PrismML

Ternary Bonsai 2.5 27B

A 27B ternary model built for agents.

∼9×	FP16-level	90%	< 10 GB
smaller than FP16	agentic tool calling	of FP16 agentic coding	strongest agentic model

PrismML · November 2026

Contents

1 Executive Summary 2

2 Model Summary 3

3 Agentic Tool Calling 3

4 Agentic Coding 4

5 General Capability 4

6 Inference Speed 5

7 Limitations and Roadmap 8

A Benchmark Evaluation Methodology 8

B Per-Benchmark Results 9

1. Executive Summary

Ternary Bonsai 2.5 27B is the agentic release of the Ternary Bonsai 27B line [1]. It keeps the Qwen3.8-27B [2] hybrid-attention architecture and the ternary weight format of Bonsai 2, so it is still 1.76 bits per weight and **5.93 GB**, against **53.8 GB** for the FP16 model. This release targets long-horizon, tool-driven work.

FP16-level agentic performance at one ninth of the size. Across five tool-calling benchmarks (τ^2-Bench, τ^3-Bench, BFCL v3, BFCL v4, and AppWorld), Bonsai 2.5 averages **77.2**, against 77.7 for FP16. On three agentic coding benchmarks (Terminal-Bench 2.1, SWE-bench Verified, and DeepSWE) it keeps **90%** of the FP16 score, up from 63% for Bonsai 2: DeepSWE rises from **0 to 27.0**, SWE-bench Verified from 60.8 to 75.6, and Terminal-Bench 2.1 from 52.8 to 60.1.

The strongest agentic model under 10 GB. We compare against conventional low-bit GGUF builds of the same base model: GSQ-RCO [3] IQ3_XXS, IQ2_S, and IQ2_XS, and Unsloth Dynamic [4] UD-IQ2_XS. Bonsai 2.5 is the smallest of them and leads every build under 10 GB on both tool calling and agentic coding. Against GSQ-RCO IQ3_XXS, which is **1.7× larger**, it is ahead on tool calling (77.2 vs. 75.3) and close on coding (54.2 vs. 55.5). Despite the name, the IQ2 builds use **2.27–2.75 bits per weight**, 29–56% more than Bonsai 2.5, and still score lower. Some of them also break down on agentic tasks: GSQ-RCO IQ2_XS scores 46.5 on AppWorld, where Bonsai 2.5 scores 90.5.

General capability stays close to FP16. Averaged over five categories (knowledge, math, coding, instruction following, and vision; 14 benchmarks), Bonsai 2.5 scores **82.3** against 84.4 for FP16, the highest of any compressed build. It is the only one that stays within four points of FP16 in every category, and it leads every conventional build, IQ3_XXS included, on math and coding.

Faster than at the Bonsai 2 release. Every GGUF file now ships with a multi-token prediction (MTP) draft head that speeds up decode by **1.5–1.8×** on NVIDIA GPUs, with output that matches

plain decoding up to rounding noise. With MTP and the kernels now in testing, decode is **2.3–3.0**× faster than on Bonsai 2's release day on six of the seven NVIDIA GPUs we measured, reaching about 300 tokens/s on H200 and B200. The Q2_0 files also run on stock llama.cpp.

The ternary format, storage accounting, and rotated weight basis are unchanged from Bonsai 2 and are described in its report [1]. Section 6 covers what is new in the runtime.

2. Model Summary

Bonsai 2.5 is built for agentic and tool-use workloads. The released weights are exactly ternary and use the Bonsai 2 format.

Table 1. System specification. Changes from Bonsai 2 are marked with †.

Item	Specification
Base model	Qwen3.8-27B [2], a 27B hybrid-attention causal language model
Parameters	24.35B language (64 blocks) + 0.47B vision tower (27 blocks) + 2.54B embedding/lm-head = 27.36B total
Architecture	Hybrid attention (∼75% linear / ∼25% full attention), SwiGLU MLP, RoPE, RMSNorm
Context length	262K tokens
Weight format	Ternary g128 (FP16 group-wise scaling) in a blockwise Hadamard basis; 1.76 bits per weight, 5.93 GB (PTQ1_0)
Focus†	Agentic and tool-use tasks
Reasoning efforts	xhigh (default, most accurate) or medium
Backends	MLX (Python, Swift) and CUDA; GGUF packs PTQ1_0 and PQ2_0
License	Apache License
Recommended sampling	Thinking mode: temperature 1.0, top-p 0.95, top-k 20, min-p 0
Speculative decoding†	Multi-token prediction (MTP) draft head included in every GGUF file

3. Agentic Tool Calling

Table 2. Agentic tool calling (%). AppWorld is the mean of task and scenario goal completion. Sizes are the published GGUF files.

Benchmark	Qwen3.8 FP16	GSQ-RCO IQ3_XXS	GSQ-RCO IQ2_S	GSQ-RCO IQ2_XS	Unsloth UD-IQ2_XS	Bonsai 2 27B	Bonsai 2.5 27B
Size (GB)	53.8	10.1	9.3	8.4	7.6	5.93	**5.93**
Bits/weight	16.0	3.00	2.75	2.50	2.27	1.76	**1.76**
τ^2-Bench	80.6	77.7	78.8	79.5	82.0	80.2	**82.0**
τ^3-Bench	72.8	70.7	69.6	67.5	67.2	69.1	**73.9**
BFCL v3 (native)	71.2	71.6	71.8	70.2	71.5	70.3	**72.6**
BFCL v4	70.8	70.6	67.7	64.2	66.2	64.4	**67.2**
AppWorld	93.2	86.1	82.8	46.5	89.6	51.2	**90.5**
Average (5)	77.7	75.3	74.1	65.6	75.3	67.0	**77.2**

All models are run the same way, in thinking mode at reasoning effort `xhigh` (details in Appendix A). On average, Bonsai 2.5 is level with FP16 on tool calling (77.2 vs. 77.7) and ahead of every conventional build. It scores above FP16 on τ^2-Bench and τ^3-Bench, where the agent serves a simulated customer over many turns of conversation and tool calls, and on BFCL v3 with native tool calls. BFCL v4 is the exception: there it trails FP16 by 3.6 points and IQ2_S by 0.5.

AppWorld asks the agent to write code against the APIs of nine everyday apps. Bonsai 2.5 reaches 90.5, up from 51.2 for Bonsai 2 and ahead of every conventional build, including IQ3_XXS at 86.1. Most of that gap comes from finishing whole scenarios: Bonsai 2.5 completes every task in a scenario 89.3% of the time, against 80.4% for IQ3_XXS and 75.0% for IQ2_S.

Low-bit builds can also fail in ways their other scores do not predict. GSQ-RCO IQ2_XS, 1.4× the size of Bonsai 2.5, scores 79.5 on τ^2-Bench but only 46.5 on AppWorld. In about one task in five, it writes a long script that guesses every API call and marks the task complete before seeing any output.

4. Agentic Coding

Table 3. Agentic coding (%). Terminal-Bench 2.1 uses the Terminus-2 agent; SWE-bench Verified and DeepSWE 1.1 use the mini-SWE-agent scaffold. Setup details are in Appendix A.

Benchmark	Qwen3.8 FP16	GSQ-RCO IQ3_XXS	GSQ-RCO IQ2_S	GSQ-RCO IQ2_XS	Unsloth UD-IQ2_XS	Bonsai 2 27B	Bonsai 2.5 27B
Size (GB)	53.8	10.1	9.3	8.4	7.6	5.93	**5.93**
Bits/weight	16.0	3.00	2.75	2.50	2.27	1.76	**1.76**
Terminal-Bench 2.1	70.8	66.3	59.6	53.4	53.9	52.8	**60.1**
SWE-bench Verified	80.6	73.2	75.4	74.4	71.8	60.8	**75.6**
DeepSWE	30.1	27.0	22.6	17.7	17.7	0.0	**27.0**
Average (3)	60.5	55.5	52.5	48.5	47.8	37.9	**54.2**

Compression has usually hurt agentic coding the most, because one bad action can derail a long episode. Bonsai 2 kept 63% of the FP16 score across the three coding benchmarks and did not solve a single DeepSWE task. Bonsai 2.5 keeps 90%: DeepSWE goes from 0 to 27.0 (90% of FP16's 30.1), SWE-bench Verified from 60.8 to 75.6 (94%), and Terminal-Bench 2.1 from 52.8 to 60.1.

Against the conventional builds, Bonsai 2.5 beats all three IQ2 builds on Terminal-Bench 2.1 and DeepSWE, and it scores highest on SWE-bench Verified, with IQ2_S 0.2 points behind. Only IQ3_XXS, which is 1.7× larger, has a higher coding average, on the strength of its Terminal-Bench 2.1 score.

5. General Capability

The agentic gains should not come at the cost of everything else, so we also measure general capability. Table 4 gives category averages over 14 benchmarks, with per-benchmark scores in Appendix B. Averaged over the five categories, Bonsai 2.5 scores 82.3 against 84.4 for FP16, the highest of the compressed builds, ahead of IQ3_XXS and Bonsai 2 (both 81.9). Its largest gap to FP16 is 3.1 points, on math; every other compressed build falls at least four points behind FP16 in some category. Bonsai 2.5 leads every conventional build on math and coding, IQ3_XXS

included. Instruction following and vision are the closest categories: there IQ3_XXS and IQ2_S are up to 1.1 points ahead, and Bonsai 2 leads on instruction following by 1.7.

Table 4. General capability by category, thinking mode (category averages, %). Sizes are the model files. Per-benchmark scores are in Table 8.

Category	Qwen3.8 FP16 53.8 GB	GSQ-RCO IQ3_XXS 10.1 GB	GSQ-RCO IQ2_S 9.3 GB	GSQ-RCO IQ2_XS 8.4 GB	Unsloth UD-IQ2_XS 7.6 GB	**Bonsai 2 27B** 5.93 GB	**Bonsai 2.5 27B** 5.93 GB
Knowledge & reasoning MMLU-Redux, GPQA Diamond, AA-LCR	86.67	85.57	83.63	82.40	80.97	83.80	**84.27**
Math AIME26, HMMT26	93.15	88.15	86.00	84.80	80.75	89.00	**90.10**
Coding LiveCodeBench v6, BigCodeBench	74.55	68.20	67.80	63.80	64.70	70.50	**71.65**
Instruction following IFBench, IFEval	81.15	81.95	81.40	79.25	78.20	82.55	**80.90**
Vision CharXiv, A-OKVQA, OmniDocBench v1.6, RealWorldQA, HallusionBench,MMU-PRO,MathVision	86.07	84.73	84.07	83.16	82.64	81.89	**83.66**
Average (5)	84.32	81.72	80.58	78.68	77.45	81.55	**82.12**

6. Inference Speed

Table 5 reports plain decoding (no MTP or speculative decoding) for this model's PQ2_0 and PTQ1_0 files on the current PrismML llama.cpp runtime, with the speedup over the September 2026 Bonsai 2 release build on the same files in parentheses. Decode is `llama-bench` tg128 and prefill is pp2048, batch 1, one GPU, flash attention on, five repetitions in two order-swapped runs (run-to-run spread under 1%). Perplexity is the same on both builds, so the speedups do not come from changed output.

Table 5. Plain-decoding speed (tokens/s) on the current runtime; in parentheses, speedup over the September 2026 release build. *Apple M5 Pro is Metal, PQ2_0 only, and its prefill column is pp512.

GPU	PQ2_0 decode	PTQ1_0 decode	PQ2_0 prefill	PTQ1_0 prefill
NVIDIA H200	184 (1.56×)	189 (2.12×)	3441 (1.19×)	2962 (2.35×)
NVIDIA H100	177 (1.50×)	186 (2.05×)	3401 (1.18×)	2987 (2.39×)
NVIDIA B200	149 (1.29×)	192 (2.11×)	3376 (1.02×)	3228 (2.23×)
NVIDIA A100	86 (1.16×)	107 (1.96×)	1387 (1.03×)	1405 (1.98×)
RTX PRO 6000	137 (1.10×)	158 (1.32×)	4223 (1.02×)	4283 (2.11×)
RTX 5090	143 (1.07×)	163 (1.28×)	4163 (1.00×)	4198 (2.22×)
RTX 4090	98 (1.09×)	112 (1.17×)	3444 (1.06×)	3452 (2.06×)
NVIDIA L40S	82 (1.09×)	95 (1.14×)	3057 (1.07×)	3018 (1.87×)
Apple M5 Pro*	30.2 (1.07×)	–	416 (1.05×)	–

Kernels in testing. A runtime update now in validation adds a chunk-staged Gated DeltaNet

prefill, a fused norm and Hadamard path for PTQ1_0, a Hopper prefill kernel for PTQ1_0, and the PQ2_0 multi-column mat-vec for single-token decode on A100 and B200. On H200 it raises prefill by 19% (PQ2_0) and 29% (PTQ1_0) and PTQ1_0 decode by 4%; across H100, B200, A100, and RTX PRO 6000 prefill rises 9–35% and PQ2_0 decode up to 24% (A100, B200). Greedy output is byte-identical to the current runtime on every GPU tested, except PQ2_0 decode on B200, where the multi-column mat-vec sums in a different order (mean KL divergence 1.2×10^{-4}, the same size as changing the batch size). It also includes kernels for speculative decoding (see below): the draft head reads the prompt alongside the main model, and Metal and SYCL kernels check several drafted tokens at once. On Apple GPUs, an opt-in switch sends verify steps of 2–4 tokens to a base-3 collapse kernel for PTQ1_0, which computes the same integer dot product (proved exact in Lean) and raises verify throughput by 26%, 36%, and 72% at 2, 3, and 4 tokens on an M5 Pro, with KL divergence at the measurement floor; it stays opt-in because float accumulation order differs.

Upstream llama.cpp. Q2_0 files of this model run on stock llama.cpp: runtime support for the Bonsai 27B family was merged as ggml-org/llama.cpp PR #29600, and the blockwise Hadamard (FWHT) kernels it needs are upstream for CPU (#27779), CUDA (#29096, #29100), Metal (#29094, #29095, #29602), Vulkan (#29772), SYCL (#29243, #29605), and MUSA (#30167). On an H200, stock llama.cpp matches the PrismML runtime on the Q2_0 file (mean KL divergence 3.6×10^{-4}, same top token at 99.3% of positions, identical greedy text) and decodes at 109 tokens/s against 150 on the PrismML runtime. PTQ1_0 is under upstream review (#29672 for the format and CPU, #29676 for CUDA). PQ2_0 and the kernels above currently ship in the PrismML fork.

MLX. A text-only MLX build of this model (affine 2-bit, group 128, with a bundled loader that applies the Hadamard transforms) reproduces llama.cpp closely: mean KL divergence 4×10^{-6} to 3×10^{-5} per chunk and the same top token at all 1,020 positions checked. On an Apple M5 Pro it prefills at 348 tokens/s (pp512) and decodes at 21.7 tokens/s, against 416 and 30.2 for llama.cpp on the same machine.

Speculative decoding with the MTP head. Every GGUF file of this release includes a multi-token prediction (MTP) draft head [20]: one extra layer, built by PrismML, that guesses the next few tokens. With speculative decoding [19], the full model checks all the guesses in one pass and keeps only the tokens it would have produced itself, so output matches decoding without drafting up to the rounding noise of checking several tokens at once: on an H200 with the PTQ1_0 file at the recommended setting, mean KL divergence between drafting on and off is 5.2×10^{-5}, no higher than in a control that accepts every guess (not yet checked on Metal, SYCL, or the NPU). The head speeds up decode by 1.5–1.8$\times$ on NVIDIA GPUs and 1.1–2.1$\times$ on the other devices (Table 6). Because it makes its own pass over the prompt, it slows prefill by 9–23% on NVIDIA GPUs and 2–8% elsewhere; with the drafting kernels in testing, prefill with MTP beats plain prefill on every NVIDIA GPU, and the M4 Pro's decode gain rises to 1.27$\times$.

How the MTP tables are measured. Drafting pays off only when the head guesses right, which depends on the text, so Tables 6 and 7 cannot use `llama-bench`'s synthetic tokens as Table 5 does. They instead time `llama-server`, as a user would run it, on 36 real prompts (chat, agents and tool calls, documents, structured output, code, reasoning, and long documents) on separate machines; prefill is the server's rate on a 2,048-token prompt. With MTP off, decode agrees with Table 5 within 3% on six of the seven GPUs (7% higher on the RTX 5090), and server prefill reads 4–19% lower because it includes per-request work that `llama-bench` omits; plain decode on the development build used here is within 0.2% of the current release, so the build is not the cause. Each ratio compares two runs on one machine and build, so read these tables on their own, not against Table 5.

Table 6. What MTP adds on each device today, and what the kernels in testing add on top (tokens/s). "+kernels": MTP on plus the kernels in testing that apply to the device's file (the PQ2_0 mat-vec is not used on PTQ1_0 rows). In parentheses, speedup over MTP off on the same build (for "+kernels", MTP and kernels combined). Setting: tokens drafted per pass, minimum draft probability.

		Decode			Prefill (2K)		
Device	Setting	Off	On	+kernels	Off	On	+kernels
NVIDIA H200	3, 0.1	194	296 (1.53×)	306 (1.58×)	2475	2252 (0.91×)	3362 (1.36×)
NVIDIA H100	3, 0.1	184	282 (1.53×)	291 (1.58×)	2746	2151 (0.78×)	3196 (1.16×)
NVIDIA B200	3, 0.1	191	296 (1.55×)	309 (1.61×)	3076	2620 (0.85×)	3493 (1.14×)
NVIDIA A100	3, 0.1	105	171 (1.63×)	175 (1.66×)	1342	1201 (0.90×)	1435 (1.07×)
RTX PRO 6000	3, 0.1	158	277 (1.76×)	283 (1.80×)	3930	3358 (0.85×)	4445 (1.13×)
RTX 5090	3, 0.1	174	305 (1.75×)	312 (1.79×)	3854	2998 (0.78×)	4173 (1.08×)
RTX 4090	3, 0	111	204 (1.83×)	204 (1.83×)	2808	2166 (0.77×)	3014 (1.07×)
Apple M4 Pro*	1, 0.6	22.9	25.3 (1.10×)	29.2 (1.27×)	126	122 (0.98×)	126 (1.00×)
Core Ultra laptop†	8, 0.6	11.1	23.1 (2.08×)	24.5 (2.21×)	236	218 (0.92×)	n/a
Galaxy S26 Ultra‡	1, 0	6.34	8.79 (1.39×)	n/a	137	132 (0.97×)	n/a

Unmarked rows are PTQ1_0.
*Metal, PQ2_0; an M4 Pro, not Table 5's M5 Pro; 18 of the 36 prompts.
†Integrated GPU, SYCL.
‡Q2_0, whole model on the Hexagon NPU, with a fix not yet released; 10 of the prompts.
n/a: no kernel in testing applies.

Since the Bonsai 2 release. Table 7 compares today with Bonsai 2 27B, released on September 17, 2026 [1] without a draft head. For a strong baseline, each Bonsai 2 cell is the fastest setup a user could run in its release week on the release build: PTQ1_0 or PQ2_0 without drafting, or PTQ1_0 with the community head published three days later. Decode with MTP is now 2.3–3.0× faster on six of the seven NVIDIA GPUs, 1.9× on the RTX 4090, 1.44× on the M4 Pro, and 2.0× on the Core Ultra laptop. Drafting has turned around: on release day it lost to the fastest plain setup everywhere except the RTX 4090 (on the M4 Pro it ran at 0.38× of plain PTQ1_0 decoding), and today it pays on every device. File format mattered too: smaller files usually decode faster, since decode is limited by memory bandwidth, yet in September PQ2_0's simpler 2-bit packing beat the smaller PTQ1_0 file on most GPUs (114 vs. 89 tokens/s on the H200); the current kernels have reversed that for decode on every NVIDIA GPU in Table 5. On the H100 the gain splits about evenly: 1.65× from the newer runtime, model, and file without drafting, and a further 1.58× from MTP with PrismML's kernels. Prefill gains are smaller, 1.1–1.4×, because PQ2_0 already read prompts quickly in September; they reach 2.1× on the RTX 4090 and 2.6× on the laptop, whose fastest earlier setups drafted, and are flat on the M4 Pro.

Table 7. How much faster a user is today than on Bonsai 2's release day (tokens/s). Bonsai 2: the fastest setup on the September 17, 2026 release build, named under Setup ("+MTP": community head, one draft). Bonsai 2.5: the "+kernels" columns of Table 6; in parentheses, speedup over Bonsai 2. Same prompts and timers on both sides; PQ2_0 rows ran on different hosts. Blackwell GPUs run the release source built with CUDA 12.8 (the September binaries had no Blackwell code). L40S not measured.

		Bonsai 2, Sep 2026		Bonsai 2.5, +kernels	
Device	Setup	Decode	Prefill (2K)	Decode	Prefill (2K)
NVIDIA H200	PQ2_0	114	2618	306 (2.69×)	3362 (1.28×)
NVIDIA H100	PQ2_0	112	2648	291 (2.60×)	3196 (1.21×)
NVIDIA B200	PQ2_0	104	2778	309 (2.98×)	3493 (1.26×)
NVIDIA A100	PQ2_0	73	1281	175 (2.39×)	1435 (1.12×)
RTX PRO 6000	PQ2_0	120	3226	283 (2.36×)	4445 (1.38×)
RTX 5090	PQ2_0	135	3645	312 (2.31×)	4173 (1.14×)
RTX 4090	PTQ1_0+MTP	106	1446	204 (1.92×)[§]	3014 (2.08×)
Apple M4 Pro[*]	PQ2_0	20.3	125	29.2 (1.44×)	126 (1.01×)
Core Ultra laptop[†]	PTQ1_0+MTP[¶]	11.5	85	22.9 (2.00×)	218 (2.56×)
Galaxy S26 Ultra[‡]	not available at release			8.79	132

[*]Metal; a community PQ2_0 head needed a separate runtime without Metal support.
[†]Integrated GPU, SYCL.
[‡]Hexagon NPU.
[§]Conservative: different hosts, and a faster RTX 4090 reached 219 tokens/s on the same build.
[¶]Nothing ran Bonsai 2 27B here on September 17: first SYCL release (October 9, 2026), community head at two drafts (prefill at one); both sides on the same 18 prompts, hence 22.9 here vs. 24.5 over all 36 in Table 6.

7. Limitations and Roadmap

Longer-horizon agentic work. Terminal-Bench 2.1 is still the agentic benchmark where we are furthest from FP16, and even our longest evaluations are bounded, single-task episodes. We have not yet measured workloads that run for hours across many context windows, such as machine-learning engineering (MLE-bench [16]), long-running autonomous operation (Vending-Bench [17]), and harder multi-language repository work (SWE-bench Pro [18]). Making steady progress over these horizons is the focus of the next release.

Kernels and KV cache. The kernel and KV-cache roadmap from Bonsai 2 [1] still applies: making PTQ1_0 as fast as PQ2_0 on every target, sub-4-bit KV caches, and removing the activation-side rotation overhead at batch size 1.

A. Benchmark Evaluation Methodology

The evaluation setup follows Bonsai 2 [1]: one harness and serving path for every model, thinking mode with reasoning effort `xhigh`, the reasoning block stripped before grading, and the sampling recommended by the Qwen3.8 model card (temperature 1.0, top-p 0.95, top-k 20, no presence or repetition penalty). Each task is attempted once (pass@1); where a benchmark was run more than once, we report the mean. Outside the code-execution benchmarks, a response that reaches the output limit before answering is scored as incorrect. Non-agentic benchmarks use EvalScope [5] with the vLLM [6] backend and per-benchmark output budgets of 16,384–81,920 tokens (81,920 for AIME26 and HMMT26). All models are served with vLLM on NVIDIA H100 GPUs. The conventional builds are the GSQ-RCO [3] and Unsloth Dynamic [4] GGUF files of Qwen3.8-27B, unpacked to BF16 and served through the same path; their sizes are the published GGUF files, and bits per weight is file size over the 26.9B language-model parameters.

Because τ^2-Bench and τ^3-Bench now use a thinking-mode user simulator and BFCL v3 uses native tool calls, these figures are not comparable with the Bonsai 2 report. The agentic benchmarks are configured as follows.

Terminal-Bench 2.1 [7]: the Terminus-2 agent on all 89 tasks, a 262,144-token context, and an 80,000-token output budget per turn, with network access inside the task container. **SWE-bench Verified** [14]: all 500 tasks with the mini-SWE-agent scaffold and a 262,144-token context, graded by the official test harness. **DeepSWE 1.1** [12]: 113 repository tasks with the mini-SWE-agent scaffold and a 262,144-token context, graded by the hidden tests on the agent's final commit. **τ^3-Bench** [9] and **τ^2-Bench** [8]: 375 and 278 tasks respectively, single trial. The user simulator for every model is `deepseek-v4-flash` in thinking mode (reasoning effort high, 32,768-token budget, temperature 0). **AppWorld** [10]: the 168 test-normal tasks with a ReAct-style code-action agent, a 40,000-token output budget, and a 131,072-token context. We report the mean of task goal completion and scenario goal completion. **BFCL v3** [11]: all 17 subsets including multi-turn (4,441 items) with native tool-call messages; we report the sample-weighted mean accuracy. **BFCL v4** [11]: 4,906 items, adding the agentic memory categories (key-value, vector, and recursive-summary memory) to the v3 subsets, with native tool calls and a 32,768-token output budget; the web-search category is not included.

B. Per-Benchmark Results

Table 8. General capability per benchmark, thinking mode (%).

Category	Benchmark	Qwen3.8 FP16	GSQ-RCO IQ3_XXS	GSQ-RCO IQ2_S	GSQ-RCO IQ2_XS	Unsloth UD-IQ2_XS	Bonsai 2 27B	Bonsai 2.5 27B
Knowledge & reasoning	MMLU-Redux	91.5	90.7	89.2	88.1	88.6	88.6	89.1
	GPQA Diamond	90.5	88.0	84.7	83.1	80.3	85.8	87.7
	AA-LCR	78.0	78.0	77.0	76.0	74.0	77.0	76.0
Math	AIME26	94.6	93.3	91.7	90.8	89.2	95.8	94.2
	HMMT26	91.7	83.0	80.3	78.8	72.3	82.2	86.0
Coding	LiveCodeBench v6	87.6	78.6	77.1	73.7	74.2	83.8	83.0
	BigCodeBench	61.5	57.8	58.5	53.9	55.2	57.2	60.3
Instruction following	IFBench	71.0	71.3	71.7	68.3	65.3	74.0	71.0
	IFEval	91.3	92.6	91.1	90.2	91.1	91.1	90.8
Vision	CharXiv	82.0	82.0	81.7	80.7	81.4	80.0	81.6
	A-OKVQA	89.3	88.7	89.0	87.4	86.7	86.8	88.6
	OmniDocBench v1.6	92.5	92.2	91.5	90.9	90.8	89.1	90.1
	RealWorldQA	83.4	82.5	81.8	80.9	79.0	80.1	81.7
	HallusionBench	84.8	83.1	82.9	83.1	81.9	81.1	81.5
Vision	CharXiv	82.0	82.0	81.7	80.7	81.4	80.0	81.6
	A-OKVQA	89.3	88.7	89.0	87.4	86.7	86.8	88.6
	OmniDocBench v1.6	92.5	92.2	91.5	90.9	90.8	89.1	90.1
	RealWorldQA	83.4	82.5	81.8	80.9	79.0	80.1	81.7
	MMU-PRO	81.7	79.3	77.6	76.8	77.4	75.0	77.6
	MathVision	88.7	85.3	83.9	82.3	81.3	81.1	84.5

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] Qwen Team. Qwen3.8-27B. https://huggingface.co/Qwen/Qwen3.8-27B, 2026.

[3] IST Austria Distributed Algorithms and Systems Lab. Qwen3.8-27B GSQ-RCO GGUF. https://huggingface.co/ISTA-DASLab/Qwen3.8-27B-GSQ-RCO-GGUF, 2026.

[4] Unsloth. Qwen3.8-27B GGUF (Unsloth Dynamic). https://huggingface.co/unsloth/Qwen3.8-27B-GGUF, 2026.

[5] Alibaba ModelScope. EvalScope: Evaluation framework for large language models. https://github.com/modelscope/evalscope, 2024.

[6] 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 SOSP*, 2023.

[7] Terminal-Bench Team. Terminal-Bench 2.1. https://www.tbench.ai/news/terminal-bench-2-1, May 2026.

[8] 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.

[9] Q. Shi, A. Zytek, P. Razavi, K. Narasimhan, and V. Barres. τ-Knowledge: Evaluating Conversational Agents over Unstructured Knowledge. arXiv:2603.04370, 2026. Benchmark released as τ^3-Bench: https://github.com/sierra-research/tau2-bench.

[10] H. Trivedi et al. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In *ACL*, 2024.

[11] S. G. Patil et al. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *ICML*, 2025.

[12] W. Huang, C. Lee, L. Tng, and S. Ge. DeepSWE: Measuring Frontier Coding Agents on Original, Long-Horizon Engineering Tasks. arXiv:2607.07946, 2026. Datacurve; benchmark v1.1: https://github.com/datacurve-ai/deep-swe.

[13] C. Rawles et al. AndroidWorld: A Dynamic Benchmarking Environment for Autonomous Agents. arXiv:2405.14573, 2024.

[14] OpenAI. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/, August 2024.

[15] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *ICLR*, 2024.

[16] J. S. Chan et al. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. In *ICLR*, 2025.

[17] A. Backlund and L. Petersson. Vending-Bench: A Benchmark for Long-Term Coherence of Autonomous Agents. arXiv:2502.15840, 2025.

[18] X. Deng et al. SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks? arXiv:2509.16941, 2025.

[19] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023. https://arxiv.org/abs/2211.17192.

[20] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. arXiv:2404.19737, 2024. https://arxiv.org/abs/2404.19737.