Millstone AI Solutions

*On-Prem AI Solutions*

BENCHMARK REPORT

# DeepSeek-V4.1-Flash

## Performance Analysis on 4x RTX Pro 6000 Blackwell

### MODEL

| | |
|---|---|
| Organization | **DeepSeek** |
| Parameters | **552B** |
| Precision | **FP4+FP8** |

### TEST HARDWARE

| | |
|---|---|
| GPU | **4x RTX Pro 6000 Blackwell** |
| VRAM | **384GB** |
| Engine | **vLLM** |

### HIGHLIGHTS

**637.6**
Tok/s Peak Throughput
@ 10 Concurrent Requests

**100.0%**
Success Rate
Across All Scenarios

**11**
Concurrent Users
@ 32K Context

October 01, 2026

MillstoneAI.com

CONTENTS

# Table of Contents


**Executive Summary** 3

**Use Case Guidance** 4

**Performance Analysis** 5

- System Throughput 5
- Per-User Generation Speed 6
- Time to First Token 7

**Capacity Analysis** 8

- Code Completion (1K Context) 8
- Short-form Chatbot (8K Context) 9
- General Chatbot (32K Context) 10
- Long Document Processing (64K Context) 11
- Automated Coding Assistant (96K Context) 12

**Technical Deep Dive** 13

- Queue Wait Times 13
- Per-User Prefill Speed 14
- Inter-Token Latency 15
- Scaling Efficiency 15

**Power & Efficiency** 16


> **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/deepseek-v4-1-flash-fp4fp8-4x-rtx-pro-6000-blackwell

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 **DeepSeek-V4.1-Flash** (DeepSeek, 552B parameters, Mixture-of-Experts) running in FP4+FP8 precision on 4x RTX Pro 6000 Blackwell (384GB VRAM).

**Test parameters:** Context lengths from 1K - 1M tokens. Concurrency from 1 - 10 requests. 1024 output tokens per request. No prompt caching. Speculative decoding enabled. FP8 KV cache.

Benchmark methodology →

## Key Findings

| | |
|---|---|
| **Peak System Throughput** | 637.6 tok/s @ 10 concurrent requests, 1K context |
| **TTFT Single Request** | 320ms (1K context) → 119.6s (1024K context) |
| **Generation Speed Single Request** | 166.3 tok/s (1K context) → 120.6 tok/s (1024K context) |
| **Chatbot Capacity** | 11 concurrent requests @ 32K context |
| **Throughput Scaling** | 4.8× from 1 to 10 concurrent requests |
| **Success Rate** | 100.0% across 2.7K 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 **5** concurrent requests within accepted thresholds. |
| **Short-form Chatbot** | 10s | 10 tok/s | Supports **~41** concurrent requests within accepted thresholds. |
| **General Chatbot** | 8s | 15 tok/s | Supports **11** 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. |

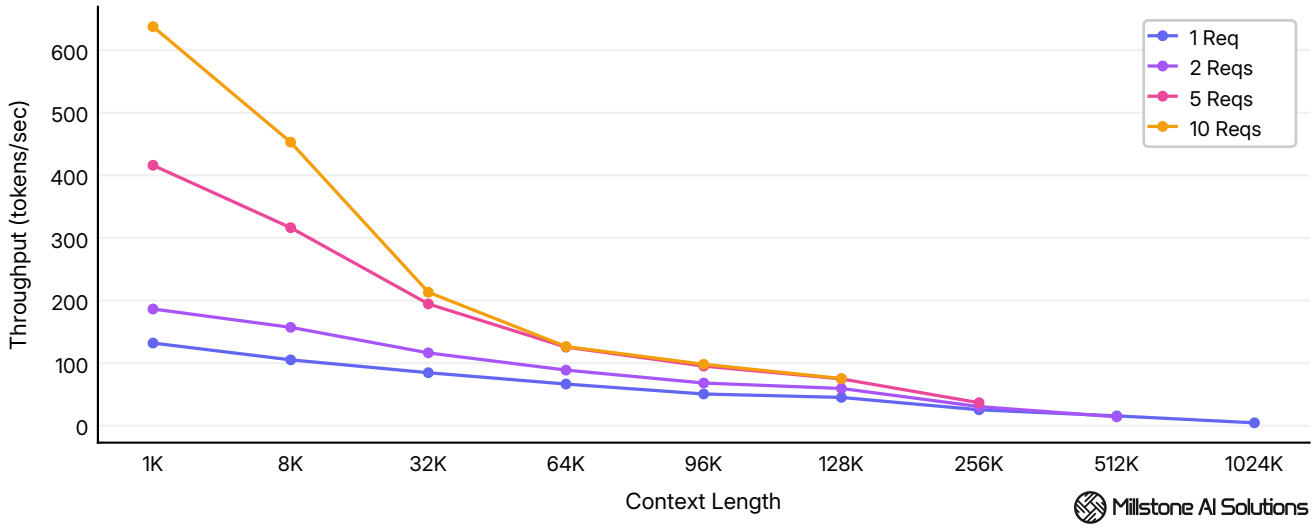
The limits shown are conservative. Beyond these points, the system continues functioning with slower response times that may still be acceptable for your specific use case.

> 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 - 1024K tokens context lengths at 1 - 10 concurrency levels.*

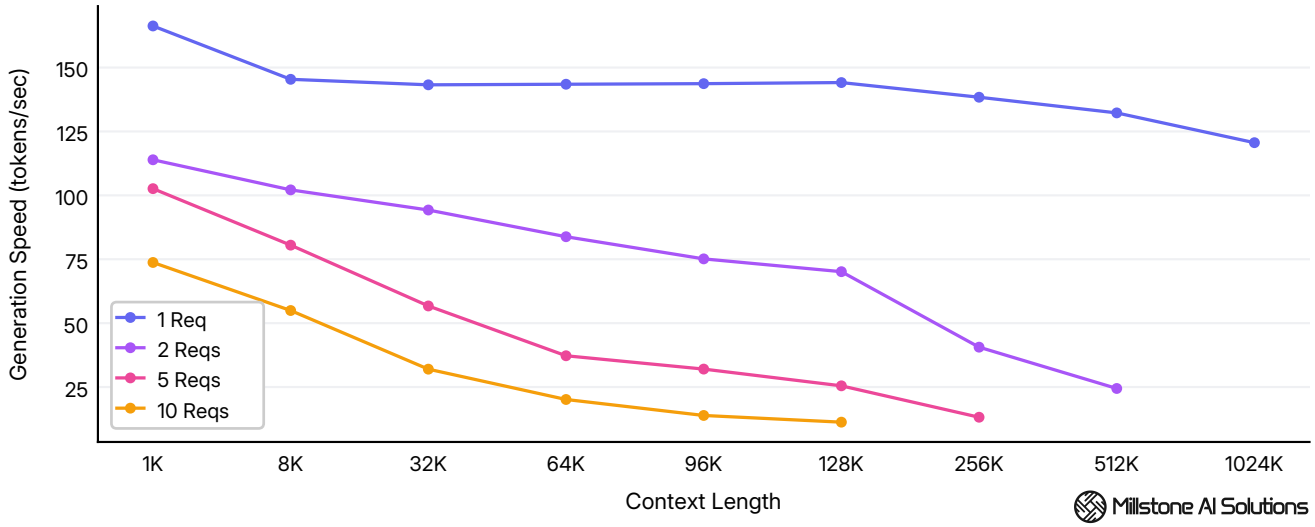
| CONDITION | THROUGHPUT |
|---|---|
| Peak (1K context, 10 requests) | **637.6** tok/s |
| 32K context, 10 requests | **213.3** tok/s |
| 128K context, 10 requests | **75.5** tok/s |

At peak throughput, this configuration produces approximately **2.3 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 - 1024K 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 **11.3** tok/s (the lowest measured point: 128K context, 10 concurrent requests), this configuration slows below fast reading speed in the most demanding scenarios. Single-user performance at 1K context reaches **166.3** tok/s.

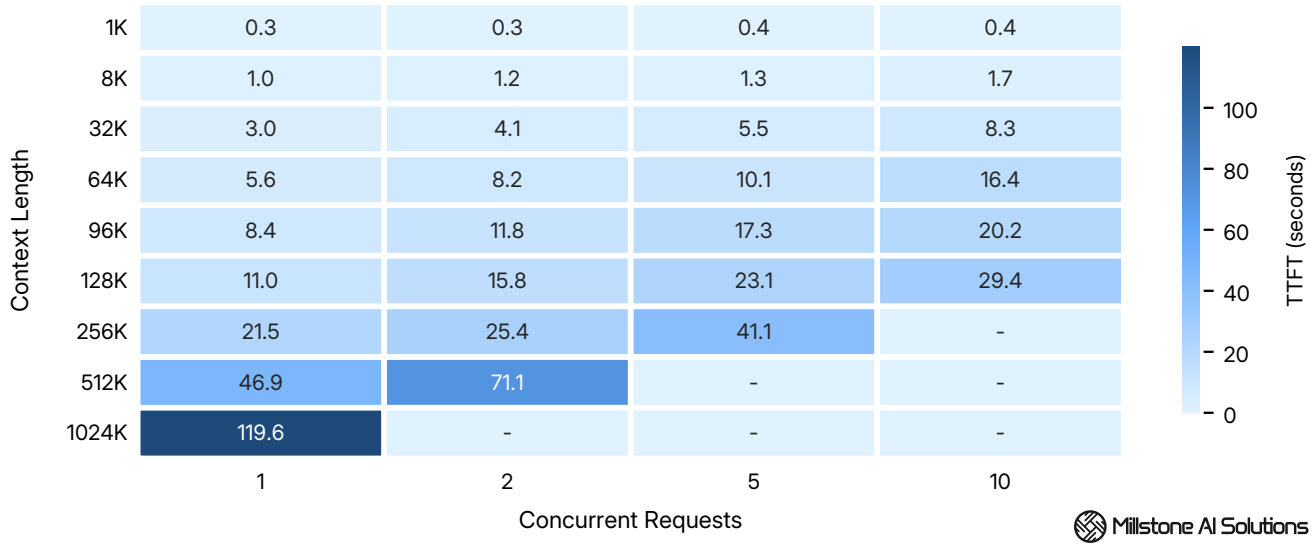
LATENCY

# 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.




*Average time to first token across 1K - 1024K 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 PLANNING

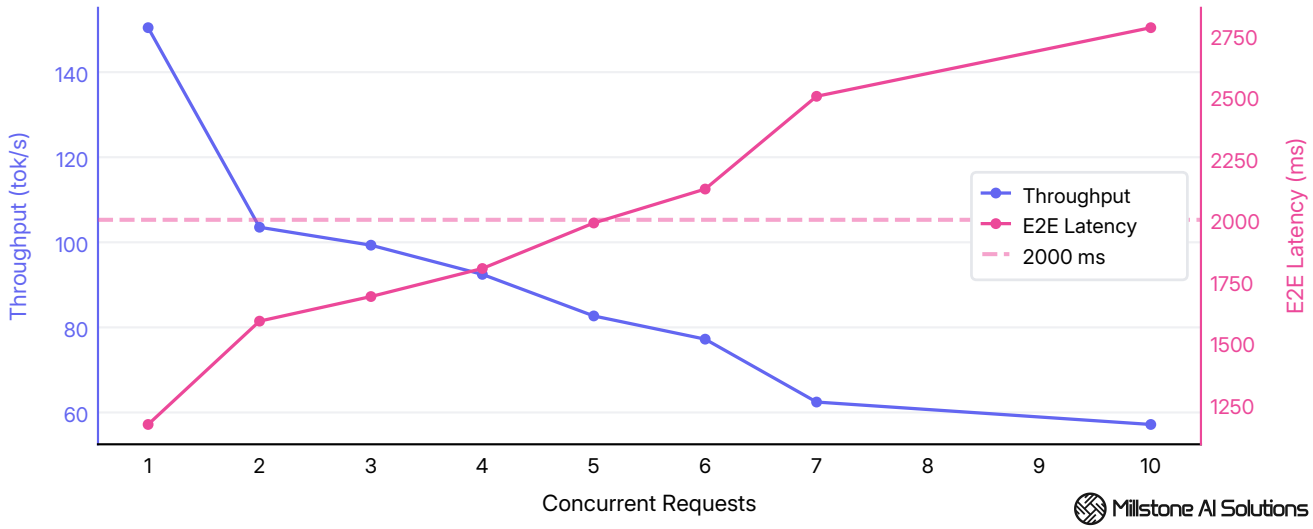
# Capacity Analysis

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 | @ 5 requests | @ 10 requests |
|---|---|---|---|
| End-to-end latency | 1,168ms | 1,987ms | 2,781ms (threshold exceeded) |
| Throughput | 150 tok/s | 83 tok/s | 57 tok/s |

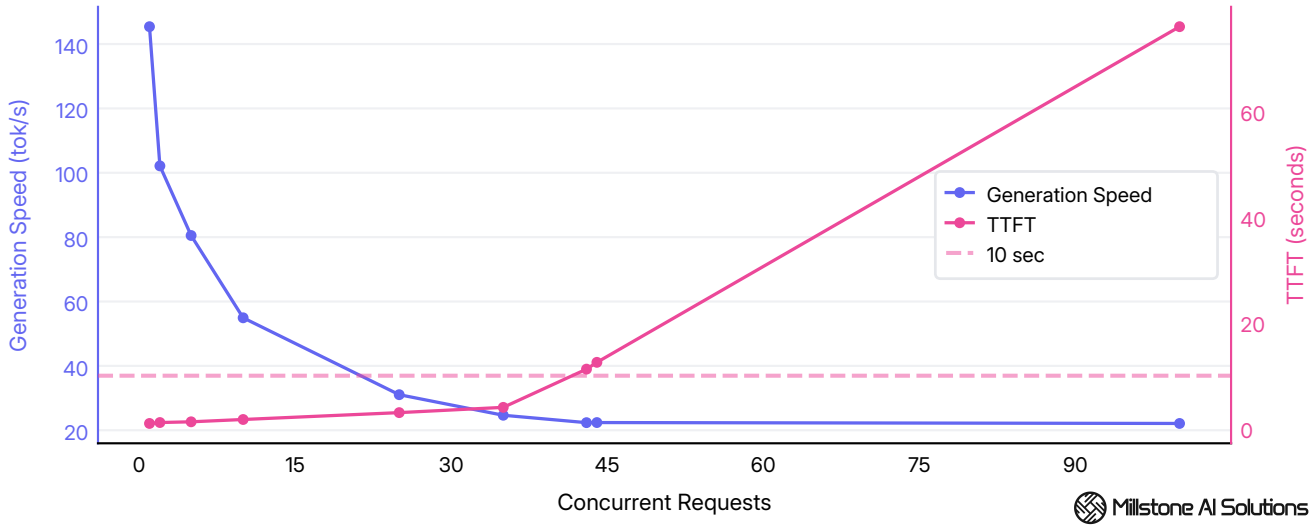
**Capacity limit: 5 concurrent requests**

At 5 concurrent requests, end-to-end latency reaches 1,987ms, just under 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 | @ 41 requests | @ 100 requests |
|---|---|---|---|
| TTFT | 1.0s | ~9.4s | 75.9s (threshold exceeded) |
| Generation speed | 145 tok/s | ~23 tok/s | 22 tok/s |

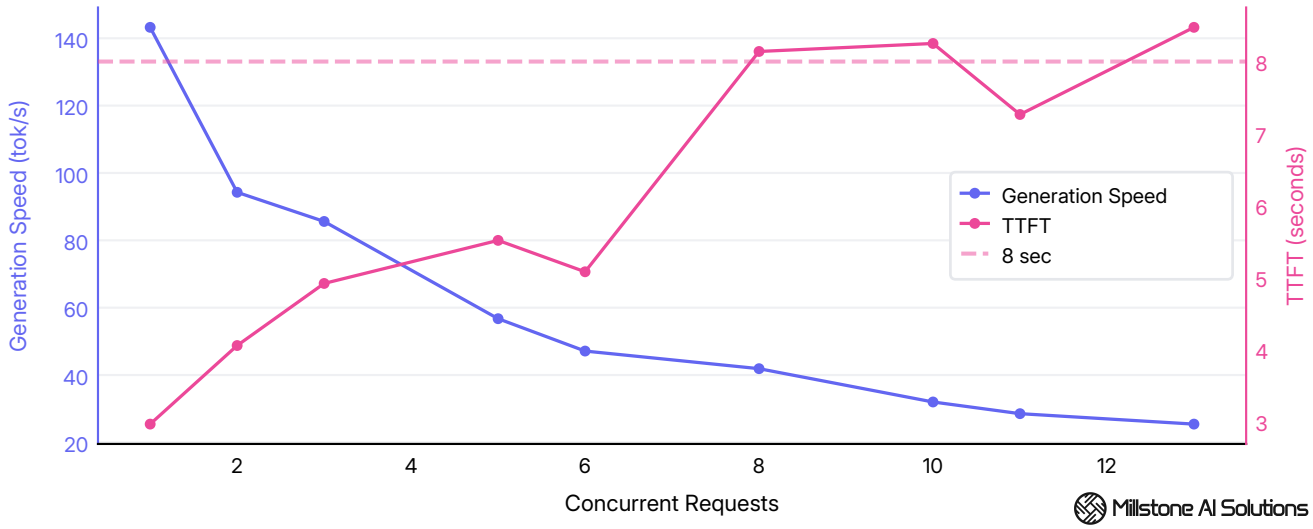
**Capacity limit: ~41 concurrent requests**

At 41 concurrent requests, TTFT reaches ~9.4 seconds, just under the 10-second threshold. Generation speed at this concurrency is ~23 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.

**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 | @ 11 requests | @ 13 requests |
|---|---|---|---|
| TTFT | 3.0s | 7.3s | 8.5s (threshold exceeded) |
| Generation speed | 143 tok/s | 29 tok/s | 25 tok/s |

**Capacity limit: 11 concurrent requests**

At 11 concurrent requests, TTFT reaches 7.3 seconds, just under the 8-second threshold. Generation speed at this concurrency is 29 tok/s, above the 15 tok/s minimum.

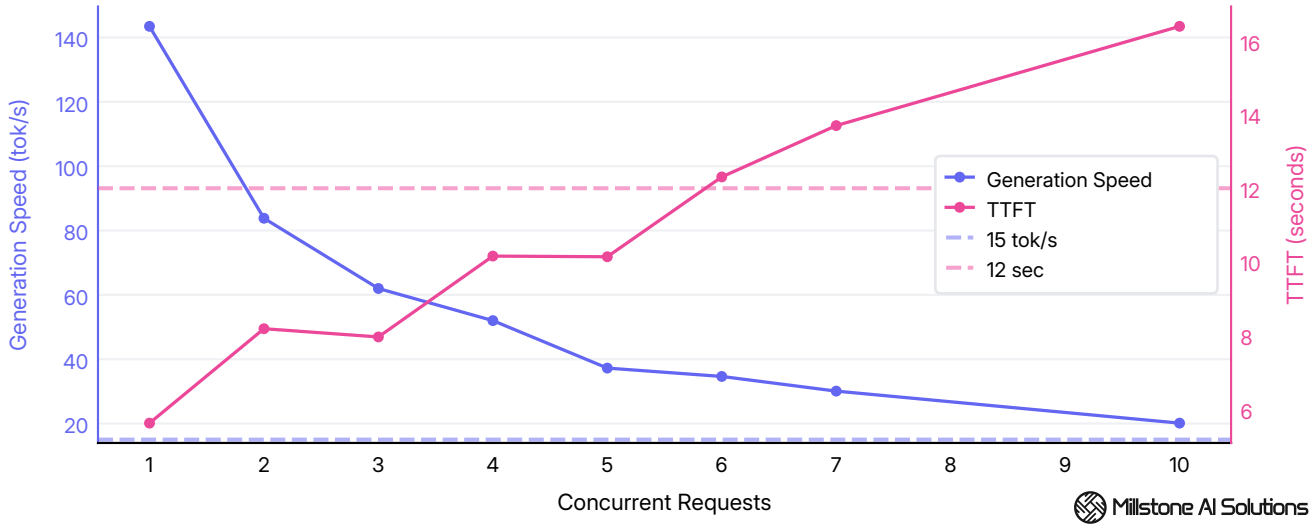
> **Note about caching:** Most chatbot users build context incrementally over a conversation. With caching properly configured, TTFT is dramatically reduced since only new tokens require processing. These results represent worst-case TTFT where all context is processed at once.

# 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 | 5.6s | 10.1s | 16.4s (threshold exceeded) |
| Generation speed | 143 tok/s | 37 tok/s | 20 tok/s |

**Capacity limit: 5 concurrent requests**

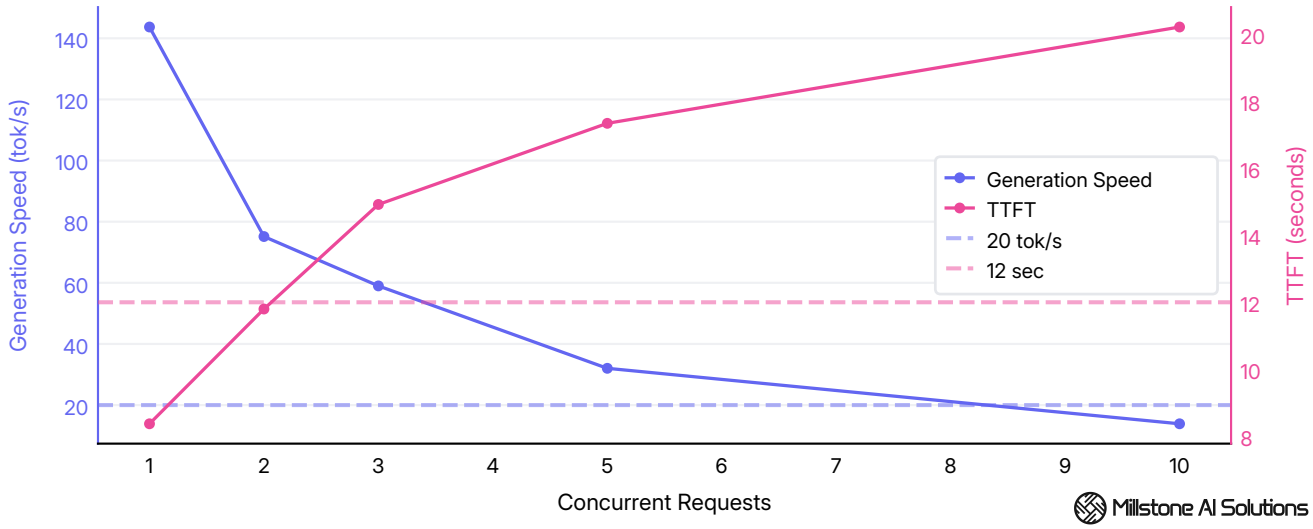
At 5 concurrent requests, TTFT reaches 10.1 seconds, just under the 12-second threshold. Generation speed at this concurrency is 37 tok/s, above the 15 tok/s minimum.

# 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 | 8.4s | 11.8s | 20.2s (threshold exceeded) |
| Generation speed | 144 tok/s | 75 tok/s | 14 tok/s (below threshold) |

**Capacity limit: 2 concurrent requests**

At 2 concurrent requests, TTFT reaches 11.8 seconds, just under the 12-second threshold. Generation speed at this concurrency is 75 tok/s, above the 20 tok/s minimum.

DEEP DIVE

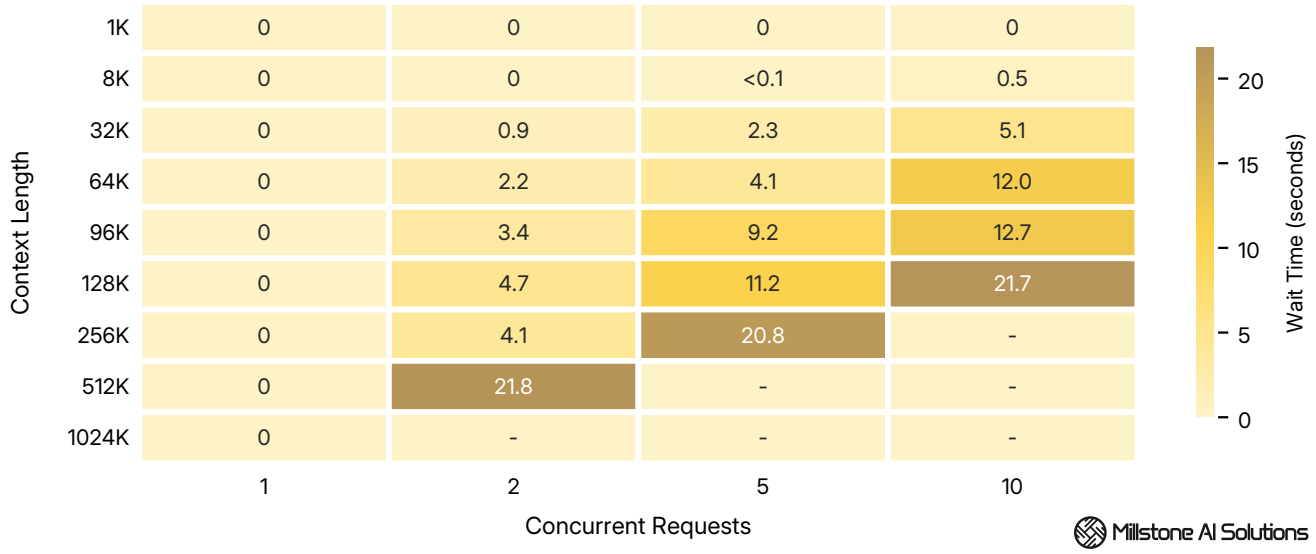
# Technical Analysis

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).



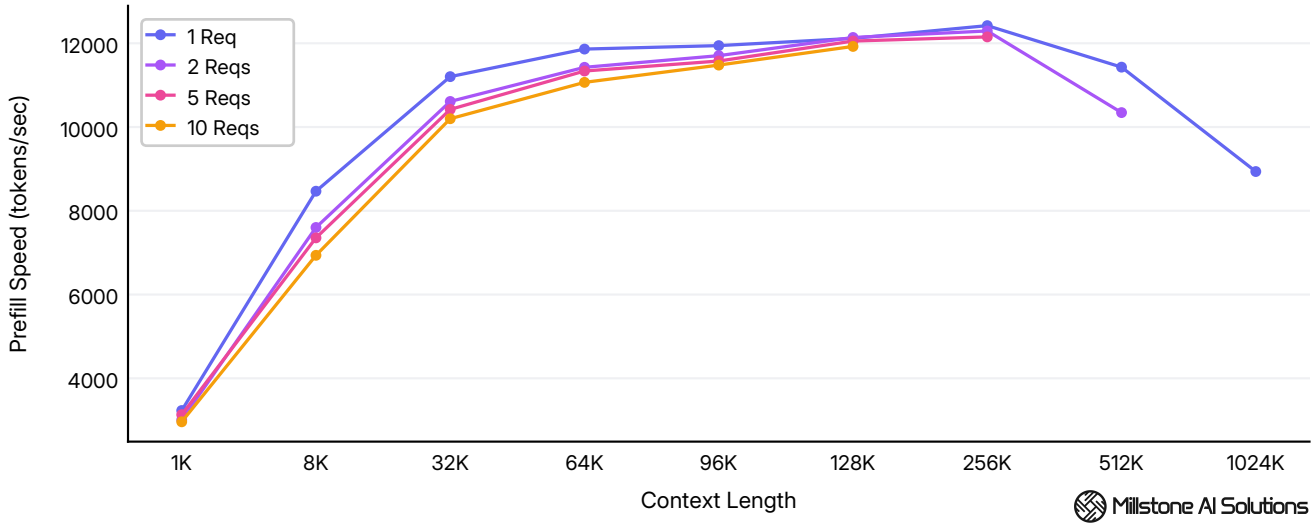

*Average queue wait time across 1K - 1024K tokens context at 1 - 10 concurrent requests.*

At single concurrency, queue wait is effectively zero regardless of context length. At **5 concurrent requests** with **256K context**, queue wait reaches **20.8 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.



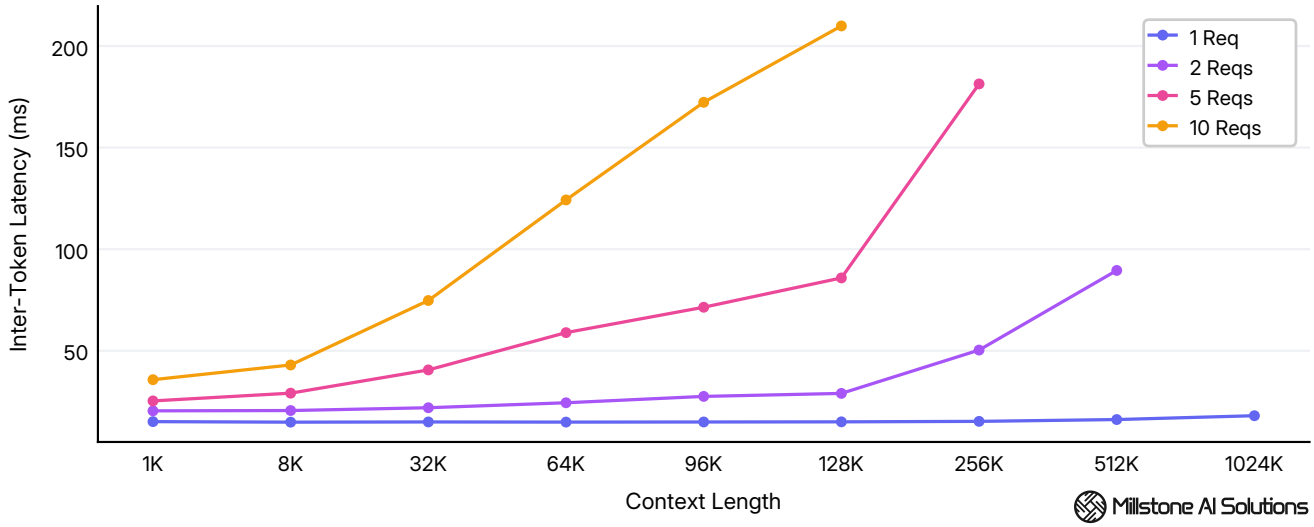

*Average per-user prefill speed across 1K - 1024K tokens context at 1 - 10 concurrent requests.*

| CONCURRENT REQUESTS | PEAKS AT | PEAK SPEED |
|---|---|---|
| **1** | 256K context | **12,421** tok/s |
| **2** | 256K context | **12,294** tok/s |
| **5** | No saturation | — |
| **10** | No saturation | — |

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.

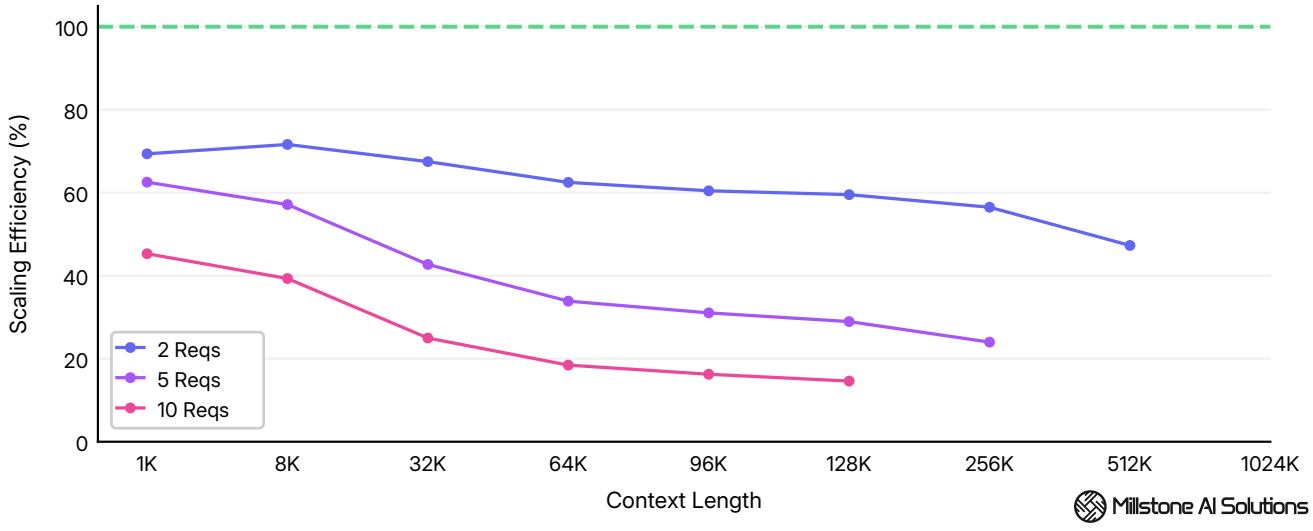



*Average inter-token latency across 1K - 1024K tokens context at 1 - 10 concurrent requests.*

At single-user short context, inter-token latency is imperceptible (**15ms**). The highest latency observed was **210ms** at **128K** context with **10 concurrent requests**, where individual tokens become visible as they stream.

## 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.



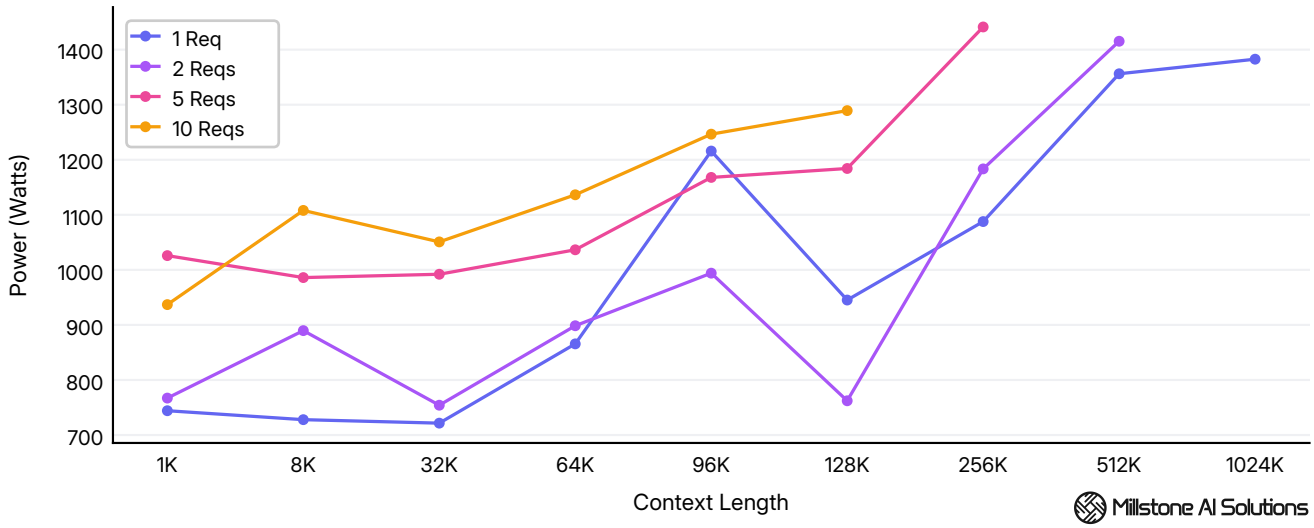

*Scaling efficiency across 1K - 512K tokens context at 1 - 10 concurrent requests.*

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 - 1024K tokens context at 1 - 10 concurrent requests.*

Power consumption scales with both context length and concurrency. The highest power draw observed was **1441W** at **256K** context with **5 concurrent requests**, costing approximately **$0.14/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.