than the published InferenceX point at 16 agents (Figure 6). Stock was not run at 24 agents, so this compares C at 24 with stock at 16, not with stock at 24. It is a different number from the 52.65% gain in P90 E2E-normalized interactivity at 16 agents.

The cost is a higher median time per token (11.6 against 9.7 ms) and a P90 TTFT of 1.20 to 1.25 s. That TTFT is still half of the 2.53 s of stock at 16 agents, and P90 E2E-normalized interactivity is 41.6 against 36.6. At 20 agents C gave 45,837, 45,829 and 45,624 tokens per second per GPU (mean 45,763) at a P90 interactivity of 70.1, which is 27% above stock at 16 agents, with a P90 TTFT of 1.02 s.

**Table 7.** Cache counters, means over three runs (six stock points at 16 agents). Limit: the theoretical prefix-cache hit rate that the AgentX harness reports for the trace. Computed: prompt tokens the server had to compute in the 1-hour window, input tokens × (1 − hit rate). Required: the same at the limit. Excess: computed minus required. Peak active use: the largest share of the pool held by running requests in the window. Columns are rounded independently.

| Agents, arm | Pool, M tokens | Peak active use | Hit rate | Limit | Computed, M | Required, M | Excess, M | Requests |
|---|---|---|---|---|---|---|---|---|
| 8, stock | 2.10 | 41% | 94.4% | 96.0% | 7.6 | 5.5 | 2.1 | 1,428 |
| 8, C | 3.96 | 22% | 95.1% | 96.0% | 6.7 | 5.5 | 1.2 | 1,421 |
| 14, stock | 2.10 | 78% | 90.4% | 95.9% | 21.2 | 9.2 | 12.1 | 2,324 |
| 14, C | 3.96 | 38% | 94.8% | 95.9% | 12.1 | 9.6 | 2.5 | 2,409 |
| 16, stock | 2.10 | 99% | 89.3% | 96.1% | 26.0 | 9.6 | 16.4 | 2,440 |
| 16, C | 3.96 | 48% | 95.3% | 96.3% | 12.7 | 10.1 | 2.5 | 2,628 |
| 20, C | 3.96 | 69% | 95.2% | 96.3% | 15.6 | 12.2 | 3.4 | 3,130 |
| 24, C | 3.97 | 75% | 95.1% | 96.5% | 19.0 | 13.4 | 5.5 | 3,627 |

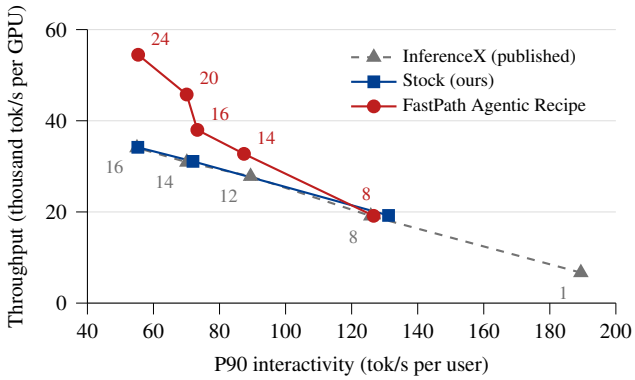


**Figure 6.** Throughput against P90 interactivity. Labels give the number of agents. At 24 agents the FastPath Agentic Recipe keeps the P90 interactivity of stock at 16 agents (about 55 tok/s per user) at 1.6 times the throughput.

## 7 Threats to Validity and Deviations

- **No gain without pressure.** At 8 agents C gave +0.22% and +0.17% on node A and −1.74% on node B. The rules allow a loss at 8 agents up to the noise bound. The final bound is 2.47%, set by the stock spread on node B, so run 3 passes with a margin of 0.72 points. We did not change the rule.
- **Interactivity at low load.** At 8 agents, P90 interactivity was 3% to 4% lower with C in all three runs. This is inside the pre-registered bound for this metric (5.6%, the stock spread measured in the pilot), but the sign is consistent. A likely cause is FP8 conversion in attention when the cache is not a constraint. We did not test this cause.
- **Limited ablation.** In 20-minute pilot points at 16 agents, FP8 alone gave +2.6%, HiCache alone +5.6% and both (C) +5.2%. Full 1-hour ablations were not run, so we cannot split the gain between the two features. In the pilot HiCache alone gave more than FP8 alone. The counters of Section 6.2 show which tier served the tokens in C. They do not show what each feature would give alone.
- **Stock not run at 24 agents.** The capacity result compares C at 24 agents with stock at 16 agents at the same interactivity.
- **Kernel work did not transfer.** A separate set of tuned kernels cut mean time per token by 6.35% at 1 agent on the same cell (three runs on two nodes, FastPathLabs run record of 17 November 2026). In the pilot, the same kernels (arm ALL) changed throughput at 16 agents by −0.28%. A gain at one load does not carry to another.
- **Deviation: setup failures in run 3.** Run 3 failed three times in setup before any measured point: a network reset during a download, a disk quota error during the model fetch, and an install that passed its time limit (logs kept with the run 3 data). The plan allows one repeat for a cause outside the arm and says that a second failure ends the study as NO_RESULT. We continued after the second and third failures. This is a deviation from the plan. No measured point was lost or repeated, and the causes were outside both arms.
- **Deviation: stock out of memory in run 2.** During the second stock point at 16 agents in run 2, the stock server ran out of GPU memory: a 434 MB allocation failed with 78 MB free, about 105 s after start. Running requests held at most 30% of its KV pool in that attempt; the memory was already allocated at start. It served 34 requests in the window, and then its watchdog stopped it. The kit marked the run invalid. We kept that attempt as data and ran only that stock point again with the same commands. The new point was valid (34,049 tokens per second per GPU, 2,438 requests, 0 errors) and agrees with S1 of the same run (33,934). C was not affected. The plan does not allow a repeat for a cause inside an arm, so this is the second deviation. The event also shows that the published recipe runs this cell with almost no free GPU memory.
- **Scope.** One model, one GPU type and one agent workload. We did not measure chat or short prompts. The runs count agents (8 to 24) on InferenceX AgentX. There are no user-counted (128 or 256 users) runs.
- **Accuracy check is narrow.** GSM8K on the first 100 questions detects only large damage. It does not prove that long-context or agentic output is unchanged with an FP8 KV cache. No quality benchmark has been executed against the target model on long-context or agentic workloads. InferenceX runs the full GSM8K test set for this cell (1,319 questions, 97.65% for the published run [8]).
- **Cross-machine comparison.** The InferenceX point comes from its own cluster. Only InferenceX running C on its own hardware gives an official comparison.

- **Throughput counts cached input.** InferenceX counts every input token, cached or not. Most of the 34,000 tokens per second are cached input; output is about 300. We report the output gain next to the throughput gain.
- **Counters summed twice.** The metrics export scraped the same server at two addresses (127.0.0.1 and localhost) and added them. The server counter of computed prompt tokens therefore reads 2.0 times the value that follows from the measured throughput and hit rate in all 27 points (1.95 to 2.02 times; for stock at 16 agents 52.3M against 26.0M; TP rank 1 reports 0). The KV pool gauge, summed over the 2 addresses and the 2 TP ranks, reads 4 times the per-server pool of the server log. We report the per-server pool and the computed tokens that follow from throughput and hit rate. At 16 agents the ratio between the arms is the same on both bases (0.49).
- **Forced MTP acceptance.** As InferenceX does for speculative decoding, every timing point ran with a fixed acceptance length (`SGLANG_SIMULATE_ACC_LEN=2.32`), the same in both arms. Throughput and time per token therefore do not include any change in real acceptance. On the GSM8K servers, which use real acceptance, the FP8 arms accepted about 4% fewer tokens per step at full batch (3.13 against 3.27). This is weak evidence (GSM8K prompts, 12 to 13 log lines per run), but a real-acceptance run on agent traces could show a smaller gain.
- **Power.** InferenceX reports 499 W average per GPU for its point. We did not measure power and make no per-watt claim.
- **One window per point.** Each point is one 1-hour window. The spread between the two stock points in one run was 0.34% to 2.47%.

## 8 Related Work

**Paged and prefix-reusing KV caches.** PagedAttention (vLLM) manages the KV cache in pages to reduce fragmentation and allow larger batches [11]. SGLang reuses prefixes across requests with RadixAttention [5]. Both make the GPU pool more efficient. Neither makes it larger.

**KV tiers below the GPU and low-precision KV.** SGLang HiCache adds host-DRAM and storage tiers [3]. SGLang and LMSYS documentation and blog posts already describe the FP8 KV cache and HiCache as standard features for long-context and multi-turn workloads. The HiCache announcement quotes one deployment in which a storage tier raised the hit rate from 40% to 80% and cut average TTFT by 56% [12]. InferenceX has HiCache DRAM arms in other cells, among them this model on GB300, B300, B200 and MI355X (PR #4779 [9]) [6]. SSEngine-GLM5.3-Flash [13] builds a new prefill-only engine with HBM, DRAM and SSD tiers and shortest-remaining-prefill-first scheduling. On agent session traces it reports 40.51 requests per second against 4.64 for vLLM DP4/EP4 at 400 sessions on 4× B200. It reaches the same diagnosis as we do: HBM plus DRAM must hold the retained context, or the hit rate falls and the GPUs recompute. Its baselines, hardware, model and metric differ from ours, so the ratios are not on one scale.

**Agentic KV management.** Penelope [14] rethinks eviction for agentic workloads. It preserves shared prefixes, evicts from the beginning of the remaining history and keeps the cached suffix, and it co-designs recomputation and unified GPU-CPU cache management in SGLang. Against SGLang on real agent traces it reports up to 60% lower mean TTFT and up to 37% lower job completion time with a GPU-only cache, and up to 76% and 60% with a two-tier GPU-CPU cache. Other recent work includes workflow-aware eviction (KVFlow [15]), phase-aware KV eviction (AgentKV [16]) and TTL-based retention (Continuum [17]). These works change the eviction policy. Our recipe keeps the stock policy and makes the cache larger.

**This work.** FP8 KV caches and host-memory tiers are not new. Each is a documented SGLang feature [3], and InferenceX uses both in other cells. We add a measurement: on a public agent benchmark cell, the published recipe leaves both off, the cache is the bottleneck at the top of the published range, and turning them on gives a pre-registered gain with accuracy held and no code change. At 16 agents almost all cache hits came from the 1.9 times larger GPU pool; the split of the gain between the two features is not measured. Table 8 lists the works side by side.

## 9 Conclusion

On the InferenceX AgentX cell for Qwen3.8-Flash-Next on 2× H200, the published recipe is limited by KV-cache capacity at 16 agents. The FastPath Agentic Recipe turns on two SGLang features that already exist, an FP8 KV cache and a host-memory tier, with six flags. In three pre-registered runs on two nodes it raised throughput by 11.18% over the same-node stock server and by 11.5% over the published point, cut P90 TTFT from 2.53 to 0.93 s and held accuracy. The gain needs cache pressure and grows with prompt length. At the headline load almost all cache hits come from the larger FP8 GPU pool; the split of the gain between the two features is not measured. The next step is to submit the recipe to InferenceX, so that InferenceX measures it on its own hardware.

## Artifact Availability

- **Kit.** The benchmark kit (`bench/hl1`: `run_hl1.sh`, `hl1_lib.py`), the pre-registration and the summary CSVs will be released in a public repository. Runs 1 and 2 used kit commit `4a5231a`. Run 3 used `d287285`, which adds an optional local install cache and changes no bar and no measured step.
- **Image.** `lmsysorg/sglang:v0.5.23-cu130@sha256:f832b5c39559c9caa900b22ad37c76fcce66470676855c42052102c537812f1f`, the image pinned by InferenceX for this cell [6].
- **Data.** The InferenceX result JSON and the per-request export of every point are archived. Summary tables: `hl1_points_final.csv` (27 points and the published point) and `hl1_ttft_by_prompt_length.csv`. The summary tables hold values from the per-request files and

**Table 8.** Related work. The setups differ, so the headline numbers are not on one scale.

| Work | Approach | Setup | Headline |
|---|---|---|---|
| InferenceX AgentX, this cell [1] | Public benchmark, published recipe | Qwen3.8-Flash-Next FP8, 2× H200 | 34,070 tok/s/GPU at 16 agents |
| InferenceX, other cells for this model [6, 9] | FP8 KV cache and HiCache DRAM arms | GB300, B300, B200, MI355X | The B200 arms set the same six flags as C, with a 200 GB tier |
| PagedAttention / vLLM [11] | Paged KV memory | various | 2–4× throughput vs FasterTransformer, Orca |
| SGLang RadixAttention [5] | Prefix reuse in a radix tree | various | Reuse across requests |
| SGLang HiCache [12] | Host-DRAM and storage tiers | one production deployment, with a storage tier | Hit rate 40% to 80%, average TTFT −56% |
| SSEngine [13] | New prefill-only engine, HBM/DRAM/SSD tiers | GLM-5.3-Flash NVFP4, 4× B200 | 40.51 vs 4.64 req/s at 400 sessions |
| Penelope [14] | Evicts from the head of the history, keeps the suffix; GPU-CPU cache in SGLang | three Qwen3 sizes, real agent traces | Mean TTFT up to −60%, job completion time up to −37% vs SGLang (GPU-only cache) |
| KVFlow [15] | Workflow-aware eviction and KV prefetch | multi-agent workflows | Up to 1.83× (one workflow) and 2.19× (many) vs SGLang with hierarchical radix cache |
| This work | Six flags on the published recipe | 2× H200, 8–24 agents | +11.18% same-node, +11.5% vs published point |

the InferenceX result JSON.

- **Recipe.** Stock: InferenceX `tp2_c16` [2, 8]. C: stock plus the six flags in Table 1.

## References


[1] SemiAnalysis. InferenceX, AgentX agentic result #447792, Qwen3.8-Flash-Next, 2× H200, run date 9 November 2026. https://inferencex.semianalysis.com/infx/inference/agentic/447792?g_model=Qwen3.8-Flash-Next&i_seq=agentic-traces&g_rundate=2026-11-09

[2] SemiAnalysis. InferenceX pull request #4272: move Qwen3.8-Flash-Next H200 SGLang curve to v0.5.22 with TP2 points (adds tp2_c16). Merged 19 October 2026. https://github.com/SemiAnalysisAI/InferenceX/pull/4272

[3] SGLang Team. SGLang v0.5.23: server arguments for the KV cache data type and the hierarchical cache (HiCache), in python/sglang/srt/arg_groups/fields, and the HiCache design documentation. https://github.com/sgl-project/sglang (tag v0.5.23) and https://docs.sglang.io/docs/advanced_features/hicache_design

[4] SGLang Team. SGLang cookbook: Qwen3.8-Flash-Next (model description and verified launch commands, as shipped with v0.5.23). https://docs.sglang.io/cookbook/autoregressive/Qwen/Qwen3.8-Flash-Next

[5] L. Zheng et al. SGLang: Efficient Execution of Structured Language Model Programs. In *NeurIPS*, 2024. arXiv:2312.07104.

[6] SemiAnalysis. InferenceX repository at commit 6ed3774 (19 November 2026): inferencex-e2e/configs, the qwen3.8next recipes under srt-slurm-recipes, docs/MODELS.md and docs/agentx-standalone.md. https://github.com/SemiAnalysisAI/InferenceX

[7] SemiAnalysis. InferenceX pull request #2753: Qwen3.8-Flash-Next FP8 SGLang AgentX on H200 (adds the cell). Merged 27 August 2026. https://github.com/SemiAnalysisAI/InferenceX/pull/2753

[8] SemiAnalysis. InferenceX pull request #4888: Update Qwen3.8-Flash-Next FP8 H200 SGLang AgentX MTP image to v0.5.23-cu130. Merged 9 November 2026. https://github.com/SemiAnalysisAI/InferenceX/pull/4888

[9] SemiAnalysis. InferenceX pull request #4779: Qwen3.8-Flash-Next FP4 SGLang agentic MTP on MI355X, baseline extended to 24, 48 and 80 agents with HiCache DRAM offload. Merged 6 November 2026. https://github.com/SemiAnalysisAI/InferenceX/pull/4779

[10] K. Cobbe et al. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168, 2021.

[11] W. Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP*, 2023. arXiv:2309.06180.

[12] Z. Xie. SGLang HiCache: Fast Hierarchical KV Caching with Your Favorite Storage Backends. LMSYS Org blog, 10 September 2025. https://lmsys.org/blog/2025-09-10-sglang-hicache

[13] SyFI-ServingStudio. SSEngine-GLM5.3-Flash: pipeline-parallel prefill server with HBM, DRAM and SSD KV tiers. https://github.com/SyFI-ServingStudio/SSEngine-GLM5.3-Flash (read 19 November 2026).

[14] J. Fan et al. Rethinking KV Cache Eviction for Agentic LLM Serving. arXiv:2611.00391, 2026.

[15] Z. Pan et al. KVFlow: Efficient Prefix Caching for Accelerating LLM-Based Multi-Agent Workflows. In *NeurIPS*, 2025. arXiv:2507.07400.

[16] T. Liu et al. AgentKV: Phase-Aware KV Eviction for Agentic LLMs. arXiv:2609.14872, 2026.

[17] H. Li et al. Continuum: Efficient and Robust Multi-Turn LLM Agent Scheduling with KV Cache Time-to-Live. arXiv:2511.02230, 2025.