Ternary Bonsai 2 on six GPUs: *the smaller file is slower on every discrete card, and the MTP head pays on all six.*

PrismML's Ternary Bonsai 2 stores its weights as ternary values, each one of three. Each size — 2B, 4B and 9B — comes in two packings, PQ2_0 and the smaller PTQ1_0, and the 4B and 9B also with a multi-token-prediction draft head. We measured all of them on six GPUs, from a DGX Spark to an RTX PRO 6000: the engine's own decode and prefill, then single-user serving with real sampling, with and without the draft head.

Bottom line

- **Engine decode, PQ2_0 file:** from the DGX Spark to the RTX PRO 6000, the 2B runs at 288–894 tok/s, the 4B at 142–556 and the 9B at 89–409 (the 9B top figure is from our 6 October run). An RTX 5080 does 636 / 365 / 254 tok/s.
- **Serve PQ2_0, not the smaller PTQ1_0.** PTQ1_0 holds 17% fewer bytes but decodes 18–27% slower on the 2B, 14–21% on the 4B and 3–7% on the 9B, on every discrete card. The exception is the DGX Spark, where PTQ1_0 is 13% faster on the 9B (3% on the 4B) — and even there the PQ2_0 file with the draft head is faster still.
- **Add the MTP draft head on the 4B and 9B:** real single-user decode rises ×1.07–1.30, with 60–63% of drafted tokens accepted. The gain is largest on the DGX Spark (×1.27 4B, ×1.30 9B) and smallest on the RTX 5070 (×1.07, ×1.15).
- **Real use runs below the engine number.** With sampling and the server in the loop, plain decode lands 8–31% under the same build's llama-bench figure: about 0.5–1.5 ms per token spent outside the GPU, so the fastest cards lose the most — RTX 5080, 4B: 373 → 257 tok/s.
- **What to run:** the `PQ2_0-MTP-Q8_0` file on fork release b10777 with PrismML's MTP flags — measured in real use at 484 tok/s (4B) on an RTX PRO 6000, 293 / 241 (4B / 9B) on an RTX 5080 and 153 / 104 on a DGX Spark. The 2B has no draft head: PQ2_0.

1 · What we ran

- **Model:** PrismML's Ternary Bonsai 2 at three sizes — 1.9, 4.2 and 9.0 billion parameters. Its weights are ternary — each one of three values — with a scale per group of 128. Each size comes in two packings: `PQ2_0` at 2.13 bits per weight and the tighter `PTQ1_0` at 1.75. The 4B and 9B also ship as a `PQ2_0-MTP-Q8_0` file: the same packing plus a multi-token-prediction (MTP) head that drafts the next tokens for the model to check.
- **Files:** pre-release files from PrismML's private beta, sha256-checked at both ends of every copy against PrismML's own list. These are the files we measured:

File	Pre-release of	Used for	SHA-256 (as measured)
`Ternary-Bonsai-2-2B-PQ2_0.gguf`	6 Oct	engine; real use, plain	`46871fccc387f1b89dfc671aa287fe4789c82084c2c5792e68a8afbb1465354c`
`Ternary-Bonsai-2-4B-PQ2_0.gguf`	6 Oct	engine; real use, plain	`a7ca0fb0760e07eb4770bef9930dcfe041f2f97be37826ed8a0accf75448c7c8`
`Ternary-Bonsai-2-9B-PQ2_0.gguf`	6 Oct	engine; real use, plain	`986ed292025b13894df089c7c65bb35b4ff4b8c20d95767f2124018c4244fc1a`
`Ternary-Bonsai-2-2B-PTQ1_0.gguf`	6 Oct	engine	—
`Ternary-Bonsai-2-4B-PTQ1_0.gguf`	6 Oct	engine	—
`Ternary-Bonsai-2-9B-PTQ1_0.gguf`	6 Oct	engine	—
`Ternary-Bonsai-2-4B-PQ2_0-MTP-Q8_0.gguf`	13 Oct, beta revision `e9fe290`	real use, MTP	—
`Ternary-Bonsai-2-9B-PQ2_0-MTP-Q8_0.gguf`	13 Oct, beta revision `c507c83`	real use, MTP	—

From `prism-ml/Ternary-Bonsai-2-{2B,4B,9B}-gguf`. The PTQ1_0 and MTP files were checked the same way; their sums are not recorded in this study's run directory. The repositories are public now, as one commit each (14–15 October), and their PQ2_0 files do not carry the SHA-256 sums above: we have not re-measured the public files (see the caveats).

Size	Parameters	PQ2_0 weights	PTQ1_0 weights	Change
2B	1.9 B	506 MB	418 MB	−17%
4B	4.2 B	1,135 MB	939 MB	−17%
9B	9.0 B	2,406 MB	1,986 MB	−17%

Weight bytes as llama-bench reports them for each file (tensor data, without the file's metadata).

- **Engine:** PrismML's llama.cpp fork, which these formats need, in two releases: `prism-b10743-adfffbe`, the build behind our earlier Bonsai 2 figures and the one PrismML's demo pins, and `prism-b10777-e3de715`, the release PrismML's MTP instructions are written for. Every machine built from the same source tarball per release, with the flags of our original RTX PRO 6000 build (`GGML_CUDA=ON`, `GGML_CUDA_FORCE_MMQ=ON`, release).

- **Engine speed:** `llama-bench -ngl 99 -fa 1 -p 512 -n 128 -d 0,8192 -r 5` on b10743, both packings — prefill of a 512-token prompt and decode of 128 tokens, on an empty context and after 8,192 tokens. One card at a time, no other work on it (one exception, in the caveats). The same test on b10777 (PQ2_0, empty context) measures the change between releases.
- **Real use:** `llama-server` b10777, one request at a time (`-ngl 99 -fa on -c 16384 -np 1 --jinja`, f16 KV cache), 24 prompts in 8 kinds (templated, json, code, summary, prose, chat, math, creative; 3 each), 1,024 generated tokens each, the model card's sampling (temperature 1.0, top-p 0.95, top-k 20, min-p 0, presence penalty 1.0), thinking on, one warm-up request, and a per-prompt seed shared by both arms. *Plain* = the PQ2_0 file; *MTP* = the PQ2_0-MTP-Q8_0 file with PrismML's flags `--spec-type draft-mtp --spec-draft-n-max 2 --spec-draft-n-min 0 --spec-draft-p-min 0`. Decode is the server's own count of generated tokens over its own generation time, summed over the arm; acceptance is accepted ÷ drafted tokens. The 2B has no MTP file and ran the engine test only.
- **Telemetry:** `nvidia-smi` once a second through every run.

Card	Where	Build	Power limit	Note
RTX PRO 6000 Blackwell (Workstation Edition)	our 4-GPU server	Linux, CUDA 12.9	600 W	permanent overclock: GPC +350, MEM +6000
RTX 5080	our second host	Linux, CUDA 12.9	360 W	
RTX PRO 4500 Blackwell (Server Edition)	our second host	Linux, CUDA 12.9	165 W	ran at its power limit
RTX 3090	our second host	Linux, CUDA 12.9	370 W	
RTX 5070	a desktop PC	Windows, CUDA 12.9	250 W	the PC's display card: screen on, applications open
DGX Spark (GB10)	—	Linux, CUDA 13.0	—	unified memory; its standing workload was stopped for the run

Power limits as each card reported them before the runs. The RTX PRO 6000 row for the 9B is our 6 October measurement of the same test on the original build (3 repetitions); its other 9B cells were not run, because the card was busy with another job when the 9B was due.

2 · Engine speed: decode and prefill

Card	2B decode	4B decode	9B decode
RTX PRO 6000 Blackwell	**894 ± 8**	**556 ± 4**	409 ± 3 (6 Oct)
RTX 5080	636 ± 5	365 ± 3	254 ± 2
RTX PRO 4500 Blackwell	530 ± 4	302 ± 2	209 ± 1
RTX 3090	483 ± 2	286 ± 2	191 ± 0
RTX 5070	399 ± 3	225 ± 4	154 ± 7
DGX Spark (GB10)	288 ± 1	142 ± 0	89 ± 0

Tokens per second, PQ2_0, empty context: llama-bench tg128. PrismML llama.cpp fork `prism-b10743-adfffbe`, llama-bench `-ngl 99 -fa 1`, mean ± standard deviation of 5 repetitions. The RTX PRO 6000 runs our permanent core and memory overclock (GPC +350, MEM +6000); the RTX 5070 is a desktop PC's display card, measured with the screen on and applications open; the DGX Spark's GPU shares unified memory with its CPU.

Card	2B prefill	4B prefill	9B prefill
RTX PRO 6000 Blackwell	31,157	18,074	12,957 (6 Oct)
RTX 5080	21,241	11,470	7,689
RTX PRO 4500 Blackwell	16,647	8,657	5,718
RTX 3090	12,817	6,751	4,463
RTX 5070	14,045	6,342	4,154
DGX Spark (GB10)	9,439	4,632	3,191

Tokens per second, PQ2_0, empty context: llama-bench pp512 (a 512-token prompt), same runs. PrismML llama.cpp fork `prism-b10743-adfffbe`, llama-bench `-ngl 99 -fa 1`, mean ± standard deviation of 5 repetitions; the prefill standard deviations run 1–15% of the mean, so read small prefill differences as noise. RTX PRO 6000 overclocked (GPC +350, MEM +6000); RTX 5070 = a display card, screen on; DGX Spark = unified memory.

Card	2B decode	4B decode	9B decode	2B prefill	4B prefill	9B prefill
RTX PRO 6000 Blackwell	786 (88%)	480 (86%)	not run	28,834 (93%)	16,642 (92%)	not run
RTX 5080	555 (87%)	311 (85%)	228 (90%)	19,658 (93%)	10,399 (91%)	7,197 (94%)
RTX PRO 4500 Blackwell	467 (88%)	259 (86%)	190 (91%)	15,299 (92%)	7,910 (91%)	5,456 (95%)
RTX 3090	454 (94%)	263 (92%)	181 (95%)	12,236 (95%)	6,146 (91%)	4,170 (93%)
RTX 5070	359 (90%)	199 (88%)	142 (92%)	12,293 (88%)	5,797 (91%)	4,292 (103%)
DGX Spark (GB10)	256 (89%)	123 (86%)	80 (91%)	8,781 (93%)	4,265 (92%)	2,999 (94%)

Tokens per second, PQ2_0, after 8,192 tokens of context (llama-bench tg128 and pp512 with `-d 8192`), and in brackets the share of the empty-context figure. PrismML llama.cpp fork `prism-b10743-adfffbe`, llama-bench `-ngl 99 -fa 1`, mean ± standard deviation of 5 repetitions. RTX PRO 6000 overclocked (GPC +350, MEM +6000); RTX 5070 = a display card, screen on; DGX Spark = unified memory.

- **The order of the cards is the same at every size**: RTX PRO 6000, RTX 5080, RTX PRO 4500, RTX 3090, RTX 5070, DGX Spark. The RTX 5080 runs the 4B at 66% of the RTX PRO 6000's speed; the RTX 3090, the oldest card here, is within 5% of the RTX PRO 4500 Blackwell on the 4B.
- **8,192 tokens of context cost 5–15% of decode** and 5–12% of prefill (the RTX 5070's 9B prefill read higher after 8K than on an empty context — noise on a card driving a display).
- **The newer release is slightly faster:** b10777 decodes 0.9–4.1% faster than b10743 on every card and size. The tables keep b10743 so they line up with our published RTX PRO 6000 figures.
- **The builds are equivalent:** on one RTX PRO 6000, the original 6 October binary and this run's build decoded the 4B at 554.8 ± 4.2 and 555.0 ± 3.9 tok/s back to back. The 2B on that card read 893.5 tok/s against 886.6 on 6 October.

3 · PQ2_0 or PTQ1_0: smaller is slower, except on the DGX Spark

Card	2B: PQ2_0 → PTQ1_0	4B	9B	Prefill change
RTX PRO 6000 Blackwell	**894 → 730 (−18%)**	**556 → 478 (−14%)**	not run	+1 to +2%
RTX 5080	**636 → 509 (−20%)**	**365 → 314 (−14%)**	254 → 247 (−3%)	0 to +3%
RTX PRO 4500 Blackwell	**530 → 416 (−21%)**	**302 → 257 (−15%)**	209 → 202 (−4%)	−1 to +3%
RTX 3090	**483 → 363 (−25%)**	**286 → 230 (−20%)**	191 → 180 (−6%)	0 to +3%
RTX 5070	**399 → 291 (−27%)**	**225 → 179 (−21%)**	154 → 144 (−7%)	−5 to −1%
DGX Spark (GB10)	288 → 271 (−6%)	**142 → 146 (+3%)**	**89 → 101 (+13%)**	−5 to −3%

Decode, tokens per second, empty context (llama-bench tg128), PQ2_0 → PTQ1_0 (change). Prefill change = PTQ1_0 against PQ2_0 (pp512) over the three sizes. PrismML llama.cpp fork `prism-b10743-adfffbe`, llama-bench `-ngl 99 -fa 1`, mean ± standard deviation of 5 repetitions. The RTX PRO 6000's 9B was not run in both packings. RTX PRO 6000 overclocked (GPC +350, MEM +6000); RTX 5070 = a display card, screen on; DGX Spark = unified memory.

- **On every discrete card, PTQ1_0 decodes slower at every size** — by 18–27% on the 2B, 14–21% on the 4B and 3–7% on the 9B — though it holds 17% fewer bytes. Prefill is level within the measurement's spread (−5 to +3%).
- **The DGX Spark is the exception, and only for the larger files:** there PTQ1_0 decodes 6% slower on the 2B but 3% faster on the 4B and 13% faster on the 9B.

Our reading. Decoding one token reads every weight once. Multiply each file's weight bytes by its decode rate and you get the rate at which the card is pulling weights from memory:

Card	2B	4B	9B	9B ÷ 2B
RTX PRO 6000 Blackwell	452	631	—	—
RTX 5080	322	415	611	1.90
RTX PRO 4500 Blackwell	268	343	503	1.87
RTX 3090	244	325	458	1.88
RTX 5070	202	256	371	1.84
DGX Spark (GB10)	146	161	214	1.47

GB/s of weight reads implied by PQ2_0 decode: weight bytes × llama-bench tg128 tokens per second, empty context; the KV cache and activations come on top. PrismML llama.cpp fork `prism-b10743-adfffbe`, llama-bench `-ngl 99 -fa 1`. RTX PRO 6000 overclocked (GPC +350, MEM +6000); RTX 5070 = a display card, screen on; DGX Spark = unified memory.

On every card the implied rate climbs with the size of the model: the small files do not keep the memory busy, so the time goes to unpacking and arithmetic — exactly what PTQ1_0 makes more expensive. The PTQ1_0 penalty shrinks as the files grow, and on the DGX Spark, whose GPU reads from unified memory shared with the CPU, the larger files are where the bytes saved outweigh the extra unpacking — the 9B by the most. That is an explanation that fits the numbers; we did not profile the kernels.

> **Which file:** PQ2_0 everywhere. The only case for PTQ1_0 on these cards is the 4B or 9B on a DGX Spark run without the draft head — and the draft head ships only on PQ2_0: on the Spark the PQ2_0-MTP file measured 104 tok/s in real use against PTQ1_0's 101 engine figure (153 against 146 on the 4B), and in every run here real use came in below the engine figure.

4 · Real use, with and without the MTP draft head

Card	4B plain	4B MTP	Gain	Accepted	9B plain	9B MTP	Gain	Accepted
RTX PRO 6000 Blackwell	403	**484**	**×1.20**	61%	not run			
RTX 5080	257	**293**	×1.14	61%	197	**241**	**×1.23**	63%
RTX PRO 4500 Blackwell	224	**253**	×1.13	62%	169	**201**	×1.19	62%
RTX 3090	217	**242**	×1.12	60%	159	**192**	**×1.21**	62%
RTX 5070	197	**210**	×1.07	62%	146	**167**	×1.15	62%
DGX Spark (GB10)	121	**153**	**×1.27**	62%	80	**104**	**×1.30**	62%

Tokens per second, one request at a time: llama-server, PrismML fork `prism-b10777-e3de715`, f16 KV cache, 24 prompts × 1,024 tokens, the model card's sampling, thinking on. Plain = PQ2_0; MTP = PQ2_0-MTP-Q8_0 with up to 2 drafted tokens per step. Accepted = the share of drafted tokens kept. The RTX PRO 6000 runs our permanent core and memory overclock (GPC +350, MEM +6000); the RTX 5070 is a desktop PC's display card, measured with the screen on and applications open; the DGX Spark's GPU shares unified memory with its CPU.

- **The draft head pays on every card**, ×1.07–1.30 overall, and its acceptance barely moves from card to card (60–63%) — as it should: acceptance depends on the model and the text, not the GPU.

- **It pays most where each token is most expensive:** ×1.27 and ×1.30 on the DGX Spark, where real use with the head (153 and 104 tok/s) beats even the plain engine figure (148 and 91). It pays least on the RTX 5070 (×1.07, ×1.15), and more on the 9B than the 4B on every card.

Kind of prompt	4B, discrete cards	4B, DGX Spark	9B, discrete cards	9B, DGX Spark
Math	×1.24–1.32	×1.46	×1.35–1.39	×1.47
JSON	×1.19–1.29	×1.56	×1.32–1.40	×1.38
Code	×1.07–1.21	×1.32	×1.25–1.28	×1.29
Templated	×1.04–1.22	×1.30	×1.17–1.23	×1.35
Summary	×1.07–1.28	×1.27	×1.11–1.25	×1.26
Chat	×1.03–1.14	×1.22	×1.04–1.16	×1.26
Prose	×1.01–1.09	×1.07	**×0.92–1.12**	×1.26
Creative	**×0.95–1.12**	×1.07	×1.03–1.08	×1.14

Real-use gain from the MTP head by kind of prompt (3 prompts each): MTP ÷ plain decode, llama-server, PrismML fork `prism-b10777-e3de715`, PQ2_0-MTP-Q8_0 against PQ2_0. Discrete cards = the range over the RTX PRO 6000 (4B only; overclocked), RTX 5080, RTX PRO 4500, RTX 3090 and RTX 5070 (a display card). Sorted by mean gain.

On average, structured and predictable text gains most — math and JSON — and free prose and creative writing least. Two cells ran *slower* with the head: the RTX 5070's 9B on prose (×0.92) and the RTX 5070's 4B on creative (×0.95). With 3 prompts a kind, read these as tendencies.

If you serve the 4B or 9B: load the `PQ2_0-MTP-Q8_0` file on fork release b10777 with `--spec-type draft-mtp --spec-draft-n-max 2 --spec-draft-n-min 0 --spec-draft-p-min 0`. On an RTX 5070-class card driving a display, expect little from it on prose.

5 · What real use costs against the engine number

Card	Size	Engine, b10777	Real use, plain	Change	Extra per token
RTX PRO 6000 Blackwell	4B	563	403	**−28%**	0.71 ms
RTX 5080	4B	373	257	**−31%**	1.21 ms
	9B	258	197	**−24%**	1.22 ms
RTX PRO 4500 Blackwell	4B	311	224	**−28%**	1.25 ms
	9B	213	169	**−21%**	1.25 ms
RTX 3090	4B	290	217	**−25%**	1.16 ms
	9B	192	159	−17%	1.09 ms
RTX 5070	4B	232	197	−15%	0.77 ms
	9B	158	146	−8%	0.55 ms
DGX Spark (GB10)	4B	148	121	−18%	1.51 ms
	9B	91	80	−12%	1.54 ms

Tokens per second, PQ2_0. Engine = llama-bench tg128 on the same release, PrismML fork `prism-b10777-e3de715`, empty context, mean of 5; real use = llama-server, PrismML fork `prism-b10777-e3de715`, the plain arm above. Extra per token = 1 ÷ real use − 1 ÷ engine. The RTX PRO 6000 runs our permanent core and memory overclock (GPC +350, MEM +6000); the RTX 5070 is a desktop PC's display card, measured with the screen on and applications open; the DGX Spark's GPU shares unified memory with its CPU.

- **The gap behaves like a near-fixed cost per token outside the GPU** — sampling on the CPU, the server's loop, and a context that grows to the prompt plus 1,024 tokens: 1.1–1.2 ms on our second host (AMD EPYC 7763, the cards passed through to a virtual machine), 1.5 ms on the DGX Spark's own Arm CPU, 0.7 ms on the RTX PRO 6000's server (AMD Ryzen Threadripper PRO 9985WX) and 0.5–0.8 ms on the RTX 5070's PC (AMD Ryzen 9 9950X3D).
- **A fixed cost hurts fast cards most:** the RTX 5080's 4B gives up 31% of its engine speed, the RTX 5070's 9B 8%. At these speeds the CPU and the serving loop are part of the measurement.
- Not tested here: llama.cpp's GPU-side sampling (`--backend-sampling`), which targets exactly this cost.

6 · What to run on each card

Card	2B, PQ2_0 (engine)	4B, PQ2_0 + MTP (real use)	9B, PQ2_0 + MTP (real use)
RTX PRO 6000 Blackwell	894	**484**	not run
RTX 5080	636	**293**	**241**
RTX PRO 4500 Blackwell	530	**253**	**201**
RTX 3090	483	**242**	**192**
RTX 5070	399	**210**	**167**
DGX Spark (GB10)	288	**153**	**104**

Tokens per second, one stream. The 2B column is the llama-bench tg128 figure on PQ2_0 (PrismML fork `prism-b10743-adfffbe`), since the 2B has no draft head and no real-use run; expect real use below it, by the per-token cost in section 5. The MTP columns are measured real use on the PQ2_0-MTP-Q8_0 files (llama-server, PrismML fork `prism-b10777-e3de715`). The RTX PRO 6000 runs our permanent core and memory overclock (GPC +350, MEM +6000); the RTX 5070 is a desktop PC's display card, measured with the screen on and applications open; the DGX Spark's GPU shares unified memory with its CPU.

- **Every card:** PQ2_0, and for the 4B and 9B the `PQ2_0-MTP-Q8_0` file with PrismML's flags on release b10777. Every file ran on every card here (the largest PQ2_0 weights are 2.41 GB), so PTQ1_0's smaller footprint only matters on hardware with far less memory, which we did not test.
- **DGX Spark:** the same. PTQ1_0 is faster only without the draft head, and the head is worth more (×1.30 in real use against PTQ1_0's +13% on the 9B engine figure).
- **RTX 5070 (here a display card, screen on):** the head's gains were the smallest and the run-to-run spread the widest; it still came out ahead overall.
- **Which size** is a quality question this report does not answer; this is speed only.

---

7 · Caveats

- **We measured pre-release files.** The public repositories' PQ2_0 files carry different SHA-256 sums from the ones listed in section 1. They are the same formats, and decode speed for a given format is set by its tensor layout rather than the weight values, but we have not re-measured them — nor the public MTP files, whose acceptance depends on the weights.
- **The plain and MTP arms ran different weight files.** Plain used the 6 October PQ2_0 file; MTP used the 13 October file, the only one with the head, whose weights were updated (same tensor shapes and types). With sampled decoding the two arms therefore wrote different text from the same prompts and seeds; a gain is a reading of 24 prompts each, one pass per arm.
- **Most of the 1,024 tokens are reasoning:** with thinking on, 19–24 of the 24 replies per arm had not started their answer when the cap was reached. Prompts ran 38–4,536 tokens, so real use covers contexts up to about 5.6K, single stream; batching and concurrency were not measured.
- **Partial rows:** the RTX PRO 6000's 9B is the 6 October engine figure only (decode and prefill, 3 repetitions, the original build); the 2B ran no real-use arm on any card. The RTX PRO 6000's 2B ran while another job's idle server held memory on the card (it read 0% utilization before the run) and matched its 6 October figure.
- **Labelled cards:** the RTX PRO 6000 runs a permanent overclock (GPC +350, MEM +6000), so its figures are above a stock card's. The RTX 5070 is a Windows PC's display card, measured with the screen on and applications open: its decode spread is the largest here (standard deviations up to 6.7% of the mean, against at most 1.0% on any other card). The DGX Spark is unified memory, and its standing workload was stopped for the run. The RTX PRO 4500 Blackwell (Server Edition) ran at its 165 W power limit.
- **A check that came out clean:** on our second host the container image's own CUDA backend loaded beside our build's. On the RTX 5080 we ran the 4B both ways: decode 365.7 against 365.9 tok/s — no effect.
- **Not measured:** output quality (this is a speed report), the Q2_0 files and the DFlash2 drafters that also ship in these repositories, PTQ1_0 in real use (it has no MTP variant), GPU-side sampling, contexts past 8K in the engine test, and our RTX 5090.

Card	Peak °C, 2B / 4B / 9B	Peak W	Power limit
RTX PRO 6000 Blackwell	38 / 42 / –	421	600 W
RTX 5080	42 / 49 / 54	321	360 W
RTX PRO 4500 Blackwell	38 / 49 / 52	165	165 W
RTX 3090	42 / 49 / 49	345	370 W
RTX 5070	36 / 43 / 46	228	250 W
DGX Spark (GB10)	69 / 70 / 77	85	—

Peak readings from one-second telemetry over each card's runs. The DGX Spark reports no power limit.

Every figure on this page was computed from the raw rows of runs made on 16 October 2026 on our own machines, with the RTX PRO 6000's 9B and the PQ2_0 checksums from our 6 October run. Run directories: `bench/runs/20261016-bonsai2-speed` (the scripts, every llama-bench and server row, the telemetry) and `bench/runs/20261006-bonsai2-recipe`. The rows are kept; to see them, or to send a model you want measured, write to hello@netsphere.com.ai. Online: netsphere.com.ai/reports/bonsai-2-speed/