



Ternary Bonsai 2

2B · 4B · 9B

Small ternary vision-language models for phones, laptops, and GPUs.

~9x

smaller

90 tok/s

2B on iPhone 17 Pro Max

92 tok/s

9B on MacBook Pro M5 Pro

1. Executive Summary

Ternary Bonsai 2 extends the representation of Ternary Bonsai 2 27B [1] to three small vision-language models built on **Qwen3.5-2B, 4B, and 9B** [3]. The architectures are unchanged. The language-model matrix weights are ternary, $\{-1, 0, +1\}$ with FP16 group scales, and are used with a fixed Walsh-Hadamard rotation (Section 2). Their language models occupy **0.43 GB, 0.95 GB, and 2 GB** of memory, respectively, against 3.76 GB, 8.41 GB, and 17.91 GB for the BF16 baselines' language models.

Across the evaluation suite of Section 6, which covers knowledge and reasoning, math, agentic tool calling, coding, instruction following, and vision, the three models stay close to their Qwen3.5 baselines at roughly a ninth of the size. Retention is strongest in the skills an on-device assistant or agent relies on most. Instruction following matches or exceeds the baselines at every size, by up to 21% at 2B. Math retention is 98–108%, multi-step reasoning on MuSR is 94–100%, and the 9B scores 82.5 on AIME26. Tool calling keeps 90–95% of baseline performance, and coding matches or exceeds the baselines at every size. On images, the models retain 93–97% of their baselines' scores on chart understanding (CharXiv) and visual question answering (A-OKVQA). At 2B, the ternary model outscores its full-precision baseline on five benchmarks: GSM8K, HumanEval+, MBPP+, IFEval, and IFBench.

The small footprint translates directly into speed. Decode is $1.1\times$ – $2.1\times$ faster than the stock Qwen3.5 4-bit build and $1.5\times$ – $5.2\times$ faster than BF16 on the same device and runtime. The 2B runs at 90 tok/s on an iPhone 17 Pro Max, and the 9B runs there at 22 tok/s; no Qwen3.5-9B reference fits the phone's per-app memory limit. On the M5 Pro, speculative decoding adds a further $1.7\times$ – $1.8\times$ on copy-heavy agentic prompts, and up to $3.9\times$ on the phone.

2. Model Summary

The three models use the packing formats and kernels of Ternary Bonsai 2 27B [1], and use their weights with the same kind of fixed rotation. Ternary matrix weights W are stored in memory, and each layer's input is transformed blockwise by $R = \frac{1}{\sqrt{n}} H_n S$, where H_n is an $n \times n$ Walsh-Hadamard matrix and S a fixed diagonal of ± 1 signs [4]. Inference at each layer, where x is the output of the previous layer, is then $f(x) = W(Rx)$.

Table 1. Ternary Bonsai 2 2B, 4B, and 9B at a glance.

Item	Specification
Base models	Qwen3.5-2B, 4B, and 9B [3], with text and image input; architecture unchanged
Size (language model)	0.43 GB / 0.95 GB / 2 GB, against 3.76 GB / 8.41 GB / 17.91 GB in BF16 (8.8–9.0× smaller)
Weights	Ternary, group size 128, with a fixed Walsh-Hadamard rotation (block width 512 for the 2B and 4B, 4096 for the 9B)
Packs	PTQ1_0 (1.75-bit), PQ2_0 (2-bit), Q2_0 (2-bit, group 64), MLX 2-bit (Table 2)
Backends	llama.cpp (Metal, CUDA, iOS) and MLX (Python, Swift)
Vision encoder	Separate mmproj file for llama.cpp, in BF16 or Q8_0
Speculative decoding	Prompt lookup (all sizes); MTP draft head and DFlash 2 drafter (4B, 9B)
License	Apache License

Table 2 lists the released files. For llama.cpp, the vision encoder ships as a separate mmproj file that is loaded alongside any of the language-model packs. The MLX release is a single 2-bit model.safetensors, with the MTP head in a separate folder for the 4B and 9B.

Table 2. Released files and sizes. “—”: not shipped for that size.

Runtime	File	2B	4B	9B
llama.cpp	PTQ1_0 (1.75-bit)	429 MB	950 MB	2.00 GB
llama.cpp	PQ2_0 (2-bit)	517 MB	1.15 GB	2.42 GB
llama.cpp	Q2_0 (2-bit, group 64)	547 MB	1.21 GB	2.56 GB
llama.cpp	PQ2_0 + MTP head (Q8_0)	—	1.27 GB	2.68 GB
llama.cpp	DFlash 2 drafter	—	1.58 GB (BF16)	1.57 GB (Q8_0)
llama.cpp	Vision mmproj (BF16)	671 MB	676 MB	922 MB
llama.cpp	Vision mmproj (Q8_0)	365 MB	367 MB	624 MB
MLX	2-bit model.safetensors	1.34 GB	2.05 GB	3.46 GB

3. Throughput

Table 3 reports batch-1 decode throughput (tg128) in the fastest pack per device. The speedups compare against the stock Qwen3.5 4-bit and BF16 builds of the same size, on the same machine and runtime build (Appendix A).

Table 3. Decode throughput in tok/s (tg128, batch 1), fastest pack per device. In parentheses: speedup over Qwen3.5 4-bit / BF16. “—”: no reference fits the device.

Device	Runtime (pack)	2B	4B	9B
iPhone 17 Pro Max	llama.cpp Metal (PQ2_0)	89.9 (2.0 / 5.2×)	39.8 (2.1 / —)	22.3 (—)
MacBook Pro M5 Pro	llama.cpp Metal (PQ2_0)	229.6 (1.6 / 3.6×)	120.1 (1.6 / 3.8×)	76.5 (1.6 / 4.4×)
MacBook Pro M5 Pro	MLX Swift (2-bit)	276.3 (1.3 / 3.9×)	145.2 (1.4 / 4.5×)	91.7 (1.6 / 5.2×)
RTX 4090	llama.cpp CUDA (PTQ1_0)	603.6 (1.5 / 2.9×)	368.6 (1.6 / 3.7×)	248.6 (1.7 / 4.4×)
RTX 5090	llama.cpp CUDA (PTQ1_0)	683.4 (1.1 / 2.0×)	454.7 (1.4 / 2.8×)	325.2 (1.4 / 3.4×)
RTX PRO 6000	llama.cpp CUDA (PTQ1_0)	793.3 (1.5 / 2.6×)	488.6 (1.7 / 3.3×)	330.5 (1.6 / 3.8×)
L40S	llama.cpp CUDA (PTQ1_0)	569.4 (1.6 / 3.4×)	334.2 (1.8 / 4.2×)	217.4 (1.9 / 4.9×)
A100	llama.cpp CUDA (PTQ1_0)	427.9 (1.3 / 1.6×)	265.6 (1.4 / 2.0×)	205.2 (1.4 / 2.3×)
H100	llama.cpp CUDA (PTQ1_0)	681.9 (1.3 / 1.6×)	446.4 (1.4 / 2.0×)	347.4 (1.4 / 2.3×)
H200	llama.cpp CUDA (PTQ1_0)	699.3 (1.3 / 1.5×)	456.9 (1.4 / 1.9×)	354.2 (1.4 / 1.9×)
B200	llama.cpp CUDA (PTQ1_0)	641.3 (1.3 / 1.5×)	427.2 (1.5 / 1.9×)	347.2 (1.5 / 2.2×)

All three models fit the iPhone’s ~6 GB per-app limit. The 9B decodes there at 22 tok/s, where no Qwen3.5-9B reference can be loaded. On CUDA, the dense PTQ1_0 pack is the fastest format at every size. Time to first token is not a uniform win. With llama.cpp, the Bonsai models reach the first token of a 512-token prompt sooner than both Qwen3.5 references on the M5 Pro, and sooner than Qwen3.5 4-bit on the phone. On A100, H100, H200, and B200, compute-bound BF16 prefill is faster, and on the M5 Pro the Qwen3.5 MLX builds beat our MLX path.

Speculative decoding. The 4B and 9B drafters use z-lab’s DFlash 2 architecture [9]. We start from z-lab’s public DFlash weights for Qwen3.5-4B and Qwen3.5-9B [8, 10], convert them to

DFlash 2 with outputs unchanged, and fine-tune them against the ternary target. With the best mode per device, copy-heavy agentic prompts decode up to $1.8\times$ faster on GPUs and the M5 Pro and $3.9\times$ on the iPhone (Section 4). The 2B ships without an MTP head and uses prompt lookup alone.

Long context. Speculative gains grow with context length: plain decoding slows as the context fills, while the drafter’s cost per round stays flat. Section 5 compares the 9B at three precisions up to 64k tokens. In a separate ternary-only run out to 120k tokens, DFlash 2 with prompt lookup stayed $1.1\times$ – $1.6\times$ faster than plain decoding, code edits matched the reference exactly at every length, and the drafter added about 60% to prefill.

Energy. On the M5 Pro, the ternary models use $1.7\times$ – $2.0\times$ less energy per generated token than the BF16 references of the same size: 0.125 vs 0.216 mWh (milliwatt-hours) per token at 9B, 0.080 vs 0.148 at 4B, and 0.034 vs 0.067 at 2B. These are CPU and GPU power estimates and exclude DRAM, so they are not comparable with GPU board power. On an RTX 4090, measured as board power including GPU memory, the ternary models use $3.6\times$ – $4.2\times$ less energy per token than BF16: 0.331 vs 1.377 mWh at 9B, 0.197 vs 0.786 at 4B, and 0.104 vs 0.371 at 2B.

4. Speculative Decoding Highlights

Table 4 gives the fastest speculative mode we measured for each model and device, as decode tok/s with the speedup over plain decoding on the same prompts in parentheses. It uses the released files at batch 1 with greedy decoding (Appendix A). For the 4B and 9B the best mode is usually the MTP file or the DFlash 2 drafter with prompt lookup. In three 4B cells a simpler mode wins: prompt lookup alone on copy-heavy prompts on the M5 Pro and the H200, and MTP without prompt lookup on everyday prompts on the H200.

Table 4. Speculative decoding, best mode per cell: tok/s (speedup over plain decoding on the same prompts). Copy-heavy: 22 agentic prompts. Everyday: 64 prompts over 8 categories. “plain”: no speculative mode beats plain decoding.

Device	Prompts	4B	9B
iPhone 17 Pro Max	copy-heavy	152 (3.9 $\times$)	89 (3.9 $\times$)
	everyday	37 (1.1 $\times$)	plain
MacBook Pro M5 Pro	copy-heavy	207 (1.8 $\times$)	122 (1.7 $\times$)
	everyday	150 (1.3 $\times$)	98 (1.3 $\times$)
RTX 4090	copy-heavy	432 (1.6 $\times$)	344 (1.7 $\times$)
	everyday	354 (1.3 $\times$)	285 (1.4 $\times$)
RTX PRO 6000	copy-heavy	558 (1.6 $\times$)	481 (1.7 $\times$)
	everyday	472 (1.3 $\times$)	420 (1.4 $\times$)
H200	copy-heavy	419 (1.4 $\times$)	333 (1.5 $\times$)
	everyday	334 (1.1 $\times$)	291 (1.3 $\times$)

- **Copy-heavy work gains most.** The 4B and 9B decode 1.4 – $1.8\times$ faster on GPUs and the M5 Pro, and $3.9\times$ on the iPhone, where plain decoding is slowest.
- **Everyday prompts still gain on 4B and 9B:** 1.1 – $1.4\times$ on GPUs and the M5 Pro. On the phone, plain decoding is as fast or faster for the 9B.
- **MTP with prompt lookup is the default.** It is the best or near-best mode in almost every cell ($K=2$ on CUDA, $K=3$ on the M5 Pro). Prompt lookup alone matches it on copy-heavy prompts

on the M5 Pro. DFlash 2 is best on the iPhone 4B and at long context (Section 5), but slower than plain on everyday prompts.

- **Output.** On Apple silicon, prompt lookup matched plain output on every prompt. On CUDA, speculative outputs can differ from plain decoding at numerical near-ties.

5. Long-Context Scaling (9B)

Table 5 compares how decode speed holds up as the context grows for the 9B at three precisions: Ternary Bonsai 2 9B (PQ2_0), and the Qwen3.5-9B base model as Q4_K_M and BF16 GGUFs. Each model runs plain decoding and its own DFlash drafter with prompt lookup: our DFlash 2 drafter for the ternary model, and the public z-lab DFlash drafter for Q4_K_M and BF16. Each request places a target passage in the middle of a long context and asks for a code edit, a summary, or a quoted explanation; the model generates 384 tokens greedily.

Table 5. 9B long-context decode on one H200: tok/s pooled over three tasks and two runs. Speculative rows use each model’s DFlash drafter with prompt lookup (speedup over that model’s plain decoding in parentheses). Bold: fastest speculative result per depth. Runs shared a node with other jobs; differences under about 10–15% are within noise. [†]One of the two plain runs was slow; the other gave 183 tok/s, a 1.33× speedup.

Mode	Context	Thinking off			Thinking on		
		Ternary	Q4_K_M	BF16	Ternary	Q4_K_M	BF16
Plain	4k	225	204	158	201	203	158
	16k	215	195	137	215	195	153
	32k	203	185	137	203	185	125
	64k	157 [†]	169	136	183	168	136
DFlash + lookup	4k	247 (1.10)	205 (1.01)	255 (1.61)	310 (1.55)	289 (1.42)	313 (1.98)
	16k	252 (1.17)	200 (1.02)	195 (1.42)	271 (1.26)	243 (1.25)	254 (1.66)
	32k	223 (1.10)	190 (1.02)	201 (1.47)	277 (1.37)	232 (1.25)	240 (1.92)
	64k	244 (1.56 [†])	160 (0.95)	183 (1.35)	265 (1.45)	169 (1.00)	235 (1.72)

- **The ternary model holds its speed best.** From 4k to 64k tokens, ternary with DFlash 2 loses 1% (thinking off) and 15% (thinking on) of its decode speed, against 28% and 25% for BF16 and 22% and 42% for Q4_K_M with the z-lab drafter.
- **Fastest from 16k on.** At 4k, BF16 with the z-lab drafter ties ternary with DFlash 2. From 16k to 64k, ternary with DFlash 2 is the fastest configuration we measured in every cell, by 3–37 tok/s over the best BF16 result. That includes BF16 with Qwen’s own MTP head (K=2, 1.5×–1.8×, not shown in the table).
- **Speedup ratios favor BF16 because its plain decoding is slower.** BF16 shows the largest speedups over its own plain decoding, but ternary plain decoding is already 20–80 tok/s faster, so ternary leads in absolute speed.
- **The Q4_K_M model gains little from speculative decoding at depth** (0.95×–1.42×). The drafter acceptance rate is close to BF16, so the limit is likely the cost of verifying several tokens at once on Q4_K_M weights.

6. Benchmarks

Each Bonsai model is compared with its own Qwen3.5 BF16 baseline on the same harness (Appendix A). The 2B suite has 17 benchmarks, the 4B adds τ^2 -Bench, and the 9B adds AIME26,

so averages compare within a size only. The benchmarks are grouped into six skill categories. A category's retention is the Bonsai category average divided by the baseline's.

Table 6. Per-benchmark results (%). "—": not in that size's suite.

Benchmark	2B		4B		9B	
	Qwen3.5	Bonsai	Qwen3.5	Bonsai	Qwen3.5	Bonsai
<i>Knowledge & reasoning</i>						
MMLU-Redux	79.39	67.60	88.63	81.98	91.11	86.95
GPQA Diamond	58.48	39.60	78.08	63.84	82.53	69.90
MuSR	62.83	59.13	62.43	62.57	66.67	66.53
<i>Math</i>						
MATH-500	85.40	82.40	99.00	97.60	99.00	98.60
GSM8K	65.96	80.74	94.54	94.54	95.45	95.07
AIME26	—	—	—	—	86.67	82.50
<i>Agentic / tool calling</i>						
BFCL v3	62.06	60.53	71.52	70.16	73.81	71.81
BFCL v4 Mobile	66.13	61.85	78.42	76.49	82.02	78.02
τ^2 -Bench	—	—	81.65	61.87	85.25	73.02
<i>Coding</i>						
HumanEval+	45.12	50.00	83.54	85.37	87.80	89.02
MBPP+	44.18	52.65	70.11	69.31	73.02	71.96
<i>Instruction following</i>						
IFEval	75.79	85.77	86.69	88.91	89.65	90.02
IFBench	34.33	48.00	50.33	56.33	57.33	57.00
<i>Vision</i>						
MMMU-Pro	59.25	50.23	68.09	64.86	72.43	68.73
A-OKVQA	81.05	78.60	85.94	82.53	88.38	84.02
CharXiv	72.06	69.86	78.06	72.46	80.10	76.20
OmniDocBench	71.78	55.64	86.51	71.22	88.73	73.80
Real World QA	70.70	63.90	78.30	69.80	80.00	75.20
OCRBench v2	52.60	44.20	53.68	50.60	58.31	55.04
Average	63.95	61.81	77.53	73.36	80.96	77.02

Table 7. Retention by skill category, relative to the same-size Qwen3.5 BF16 baseline.

Category	2B	4B	9B
Knowledge & reasoning	82.9%	90.9%	93.0%
Math	107.8%	99.3%	98.2%
Agentic / tool calling	95.5%	90.0%	92.4%
Coding	115.0%	100.7%	100.1%
Instruction following	121.5%	106.0%	100.0%
Vision	89.0%	91.3%	92.5%

Instruction following and coding. These are the strongest categories. Bonsai matches or beats its baseline on IFEval at every size, and on IFBench at 2B and 4B (+13.7 and +6.0 points); the 9B is within 0.4 points on both. HumanEval+ improves at every size (+1.2 to +4.9 points). MBPP+ rises 8.5 points at 2B and stays within about a point at 4B and 9B. For an assistant that has to respect formatting constraints and emit well-formed tool calls or code, this is the behaviour that matters most.

Math and reasoning. Math is essentially preserved at 4B and 9B. MATH-500 and GSM8K are within 1.4 points, the 4B ties its baseline on GSM8K, and the 9B scores 82.5 on AIME26 against 86.7. Multi-step reasoning on MuSR is level at 4B and 9B (within 0.2 points) and 3.7 points lower at 2B. The 2B's math average rises above its baseline on the strength of GSM8K (80.7 against 66.0), while its MATH-500 score is 3.0 points lower.

Tool calling. Function calling stays close to the baselines: BFCL v3 is within 2.0 points at every size, and BFCL v4 Mobile within 1.9–4.3 points. The gap opens on τ^2 -Bench, where the agent holds a long multi-turn conversation, follows a domain policy, and shares the environment with a simulated user who also acts on it. There the 4B retains 75.8% and the 9B 85.7%, the largest single loss at 4B.

Vision. General visual understanding holds up well. A-OKVQA retains 95–97%, CharXiv chart reasoning 93–97%, MMMU-Pro about 95% at 4B and 9B, and Real World QA 89–94%. Text recognition on OCRBench v2 retains 94% at 4B and 9B and 84% at 2B. The main loss is document parsing: OmniDocBench is 15–16 points lower at every size, so unlike the knowledge gap, it does not close with a larger model.

Knowledge. Knowledge-heavy benchmarks show losses that shrink with size. MMLU-Redux is 11.8 points lower at 2B, 6.7 at 4B and 4.2 at 9B, and GPQA Diamond is 18.9, 14.2 and 12.6 points lower. Where broad factual recall matters, the 4B or 9B is the better choice, or a smaller model paired with retrieval.

6.1 Against Smaller Full-Precision and Lower-Bit Models

On a device, the question is what else fits in the same memory. Table 8 compares each ternary model with a model that has fewer parameters but still needs more memory. The ternary 2B and 4B face the next smaller Qwen3.5 models in BF16, and the 2 GB ternary 9B faces Qwen3.5-4B in the popular Q3_K_M GGUF format (about 4.3 bits per weight, 2.3 GB).

Table 8. Category averages (%) against smaller references. Sizes are language-model weights. Bold: higher score in each pair. *9B suite only. †4B and 9B suites only. Per-benchmark reference scores are in Appendix B.

	Bonsai 2B	Qwen3.5-0.8B	Bonsai 4B	Qwen3.5-2B	Bonsai 9B	Qwen3.5-4B
Category	0.43 GB	BF16, 1.51 GB	0.95 GB	BF16, 3.76 GB	2 GB	Q3_K_M, 2.3 GB
Knowledge & reasoning MMLU-Redux, GPQA Diamond, MuSR	55.44	50.91	69.46	66.90	74.46	73.30
Math MATH-500, GSM8K, AIME26*	81.57	51.70	96.07	75.68	92.06	79.16
Agentic / tool calling BFCL v3, BFCL v4 Mobile, τ^2 -Bench†	61.19	45.90	69.51	64.91	74.28	72.60
Coding HumanEval+, MBPP+	51.33	18.49	77.34	44.65	80.49	75.76
Instruction following IFEval, IFBench	66.89	34.99	72.62	55.06	73.51	68.68
Vision MMMU-Pro, A-OKVQA, CharXiv, OmniDocBench, Real World QA, OCRBench v2	60.41	54.21	68.58	67.91	72.17	74.42
Category average	62.80	42.70	75.60	62.52	77.83	73.99

The ternary 2B is $3.5\times$ smaller than Qwen3.5-0.8B and leads in every category, by 20.1 points on the category average; the 0.8B model is ahead on only one benchmark, OmniDocBench. The ternary 4B is $4\times$ smaller than Qwen3.5-2B and also leads in every category, by 13.1 points, with the largest margins in coding (+32.7), math (+20.4), and instruction following (+17.6); vision is level.

The ternary 9B, despite being smaller than the 4-bit 4B, leads in five of six categories and by 3.8 points on the category average. Its clearest advantage is math (+12.9, including 82.5 against 54.2 on AIME26), followed by instruction following (+4.8) and coding (+4.7). The 4-bit 4B is ahead on vision, mainly on OmniDocBench and MMMU-Pro. Agentic tool calling, the capability on-device agents depend on most, is ahead at every size: by 15.3 points at 2B, 4.6 at 4B, and 1.7 at 9B, where the ternary model leads on all three tool-calling benchmarks. At 4B the gain comes from BFCL v3 and v4 Mobile (+8.1 and +10.4), while Qwen3.5-2B is ahead on τ^2 -Bench. At a fixed memory budget, a larger model in the ternary representation is the better choice than a smaller model at higher precision. Appendix B lists the per-benchmark scores behind Table 8.

References

- [1] PrismML. Ternary Bonsai 2 27B. Technical report, PrismML, September 2026. <https://github.com/PrismML-Eng/Bonsai-demo/blob/main/bonsai-2-27b-whitepaper.pdf>
- [2] PrismML. 1-bit Bonsai 8B: End-to-end 1-bit language model deployment. Technical report, PrismML, March 2026.
- [3] Qwen Team. Qwen3.5: Qwen3.5-2B, Qwen3.5-4B and Qwen3.5-9B. <https://huggingface.co/Qwen>, 2026.
- [4] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort. SpinQuant: LLM quantization with learned rotations. arXiv:2405.16406, 2024.
- [5] G. Gerganov et al. llama.cpp: LLM inference in C/C++. <https://github.com/ggml-org/llama.cpp>, 2023-2026.
- [6] ggml-org. llama.cpp tensor encoding schemes. <https://github.com/ggml-org/llama.cpp/wiki/Tensor-Encoding-Schemes>, 2026.

- [7] A. Hannun, J. Digani, A. Katharopoulos, and R. Collobert. MLX: An array framework for Apple silicon. <https://github.com/ml-explore/mlx>, 2023.
- [8] J. Chen, Y. Liang, and Z. Liu. DFlash: Block Diffusion for Flash Speculative Decoding. *arXiv preprint arXiv:2602.06036*, 2026.
- [9] Inco AI. DFlash 2: Keep Drafting Parallel. <https://inco.ai/blog/dflash2/>, August 2026.
- [10] Z-Lab and Modal. Qwen3.5-4B-DFlash and Qwen3.5-9B-DFlash draft models. <https://huggingface.co/z-lab/Qwen3.5-4B-DFlash>, <https://huggingface.co/z-lab/Qwen3.5-9B-DFlash>, 2026.
- [11] Alibaba ModelScope. EvalScope: Evaluation framework for large language models. <https://github.com/modelscope/evalscope>, 2024.
- [12] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of SOSOP*, 2023.
- [13] A. P. Gema et al. Are We Done with MMLU? In *NAACL*, 2025.
- [14] D. Rein et al. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. *arXiv:2311.12022*, 2023.
- [15] Z. Sprague, X. Ye, K. Bostrom, S. Chaudhuri, and G. Durrett. MuSR: Testing the Limits of Chain-of-thought with Multistep Soft Reasoning. In *ICLR*, 2024.
- [16] D. Hendrycks et al. Measuring Mathematical Problem Solving with the MATH Dataset. *arXiv:2103.03874*, 2021.
- [17] K. Cobbe et al. Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168*, 2021.
- [18] Y. Zhang and Math-AI Team. American Invitational Mathematics Examination (AIME) 2026. 2026. <https://huggingface.co/datasets/math-ai/aime26>
- [19] S. G. Patil et al. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *ICML*, 2025.
- [20] V. Barres, H. Dong, S. Ray, X. Si, and K. Narasimhan. τ^2 -Bench: Evaluating Conversational Agents in a Dual-Control Environment. *arXiv:2506.07982*, 2025.
- [21] J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. In *NeurIPS*, 2023.
- [22] J. Zhou et al. Instruction-Following Evaluation for Large Language Models. *arXiv:2311.07911*, 2023.
- [23] V. Pyatkin et al. Generalizing Verifiable Instruction Following. *arXiv:2507.02833*, 2025.
- [24] X. Yue et al. MMMU-Pro: A More Robust Multi-discipline Multimodal Understanding Benchmark. *arXiv:2409.02813*, 2024.
- [25] D. Schwenk, A. Khandelwal, C. Clark, K. Marino, and R. Mottaghi. A-OKVQA: A Benchmark for Visual Question Answering using World Knowledge. *arXiv:2206.01718*, 2022.
- [26] Z. Wang et al. CharXiv: Charting Gaps in Realistic Chart Understanding in Multimodal LLMs. In *NeurIPS*, 2024.
- [27] L. Ouyang et al. OmniDocBench: Benchmarking Diverse PDF Document Parsing with Comprehensive Annotations. *arXiv:2412.07626*, 2024.
- [28] xAI. RealWorldQA, released with Grok-1.5 Vision Preview. <https://x.ai/news/grok-1.5v>, 2024.
- [29] L. Fu et al. OCRBench v2: An Improved Benchmark for Evaluating Large Multimodal Models on Visual Text Localization and Reasoning. *arXiv:2501.00321*, 2025.

A. Methodology

Benchmarks. Each model and its baseline are evaluated with EvalScope [11] on vLLM [12], following the protocol in Appendix B of the 27B report [1]: thinking on, temperature 1.0, top-p 0.95, top-k 20, and the same output budgets. The suite covers MMLU-Redux [13], GPQA Diamond [14], MuSR [15], MATH-500 [16], GSM8K [17], AIME26 [18], BFCL v3 and v4 Mobile [19], τ^2 -Bench [20], HumanEval+ and MBPP+ [21], IFEval [22], IFBench [23], MMMU-Pro [24], A-OKVQA [25], CharXiv [26], OmniDocBench [27], Real World QA [28], and OCRBench v2 [29].

Throughput. Qwen3.5 references are the stock GGUF (BF16, Q4_K_M [6]) and MLX (BF16, 4-bit) builds, run on the same machine and runtime build as the Bonsai rows. Decode and prefill use llama-bench [5] tg128/pp512 with flash attention on, mean of 5; MLX [7] rows use the mlx_lm benchmark and an MLX Swift harness. Time to first token is measured, not derived: an exact 512-token prompt, one output token, no prompt cache, one warm-up, and the median of 5. The iPhone runs llama.cpp in our test app with Metal tensor ops on.

Speculative decoding. We compare pooled tok/s with and without speculation on the same prompts and machine, using llama-server at batch 1 with greedy decoding. *Copy-heavy* is 22 agentic prompts and *everyday* is 64 tasks over 8 categories. On the iPhone, each workload is a single built-in task in the test app. On Apple silicon, prompt lookup matched plain output on every prompt. On CUDA, outputs can differ from plain decoding at numerical near-ties. Long-context runs place one needle (a source file or a Wikipedia article) halfway into a 4k–120k-token context and generate 384 tokens, on one H200 with two repetitions.

Energy. Energy on the M5 Pro integrates powermetrics CPU and GPU power, sampled every 100 ms, over each request (a 64-token prompt, 512 generated tokens, a warm model, and five repetitions); idle power is reported separately. On the RTX 4090, energy is NVML board power sampled every 100 ms with the same request shape, and each of the five repetitions is nine back-to-back requests so that the sampling window is long enough.

B. Reference Model Scores

Table 9 lists the per-benchmark scores behind Table 8, each reference beside the ternary model it is compared with. The third comparison, the ternary 4B against Qwen3.5-2B in BF16, uses the scores in Table 6, plus Qwen3.5-2B on τ^2 -Bench (66.55), which is run only for the 4B suite.

Table 9. Per-benchmark scores (%) for the smaller references in Table 8. Bold: higher score in each pair. "—": not in that size's suite.

Benchmark	Bonsai 2B	Qwen3.5-0.8B BF16	Bonsai 9B	Qwen3.5-4B Q3_K_M
<i>Knowledge & reasoning</i>				
MMLU-Redux	67.60	62.40	86.95	86.40
GPQA Diamond	39.60	32.12	69.90	70.41
MuSR	59.13	58.20	66.53	63.10
<i>Math</i>				
MATH-500	82.40	50.00	98.60	91.20
GSM8K	80.74	53.40	95.07	92.12
AIME26	—	—	82.50	54.17
<i>Agentic / tool calling</i>				
BFCL v3	60.53	47.92	71.81	69.83
BFCL v4 Mobile	61.85	43.87	78.02	77.45
τ^2 -Bench	—	—	73.02	70.51
<i>Coding</i>				
HumanEval+	50.00	19.51	89.02	85.37
MBPP+	52.65	17.46	71.96	66.14
<i>Instruction following</i>				
IFEval	85.77	55.64	90.02	86.69
IFBench	48.00	14.33	57.00	50.67
<i>Vision</i>				
MMMU-Pro	50.23	39.88	68.73	72.54
A-OKVQA	78.60	70.22	84.02	84.68
CharXiv	69.86	54.62	76.20	78.20
OmniDocBench	55.64	61.02	73.80	80.13
Real World QA	63.90	57.40	75.20	75.00
OCRBench v2	44.20	42.14	55.04	55.95