Running a 125-Billion-Parameter Model on a Normal PC

Strata: a CUDA and CPU inference engine that serves Qwen3.8-Flash-Next from one 12 GB graphics card, 64 GB of RAM and a six-core desktop processor · September 2026

Abstract. Qwen3.8-Flash-Next is a 125-billion-parameter mixture-of-experts model that activates only 6 billion parameters per token[1–3]. Its smallest good-quality quantization is a 66 GB file[4], five and a half times the memory of a 12 GB graphics card. Strata is an inference engine written for this one model: it keeps the small, always-used part of the network on the GPU, keeps all 24,576 experts in system RAM, computes the experts the GPU does not hold on the CPU in the same instant, and uses the model's own multi-token-prediction layer to verify several tokens per pass. On an RTX 5070 (12 GB), a Ryzen 5 7600 and 64 GB of DDR5, it generates 95 tokens per second at a 4K context with the 2-bit Q2_0 file, 45–95 tokens per second across all three quantizations and contexts from 1K to the full 262K, and reads prompts at 285–571 tokens per second. This paper explains how the engine works, what limits it, what we measured and what could make it faster, in plain language wherever possible.

1 The problem in one paragraph

A large language model produces text one token (roughly one word piece) at a time, and for every token the hardware has to read the model's weights that the token uses. On a normal PC the fastest memory is the graphics card's VRAM (hundreds of GB/s), the next is system RAM (tens of GB/s) and the slowest is the SSD. A 66 GB model cannot live in a 12 GB card. The usual answer, running the rest from system RAM on the CPU with a general-purpose engine such as llama.cpp[9], works but leaves most of the machine idle most of the time. Strata's answer is to specialize: to write the engine around the exact shape of this model and this kind of PC, so that the GPU, the CPU, the RAM and the PCIe link are all busy with the work each is best at, at the same time.

2 Why this model fits a small GPU at all

The model's designers priced every architectural choice in bytes moved per token[1], and it shows. Of its 48 layers, 36 use Gated DeltaNet[10], a recurrent mixer whose memory is a fixed-size state instead of a growing cache; the other 12 use Qwen Sparse Attention, which has only two key/value heads and attends to at most about 2,048 selected positions no matter how long the context is[1, 21]. Every layer ends in a mixture of 512 experts of which the router picks 10, so a token touches only 2% of the expert weights. A 51-billion-parameter n-gram embedding table is addressed by the last three token ids, so its rows are known before they are needed. Table 1 is the byte census of the 2-bit file.

Part of the model	Stored (Q2_0 file)	Read per token	Where Strata keeps it
Routed experts (48 layers × 512)	34.0 GB	0.66 GB (2%)	**RAM** (all) + **VRAM** cache (the most-used ~4,500)
Mixers, attention, shared experts, routers	1.8 GB	1.8 GB	**VRAM**
Gated residual (hyper-connections)	1.3 GB	1.3 GB	**VRAM**
Output head	0.44 GB	0.44 GB	**VRAM**
N-gram embedding table	28.8 GB	≤ 23 KB	**SSD** + OS page cache
Multi-token-prediction (MTP) layer	0.8 GB (from the BF16 checkpoint)	per draft	**VRAM**

Table 1. Where the bytes are. The GPU-resident part is about 3.5 GB; the expert traffic, 0.66 GB per token, is the quantity the design is organized around[4, 9].

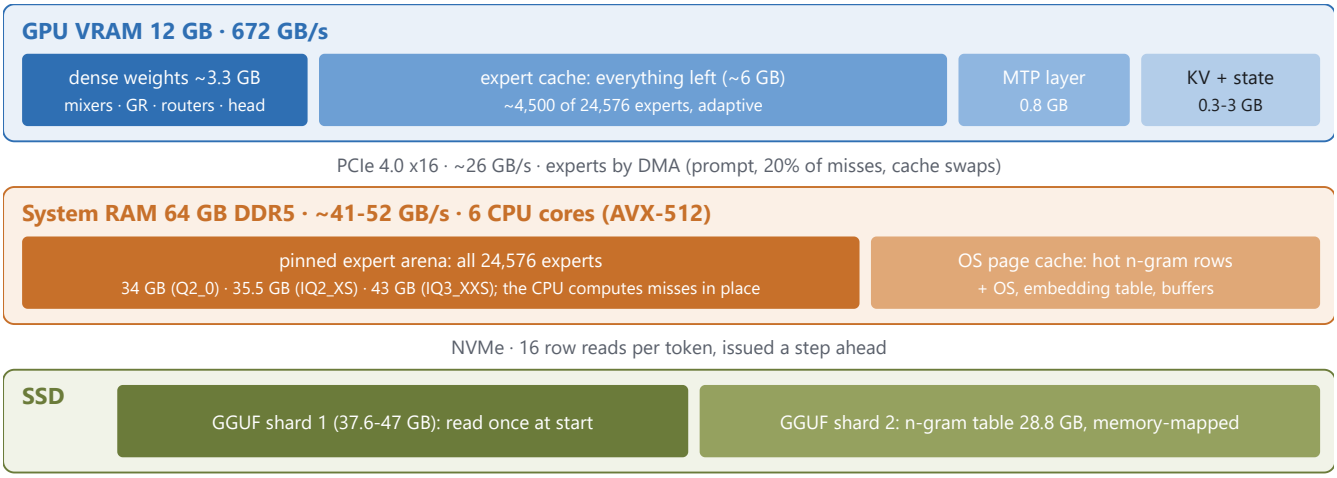

Figure 1. The three memory tiers and what Strata places in each (RTX 5070 12 GB, 64 GB DDR5, PCIe 4.0 x16, NVMe SSD). Bandwidths are measured on the test machine.

3 How Strata works

3.1 Three memory tiers

VRAM holds everything that every token needs: the attention and DeltaNet mixers, the gated-residual weights, the routers, the shared experts, the output head, the MTP draft layer, the key/value cache and the recurrent state, and an *expert cache* that fills whatever VRAM is left. **System RAM** holds all 24,576 experts in one pinned (page-locked) arena, so the CPU can compute any of them in place and the GPU can pull any of them over PCIe by DMA. The **SSD** keeps the 28.8 GB n-gram table; the engine reads the 16 rows each token needs through the operating system's page cache, one step ahead of the layer that uses them.

3.2 One layer, three workers at once

For every layer the GPU runs the mixer and the router and writes the ten chosen expert ids into a small block of pinned host memory (a “doorbell”). The CPU, which spins on that doorbell instead of waiting on the driver, immediately splits the experts: the ones already in the VRAM cache go to the GPU, a share of the others is copied to the GPU over PCIe by its copy engine while everything else runs (55% of the misses for the i-quant files, 20% for Q2_0, whose CPU needs the RAM bandwidth itself), and the rest are computed by the CPU's cores next to the RAM. The CPU writes its results back into mapped memory, the GPU adds

everything up, and the next layer starts. The whole 48-layer pass is one captured CUDA graph per window size, so the host never waits on a synchronizing driver call.

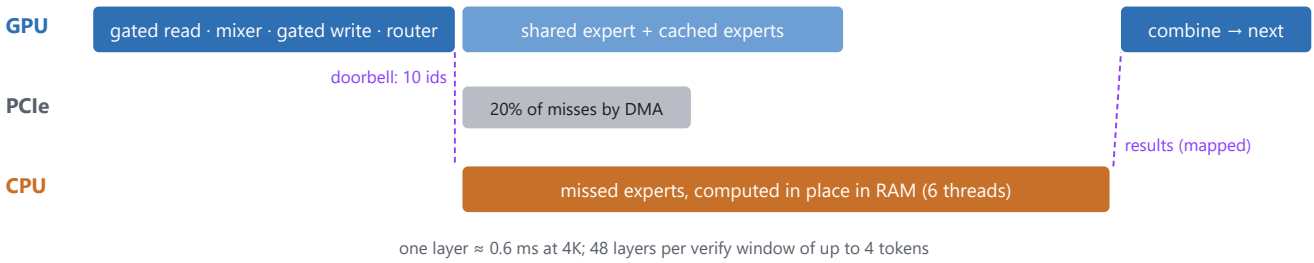

Figure 2. One layer of a verify window. The GPU's resident experts and the CPU's missed experts run at the same time; the next layer waits for the slower of the two.

3.3 Checking several tokens per pass: speculative decoding with the model's own MTP layer

Reading 3.5 GB of dense weights for every single token is the GPU's main cost, and it barely grows when four tokens are processed together. Qwen3.8-Flash-Next ships a one-layer multi-token-prediction (MTP) head[1, 12] that guesses the next tokens cheaply. The GGUF files do not include it, so Strata fetches only its tensors (5 GB) from the original BF16 checkpoint[3] and keeps it in VRAM with its experts in 2-bit form and a reduced output head over 40,525 likely tokens[7, 8]. Each round, the last accepted token and up to three drafts go through all 48 layers as one *verify window*; the engine keeps every draft the full model agrees with and one more token from the model itself[11]. A draft only enters the window while the MTP layer is at least 50% confident. The result is exact: the text is identical, token for token, to plain greedy decoding, which we test with forced wrong drafts and rollbacks.

3.4 An expert cache that follows the conversation

Routing is uneven: some experts are chosen far more often than others. At startup Strata fills the VRAM cache with the most-used experts from a profile recorded on other prompts; every four rounds it swaps up to 96 experts that the current conversation keeps asking for into the slots of the least-used ones, while the GPU is drafting. Figure 3 shows how the share of experts served from VRAM (the hit rate) grows with the number of experts that fit.

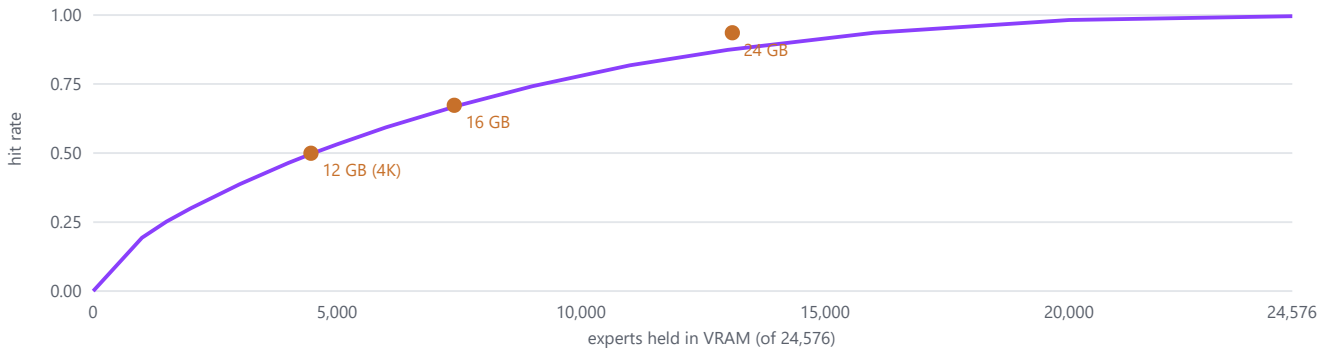

Figure 3. Hit rate of a profile-filled expert cache against its size, measured on held-out prompts (each prompt scored against a profile built from the other nine). The adaptive swapping adds to this: on the 12 GB card the engine measures 0.72 at 4K context instead of the curve's 0.50.

3.5 Reading the prompt: streaming experts to the GPU

Prompts are processed in chunks of 2,048 tokens. With that many tokens every expert is used many times, so the right place to compute is the GPU: for each layer, the experts that are not cached are copied from the pinned arena over PCIe while the GPU computes the previous ones, dequantized to half precision and

multiplied with cuBLAS[18]. The buffers are borrowed from the expert cache and refilled afterwards. Prompt speed stays flat from 4K to 262K tokens because the sparse attention reads a bounded set of positions.

3.6 Three quantizations of the same model

ISTA-DASLab publishes three GSQ-RCO quantizations[4]: **Q2_0** (2.25 bits per expert weight, the fastest), **IQ2_XS** and **IQ3_XXS** (llama.cpp's "i-quants", which encode groups of eight weights as an index into a codebook of lattice points[9]). Q2_0 is a plain grid {-1, 0, 1, 2} × scale, which suits an integer dot-product instruction; Strata repacks its experts once and computes them with a hand-written AVX-512 VNNI kernel that runs near the RAM's bandwidth. The i-quants cannot be repacked into that form without losing information, so for them Strata keeps every tensor exactly as the GGUF stores it: the GPU uses dot products and dequantizers transcribed from llama.cpp's CUDA code, the CPU uses ggml's own vector dot products, and a new AVX-512 kernel handles the Q2_0 down projections and the experts that several tokens of a window share. Each of these was checked against llama.cpp's reference arithmetic on real layers of the files.

3.7 Long context

Past 8K tokens the key/value cache is stored in 8-bit integers. The MTP layer attends to the last 32,768 positions, which keeps drafting cheap at 262K. The cost of a long context is mainly VRAM: at 262K the key/value cache of the 12 sparse attention layers, their indexer keys and the larger prompt buffers take about 4 GB, so the automatic sizing fits about 1,600 experts on the GPU instead of about 4,500.

4 Results

Machine: RTX 5070 12 GB (PCIe 4.0 x16, driving the display), AMD Ryzen 5 7600 (6 cores, AVX-512), 64 GB DDR5-5200 (EXPO off), NVMe SSD, Windows 10. **Method:** one code-agent prompt per length (a real code-review task over long source files), 256 generated tokens, greedy decoding, MTP speculation on (up to 4 tokens per window, drafts kept while their probability is at least 0.5), expert cache sized automatically from free VRAM, 8-bit KV cache from 32K up. The expert profile was recorded on different prompts. Every number is one full run of the engine from a cold start of the prompt; prompt speed includes all 48 layers and the n-gram lookups. Q2_0 runs from its repacked (canonical) pack, IQ2_XS and IQ3_XXS from native packs (section 3.6).

Prompt processing (tokens per second)

Model	1K	4K	32K	64K	128K	262K
Q2_0	389	539	571	561	543	496
IQ2_XS	332	463	495	486	472	437
IQ3_XXS	285	410	435	427	414	–

Table 2. Prompt processing speed, measured. Prompt lengths: 1,017 / 3,562 / 31,566 / 64,162 / 128,478 / 259,943 tokens (the last is the model's full 262,144-token window). IQ3_XXS at 262K was not measured: with its 43 GB expert arena, a 260K-token context and the operating system, the run brings a 64 GB machine to its memory limit, and it was stopped there. Treat 128K as the practical ceiling for IQ3_XXS on 64 GB.

Output (tokens per second)

Model	1K	4K	32K	64K	128K	262K
Q2_0	88.7	94.6	87.5	76.4	65.1	56.3
IQ2_XS	82.0	78.0	65.3	63.7	52.0	48.0
IQ3_XXS	64.6	65.6	57.3	54.4	44.8	–

Table 3. Generation speed with MTP speculative decoding, measured over 256 generated tokens.

Details

Run	1K	4K	32K	64K	128K	262K
Q2_0: time to first token (s)	2.7	6.7	55.3	114.4	236.7	523.8
Q2_0: tokens per verify window	2.71	3.23	3.11	2.80	2.65	2.46
Q2_0: experts in VRAM	4,517	4,464	4,174	3,809	3,079	1,588
IQ2_XS: time to first token (s)	3.1	7.8	63.8	132.0	272.5	595.1
IQ2_XS: tokens per verify window	3.25	3.28	2.53	2.72	2.44	2.42
IQ2_XS: experts in VRAM	4,621	4,568	4,289	3,936	3,229	1,796
IQ3_XXS: time to first token (s)	3.7	8.8	72.7	150.2	310.1	–
IQ3_XXS: tokens per verify window	3.63	3.24	2.78	2.88	2.42	–
IQ3_XXS: experts in VRAM	3,556	3,511	3,279	2,990	2,407	–

Table 4. Time to first token, tokens produced per verify window (accepted drafts + 1) and the number of experts the automatic sizing fitted into VRAM.

5 Where the time goes

A decode round (one verify window of up to four tokens) breaks down as in Table 5. “GPU” is the time the CPU waits for the GPU's per-layer work (weights read once per window, attention, the cached experts); “CPU experts” is the time the GPU waits for the CPU to finish the experts that were not in VRAM.

Model (4K)	round ms	GPU	CPU experts	drafting	other	tokens/round	VRAM hit rate	CPU GB/s
Q2_0	34.3	14.6	13.4	2.0	4.3	3.23	0.72	41
IQ2_XS	42.0	14.7	21.1	2.2	4.0	3.28	0.78	20
IQ3_XXS	49.4	15.3	27.1	2.1	4.9	3.24	0.71	24

Table 5. One verify window at 4K context, in milliseconds (measured). “CPU GB/s” is the rate at which the CPU streams expert weights while computing.

The single biggest bottleneck, in plain words.

- **CPU:** for the 2-bit Q2_0 file the CPU is not the limit, the RAM behind it is: the expert kernel already streams weights at about 42 GB/s, close to what the memory delivers while the GPU is also pulling data. For the i-quant files the CPU *itself* is the limit: decoding a codebook weight costs several instructions (table lookups, sign masks), and six cores manage only about 25 GB/s. That is why IQ3_XXS is slower even though it is only 25% bigger.
- **RAM:** bandwidth (every missed expert is read from it each round) and capacity. 64 GB is just enough: the expert arena alone is 34 GB (Q2_0), 35.5 GB (IQ2_XS) or 43 GB (IQ3_XXS). Running the RAM at its rated EXPO speed instead of the default 5200 MT/s would speed up the CPU half directly. On a DDR4 system (roughly half the bandwidth of DDR5) the CPU half of a Q2_0 round takes about twice as long; the i-quants, which are limited by arithmetic rather than bandwidth, lose less. Either way, a bigger GPU cache is the best compensation.
- **GPU:** its *size* matters more than its speed. Every extra gigabyte of VRAM holds about 700 more experts, and every expert in VRAM is one the CPU does not have to compute. The GPU's own work (about 14 ms per window, mostly reading 3.5 GB of dense weights) is the second limit, and it is set by memory bandwidth.

Put together: on this machine the GPU and the CPU each need about half of every round, and they wait for each other at every layer. The engine is balanced, which is good, but it means that no single component upgrade doubles the speed.

6 Findings

1. **The engine is balanced, so both halves matter.** At 4K context with Q2_0 a verify window takes about 33 ms: the CPU waits about 14 ms for the GPU and the GPU about 13 ms for the CPU. Speeding up only one side cannot help by more than about a third.
2. **Speculation with the model's own MTP layer is worth 1.6–1.8×.** Plain decoding reached 47–57 tokens per second at 4K; with the verify window it is 82–92. That is less than the 2.5–3× engines report when the whole model is in VRAM[7, 8], because every extra token in the window routes to its own CPU experts, but more than the 1.1–1.7× the original design estimated.
3. **Exact is achievable and worth insisting on.** The speculative output is identical to plain greedy decoding token for token, including with deliberately wrong drafts. Against llama.cpp the mean KL divergence of the next-token distribution is 0.022, below the 0.058 that llama.cpp's own CPU and GPU builds differ by[9].

4. **An adaptive expert cache beats a static one by a wide margin.** A profile recorded on other prompts serves 50% of the routed experts from 4,500 VRAM slots; swapping in the conversation's own hot experts raises that to about 72%.
5. **Overlapping the two halves inside a layer did not pay.** Splitting each window into two token groups, so that one group's CPU experts run while the GPU processes the other group, is exact but about 7% slower (85.8 vs 91.8 tokens per second at 4K): the dense weights are read twice and each group shares fewer experts across its tokens.
6. **The biggest CPU gain came from the instruction mix.** Rewriting the Q2_0 expert kernel for 512-bit AVX-512 (one unpack, one VNNI dot product per 64 weights) brought the CPU to about 42 GB/s of expert weights, close to what the RAM delivers while the GPU is also reading from it (41 GB/s concurrent, 52 GB/s alone).
7. **The i-quants are limited by CPU arithmetic, not by RAM.** Decoding a codebook weight costs table lookups and sign handling; both ggml's AVX2 kernels and our AVX-512 ones run at about 5 GB/s per core, 23–26 GB/s on six cores. Decoding once for several tokens helps (2.0–2.4× at three tokens), but most experts serve a single token. ggml's CPU backend has no SIMD kernel for Q2_0 on x86 at all; the IQ files' Q2_0 down projections needed their own (7× faster than the scalar one).
8. **Size the cache in bytes, not in slots.** IQ3_XXS experts range from 1.3 to 2.3 MB depending on the layer. Sizing each VRAM slot to its layer instead of to the largest expert fits 3,556 experts instead of 2,673 and raised decode by 13% at 1K.
9. **The overlap that paid: the copy engine.** A share of the missed experts can be read by the GPU straight from RAM over PCIe. Done by a copy kernel, those reads sat on the GPU's critical path, and more than 20% of the misses made decoding slower. Issuing them instead as DMA copies from the CPU thread the moment it plans the layer lets the GPU's copy engine run beside both the CPU's experts and the GPU's own work; the GPU computes its cached experts first and the copied ones when they land. For the i-quant files, whose CPU is limited by arithmetic and leaves RAM bandwidth free, 55% of the misses now go over PCIe: IQ2_XS at 4K went from 70 to 78 tokens per second and IQ3_XXS from 56 to 65. Q2_0 keeps the copy kernel at 20%: its CPU already uses the RAM's bandwidth, so DMA would only take bandwidth away from it.
10. **Refill the cache without making anyone wait.** The adaptive cache used to copy up to 96 experts (130–200 MB) into VRAM between rounds, and the next round waited for the copies. Evicting the old expert at once and admitting the new one only when its copy has landed removed the wait: Q2_0 at 4K went from 91.7 to 94.4 tokens per second.
11. **Prompt speed is flat with context length.** 539 tokens per second at 4K and 496 at 262K: the sparse attention reads a bounded set of positions. It is bound by per-expert kernel launches and half-precision dequantization, using only about 7 GB/s of the 26 GB/s PCIe link.
12. **Long context costs VRAM more than compute.** At 262K tokens the 8-bit KV cache leaves room for about 1,600 experts instead of 4,500, and the MTP layer's drafts are accepted less often; together they take Q2_0 decoding from about 95 to about 56 tokens per second.
13. **Windows needs care with pinned memory.** Page-locking 34–43 GB as one range fails on Windows; per-layer ranges plus a working-set lock for the remainder work, and a PCIe read of a missed expert must never span two ranges.

7 What could make it faster

- **Keep the conversation's state between requests.** Today every request processes its whole prompt again. Saving the recurrent state at the end of each turn (a few hundred MB) and the KV cache would

turn a 30-second re-read of a long chat into a fraction of a second[5, 8]. For agents this is the largest single improvement available.

- **Overlap across layers, not within one.** Splitting each window into two token groups so that one group's CPU experts overlap the other group's GPU work is exact but slower (Finding 5); moving part of the misses to the copy engine is what paid (+10–16% for the i-quants). The next step in the same direction is to predict the next layer's experts from the current hidden state[15, 16] and start their copies before the router has run, which would widen the share the copy engine can take without the GPU waiting for it.
- **Cheaper i-quant arithmetic on the CPU.** Lookup-table kernels in the style of T-MAC[17], or converting the most-used experts to a CPU-friendly form at load time, would lift the i-quants toward the Q2_0 speed.
- **A faster prompt path.** Prompt speed is bound by per-expert kernel launches and half-precision dequantization, not by FLOPs; a fused 8-bit grouped kernel[8, 18] would help most on short and medium prompts.
- **Sampling.** Decoding is greedy today; temperature sampling needs rejection sampling in the verify window[11].
- **Hardware settings, free:** RAM at its EXPO speed, the monitor on the motherboard's output (it holds VRAM the expert cache could use), and Linux (lower driver overhead per launch than Windows' WDDM).

8 Other graphics cards (estimates)

We could only measure on the RTX 5070. Table 6 estimates two popular cards with the same CPU and 64 GB of DDR5, using the measured breakdown of every run: the GPU part is scaled by memory bandwidth, the CPU part by the share of experts that no longer fits (from the hit-rate curve of Figure 3 with the card's extra VRAM), and prompt speed by tensor throughput. The RTX 5060 Ti 16 GB has less bandwidth than the 5070 but more room for experts, so the two roughly cancel; the RTX 3090 has both more bandwidth and twice the VRAM, which removes most of the CPU work. Treat these as ±20% until someone measures them.

Card	Model	1K prompt / output	4K prompt / output	32K prompt / output	64K prompt / output	128K prompt / output	262K prompt / output
RTX 5060 Ti 16GB	Q2_0	~341 / ~80	~472 / ~87	~501 / ~81	~492 / ~72	~476 / ~62	~435 / ~53
RTX 5060 Ti 16GB	IQ2_XS	~291 / ~80	~406 / ~77	~434 / ~63	~426 / ~62	~413 / ~51	~383 / ~47
RTX 5060 Ti 16GB	IQ3_XXS	~249 / ~66	~359 / ~65	~381 / ~56	~374 / ~54	~363 / ~45	–
RTX 3090 24GB	Q2_0	~355 / ~128	~491 / ~140	~521 / ~130	~512 / ~115	~495 / ~100	~453 / ~85
RTX 3090 24GB	IQ2_XS	~303 / ~131	~422 / ~128	~451 / ~103	~444 / ~102	~430 / ~85	~398 / ~78
RTX 3090 24GB	IQ3_XXS	~260 / ~106	~374 / ~103	~396 / ~89	~390 / ~85	~378 / ~71	–

Table 6. Estimated tokens per second, same CPU (6-core AVX-512) and 64 GB DDR5. Not measured.

9 Related work

Computing mixture-of-experts layers partly on the CPU was explored by Eliseev and Mazur[22], Fiddler[13] and KTransformers[14]; MoE-Infinity[15] and Pre-gated MoE[16] study expert caching and next-layer prediction, and PowerInfer[23] exploits hot and cold activations on consumer GPUs. The idea of one engine for one model comes from Splash[5, 6], ninfer[7] and HyperQwen[8]. Speculative decoding is due to Leviathan et al.[11] and multi-token prediction heads to Gloeckle et al.[12]; lookup-table CPU kernels to T-MAC[17]; quantized GPU matrix kernels to Marlin[18]; paged attention to vLLM[19]. The i-quant formats, their dot products and the ggml CPU backend are llama.cpp's[9, 24].

References

1. Qwen Team. *On the Design of Qwen3.8-Next Architecture: Evaluation, Efficiency, and Training Stability*. arXiv:2608.30320, Aug 2026.
2. Qwen Team. *Qwen3.8-Flash-Next: A New Architecture, Towards Ultimate Cost-Efficiency*. Blog post, Aug 2026. qwen.ai/blog?id=qwen3.8-flash-next
3. Qwen. Qwen3.8-Flash-Next model card, config.json and BF16 checkpoint. huggingface.co/Qwen/Qwen3.8-Flash-Next
4. ISTA-DASLab. Qwen3.8-Flash-Next-GSQ-RCO-GGUF model card, files and tensor-allocation/*.rco-allocation.txt. huggingface.co/ISTA-DASLab/Qwen3.8-Flash-Next-GSQ-RCO-GGUF
5. Inco AI. *Splash*. Blog post, 17 Sep 2026. inco.ai/blog/splash
6. Inco AI. splash (source repository, Apache-2.0). github.com/incoai/splash
7. Neroued. ninfer: README, docs/performance.md, docs/maintainer/engine-architecture.md, docs/maintainer/replayssm-gdn.md. github.com/Neroued/ninfer
8. syv.ai. HyperQwen: README, docs/optimizations.md, docs/benchmarks.md. github.com/syv-ai/HyperQwen
9. ggml-org. llama.cpp at commit 3cf0325: ggml/src/ggml-common.h (block formats, codebooks), ggml-cuda/vecdotq.cuh and dequantize.cuh (GPU dot products and dequantizers), ggml-cpu/quants.c (CPU dot products). MIT license. github.com/ggml-org/llama.cpp
10. S. Yang, J. Kautz, A. Hatamizadeh. *Gated Delta Networks: Improving Mamba2 with Delta Rule*. 2024.
11. Y. Leviathan, M. Kalman, Y. Matias. *Fast Inference from Transformers via Speculative Decoding*. ICML 2023.
12. F. Gloeckle et al. *Better & Faster Large Language Models via Multi-token Prediction*. ICML 2024.
13. K. Kamahori et al. *Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models*. 2024.
14. H. Chen et al. *KTransformers: Unleashing the Full Potential of CPU/GPU Hybrid Inference for MoE Models*. SOSP 2025.
15. L. Xue et al. *MoE-Infinity: Efficient MoE Inference on Personal Machines with Sparsity-Aware Expert Cache*. 2024.
16. R. Hwang et al. *Pre-gated MoE: An Algorithm-System Co-Design for Fast and Scalable Mixture-of-Expert Inference*. ISCA 2024.
17. J. Wei et al. *T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge*. 2024.
18. E. Frantar et al. *MARLIN: Mixed-Precision Auto-Regressive Parallel Inference on Large Language Models*. 2024.
19. W. Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. SOSP 2023.
20. Z. Ye et al. *FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving*. MLSys 2025.
21. DeepSeek-AI. *DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models*. 2025.
22. A. Eliseev, D. Mazur. *Fast Inference of Mixture-of-Experts Language Models with Offloading*. 2023.
23. Y. Song et al. *PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU*. 2023.
24. ggml-org. ggml CPU backend type traits (ggml_get_type_traits_cpu: vec_dot, from_float) at the same llama.cpp commit; NVIDIA. CUDA C++ Programming Guide (CUDA Graphs, mapped pinned memory), CUDA 13.