Qwen3.8 Flash Next Performance on an Apple M4 Max

Ivan Murashko

Contents

1 A Focused Follow-Up 1

2 Model, Machine, and Protocol 1

3 Stock Story Performance 2

4 Sustained Story Performance 3

5 Patched Long-Summary Performance 4

6 The Ollama Prefill Issue and Patch 4

7 Conclusion 5

1 A Focused Follow-Up

Part two of this series compares the current local model packages on one Apple M4 Max. Qwen3.8 Flash Next appeared after that campaign and requires a newer Ollama release, so inserting it into the earlier table would mix runtimes. This article instead tests only `qwen3.8-flash-next:125b-mlx`: isolated story generation, one sustained story sequence, and the fixed long-summary request.

The comparison is deliberately about performance. Every generation sends `think=false`; there is no quality test, reasoning-effort diagnostic, or composite ranking. Input tokens/s, output tokens/s, latency, and memory describe different parts of the run and remain separate.

2 Model, Machine, and Protocol

The machine is a MacBook Pro with an Apple M4 Max, 16 CPU cores, 40 GPU cores, and 128GB of unified memory. It runs macOS 26.6.2 on arm64. Ollama 0.33.1 introduced MLX support for Qwen3.8 Flash Next. The tested NVFP4 safetensors package occupies 97.651GiB. Ollama reports 180.0B total parameters and a 262,144-token advertised context; the tag's 125B refers to the main model. The 262,144-token value is model metadata, not a context length measured in this campaign. The API context setting was 16,384 tokens, and the

longest measured prompt contained 9,223 tokens. The model description also lists 51B N-gram embedding parameters and 6B active parameters per token.

The campaign records the complete model and executable digests. I use the readable package tag in the article rather than printing an unexplained block of hexadecimal digits. At the measured 16K context, the freshly loaded `qwen4_exp` runner occupied 97.641GiB.

The story and synthetic 180-note summary are exactly the tasks used in part two. Table 1 gives the API settings. The story measurements use the unmodified 0.33.1 runtime. The long-summary numbers use the same version with the local prefill patch described in Section 6; they are never presented as stock-runtime results.

The gray points in Figure 2 reproduce the seven package averages published in part two; every row names its package. Those measurements used Ollama 0.32.15; Flash Next uses stock or locally patched 0.33.1 as stated above. The prompts, caps, context, and sampling settings match, but the gray data provide historical context rather than a same-runtime rerun.

Setting	Story	Long summary
Prompt tokens	38	9223
Output cap	768	180
Accepted isolated trials	10	3
Context	16,384	
Sampling	temperature 0, no streaming, `think=false`	

Table 1: Fixed request settings. The long-summary token count is the observed count for this package; the prompt text itself is unchanged.

Before an isolated request, the collector unloads any runner, preloads the model with an empty generation, validates the pinned digest, single-runner state, 16K context, and full GPU residency, and waits for 60 continuous seconds at nominal macOS thermal state. Requests run one at a time with a 30-minute keep-alive. The sustained sequence keeps the runner loaded and records thermal changes instead of rejecting them.

3 Stock Story Performance

Ten fresh-runner story trials produced 41.90 ± 0.59 output tokens/s. The slowest was 41.31 and the fastest 42.66, a narrow range. Other practical measurements are shown in Table 2. Here, loaded latency is the complete API duration after the preload; inference time is Ollama's prompt plus output evaluation time.

Measurement	Mean ± sample deviation
Output throughput	41.90 ± 0.59 tokens/s
Loaded-request latency	25.81 ± 0.45 s
Ollama-attributed inference	25.79 ± 0.45 s
Visible output	176.5 ± 1.4 characters/s
Fresh-runner preload	18.86 ± 0.34 s
Runner residency	97.641 GiB

Table 2: Ten isolated stories on the unmodified Ollama 0.33.1 runtime.

Every trial generated 768 visible tokens and stopped at the fixed cap. A token-level audit decoded each complete generated suffix and matched it byte-for-byte with the visible response; there was no hidden thinking output. The rate is therefore a clean generation measurement, but the cap means this task says nothing about whether the model would have ended the story by itself.

4 Sustained Story Performance

The back-to-back sequence begins at 41.34 tokens/s and ends at 31.93. Its sustained retention is

$$100\frac{\text{mean}(r_8, r_9, r_{10})}{\text{mean}(r_1, r_2, r_3)} = 79.5\%.$$

The macOS thermal state first changes from nominal to fair during request 6, 109.2 seconds into the sequence. The decline becomes more pronounced after that transition.

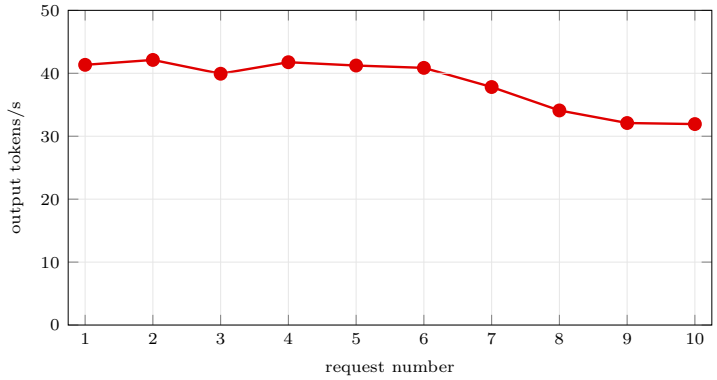

Figure 1: Flash Next output throughput over ten consecutive story requests on stock Ollama 0.33.1. Each point is one measured request.

Runner-reported residency rises from 97.64 to 99.14 GiB across the same sequence. One sequence cannot separate temperature, caching, and runtime allocation effects, so the curve is an operational observation rather than a general thermal model.

5 Patched Long-Summary Performance

I rebuilt Ollama 0.33.1 with the local patch and launched it with the intended 1024-token override. The campaign binds the patch, binaries, and collector environment, but Ollama does not echo the active chunk and no daemon startup log was retained. Table 3 reports only the successful patched measurements.

Prefill	Trials	Input rate	Output rate	Input time	Total time
1024-token override	3/3	126.27 ± 21.10	35.96 ± 1.79	74.57 ± 13.73	76.43 ± 13.82

Table 3: Three successful long-summary trials with the locally patched Ollama 0.33.1 runtime. Rates are tokens/s; times are seconds.

All three patched trials returned the same 66-token answer and ended normally; there was no thinking text or truncation. Post-request residency averaged 98.27GiB. The input variation is concentrated in the first trial: 102.06, 135.97, and 140.77 tokens/s. The three-trial mean is the published value; the two faster later trials are not substituted for it.

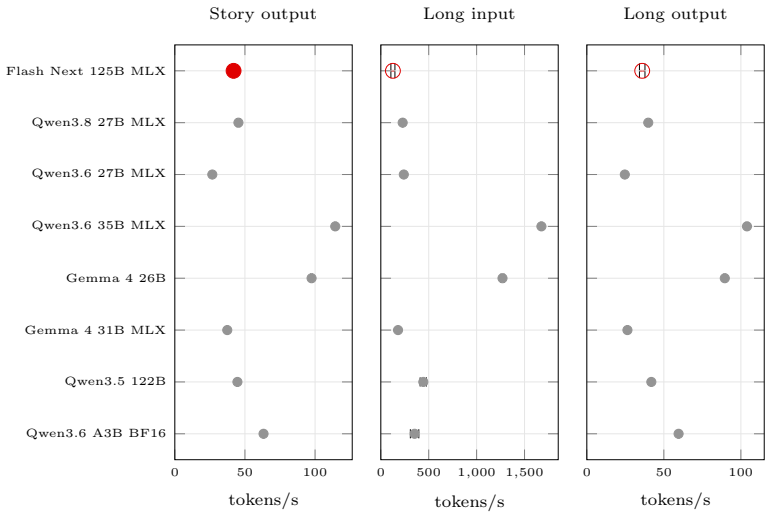

Figure 2: Mean rates with sample-deviation error bars. Flash Next is red: the filled point is the ten-trial stock story result, and the open points are the three-trial patched long-summary results. Gray points reproduce the seven package averages published in part two with Ollama 0.32.15.

6 The Ollama Prefill Issue and Patch

The original plan was to use stock Ollama 0.33.1 for both prompts. It did not work for the Flash Next long prompt, so I patched Ollama and repeated that measurement. Ollama processes the input during *prefill*, before it begins to generate the answer. The MLX runner divides this work into chunks; a larger chunk evaluates more prompt tokens at once but can require more temporary memory. The stock source fixes the MLX prefill chunk at 2048 tokens.

The local patch keeps 2048 as the default and adds the environment variable `OLLAMA_MLX_PREFILL_CHUNK_SIZE`. It accepts an integer from 1 through 2048, logs a valid override, and returns to the default for an unset or invalid value. The accompanying unit test covers the default, 1024, 512, 256, malformed, zero, negative, and oversized inputs. The benchmark uses 1024.

This is a deliberately narrow runtime change: it does not alter the model, tokenizer, prompt, context, sampling settings, or response cap. Ollama also has an upstream proposal for adaptive MLX prefill sizing; the local patch makes only the fixed setting configurable so that the measurement remains explicit and reproducible.

The original request did not complete with the stock 2048-token chunk, whereas the same package and 9,223-token request completed in all three runs with the intended 1024-token setting. This result points to temporary MLX prefill memory as the immediate obstacle. It does not show that the machine is unable to hold a 9K-token context.

The result also does not determine whether this package can reach the advertised 262K window. A published full-context experiment filled 262,144 tokens on a 128GB M5 Max using a smaller 93.7GB `UD-IQ4_XS` GGUF, llama.cpp, and a raised GPU wired-memory limit. The author estimated that a one-shot fill would take about 28 minutes, and generation at maximum depth ran at 10.9 tokens/s. That configuration differs from the 104.9GB NVFP4 MLX package tested here, so it shows that 128GB hardware is not categorically too small; it does not establish 262K operation for this Ollama package.

A closer published M4 Max experiment used a roughly 104GiB GGUF. With an 81,920-token context setting, it prefilled 81,654 tokens and generated 64; it did not complete the corresponding test at a 98,304-token setting. Reducing the prefill batch moved the stopping point. The achievable context therefore depends on the package and runtime memory strategy, not only on the advertised context or installed memory.

7 Conclusion

On this machine, Qwen3.8 Flash Next generates the stock short-story workload at about 42 tokens/s. Ten consecutive requests retain 79.5% of the initial rate, while runner residency grows by about 1.5GiB. With the local 1024-token prefill setting, the fixed long request completes at about 126 input and 36 output tokens/s.

The story and long-summary measurements use different runtime configurations, so they answer separate questions and should not be merged into a single ranking. The patched result establishes operation for the measured 9,223-token prompt, not for a 262,144-token prompt. Published experiments show that a 128GB Mac can reach the full window with a smaller 93.7GB package using a different quantization and runtime, together with a tuned wired-memory limit, while a similarly sized package on an M4 Max can stop much earlier. Practical context is therefore a property of the package, runtime, prefill strategy, tensor placement, and memory configuration together.

These measurements establish a narrow performance result on one Apple M4 Max. They do not establish quality, energy efficiency, concurrency behavior, or a general result for other Apple Silicon configurations.