Inkling 2 Mini Tested on a 3060 and a 3090: 256K Context Holds, 18 tok/s Doesn't

November 26, 2026 · Updated: November 27, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** A YouTube review says Inkling 2 Mini runs 256K context on a 12 GB card at 18 tok/s. I checked every number on a 3060 and a 3090. The context and VRAM numbers are right. The 18 tok/s didn't hold on my 3060: I got 10.4. On the 3090 box I got 16.4, a jump that matched the RAM bandwidth gap between the two machines. Then I slowed the RAM on the 3090 box, and decode didn't fall in step. Here's what actually sets the speed, which runtime to use, and why the best number in the whole test came from a different engine.

Updated 2026-11-27: a same-CPU RAM-speed test didn't support this article's original headline ("Your RAM Sets the Speed"). The title and conclusions are corrected below.

📚 **More on this topic:** MoE Models Explained · Strata vs llama.cpp on a 3060 · Ouro-2.6B, a Looped Model · Speculative Decoding Explained

If you have a 12 GB card and want long context, you've hit the same wall I have: the model fits, then the KV cache and compute buffers eat what's left, and 32K is about as far as you get. So when Codacus put out a YouTube review saying Inkling 2 Mini runs 256K context on a 12 GB card in about 7 GB of VRAM, at 18 tokens a second, I wanted to know if that held up.

Inkling 2 Mini is a looped mixture-of-experts model: 128 experts with 8 active per token, and layers 6 to 17 run three times for every token instead of once. That loop matters a lot here, and I didn't know it yet when I started.

I tested his claims on two boxes. My budget rig is an RTX 3060 12 GB with an i7-7700 and DDR4-2133. My main bench box is an RTX 3090 with a Ryzen 7 5700X and DDR4-3000. Both runs were pre-registered: predictions, rules and file hashes went public before the first measurement. The raw logs are in the 3060 record and the 3090 record.

The bottom line: his context and VRAM claims are right, almost to the hundred megabytes. His speed claim is not, on my 3060. The reason is my machine, but not one part of it. Decode on this model is a sum: reading experts from system RAM, the CPU's work on them, the GPU's share, and a fixed cost per token. My 3060 box is slower at more than one of those.

The scorecard

Codacus made about two dozen claims. These are the ones that matter if you're deciding whether to run this model.

Claim	His number	My 3060	Verdict
Mainline llama.cpp can't load 256K	doesn't load	Out of memory asking for a 32,120 MiB buffer	holds
Unsloth's build loads 256K	loads	Loads, and a 236K-token prompt completed	holds
VRAM at 32K / 64K / 256K	~3 / >3.5 / ~7.5 GB	2.8 / 3.4 / 6.9 GiB	holds
64K fits in 4 GB of VRAM	yes	Yes on Unsloth's build, no on mainline	holds
~17 GB of system RAM	~17 GB	15.6 to 16.1 GiB resident	holds
Decode ~18 tok/s, flat from 8K to 256K	18	**10.41**, flat (10.36 at 256K)	flat holds, speed doesn't
Prefill ~175 tok/s, >300 with a bigger batch	175 / >300	154 / 385	holds
Qwen3.6-35B-A3B is ~2.5x faster	2.5x	2.52x	holds
Strata SQ-10 does ~35 tok/s	35	39.9	holds
Strata tops out at 16K	16K	64K completed	doesn't hold
Strata's drafter never switches on	idle	It switched itself on	doesn't hold

His Strata version isn't known. He tested a couple of days after a release, and I ran Strata 0.2.21.1 with engine 0.1.67, so the two Strata misses could be changes between versions rather than mistakes in his video.

Why mainline llama.cpp can't do 256K and Unsloth's build can

This part of the video is right, and it's the most useful thing in it.

Codacus's explanation is that Inkling uses a position bias in attention, and mainline llama.cpp handles it by building the full attention score table, which grows with the square of the context. My numbers fit that. At 8K, mainline's compute buffer is 7.5 GiB on its own. At 32K it's 9.9 GiB and the 3060 runs out. At 256K it asks for 32,119.85 MiB, which doesn't fit on a 3090 either.

Unsloth's prebuilt llama.cpp does banded flash attention for this model. Same file, same flags:

Context	Unsloth build compute buffer	Mainline compute buffer
8K	670 MiB	7,568 MiB
32K	694 MiB	9,944 MiB
256K	919 MiB	32,120 MiB (OOM)

The quality is the same, too. Perplexity on both builds came within 0.2% of a CPU reference run.

So the rule is simple: **if you want more than about 8K on a 12 GB card, use Unsloth's build until mainline catches up.** With every expert in system RAM (`-cmoe`), the card held 2.4 GiB at 8K and 6.9 GiB at 256K. I also capped free VRAM at 4 GiB with a ballast program and ran 64K: it fit and ran at 10.4 tok/s. That's the 3060's speed with less memory free, not a 4 GB card's speed, but it shows the fit.

The speed gap: 10.4 tok/s, not 18

Here's where my numbers part ways with his. On the 3060, Unsloth's build decoded at 10.41 tok/s at 8K. Mainline did 9.92. That's 42% under his 18.

It stayed flat out to 256K for short prompts, just as he said: 10.36 tok/s. A long prompt is a different story. With 236K tokens actually in context, decode fell to 7.32, and the prompt took 29 minutes to process.

Before blaming my setup, I looked at what the model actually does per token. With `-cmoe`, every expert lives in system RAM, so every token has to stream its experts from RAM. Inkling's loop means layers 6 to 17 route three times per token, and each pass can pick a different set of 8 experts. I summed the expert tensor sizes from the GGUF:

	Expert bytes per token	Traffic at measured speed	Share of RAM read bandwidth
Inkling, one pass per layer	0.92 GB	9.6 GB/s	36%
	1.78 GB	18.6 GB/s	**69%**

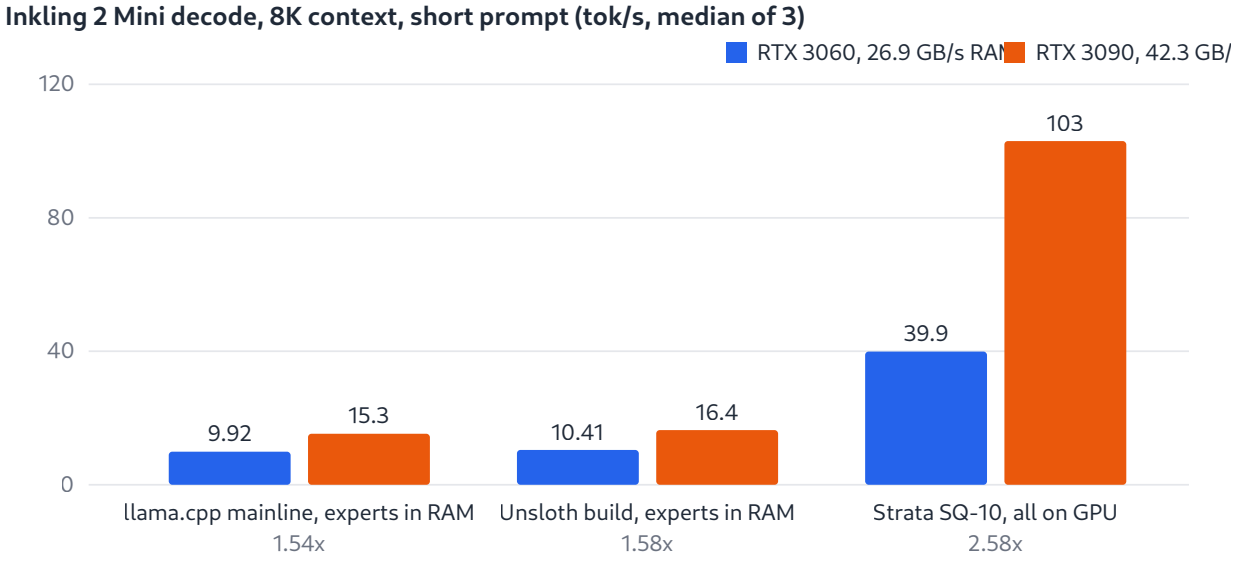
	Expert bytes per token	Traffic at measured speed	Share of RAM read bandwidth
Inkling, layers 6-17 three times			
Qwen3.6-35B-A3B	0.61 GB	16.0 GB/s	60%

My 3060 box reads 26.94 GB/s from RAM (measured, four threads, median of three runs). If the loop really reads its experts three times, Inkling's expert traffic works out to 69% of that, which is the same range expert-offloaded Qwen lands in on my rigs. That fits the Qwen claim too: 2.9x fewer expert bytes per token, 2.52x faster.

I first read that 69% as a fixed share, so that speed would simply be bandwidth divided by bytes. It isn't. The same-box RAM test further down shows that part of every token doesn't depend on RAM speed at all, so faster RAM helps less than that arithmetic says. On the 3090 box's CPU, the fit from that test puts 18 tok/s at about 50 GB/s of read bandwidth. A DDR5 desktop has that much, but it also has a different CPU, so I can't promise the number.

The 3090 test: the box gap matched the RAM gap

That was a hypothesis, so I wrote down a prediction and tested it on the 3090 box. The 3090's RAM reads 42.30 GB/s, 1.570 times the 3060 box. If decode is RAM-bound, Inkling should run about 1.57x faster there, regardless of the much faster GPU.



Runtime, short prompt	RTX 3060 box	RTX 3090 box	Ratio
Unsloth build, 8K	10.41	**16.40**	**1.575x**
Unsloth build, 32K	10.37	16.39	1.581x
Unsloth build, 256K	10.36	16.35	1.578x
Mainline, 8K	9.92	15.30	1.542x
RAM read bandwidth	26.94 GB/s	42.30 GB/s	1.570x

The predicted value was 16.35. I measured 16.40, so the prediction held. But a matching ratio doesn't prove the cause. The two boxes also differ in CPU and GPU, and the same-box test below shows those parts count too.

Codacus's 18 is within 9% of my 16.40. That doesn't tell me what his machine was. It says his number is plausible on a machine a bit faster than mine.

The GPU does matter once the context fills up. At 236K tokens in context, the 3090 box decoded at 13.36 against the 3060's 7.32, a 1.83x ratio. Attention over a huge KV cache runs on the GPU, so the GPU's share of each token grows as context grows.

What's still open

Updated 2026-11-27: I ran the RAM-only test. RAM speed matters, but it doesn't explain the whole gap between my two boxes.

The two boxes differ in more than RAM. The 5700X is a newer CPU with more cores, more cache and a different memory controller. So I dropped the 3090 box from DDR4-3000 to DDR4-2133 and changed nothing else: same CPU, GPU, build, flags and prompts. Its RAM read bandwidth fell 25%, from 42.30 to 31.54 GB/s.

If decode were purely RAM-bound, it would have fallen about 25% too. It fell 16%. Both settings ran on the same CPU and GPU:

3090 box, Unsloth build, short prompt	DDR4-3000	DDR4-2133	Ratio
8K	16.40	**13.83**	**0.843x**
32K	16.39	13.82	0.843x
RAM read bandwidth	42.30 GB/s	31.54 GB/s	0.746x

My pre-registered band for that ratio was 0.671 to 0.820, so the prediction failed. Going from 2133 to 3000 still bought 19% more speed, which is worth having. But fitting the two points,

about 28 ms of each 61 ms token doesn't depend on RAM bandwidth at all. It's CPU work, memory latency or the GPU's share, and this test can't tell which. Qwen3.8-Flash-Next showed the same pattern on this box in October.

So decode is a sum: RAM reads, CPU work, GPU work and a fixed per-token cost. Only the first part speeds up with faster RAM. The 1.575x against 1.570x match between my two boxes was partly coincidence: a bandwidth part plus parts that differ between the CPUs (or GPUs), which together happened to land on the RAM ratio. The 3060 box's older i7-7700 probably costs it some speed on its own, apart from its slower RAM.

What's still open is what that fixed ~28 ms is made of. Separating CPU from GPU needs a different test. The raw numbers are in the 3090 record.

Strata: 39.9 tok/s on the 3060, 103 on the 3090

Strata packs Inkling into a 10.5 GB SQ-10 file that sits entirely on the GPU. No RAM streaming, so none of the above applies.

	RTX 3060	RTX 3090
Strata default, 8K, short prompt	**39.9** tok/s	**103.0** tok/s
vs Unsloth build on the same card	3.8x	6.3x
Strata, drafting forced off, 8K	42.2	not run

The 3090 runs it 2.58x faster than the 3060, close to the 2.6x gap in VRAM bandwidth. That's what you expect when the whole model lives in VRAM.

Strata's drafter switches itself back on at startup. Codacus said it stays idle at the model's normal sampling settings. On both my cards, Strata's launcher ran a short probe at startup and turned drafting on. On the 3060 that was the wrong call: 39.9 tok/s with drafting, 42.2 with `--drafting off`. If you're on a 3060-class card, pass `--drafting off`.

The catch with Strata on 12 GB is context. It went to 64K for me, not 16K as in the video, but from 16K up the engine quietly moves part of the KV cache to system RAM. Deep decode drops hard: 19.5 tok/s with 16K in context, 7.2 at 59K. At 59K, Unsloth's llama.cpp build was faster (9.4). Strata's split mode, which keeps some experts in RAM, didn't crash for me as it did for Codacus, but both runs hit the NVMe during decode, so I threw out the speed numbers.

DSpark makes it slower

Inkling ships with DSpark, a small draft model for speculative decoding. Codacus found it took decode from 20 to 18 tok/s with about 40% of drafts accepted. I found the same direction, bigger.

3060, 8K, short	Without DSpark	With DSpark	Change
Unsloth build, temperature 1.0	10.41	6.55	−37%
Mainline, temperature 1.0	9.92	6.92	−30%
Unsloth build, temperature 0 (greedy)	10.41	7.54	−28%
Strata, temperature 1.0	42.2	39.9	−5%

Every combination I measured was slower with drafting on, including greedy decoding, where drafts usually pay off most. With 128 experts and a 3-pass loop, verifying a batch of drafted tokens means pulling even more experts out of RAM, and at 34-38% acceptance there's no win to pay for it. I measured DSpark only on the 3060. On the 3090, Strata's own probe claimed a 1.47x gain, but I didn't run a drafting-off comparison there.

Should you run Inkling 2 Mini?

Qwen3.6-35B-A3B, with every expert in RAM on the same 3060 and the same flags, decoded at 26.22 tok/s. That's 2.52x Inkling. With as many experts on the GPU as fit, Qwen did 35.26. And Qwen also fits 256K context in about 5.8 GiB of VRAM on mainline llama.cpp, no special build needed.

So the case for Inkling on a llama.cpp setup comes down to whether its output is better for your work, because it's never faster. On a 12 GB card with slow DDR4, 10 tok/s is usable for chat and slow for anything agentic.

If you want to run it anyway:

- Use Unsloth's llama.cpp build, not mainline, for anything past 8K. Mainline can't allocate the buffers.
- While the experts are in RAM, faster RAM helps, but it isn't the only thing that counts. Dropping my 3090 box from DDR4-3000 to 2133 cost 16% (16.4 to 13.8 tok/s). My 3060 box is also rated DDR4-2133, but it reads less bandwidth (26.9 GB/s against 31.5) and has an older CPU and a slower GPU, and it did 10.4. The rest of the machine counts too. I haven't measured DDR5.

- Skip DSpark. It was slower in every case I tested.
- With 24 GB, Strata is the one to use for anything under 16K. 103 tok/s on a 3090 is the best number in this whole test. On a 3060, turn drafting off.
- A full context means a long wait. 236K tokens took 29 minutes to prefill on the 3060 and 18 on the 3090.

```
# What I ran on the 3060 (Unsloth prebuilt llama.cpp, every expert in system RAM)
llama-server -m Inkling-2-Mini-UD-Q4_K_M.gguf -ngl 99 -cmoe \
  -ctk q8_0 -ctv q8_0 -fa auto -c 65536 -t 4 --cache-ram 0 --jinja
```

Codacus got the hard part right: Inkling really does run 256K on a 12 GB card, as long as you use the right build. The speed was the easy number to check, and it depends on your whole machine, RAM speed included, which he never mentioned.

A weekly email with every new guide and measured benchmark.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/inkling-2-mini-tested-rtx-3060-3090/

Free guides for running AI locally