Millstone AI Solutions

On-Prem AI Solutions

BENCHMARK REPORT

Qwen3-Coder-Next

Performance Analysis on 1x H200 SXM

MODEL

Organization	**Qwen**
Parameters	**80B**
Precision	**FP8**

TEST HARDWARE

GPU	**1x H200 SXM**
VRAM	**141GB**
Engine	**SGLang**

HIGHLIGHTS

852.6
Tok/s Peak Throughput
@ 10 Concurrent Requests

100.0%
Success Rate
Across All Scenarios

8
Concurrent Users
@ 32K Context

February 09, 2026

MillstoneAI.com

CONTENTS

Table of Contents

Interactive Data Available Online
This report provides a static snapshot of benchmark results. For interactive charts with hover tooltips, exact data point values, and interpolated metrics, visit the full benchmark page:
MillstoneAI.com/inference-benchmark/qwen3-coder-next-fp8-1x-h200-sxm

OVERVIEW

Executive Summary

Infrastructure decisions require real performance data. This report measures user-facing performance, showing how many concurrent users a configuration can support at a given context length before performance degrades.

This benchmark evaluates **Qwen3-Coder-Next** (Qwen, 80B parameters, Mixture-of-Experts) running in FP8 precision on 1x H200 SXM (141GB VRAM).

Test parameters: Context lengths from 1K - 256K tokens. Concurrency from 1 - 10 requests. 1024 output tokens per request. No prompt caching. No speculative decoding. Full-precision KV cache.

Benchmark methodology →

Key Findings

Peak System Throughput	852.6 tok/s @ 10 concurrent requests, 1K context
TTFT Single Request	379ms (1K context) → 20.4s (256K context)
Generation Speed Single Request	210.8 tok/s (1K context) → 152.8 tok/s (256K context)
Chatbot Capacity	8 concurrent requests @ 32K context
Throughput Scaling	5.6× from 1 to 10 concurrent requests
Success Rate	100.0% across 5.3K requests

Throughout this report, "**concurrent requests**" refers to simultaneous active requests. For applications with natural user pauses (chat interfaces, coding assistants), each request slot typically serves 4–5 active users.

RECOMMENDATIONS

Use Case Guidance

The table below maps this configuration's performance to common deployment scenarios. Capacity limits are where TTFT or generation speed falls below accepted thresholds for a comfortable user experience. Detailed charts and analysis for each use case are available in the Capacity Analysis section.

USE CASE	TTFT THRESHOLD	SPEED THRESHOLD	ANALYSIS
Code Completion	2s e2e	N/A	Supports **23** concurrent requests within accepted thresholds.
Short-form Chatbot	10s	10 tok/s	Supports **~48** concurrent requests within accepted thresholds.
General Chatbot	8s	15 tok/s	Supports **8** concurrent requests within accepted thresholds.
Long Document Processing	12s	15 tok/s	Supports **5** concurrent requests within accepted thresholds.
Automated Coding Assistant	12s	20 tok/s	Supports **2** concurrent requests within accepted thresholds.

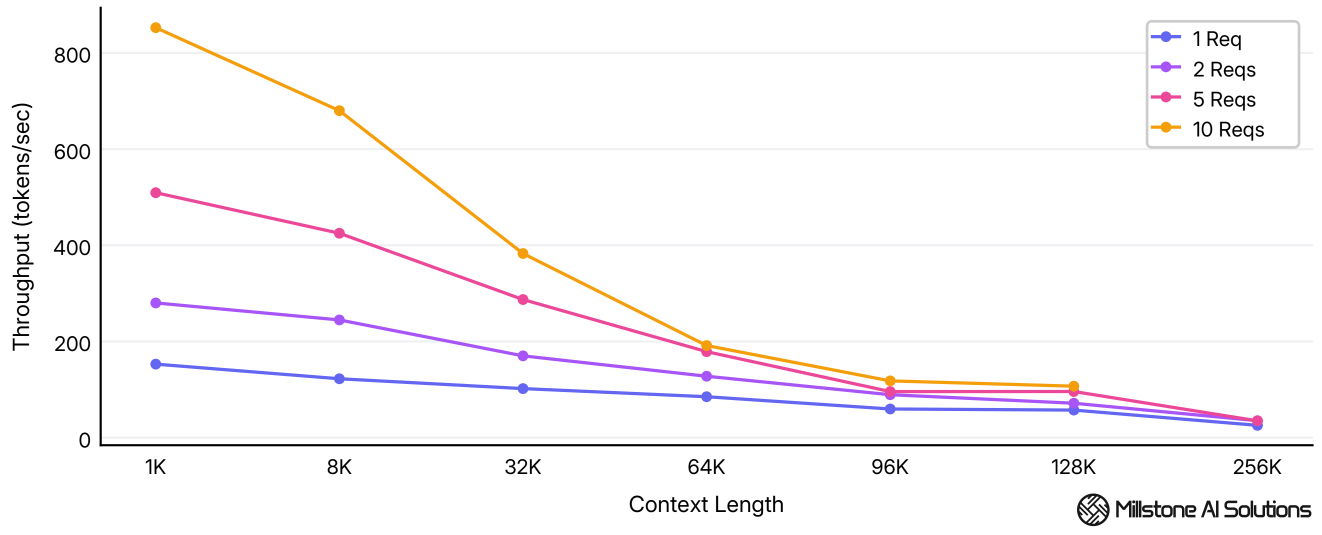
The limits shown are conservative. Beyond these points, the system continues functioning with slower response times that may still be acceptable for your specific use case.

> Want to validate your specific configuration? Request a Custom Benchmark →

PERFORMANCE

System Throughput

Aggregate token generation across all concurrent requests. Measures output tokens only. Prompt tokens processed during prefill are excluded.



Average system throughput across 1K - 256K tokens context lengths at 1 - 10 concurrency levels.

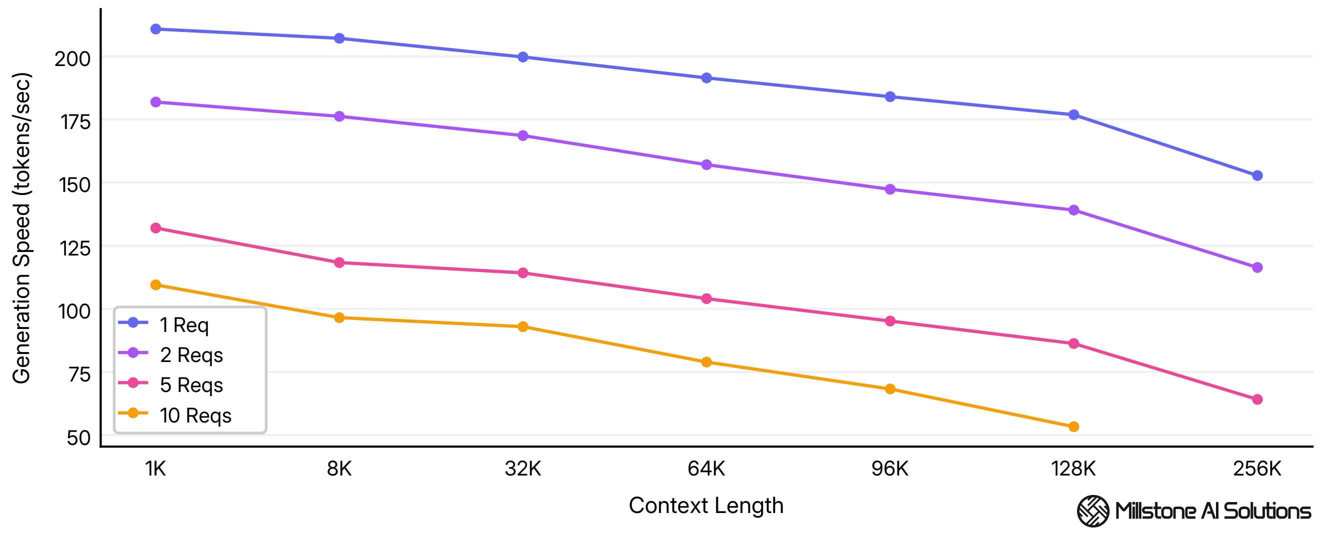
CONDITION	THROUGHPUT
Peak (1K context, 10 requests)	**852.6** tok/s
32K context, 10 requests	**382.9** tok/s
256K context, 10 requests	**106.9** tok/s

At peak throughput, this configuration produces approximately **3.1 million** tokens per hour. This is relevant for batch workloads like document processing, synthetic data generation, or offline analysis. Higher concurrency or shorter contexts can increase this further.

USER EXPERIENCE

Per-User Generation Speed

Token generation rate experienced by each individual user. This is the speed at which text streams into their response, also referred to as "decode speed" or "decode throughput." As concurrency increases, per-user speed decreases since GPU resources are shared across requests.



Average per-user generation speed across 1K - 256K tokens context lengths at 1 - 10 concurrency levels.

How Fast is This?

SPEED	USER EXPERIENCE
< 15 tok/s	Slow; may be slower than reading speed
15–25 tok/s	Acceptable; keeps pace with reading
25–50 tok/s	Fast; exceeds reading speed
> 50 tok/s	Very fast; text appears nearly instantly

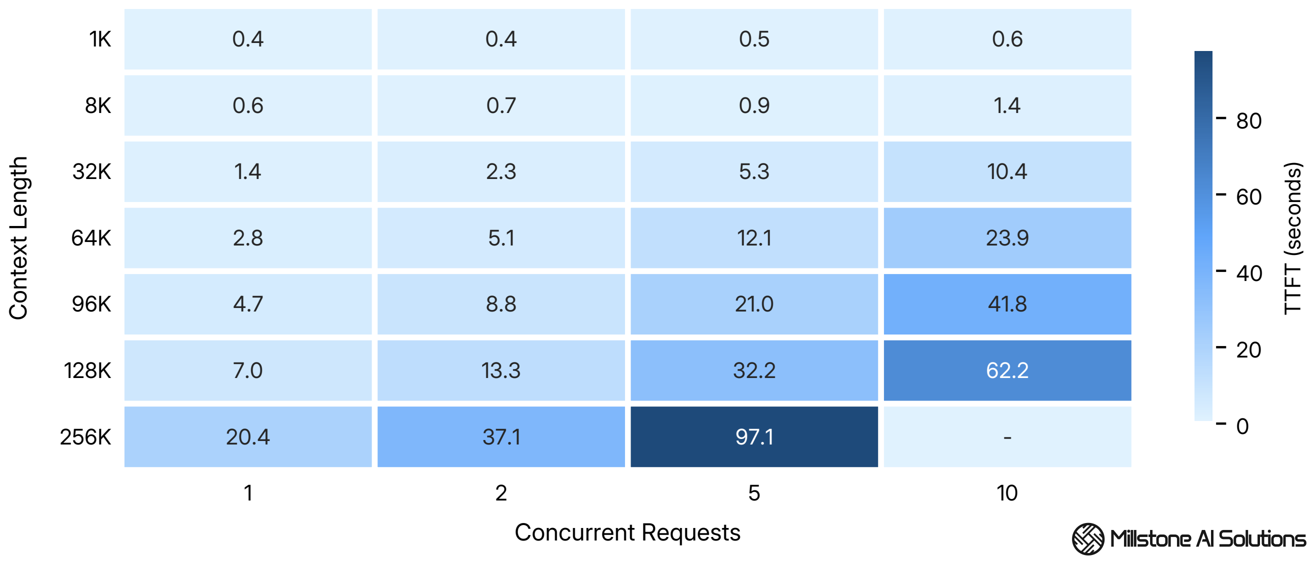
At **53.3** tok/s (the lowest measured point: 128K context, 10 concurrent requests), this configuration is very fast, even under maximum load. Single-user performance at 1K context reaches **210.8** tok/s.

Time to First Token

Time from request submission to first response token. The primary metric for perceived responsiveness. TTFT has two components:

- **Queue wait:** Time waiting for GPU availability (increases with concurrency)
- **Prefill:** Time to process input context (increases with context length)

At low concurrency, prefill dominates. Under load, queue wait takes over. See Technical Analysis for more.



Context Length	1	2	5	10
1K	0.4	0.4	0.5	0.6
8K	0.6	0.7	0.9	1.4
32K	1.4	2.3	5.3	10.4
64K	2.8	5.1	12.1	23.9
96K	4.7	8.8	21.0	41.8
128K	7.0	13.3	32.2	62.2
256K	20.4	37.1	97.1	-

Average time to first token across 1K - 256K tokens context lengths at 1 - 10 concurrency levels.

How Responsive is This?

TTFT	USER EXPERIENCE
< 500ms	Feels instant
500ms–2s	Feels responsive
2–5s	Noticeable but still acceptable
5–10s	Feels slow; generally acceptable at higher context lengths
> 10s	Can be frustrating; users may retry or abandon

Important note about caching. These benchmarks use fresh context with no caching enabled, representing worst-case TTFT. In production with caching enabled, only new tokens require processing. For example, a 64K conversation where you add 1K of new context would have a TTFT similar to the 1K results above, not the 64K results. **For most real-world use cases where context is built incrementally (chatbots, coding assistants, multi-turn agents), TTFT with caching enabled would be significantly faster than these results.**

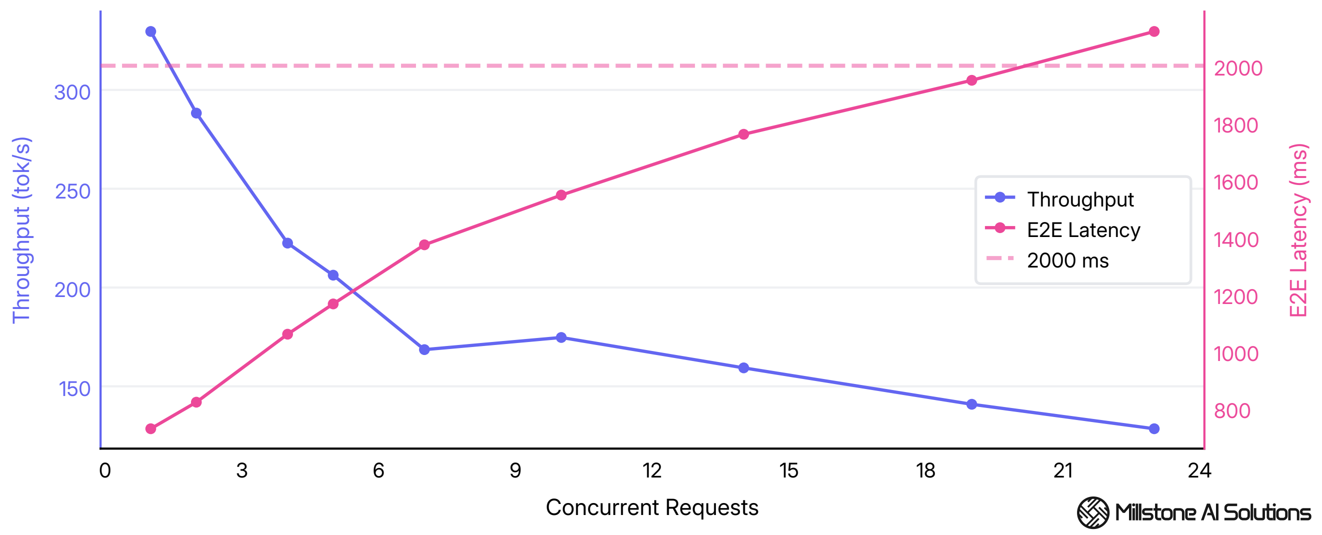
Capacity Analysis

How many concurrent requests can this configuration handle for different workloads? Each section below shows performance metrics as concurrency increases at a specific context length. Dashed lines indicate quality thresholds, the point where user experience degrades below acceptable levels. The "capacity limit" is the tested or estimated point where the first threshold is reached.

Code Completion (1K Context)

Inline code suggestions in IDEs, like autocomplete. Responsiveness is critical. This test generates 128 output tokens per request (vs. 1024 elsewhere) to match typical autocomplete length. The key metric is end-to-end latency, not TTFT.

Threshold: End-to-end latency < 2,000ms



Average end-to-end latency and throughput at 1K context. Dashed line indicates quality threshold.

METRIC	@ 1 request	@ 10 requests	@ 23 requests
End-to-end latency	729ms	1547ms	2120ms (threshold exceeded)
Throughput	330 tok/s	175 tok/s	129 tok/s

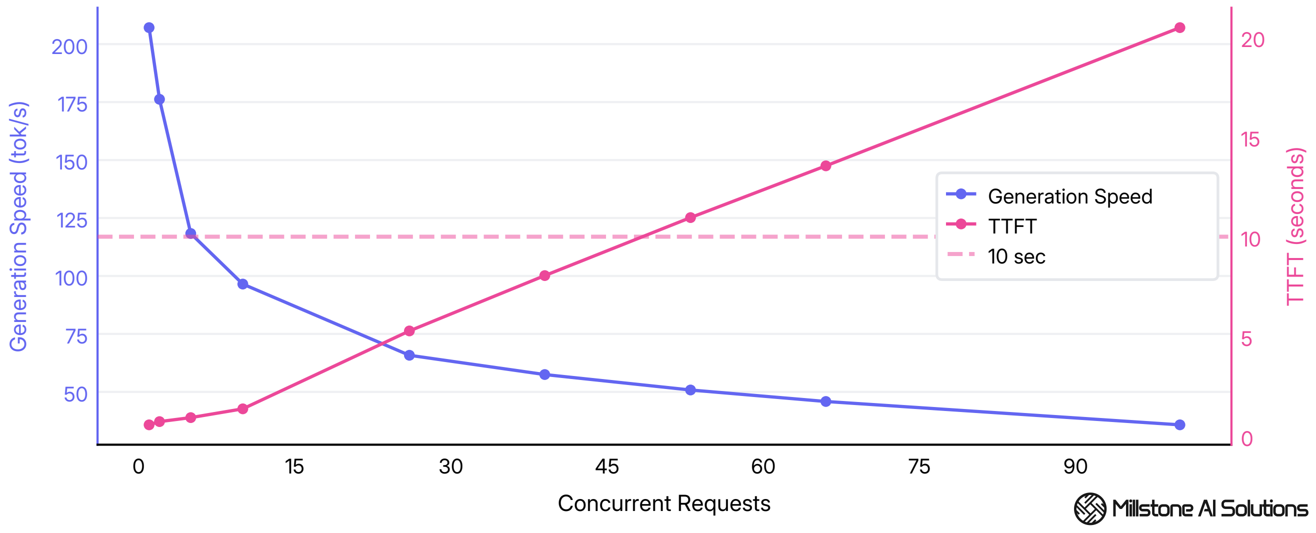
Capacity limit: 23 concurrent requests

At 23 concurrent requests, end-to-end latency reaches 2120ms, just above the 2,000ms threshold.

Short-form Chatbot (8K Context)

Quick conversational exchanges: customer support queries, simple Q&A, single-turn requests. 8K context accommodates a few back-and-forth messages plus system prompt. User expectations are more forgiving for these scenarios. 10+ tok/s is acceptable for reading streamed responses from a support chatbot.

Thresholds: TTFT < 10s, generation speed > 10 tok/s



Average per-user generation speed and TTFT at 8K context. Dashed lines indicate quality thresholds.

METRIC	@ 1 request	@ 48 requests	@ 100 requests
TTFT	0.6s	~9.9s	20.5s (threshold exceeded)
Generation speed	207 tok/s	~53 tok/s	36 tok/s

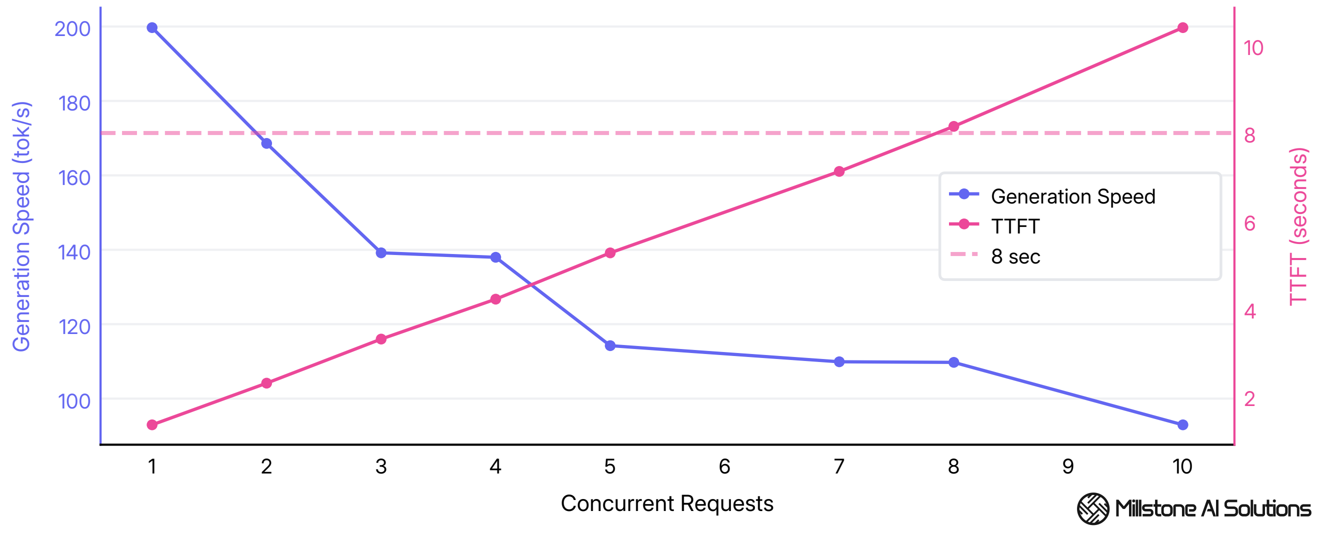
Capacity limit: ~48 concurrent requests

At 48 concurrent requests, TTFT reaches ~9.9 seconds, just under the 10-second threshold. Generation speed at this concurrency is ~53 tok/s, above the 10 tok/s minimum.

General Chatbot (32K Context)

ChatGPT-style chatbot. If you're deploying a multi-turn conversational chatbot, this benchmark shows how many concurrent requests you can support while matching acceptable responsiveness. 32K context matches ChatGPT's limit.

Thresholds: TTFT < 8s, generation speed > 15 tok/s



Average per-user generation speed and TTFT at 32K context. Dashed lines indicate quality thresholds.

METRIC	@ 1 request	@ 8 requests	@ 10 requests
TTFT	1.4s	8.2s (threshold exceeded)	10.4s (threshold exceeded)
Generation speed	200 tok/s	110 tok/s	93 tok/s

Capacity limit: 8 concurrent requests

At 8 concurrent requests, TTFT reaches 8.2 seconds, just above the 8-second threshold. Generation speed at this concurrency is 110 tok/s, above the 15 tok/s minimum.

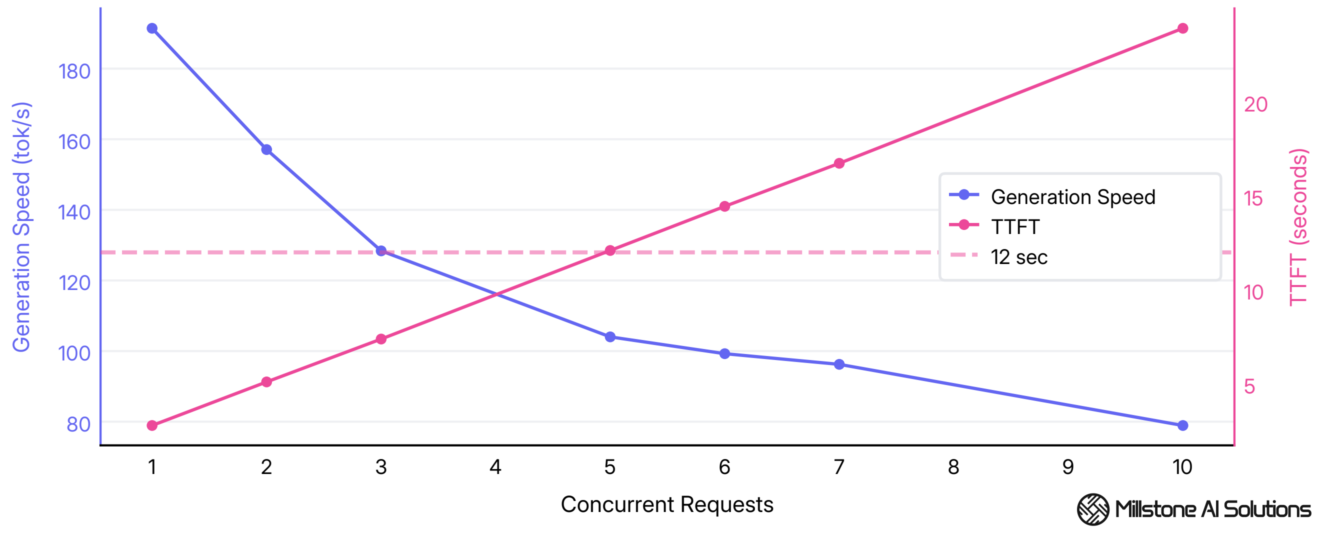
Note about caching: Most chatbot users build context incrementally over a conversation. With caching properly configured, TTFT is dramatically reduced since only new tokens require processing. These results represent worst-case TTFT where all context is processed at once.

Long Document Processing (64K Context)

Summarizing reports, extracting data from contracts, analyzing lengthy documents. 64K tokens handles documents up to roughly 125-160 pages depending on formatting and density.

Users typically tolerate higher latency for document processing since they understand large inputs require more processing time. However, generation speed still needs to stay at or above reading speed.

Thresholds: TTFT < 12s, generation speed > 15 tok/s



Average per-user generation speed and TTFT at 64K context. Dashed lines indicate quality thresholds.

METRIC	@ 1 request	@ 5 requests	@ 10 requests
TTFT	2.8s	12.1s (threshold exceeded)	23.9s (threshold exceeded)
Generation speed	191 tok/s	104 tok/s	79 tok/s

Capacity limit: 5 concurrent requests

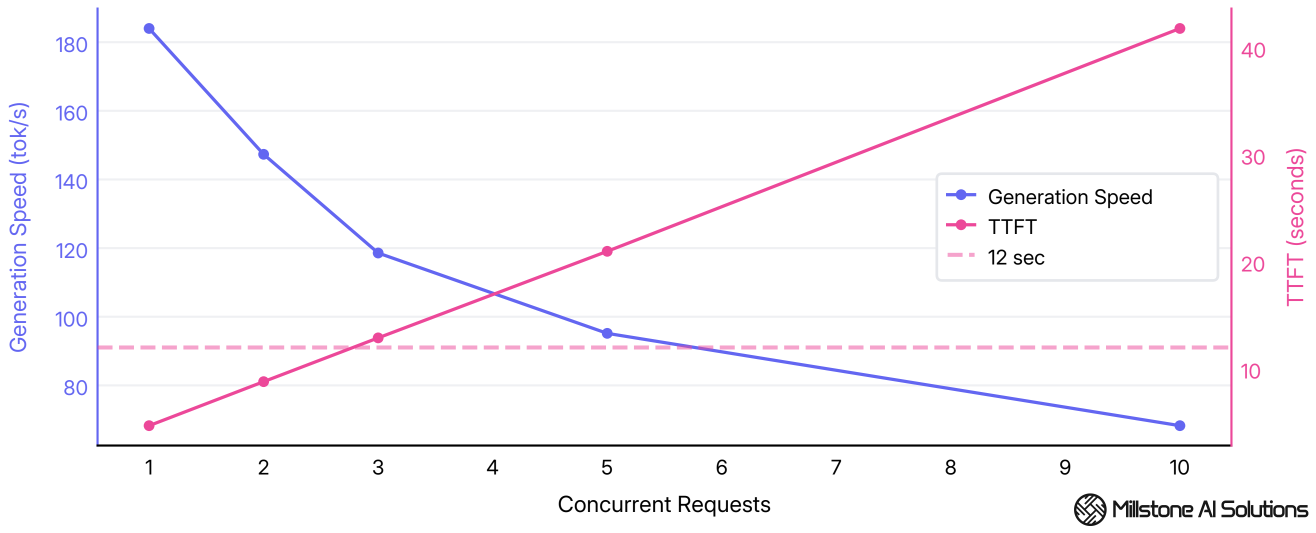
At 5 concurrent requests, TTFT reaches 12.1 seconds, just above the 12-second threshold. Generation speed at this concurrency is 104 tok/s, above the 15 tok/s minimum.

Automated Coding Assistant (96K Context)

Agentic coding workloads: AI assistants that read large portions of a codebase to answer questions, refactor code, or implement features. 96K tokens handles roughly 8,000-9,000 lines of code, enough for significant repository context.

Agentic workflows chain multiple LLM calls (tool use, retrieval, iterative refinement). With caching properly configured, context persists between requests and only new tokens require processing, dramatically reducing TTFT for each step. These results represent worst-case TTFT where all context is processed at once.

Thresholds: TTFT < 12s, generation speed > 20 tok/s



Average per-user generation speed and TTFT at 96K context. Dashed lines indicate quality thresholds.

METRIC	@ 1 request	@ 2 requests	@ 10 requests
TTFT	4.7s	8.8s	41.8s (threshold exceeded)
Generation speed	184 tok/s	147 tok/s	68 tok/s

Capacity limit: 2 concurrent requests

At 2 concurrent requests, TTFT reaches 8.8 seconds, just under the 12-second threshold. Generation speed at this concurrency is 147 tok/s, above the 20 tok/s minimum.

DEEP DIVE

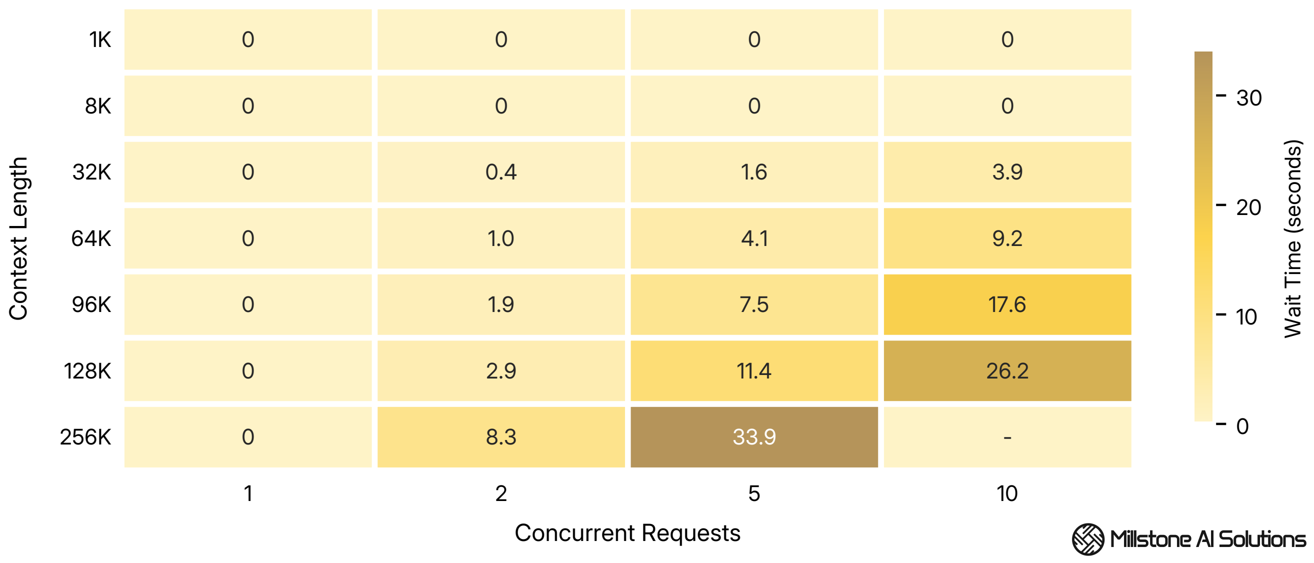
Technical Analysis

Infrastructure-level metrics that explain user-facing performance. Queue depth, prefill throughput, token generation latency, and scaling efficiency across load conditions. These help diagnose bottlenecks and validate infrastructure decisions.

Queue Wait Times

Time a request waits for GPU availability before processing begins. At low concurrency, queue wait is near zero. As load increases, requests queue and wait times grow.

Queue wait is included in TTFT. Breaking it out separately helps identify whether latency is caused by GPU saturation (high queue wait) or context processing (high prefill time).



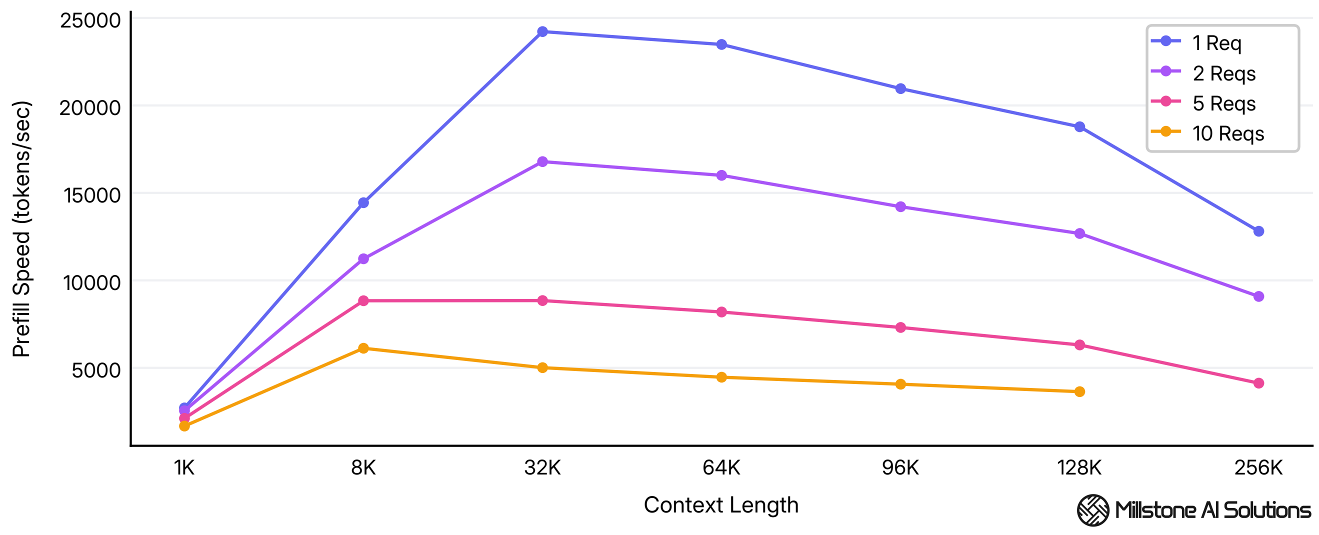
Average queue wait time across 1K - 256K tokens context at 1 - 10 concurrent requests.

At single concurrency, queue wait is effectively zero regardless of context length. At **5 concurrent requests** with **256K context**, queue wait reaches **33.9 seconds**. Rising queue times signal GPU saturation, meaning requests are waiting for resources rather than being processed immediately.

> **Interpretation:** Queue wait time and prefill time are measured independently and may not sum exactly to TTFT. Under heavy load, chunked prefill and preemptions can cause these metrics to overlap, sometimes resulting in queue wait + prefill exceeding TTFT. Use queue wait for capacity planning and identifying bottlenecks. Use TTFT for actual user wait time before streaming begins.

Per-User Prefill Speed

Rate at which the model processes input context before generating output. Prefill speed determines the non-queue portion of TTFT. Higher prefill speeds mean faster time-to-first-token at a given context length.



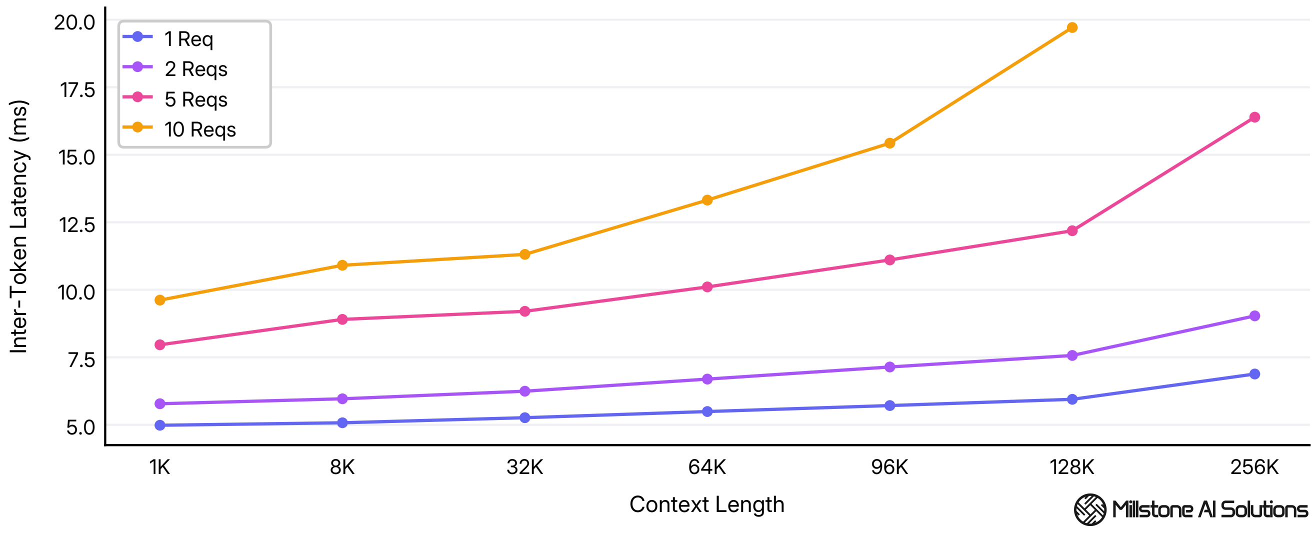
Average per-user prefill speed across 1K - 256K tokens context at 1 - 10 concurrent requests.

CONCURRENT REQUESTS	PEAKS AT	PEAK SPEED
1	32K context	**24,221** tok/s
2	32K context	**16,787** tok/s
5	32K context	**8,845** tok/s
10	8K context	**6,120** tok/s

Prefill speed peaks at a certain context length and then declines as additional context increases computational overhead. This peak can reflect GPU saturation (compute or memory bandwidth fully utilized) or engine configuration such as chunked prefill limits, which cap tokens processed per forward pass to maintain responsiveness under load. On the chart, this appears as lines that peak before reaching the longest context.

Inter-Token Latency

Time between consecutive tokens during generation. Determines the smoothness of responses. Lower latency produces more fluid output. ITL helps diagnose the underlying token-level behavior.

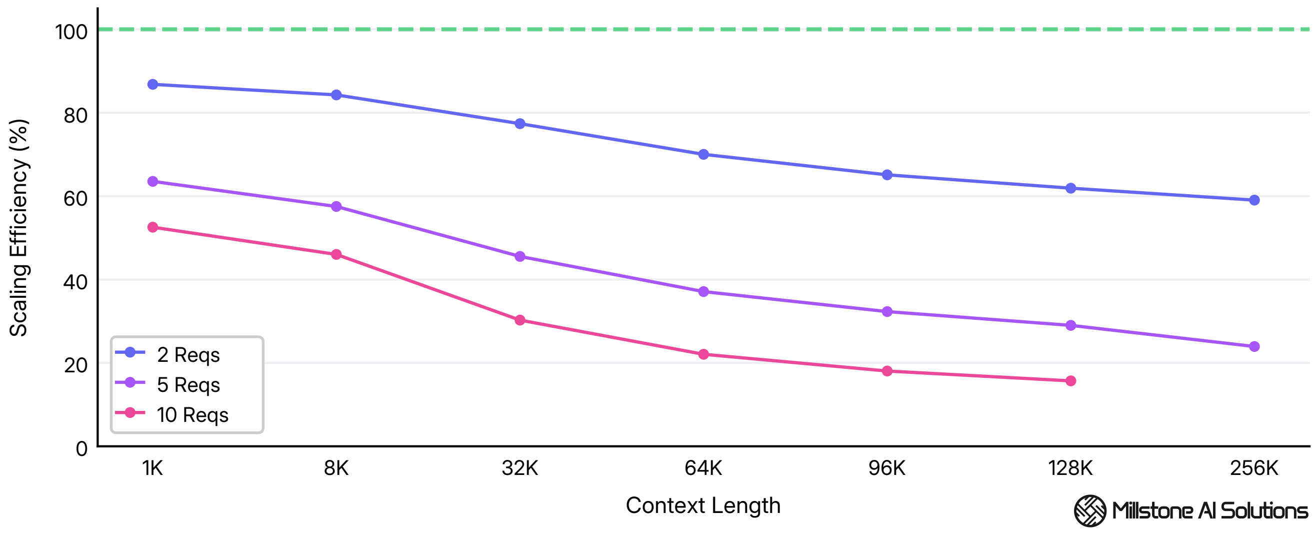


Average inter-token latency across 1K - 256K tokens context at 1 - 10 concurrent requests.

At single-user short context, inter-token latency is imperceptible (**5ms**). The highest latency observed was **20ms** at **128K** context with **10 concurrent requests**, still imperceptible to users.

Scaling Efficiency

Percentage of ideal linear scaling achieved as concurrency increases. 100% efficiency means doubling concurrent requests doubles total throughput with no per-user degradation. Real-world efficiency is always lower due to shared GPU resources.



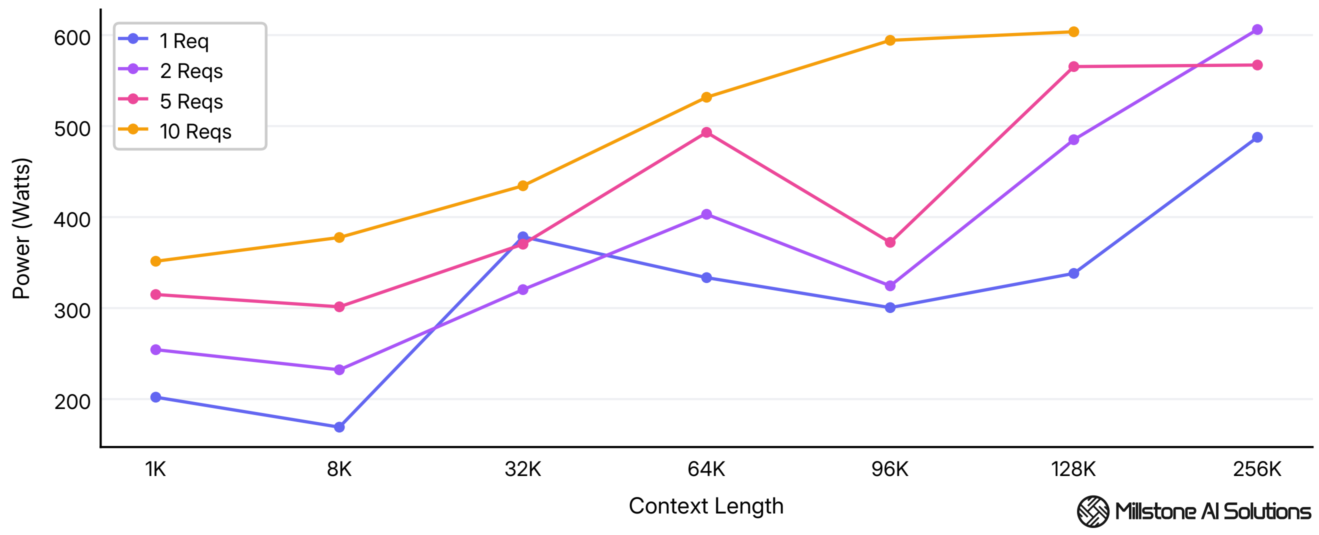
Scaling efficiency across 1K - 256K tokens context at 1 - 10 concurrent requests.

Efficiency remains high at low concurrency where GPU resources can serve requests without contention. At higher concurrency, efficiency drops as requests compete for shared resources. High efficiency at your target concurrency indicates headroom for growth. Sharply dropping efficiency signals diminishing returns.

EFFICIENCY

Power Consumption

GPU power draw under varying load conditions. Relevant for operational cost estimation, cooling requirements, and data center power budgeting.



Average GPU power draw across 1K - 256K tokens context at 1 - 10 concurrent requests.

Power consumption scales with both context length and concurrency. The highest power draw observed was **606W** at **256K** context with **2 concurrent requests**, costing approximately **$0.06/hour** at $0.10/kWh. Higher concurrency or sustained load beyond tested conditions may increase power consumption further. For infrastructure planning, budget for peak power draw.

Need Help Deciding?

Not sure what configuration you need? Our team can help you identify the right model, hardware, and deployment strategy for your specific use case.

Schedule a Conversation →

Additional data available on request: full percentile breakdowns (P50–P99) and GPU metrics.