1-bit Bonsai 8B

End-to-end 1-bit language model deployment
across Apple, GPU, and mobile runtimes

12.8–14.2×	up to 8.4×	up to 5.6×	44 tok/s
smaller vs FP16	faster generation	lower energy/token	on iPhone

PrismML · March 31, 2026

Contents

Category	Summary
Model	1-bit Bonsai 8B — end-to-end 1-bit weight LLM built from Qwen3-8B [31]
Format	Q1_0_g128: 1-bit bitpacked weights with FP16 group-wise scaling
Deployment	Apple MLX (Python, Swift), llama.cpp (CUDA, Metal)
Value	1.15 GB footprint, 5–8× faster generation, 4–6× lower energy per token
Use case	On-device, privacy-preserving, low-latency inference on real hardware
License	Apache License; weights, demos, and, integrations

1. Executive Summary

Large language models have reached a turning point. The central challenge is no longer whether powerful models can be trained, but whether they can be deployed reliably, affordably, and at scale. In modern AI systems, inference dominates real-world cost, energy use, and latency. That makes deployment efficiency—not raw model capability—the defining constraint on productization.

This constraint is especially severe on edge hardware. Phones, laptops, robots, and embedded systems operate under hard limits on memory, bandwidth, thermal headroom, battery life, and connectivity. In these environments, the bottleneck is often not arithmetic throughput, but the cost of moving model weights through memory during generation.

1-bit Bonsai 8B addresses that bottleneck directly. It is a true end-to-end 1-bit weight language model built for real deployment, with 1-bit precision applied across embeddings, attention layers, MLP layers, and the LM head. The result is an 8B-class model that preserves practical intelligence while delivering substantial gains in footprint, bandwidth efficiency, and energy use across mainstream runtimes.

The Pareto Frontier of Intelligence vs. Size

A natural tradeoff in AI is the one between the size of a model (measured by the number of bits needed to describe it) versus its intelligence (measured by its performance across some set of benchmark tasks). To gain some understanding of this tradeoff, we considered 20 leading AI instruct models in sizes ranging from 1.2 GB (Qwen 3 0.6B [31]) to 18 GB (GLM 4 9B [37]), and assessed their average performance on 6 benchmarks pertaining to knowledge, problem solving, math, coding, instruction following, and tool calling. The resulting scatter plot reveals a Pareto Frontier defined by the Qwen 3 models 0.6B, 1.7B, 4B, and 8B, as well as the Ministral 3 3B [24].

However, **1-bit Bonsai 8B**, along with its smaller sister models 1-bit Bonsai 1.7B and 4B, **significantly moves the Pareto frontier to the left**.

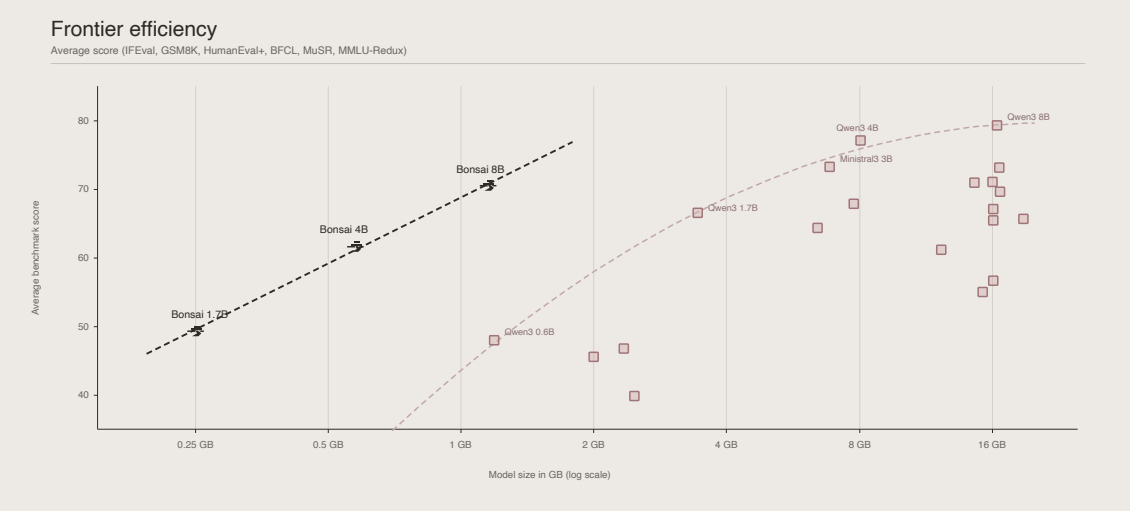

Figure 1. Benchmark score vs. model size (GB, log scale). The Bonsai family shifts the Pareto frontier of intelligence vs. size decisively to the left.

2. Efficiency as the Defining Constraint in AI Deployment

Inference defines the Economics of AI

Large language models have entered a new phase of adoption. The central question is no longer only whether highly capable models can be trained, but whether they can be deployed reliably, affordably, and at scale. In production systems, the recurring burden is inference. Every user interaction, agent step, and application call carries latency, energy use, and infrastructure cost. Deployment efficiency has therefore become a primary constraint on real-world AI adoption.

This constraint is especially visible on edge hardware. Phones, laptops, robots, wearables, and embedded systems operate within fixed limits on memory, bandwidth, thermal headroom, battery life, and connectivity. In these settings, model quality is necessary but not sufficient. The system must also fit the physical and economic limits of the device on which it runs.

Memory Bandwidth Is the Real Bottleneck

In LLM inference, especially at small batch sizes, the limiting factor is often not peak arithmetic throughput but memory movement. Performance depends heavily on how efficiently the system can fetch and stream model weights during token generation. For deployment, this makes memory bandwidth and model footprint central variables, not secondary implementation details.

This is why weight precision matters. Storing parameters at 1 bit per weight, the practical lower bound for parameter representation, does more than reduce model size. It reduces memory traffic, lowers energy per token, and improves the feasibility of running strong models on constrained hardware. The importance of 1-bit is not compression in the abstract. It is its direct effect on the main bottleneck in real inference systems.

Why 1-Bit Has Remained Out of Reach

The appeal of binary-weight neural networks has been understood for decades. In principle, they offer substantial reductions in storage, bandwidth demand, and computational cost. In practice, those gains have usually come with tradeoffs in quality, applicability, or deployment complexity.

Recent work on binary-weight Transformers [5, 6] has renewed interest in the 1-bit regime, but pushing meaningfully below 4-bit precision has remained difficult to make practical for LLMs. At these levels, the failure mode is often qualitative rather than gradual. A model may remain fluent while becoming materially less dependable on multi-step reasoning, tool use, retrieval, and edge cases. In production, that brittleness is often more damaging than a straightforward accuracy regression because it undermines trust and makes system behavior harder to predict.

Even near-1-bit approaches have often introduced enough operational complexity to limit adoption. Many rely on curated calibration sets, auxiliary metadata, custom layer handling, or bespoke runtimes and kernels that do not integrate cleanly with standard inference stacks. That creates friction exactly where efficiency is supposed to help: deployment across heterogeneous hardware, integration into existing toolchains, and reliable use without rearchitecting infrastructure.

3. What Makes PrismML Different

PrismML's differentiation begins with the method that makes 1-bit language models practical. Although the broader ecosystem for efficient inference has improved, tooling alone does not solve the central problem. At extreme compression levels, language models have historically lost too much of the reasoning quality and behavioral stability that give them value. PrismML's approach is based on mathematically grounded advances designed to preserve those properties under aggressive compression.

This foundation comes from proprietary Caltech intellectual property that addresses a long-standing research challenge through rigorous mathematics rather than ad hoc heuristics. 1-bit Bonsai 8B is therefore not a collection of narrow exceptions built for a controlled demonstration. It is a principled compression framework designed to preserve useful model behavior while materially lowering the cost of inference.

That differentiation is visible in the deployed model itself. 1-bit Bonsai 8B applies end-to-end 1-bit weight precision across the full network: embeddings, attention layers, MLP layers, and the LM head. It is not a partially quantized system that depends on higher-precision escape hatches in the critical path. This matters because such exceptions can preserve quality, but they also dilute the footprint and bandwidth reductions that make 1-bit valuable in deployment.

The result is an 8B-class model that makes strong on-device inference far more practical. Lower memory demand and reduced bandwidth pressure translate into concrete deployment advantages, including lower latency, lower energy consumption, smaller footprints, and improved feasibility on edge hardware. These properties also expand the range of applications that can run locally, especially in settings where privacy, responsiveness, or intermittent connectivity make cloud dependence undesirable.

Bonsai is designed as a deployment strategy rather than a single-device demonstration. It is engineered to run across heterogeneous hardware using mainstream backends developers already use, including Apple MLX [25] for Mac, iPhone, and iPad, and llama.cpp [13] for CUDA on NVIDIA GPUs. The significance of the system is therefore not only that 1-bit works in principle but also that it can be deployed across real runtimes, real devices, and real products.

To further demonstrate the potential of 1-bit models, we are also releasing two smaller models in the Bonsai family: 1-bit Bonsai 4B and 1-bit Bonsai 1.7B. These models extend the same design philosophy to smaller scales, showing that the advantages of 1-bit quantization are not limited to a single model size. Despite their compact footprints, both models deliver exceptionally strong throughput and energy efficiency while maintaining competitive accuracy relative to their parameter class.

Taken together, the Bonsai family illustrates that 1-bit design is not merely a compression technique, but a scalable approach to building practical, high-performance models across a range of deployment regimes. From the 1.7B model through the 8B model, Bonsai demonstrates that strong capability, efficient execution, and a small memory footprint can be achieved simultaneously rather than traded against each other. For the remainder of this manuscript, we focus primarily on 1-bit Bonsai 8B and the results of 4B and 1.7B are provided in the Appendix.

4. 1-bit Bonsai 8B Model Summary

1-bit Bonsai 8B is built from Qwen3-8B [31], a dense decoder-only causal language model. The architecture is unchanged: the novelty lies entirely in the deployment stack: End-to-end 1-bit weight storage for all major matrix-heavy components, an explicit runtime format for mainstream inference engines, and optimized kernels spanning the MLX [25] and llama.cpp [13] backends.

Table 1. System specification.

Item	Specification
Architecture model	Qwen3-8B [31] dense causal language model
Parameters	8.19 B (∼6.95 B non-embedding); 36 Transformer blocks
Architecture	GQA [2] (32 query / 8 KV heads), SwiGLU [34] MLP, RoPE [33], RMSNorm [32]
Context length	65,536 tokens
Weight formats	GGUF Q1_0_g128 and MLX 1-bit g128
1-bit weights	Embeddings, Attention projections, MLP projections, and LM head
Backends	MLX (Python, Swift) and llama.cpp (CUDA, Metal)
License	Apache License

4.1 Deployable 1-bit Format: Q1_0_g128

1-bit Bonsai uses a deployable group-wise 1-bit weight format. In llama.cpp [13] this is packaged as GGUF [12] Q1_0_g128, while MLX [25] uses the corresponding 1-bit g128 representation. The format stores one sign bit per weight and one shared FP16 scale for each group of 128 weights. This preserves a fundamentally 1-bit representation while retaining the group-wise magnitude information required for stable inference.

In storage, weights are represented as bitpacked values in $\{0, 1\}$, using exactly one bit per weight. At inference time, those bits are mapped to signed values and scaled by the FP16 factor associated with their group. The effective weight is therefore given by

$$w_i = s_g \cdot (2\, b_i - 1), \qquad b_i \in \{0,\, 1\}$$

where s_g is the shared scale for group g. Because each group of 128 weights carries one FP16 scale, the effective storage cost is

$$b_{\text{eff}} = 1 + \frac{16}{128} = 1.125 \text{ bits/weight},$$

which yields an idealized raw-weight compression of $16/1.125 \approx 14.2\times$ relative to FP16, before container overhead and alignment.

Q1_0_g128 is applied uniformly across the large matrix-heavy components of the model, including embeddings, attention projections, MLP projections, and the LM head. Normalization parameters and scale metadata remain in higher precision for numerical stability, but these account for a negligible share of memory traffic relative to the large weight tensors that dominate bandwidth during decoding.

The format is designed for direct execution rather than offline expansion. In the inference path, sign bits are decoded inline inside the matrix-multiplication kernel instead of materializing a full FP16 weight tensor in memory. This preserves the storage and bandwidth advantages of 1-bit weights where they matter most: the repeated movement of large weight matrices in the token-by-token decoding path.

MLX 1-bit g128: For MLX we need to pack our model slightly differently which has some extra overhead. MLX's quantization formats and kernels generally need both a scale (s_{mlx}) and a bias (b_{mlx}) per group such that $w = s_{\text{mlx}} \times b_i + b_{\text{mlx}}$, where $b_i \in \{0, 1\}$. To pack our scale-only 1-bit weights into this format we use

$$s_{\text{mlx}} = 2 \times s_g$$
$$b_{\text{mlx}} = -s_g$$

This reconstructs $-s_g$ when $b_i = 0$ and $+s_g$ when $b_i = 1$. Because MLX stores two FP16 values per group (scale + bias) instead of one, the effective bits per weight is 1.25 bits/weight which is slightly higher than the GGUF format. This is a current limitation of MLX, but once it supports scale-only formats, it should be able to match 1.125 bits/weight which is native to our models.

4.2 Storage Footprint

The deployed size reduction is the first-order consequence of the 1-bit representation. For the same 8.19B-parameter model, FP16 safetensors occupy 16.38 GB, while the runtime packages for 1-bit Bonsai 8B fall to approximately 1.15 GB in GGUF and 1.28 GB in MLX, respectively.

Table 2. Runtime package sizes for FP16 and 1-bit deployments.

Format	Exact bytes	GB	GiB	Reduction	Ratio
FP16 (HF safetensors)	16,381,452,288	16.38	15.25	—	1.0×
GGUF Q1_0_g128	1,151,820,864	1.15	1.07	92.9%	**14.2**×
MLX 1-bit (g128)	1,280,131,424	1.28	1.19	92.2%	**12.8**×

The deployed runtime footprints, i.e., peak memory requirements, vary slightly between GGUF and MLX builds. Runtime disk usage also includes auxiliary metadata such as config, chat template, and tokenizer files. For both GGUF and MLX, however, the total footprint remains approximately an order of magnitude smaller than the peak memory requirement of FP16.

4.3 Cross-platform Throughput

1-bit Bonsai 8B is designed for deployment across commonly used hardware platforms: Apple silicon through MLX or Metal, NVIDIA GPUs through CUDA in llama.cpp, and mobile-class devices through MLX Swift or OpenCL paths. To achieve acceleration across hardware platforms, we have developed custom inference support for Q1_0_g128 format for all target backends. For further details please refer to Appendix A. The standardized throughput summary below uses tg128 / pp512 deployment measurements, where tg128 measures token generation throughput over 128 generated tokens, and pp512 measures prompt processing throughput over 512 input tokens. Compared to full-precision 8B models, we observe **5.4–8.4**× **acceleration** in token generation for 1-bit Bonsai 8B model. Note that these values are distinct from the longer sustained workloads used for energy measurement. For more details of the methodology of throughput measurements and furtherresults on 1-bit Bonsai 4B and 1.7B, please refer to Appendix D.

Table 3. Cross-platform throughput summary and comparison of 1-bit Bonsai 8B with respect to FP16 baseline using standardized tg128 / pp512 deployment measurements rather than the sustained-run workloads.

Platform	Backend	Size	TG128 (tok/s)	PP512 (tok/s)	Base TG (tok/s)	Speedup
RTX 4090	llama.cpp CUDA	1.15 GB	368	11,809	59	**6.2**×
RTX L40S	llama.cpp CUDA	1.15 GB	327	9,592	52	**6.3**×
M4 Pro 48 GB	llama.cpp Metal	1.15 GB	85	498	16	**5.4**×
M4 Pro 48 GB	MLX (Python)	1.28 GB	131	472	16	**8.4**×
iPhone 17 Pro Max	MLX Swift	1.28 GB	44	377	13.8[1]	**3.2**×[1]
Samsung S25 Ultra	llama.cpp OpenCL	1.15 GB	19.6	30.4	—[2]	—[2]
RTX 3060 Laptop	llama.cpp CUDA	1.15 GB	81	1871	3.5	**23.0**×[3]

[1] iPhone baseline is 4-bit because an 8B-parameter FP16 model does not fit on-device.
[2] On the Samsung device, even a 4-bit 8B model did not fit.
[3] FP16 baseline requires partial GPU offloading due to insufficient memory.

Overall, these results show that 1-bit Bonsai 8B delivers substantial cross-platform throughput gains while reducing deployment footprint enough to make efficient inference practical on hardware ranging from high-end GPUs to mobile devices.

4.4 Energy per Token

Next, we examine the energy savings of the 1-bit Bonsai 8B model. Unlike deployment footprint, the energy profile is more nuanced. On both Apple silicon and NVIDIA GPUs, 1-bit Bonsai can exhibit equal or even higher instantaneous power draw during generation than FP16, because inline dequantization and bit-level arithmetic shift execution toward a more compute-intensive regime. However, token generation is substantially faster, so the energy consumed per output token is still significantly lower. In other words, higher instantaneous power does not necessarily imply higher total energy when time-to-token is reduced sufficiently. The results are provided in Table 4.

Table 4. Energy-per-token generation (E_{tg}) summary for measured and estimated deployments.

Platform	1-bit Bonsai 8B (mWh/tok)	Baseline FP16 (mWh/tok)	Advantage
Mac M4 Pro (MLX)	0.074	0.415	**5.6**×
Mac M4 Pro (Metal)	0.091	0.471	**5.1**×
RTX 4090 (CUDA)	0.276	1.134	**4.1**×
iPhone 17 Pro Max	∼0.068	4-bit: ∼0.143	**2.1**× vs 4-bit

Mac power includes CPU + GPU + ANE + DRAM and excludes system overhead. RTX 4090 measurements use GPU power only. iPhone energy is estimated from Xcode Power Profiler and battery-drain observations.

For Mac M4 Pro and RTX 4090, we generally observe a **4–6**× **reduction** in energy per generated token, whereas on iPhone we measure roughly $2\times$ improvement over 4-bit quantization which is the largest size we can fit on the device for an 8B model.

5. Benchmarks and Intelligence Density

We evaluated the performance of 1-bit Bonsai 8B on several benchmarks on across six skill categories: knowledge, reasoning, math, coding, instruction following, and tool calling. All benchmarks were evaluated using EvalScope [1] v1.4.2 with the vLLM [36] 0.15.1 backend on NVIDIA H100 GPUs. We compare 1-bit Bonsai 8B with 11 other leading models in the 6B to 9B parameter range. Every model was evaluated using identical infrastructure, generation parameters, and scoring methods to ensure a fair and reproducible comparison. Generation used greedy decoding (temperature $= 0$, top-$p = 1.0$) with thinking mode disabled. The scoring was primarily rule-based following the industry standard approaches paired with an LLM recall fallback, which is activated only when the rule-based parser failed to extract a valid answer. All models used the same judge configuration, dataset revisions, and sample sets. For further details of methodology and fairness efforts, please refer to Appendix B.

Table 5 summarizes the results on one benchmark per skill category: MMLU-Redux [26, 14] on knowledge, MuSR [27] on reasoning, GSM8K [7] on math, HumanEval+ [8, 19] on coding, IFEval [35] on instruction following, and BFCLv3 [4] on tool calling. Additional results for these categories, along with results for our 1-bit Bonsai 4B and 1.7B models, are provided in Appendix C.

Table 5. Benchmark comparison. 1-bit Bonsai 8B is compared against 11 leading conventional models of comparable scale.

Model	Size	Average	MMLU Redux	MuSR	GSM8K	Human Eval+	IFEval	BFCLv3
Qwen 3 8B [31]	16.38 GB	**79.3**	83	55	93	82.3	81.5	81
RNJ 8B [11]	16.63 GB	**73.1**	75.5	50.4	93.7	84.2	73.8	61.1
Ministral3 8B [23]	16.04 GB	**71.0**	68.9	53.8	87.9	72.6	67.4	75.4
Olmo 3 7B [29]	14.60 GB	**70.9**	72	56.1	92.5	79.3	87.1	38.4
1-bit Bonsai 8B	**1.15 GB**	**70.5**	65.7	50	88	73.8	79.8	65.7
LFM2 8B [21]	16.68 GB	**69.6**	72.7	49.5	90.1	61	82.2	62.0
Llama 3.1 8B [17]	16.06 GB	**67.1**	72.9	51.3	87	63.4	76.4	51.5
GLM 4 9B [37]	18.80 GB	**65.7**	81.9	53.2	89.4	78.7	69.3	21.9
Hermes 3 8B [28]	16.06 GB	**65.4**	67.4	52.2	82.9	51.2	69.3	69.6
Trinity Nano 6B [3]	12.24 GB	**61.2**	66.8	52.6	81.1	54	50	62.5
Marin 8B [22]	16.06 GB	**56.6**	64.8	42.6	86.1	55.9	63	27.9
DeepSeek R1 Qwen 7B [10]	15.23 GB	**55.0**	62.5	29.1	92.7	81.7	48.8	15.4

Despite being $1/14$ of the memory of the other models in Table 5, the 1-bit Bonsai 8B achieves competitive performance with the leading full-precision 8B instruct models across all six skill categories. These results demonstrate that 1-bit Bonsai 8B maintains strong general-purpose capabilities while drastically reducing memory requirements at deployment time. In other words, 1-bit Bonsai 8B challenges the conventional wisdom that dramatic gains in efficiency must come at the expense of model capability.

5.1 Intelligence Density

The benchmark results above motivate a more systematic way of characterizing the tradeoff between capability and model size. Rather than considering performance alone, we would like a measure that captures how much intelligence a model delivers per unit of memory. Figure 1

visualizes this tradeoff as a scatter plot of intelligence versus model size across models. To summarize this relationship with a single scalar quantity, we define *intelligence density* as the ratio of a model's intelligence to its size (measured in bits or, equivalently here, in GB).

At first, one may be tempted to define the intelligence of a model as the average benchmark score on a set of tasks, such as the average benchmark score in Table 5. However, this may be problematic. If one went with this definition of intelligence, then models that score 99 and 55 are both 10% more intelligent than models that score 90 and 50, respectively. But it seems rather obvious that it takes much more effort to go from a score of 90 to 99 than it does to go from 50 to 55. We therefore seek a definition of intelligence that better reflects this nonlinear scaling. To this end, we interpret

$$P_e = 1 - \frac{\text{average benchmark score}}{100}$$

as the model's probability of error. Information-theoretic arguments, rooted in rate-distortion theory, suggest that the relationship between a model's size in bits, denoted by N, and its probability of error should be exponential:

$$P_e = e^{-ND},$$

where D is the exponent. This quantity is analogous to the error exponent in the theory of error-correcting codes. We therefore identify D as the model's *intelligence density*. Solving for D gives the following formula for the intelligence density

$$D = \frac{-\log(P_e)}{N}. \tag{1}$$

Under this definition, the corresponding measure of *intelligence* is

$$-\log(P_e) \;=\; -\log\Big(1 - \frac{\text{average benchmark score}}{100}\Big). \tag{2}$$

This definition yields a much more plausible scaling than raw benchmark scores. For example, under this measure, a model scoring 99 is roughly twice as intelligent as a model scoring 90, whereas a model scoring 55 is only about 15% more intelligent than a model scoring 50. Using this definition of intelligence density, and the benchmarks studied in Table 5, we compare 20 instruct models in the 1.2B–9B parameter range, together with 1-bit Bonsai 8B and its sister models, 1-bit Bonsai 4B and 1.7B in Table 6.

Table 6. Intelligence density (1/GB). The three Bonsai models lead by a wide margin. Average score is computed over MMLU-Redux, MuSR, GSM8K, HumanEval+, IFEval, and BFCLv3. Intelligence metric is computed as shown in Equation (2) and intelligence density is computed via Equation (1) where Gigabyte (GB) is used as the unit for the model size.

Model	Intelligence Density (1/GB)	Size	Average Score
1-bit Bonsai 1.7B	**2.832**	**0.24 GB**	**49.60**
1-bit Bonsai 4B	**1.744**	**0.57 GB**	**62.72**
1-bit Bonsai 8B	**1.060**	**1.15 GB**	**70.50**
Qwen 3 0.6B [31]	0.549	1.19 GB	48.02
Qwen 3 1.7B [31]	0.318	3.44 GB	66.57
Gemma 3 1B [15]	0.304	2.00 GB	45.53
LFM2 1.2B [20]	0.269	2.34 GB	46.73
Llama 3.2 1B [17]	0.206	2.47 GB	39.88
Ministral3 3B [24]	0.192	6.86 GB	73.22
Qwen 3 4B [31]	0.183	8.04 GB	77.10
Llama 3.2 3B [17]	0.161	6.43 GB	64.35
Gemma 3 4B [15]	0.146	7.76 GB	67.88
Qwen 3 8B [31]	0.096	16.38 GB	79.30
Olmo 3 7B [29]	0.085	14.60 GB	70.90
RNJ 8B [11]	0.079	16.62 GB	73.12
Trinity Nano 6B [3]	0.077	12.24 GB	61.17
Ministral3 8B [23]	0.077	16.04 GB	71.00
LFM2 8B [21]	0.071	16.68 GB	69.58
Llama 3.1 8B [17]	0.069	16.06 GB	67.08
Hermes 3 8B [28]	0.066	16.06 GB	65.43
GLM 4 9B [37]	0.057	18.80 GB	65.73
DeepSeek R1 Qwen 7B [10]	0.052	15.23 GB	55.03
Marin 8B [22]	0.052	16.06 GB	56.55

As expected, smaller models tend to exhibit higher intelligence density. The more important result, however, is that the three 1-bit Bonsai models outperform all other models by a wide margin, indicating that their advantage is not merely a consequence of smaller size. Rather, they achieve a fundamentally better efficiency–performance tradeoff, delivering far more intelligence per GB than conventional full-precision models. This shows that Bonsai 1-bit family models not only showcase strong general-purpose capabilities, they unlock a new regime of highly efficient intelligence.

6. Use Cases

The results in the previous sections show that 1-bit Bonsai models are not merely smaller, but practically deployable in settings where strong model quality must coexist with hard system constraints. One important class of applications is on-device assistance on Mac, iPhone, and iPad, where local execution can improve responsiveness, reduce memory pressure, and strengthen privacy by keeping inference close to the user. The same deployment profile is also relevant in enterprise environments that require local or tightly controlled inference, including settings where data residency, on-prem deployment, or reduced dependence on cloud infrastructure are important.

A second class of applications is cost-sensitive serving on commodity GPUs. In these deployments, bandwidth, VRAM pressure, and energy per token often dominate operating cost more than peak compute alone. By reducing memory footprint and memory movement while improving throughput, 1-bit Bonsai 8B is well suited to serving environments where deployment efficiency is a first-order concern.

The 1-bit Bonsai models are also relevant to edge systems operating under tighter physical constraints, including robotics, autonomy, and embedded inference. In these settings, thermal limits, intermittent connectivity, and restricted memory budgets make full-precision deployment difficult. A compact model that can execute efficiently across heterogeneous hardware expands the range of systems that can run useful language-model inference locally. Taken together, these use cases point to a broader opportunity: enabling capable language models in deployment regimes where conventional models are often too large, too costly, or too infrastructure-dependent to be practical.

7. Limitations and Roadmap

The results in this whitepaper are obtained on general-purpose hardware through software and kernel optimization rather than on a native mainstream 1-bit hardware target. Accordingly, they should be interpreted as deployment results on existing platforms, not as an upper bound on what binary-weight inference could achieve with purpose-built silicon. In addition, mobile energy measurement is less direct than desktop measurement, and the iPhone energy figures reported here are therefore estimated rather than hardware-metered.

At the same time, the Bonsai methodology is architecture-agnostic and is not tied to a single transformer family. Future releases will extend Bonsai to newer model backbones, including emerging non-transformer and hybrid-efficient architectures, diffusion models, as well as to additional efficient deployment formats beyond 1-bit alone.

In the near term, this roadmap includes Bonsai variants at different bit widths together with additional efficiency mechanisms designed to improve deployment on real hardware. The broader objective is not only to demonstrate a single compact model, but to develop a family of practical low-bit inference systems that map cleanly onto different architectures, runtimes, and hardware regimes. In that sense, 1-bit Bonsai 8B should be viewed not as an endpoint, but as the first indicator of a larger shift: capable language models are beginning to move into a regime where deployment efficiency is itself a primary design objective.

A. Achieving 1-bit Inference Acceleration

Although both `llama.cpp` and Apple's MLX ecosystem already support several low-bit quantization schemes, neither natively supports the true 1-bit format Q1_0_g128 introduced with Bonsai family of models. Recall that in Q1_0_g128 format, each weight is represented only by its sign, $\{-1, +1\}$, while a shared scale factor is stored for each group of weights. Enabling efficient inference with this representation therefore required custom implementation work across all target backends, spanning multiple GPU platforms and shader languages.

Why 1-bit models are faster. Token generation in large language models is typically memory-bandwidth-bound: producing each output token requires reading essentially the full set of model weights from memory, while performing relatively little arithmetic per byte transferred. A 1-bit model greatly reduces this memory traffic by storing each weight as a single sign bit together with a per-group scale factor. In practice, as seen in 1-bit Bonsai 8B this reduces the deployed model size by roughly $14\times$ relative to FP16. Because much less data must be fetched from memory for each decoding step, token generation becomes substantially faster.

By contrast, prompt processing is less sensitive to weight bandwidth and more strongly influenced by arithmetic throughput, since many tokens are processed in parallel. As a result, prompt processing sees only modest gains from 1-bit model deployment, typically on the order of 1.0–$1.1\times$, whereas token generation benefits much more significantly.

Implementation across backends. Supporting this format required backend-specific engineering work, since each inference stack exposes different kernel abstractions and GPU programming models.

`llama.cpp` (CUDA and Metal). `llama.cpp` provides multiple hardware backends, and we extended it with custom support for Bonsai's 1-bit representation. On NVIDIA GPUs, we implemented custom CUDA kernels for both matrix-vector and matrix-matrix multiplication that efficiently unpack binary weights and apply the corresponding group scales during execution. On Apple silicon, we implemented custom Metal compute shaders for both single-token decoding and batched prompt processing using the same 1-bit format. For the implementation details, please refer to `https://github.com/PrismML-Eng/llama.cpp`.

MLX (macOS). For Apple's MLX framework on macOS, we maintain a dedicated fork that adds end-to-end support for Bonsai's 1-bit quantization format. This includes custom Metal GPU kernels integrated into the MLX execution stack so that 1-bit models can run efficiently on Mac devices while preserving a workflow that is comparable to standard MLX usage. For the implementation details, please refer to `https://github.com/PrismML-Eng/mlx`.

`mlx-swift` (iPhone/iPad). For iOS deployment, we also maintain a separate fork of Apple's `mlx-swift` framework. Since `mlx-swift` is distinct from the main MLX project, this required an independent implementation of 1-bit support tailored to the iPhone/iPad runtime. As in the macOS case, this support relies on custom Metal GPU kernels designed for efficient execution of Bonsai's packed binary weights on mobile Apple devices. For the implementation details, please refer to `https://github.com/PrismML-Eng/mlx-swift`.

Summary. Taken together, these implementations enable a consistent 1-bit inference path across server-class NVIDIA GPUs, Apple laptops and desktops, and mobile Apple devices. The core acceleration comes from reducing weight bandwidth during decoding, while the backend-specific kernel work ensures that this theoretical advantage is realized in practice on each platform.

B. Benchmark Evaluation Methodology

B.1 Evaluation Framework

All benchmark results reported in this paper were produced using a custom evaluation harness built on EvalScope v1.4.2. Models were served via vLLM 0.15.1 on NVIDIA H100 GPUs with tensor parallelism as needed. The harness automatically manages model serving, benchmark routing, scoring strategy selection, and result aggregation. Every model in the comparison, including both 1-bit Bonsai checkpoints and third-party instruct models, was evaluated using identical infrastructure, reducing the chance that observed differences are driven by evaluation artifacts rather than genuine model capability.

B.2 Benchmark Suite

In the evaluation, we report 10 benchmarks spanning 6 skill categories. In particular, we measure knowledge via MMLU-Redux [26, 14] and GPQA-Diamond [16], reasoning via MuSR [27], math via GSM8K [7] and Math-500 [18], coding via HumanEval+ [8, 19] and MBPP+ [19], instruction-following via IFEval [35] and IFBench [30], and tool-calling via BFCLv3 [4]. We selected benchmarks that (a) test capabilities relevant to real deployment scenarios, and (b) are well-established with published baselines for comparison. We deliberately exclude pretraining-saturated benchmarks such as ARC, WinoGrande, PIQA, and CommonsenseQA from the primary comparison, as these showed limited discriminative spread in our comparisons at the 8B scale, with most models landing in the 85%–95% range.

Category	Benchmarks
Knowledge	MMLU-Redux (57 subjects), GPQA Diamond (198 questions)
Reasoning	MuSR (multistep soft reasoning, 756 questions)
Math	GSM8K (1,319 problems), MATH-500 (500 problems)
Code	HumanEval+ (164 problems), MBPP+ (378 problems)
Instruction Following	IFEval (541 prompts), IFBench
Tool Calling	BFCL v3 (13 single-turn subsets)

B.3 Generation Configuration

The default generation configuration uses non-thinking mode with greedy decoding to measure direct model capability without chain-of-thought amplification. This reflects our target deployment setting, where thinking mode is typically disabled for latency-sensitive applications. For models with an explicit thinking toggle (e.g., the Qwen3 family), we disable it via `enable_thinking=false`; all other models use their standard chat template without added reasoning scaffolding.

The one exception to greedy decoding is GPQA Diamond, which uses 10 samples with sampling (`temperature=0.6`, `top_p=0.95`) and reports mean accuracy across those samples. This follows the common convention of using multiple samples on graduate-level questions where a single greedy pass is noisy.

Parameter	Value
Temperature	0.0 (greedy)
Top-p	1.0
Thinking mode	Disabled (`enable_thinking=false`)
Max tokens	2,048–16,384 (per benchmark)
Seed	42

Max output tokens are tuned per benchmark category based on truncation analysis across multiple models:

Token Budget	Benchmarks
2,048 (short)	MMLU-Redux, MuSR, GSM8K
4,096 (medium)	IFEval, IFBench, BFCLv3, MBPP+
8,192 (long)	MATH-500, GPQA Diamond, HumanEval+

B.4 Deterministic Evaluation

Reproducibility is our first-order concern. vLLM's default attention kernels (`FlashInfer`) can produce non-deterministic outputs across runs due to non-deterministic GPU thread scheduling. Without mitigation, benchmarks such as IFEval showed 2–5 percentage point variance across identical reruns, making model comparisons unreliable.

We enforce determinism through two mechanisms:

- `VLLM_BATCH_INVARIANT=1`, which ensures outputs do not change based on batch composition or scheduling order.
- `--attention-backend FLASH_ATTN`, which forces Flash Attention 2 (deterministic on H100 compute capability 9.0+) instead of the default FlashInfer backend.

These settings are applied uniformly to every model evaluation. Combined with a fixed global seed (`42`), they produced stable, reproducible outputs in repeated reruns on our H100 hardware. All scores in the paper were produced after locking down these settings.

B.5 Scoring Methodology

We use benchmark-appropriate scoring strategies, defaulting to rule-based methods wherever possible to minimize scorer variance. IFEval and IFBench used strict constraint checking matching the Open LLM Leaderboard methodology. BFCL v3 was scored by AST matching and execution verification, while code benchmarks (HumanEval+ and MBPP+) were executed in Docker sandboxes. For knowledge and math benchmarks requiring answer extraction (MMLU-Redux, GPQA Diamond, GSM8K, MATH-500, and MuSR), we used a rule-based extractor with an LLM recall fallback (Gemini 2.5 Flash Lite at temperature $= 0.0$), which activated only when the rule-based parser failed to extract a valid answer. All models used the same judge configuration, dataset revisions, and sample sets. The following gives further details per benchmark.

B.5.1 IFEval and IFBench (Rule-Based Only)

Both IFEval and IFBench use purely rule-based constraint checking with no LLM judge involvement. For IFEval, the reported metric is the Open LLM Leaderboard average:

$$\frac{\texttt{prompt\_level\_strict} + \texttt{inst\_level\_strict}}{2}.$$

This matches the standard methodology from [35] that introduced the IFEval benchmark. For IFBench, we deliberately report the same strict OLLM average rather than the prompt-level loose accuracy used in the [30], to provide a consistent cross-benchmark comparison. Both benchmarks avoid the score inflation observed when LLM recall boosting is enabled. For Qwen3-based models, `<think>... </think>` tags are stripped before constraint evaluation to prevent spurious failures on word-count and format constraints.

B.5.2 BFCL v3 (AST + Execution Verification)

BFCL runs in prompting mode (`is_fc_model=false`): tool definitions are presented in the system prompt and the model outputs function calls as text. Scoring uses AST matching and execution verification (`judge_strategy=rule`), rather than an LLM recall judge. We report the macro-average across the 13 single-turn subsets out of BFCL v3's 17 total tasks; the 4 multi-turn subsets are excluded from the main comparison due to context-window sensitivity and are not included in the reported BFCL score. We set `underscore_to_dot=false` so that dotted function names (e.g., `math.factorial`) are preserved exactly during evaluation.

B.5.3 Code Benchmarks (Sandbox Execution)

HumanEval+ and MBPP+ execute generated code in isolated Docker containers using *python:3.11-slim*. MBPP+ uses EvalPlus's augmented test suite rather than the shorter default `test_list`, applied via a local hotfix to EvalScope. Code extraction uses enhanced rule-based Python parsing (also via hotfix) to reduce false negatives from malformed code fences. Both hotfixes are idempotent scripts included in our evaluation repository for reproducibility.

B.5.4 Knowledge and Math (Rule + LLM Recall Fallback)

For benchmarks requiring answer extraction from free-text responses (MMLU-Redux, GPQA Diamond, GSM8K, MATH-500, MuSR), we use a two-stage scoring pipeline. A rule-based extractor attempts to parse the answer first. Only when rule-based extraction fails does an LLM judge (*Gemini 2.5 Flash Lite*, temperature 0.0) re-evaluate the response for answer extraction. This approach maximizes scoring consistency while recovering valid answers that the rule-based parser misses due to formatting variation. The same judge model and configuration is used for all models.

B.6 Fair Comparison Guarantees

To ensure fair comparisons, the following controls are applied uniformly:

- **Same infrastructure:** All models are served via vLLM on the same H100 hardware with identical vLLM version and configuration.
- **Same generation parameters:** Greedy decoding (`temp=0.0`, `top_p=1.0`) for all benchmarks except GPQA Diamond (sampling, `temp=0.6`). For models with an explicit thinking toggle, it is disabled; all models use their default chat template. Identical per-benchmark token budgets are used.
- **Same benchmark versions:** All models are evaluated on identical dataset revisions, prompts, and sample sets.
- **Same scoring pipeline:** Identical judge model (*Gemini 2.5 Flash Lite*), same rule-based extractors, and same sandbox execution environment for code.
- **Deterministic execution:** Flash Attention 2 with batch-invariant scheduling.
- **No model-specific tuning:** No per-model prompt engineering, system prompts, or generation parameter adjustments. Each model's default chat template is used as-is.

B.7 Software Versions

Component	Version
EvalScope	1.4.2
vLLM	0.15.1
Python	3.10 (conda)
BFCL evaluator	bfcl-eval 2025.10.27.1
Docker sandbox	python:3.11-slim
Judge model	Gemini 2.5 Flash Lite (temperature 0.0)
Hardware	NVIDIA H100 80GB ($8\times$ H100 node)
Attention backend	Flash Attention 2 (`FLASH_ATTN`)

B.8 Reported Metrics

The following metrics are reported per benchmark:

Benchmark	Metric
MMLU-Redux	Accuracy (57 subjects)
GPQA Diamond	Mean accuracy across 10 samples (sampling, temperature 0.6)
MuSR	Accuracy (multistep soft reasoning)
GSM8K	Accuracy (exact match on final numerical answer)
MATH-500	Accuracy (exact match)
HumanEval+	pass@1 (sandbox execution with augmented tests)
MBPP+	pass@1 (sandbox execution with EvalPlus test suite)
IFEval	OLLM avg $= (\texttt{prompt\_strict} + \texttt{inst\_strict})/2$
IFBench	OLLM avg $= (\texttt{prompt\_strict} + \texttt{inst\_strict})/2$
BFCL v3	Macro-average across 13 single-turn subsets (of 17 total BFCL v3 tasks)

C. All Benchmark Results

We present results on the full benchmark suite described in Appendix B. Tables 7, 8, and 9 report comparisons with 20 reference models for 1-bit Bonsai 8B, 4B, and 1-bit Bonsai 1.7B, respectively.

Table 7. Full Benchmark Suite Comparison. 1-bit Bonsai 8B is compared against 11 leading conventional models of comparable scale. Intelligence density is reported in 1/GB using 10 benchmarks using (1) .

Model	Intelligence Density	Avg.	MMLU Redux	GPQA Diamond	MuSR	IFEval	IFBench	GSM8K	MATH 500	Human Eval+	MBPP+	BFCLv3
1-bit Bonsai 8B	**0.792**	**59.86**	**65.7**	**30.0**	**50.0**	**79.8**	**19.8**	**88.0**	**66.0**	**73.8**	**59.8**	**65.7**
Olmo 3 7B	0.077	67.34	72.0	40.4	56.1	87.1	45.1	92.5	91.6	79.3	70.9	38.4
Qwen 3 8B	0.076	71.02	83.0	49.3	55.0	81.5	27.2	93.0	84.4	82.3	73.5	81.0
RNJ 8B	0.065	66.19	75.5	34.0	50.4	73.8	30.7	93.7	82.6	84.2	75.9	61.1
LFM2 8B	0.061	63.89	72.7	31.6	49.5	82.2	40.7	90.1	84.8	61.0	64.3	62.0
Trinity Nano 6B	0.060	51.79	66.8	22.0	52.6	54.0	15.3	81.1	61.2	50.0	52.4	62.5
Mistral3 8B	0.058	60.51	68.9	29.2	53.8	67.4	28.8	87.9	56.0	72.6	65.1	75.4
DS R1 Qwen 7B	0.053	55.39	62.5	41.5	29.1	48.8	25.7	92.7	89.6	81.7	66.9	15.4
Llama 3.1 8B	0.053	56.97	72.9	27.2	51.3	76.4	25.8	87.0	50.2	63.4	64.0	51.5
Hermes 3 8B	0.049	54.13	67.4	29.5	52.2	69.3	29.6	82.9	32.2	51.2	57.4	69.6
GLM 4 9B	0.047	58.88	81.9	37.3	53.2	69.3	23.8	89.4	68.2	78.7	65.1	21.9
Marin 8B	0.042	49.04	64.8	24.8	42.6	63.0	15.3	86.1	56.0	54.9	55.0	27.9

Table 8. Full Benchmark Suite Comparison. 1-bit Bonsai 4B is compared against 4 leading conventional models of comparable scale. Intelligence density is reported in 1/GB using 10 benchmarks using (1) .

Model	Intelligence Density	Avg.	MMLU Redux	GPQA Diamond	MuSR	IFEval	IFBench	GSM8K	MATH 500	Human Eval+	MBPP+	BFCLv3
1-bit Bonsai 4B	**1.427**	**55.39**	**58.7**	**28.7**	**41.4**	**69.6**	**25.2**	**87.3**	**65.8**	**71.3**	**57.9**	**48.0**
Ministral3 3B	0.162	66.99	77.5	41.7	56.5	73.1	37.9	91.4	84.6	69.5	66.4	71.3
Qwen 3 4B	0.143	68.31	79.8	42.9	57.4	80.0	24.3	92.1	83.2	74.4	70.1	78.9
Llama 3.2 3B	0.125	55.15	65.5	27.2	48.9	78.3	31.5	80.1	49.0	52.4	57.7	60.9
Gemma 3 4B	0.120	60.47	66.0	28.7	46.3	73.0	23.7	89.8	75.2	67.1	69.8	65.1

Table 9. Full Benchmark Suite Comparison. 1-bit Bonsai 1.7B is compared against 5 leading conventional models of comparable scale. Intelligence density is reported in 1/GB using 10 benchmarks using (1) .

Model	Intelligence Density	Avg.	MMLU Redux	GPQA Diamond	MuSR	IFEval	IFBench	GSM8K	MATH 500	Human Eval+	MBPP+	BFCLv3
1-bit Bonsai 1.7B	**2.172**	**40.88**	**43.2**	**20.7**	**45.1**	**63.0**	**13.8**	**66.3**	**34.4**	**45.1**	**42.3**	**34.9**
Qwen3 0.6B	0.477	43.34	47.5	26.0	41.5	62.8	17.9	64.1	55.6	30.5	45.8	41.7
Gemma3 1B	0.268	41.49	43.2	24.0	37.0	61.9	14.6	64.4	46.2	40.2	56.9	26.5
Qwen3 1.7B	0.254	58.24	66.8	32.5	50.1	70.3	18.3	83.1	74.0	57.3	58.2	71.8
LFM2 1.2B	0.226	41.08	52.9	23.5	25.4	77.5	27.7	62.2	42.2	36.0	37.0	26.4
Llama 3.2 1B	0.182	36.15	47.2	22.5	29.2	47.7	18.0	49.0	34.4	35.4	47.3	30.8

Viewed through raw benchmark scores alone, the Bonsai models are competitive across all three scales. Viewed through intelligence density, however, they emerge as clear outliers. Across the 8B, 4B, and 1.7B regimes, the Bonsai family delivers substantially more intelligence per GB than nearby conventional base-

lines. In particular, 1-bit Bonsai 8B achieves over $10.2\times$ the intelligence density of the closest model in its weight class, 1-bit Bonsai 4B achieves over $8.8\times$, and 1-bit Bonsai 1.7B achieves over $4.5\times$. This shows that 1-bit Bonsai is not simply more compact than conventional models, but markedly more efficient at translating model size into usable intelligence.

D. Throughput and Energy Measurement Notes

Throughput and power numbers come from separate harnesses. The comparison of tg128 / pp512 in Table 3 is the standardized deployment summary. Energy numbers come from sustained-run workloads with different absolute token rates. In this section, we first discuss the methodology and the extensive results on cross-platform throughput of 1-bit Bonsai family and later focus on the methodology and results on the energy consumption of the models.

D.1 Inference Speed and Throughput Evaluations

We measure two distinct phases of inference across all platforms:

Prompt processing (pp). The model ingests a batch of input tokens in parallel, populating the key-value cache. This phase is compute-bound and benefits from high arithmetic throughput. We report **pp512**—a benchmark that feeds 512 input tokens to the model—as the primary prompt processing metric.

Token generation (tg). The model produces output tokens one at a time in an autoregressive loop. This phase is memory-bandwidth-bound because each step reads the full model weights to produce a single token. We report **tg128**—a benchmark that generates 128 output tokens—as the primary generation metric.

Both metrics are reported in tokens per second (`tok/s`). Multiple iterations and repetitions are run for each configuration to reduce variance, and we report the mean.

llama.cpp. All `llama.cpp` speed benchmarks use `llama-bench`, the standard benchmarking utility included with `llama.cpp`. It directly measures pp and tg throughput with configurable prompt sizes, generation lengths, and backend settings. We evaluate `llama.cpp` across CUDA (NVIDIA GPUs) and Metal (Apple Silicon).

MLX Python. For the Apple MLX framework, we use custom benchmark scripts that produce directly comparable pp512 and tg128 measurements. Each configuration is repeated multiple times and we report the mean.

mlx-swift (iPhone/iPad). `mlx-swift` benchmarks are run on an iPhone 17 Pro Max (A19 Pro, 12 GB). All tests use the same $\sim$250-token prompt and allow the model to generate 250 output tokens, ensuring consistent and comparable results across configurations.

Token generation shows the largest speedups for all models (see Table 3 for 1-bit Bonsai 8B), while prompt processing typically improves more modestly, by up to $\sim$10% in most cases. See Table 10 for the prompt processing comparison for the entire 1-bit Bonsai family models, and Tables 11 and 12 for the specific throughput results for the Bonsai 4B and 1.7B models, respectively.

Table 10. Prompt processing throughput comparison between 1-bit Bonsai and the FP16 baseline using standardized pp512 deployment measurements. Prompt processing is compute-bound, so the speedup from 1-bit is modest compared to token generation. In most cases 1-bit matches or slightly exceeds FP16 prompt processing speed.

Model	Platform	Backend	1-bit (tok/s)	FP16 (tok/s)	Speedup
8B	RTX 4090	llama.cpp CUDA	11,809	10,453	**1.13**×
8B	M4 Pro 48 GB	MLX (Python)	472	434	**1.09**×
8B	M4 Pro 48 GB	llama.cpp Metal	498	490	**1.02**×
4B	RTX 4090	llama.cpp CUDA	18,135	16,310	**1.11**×
4B	M4 Pro 48 GB	MLX (Python)	806	728	**1.11**×
4B	M4 Pro 48 GB	llama.cpp Metal	915	915	**1.00**×
1.7B	RTX 4090	llama.cpp CUDA	31,899	30,630	**1.04**×
1.7B	M4 Pro 48 GB	MLX (Python)	1,759	1,585	**1.11**×
1.7B	M4 Pro 48 GB	llama.cpp Metal	2,305	2,291	**1.01**×

Table 11. Cross-platform token generation throughput summary for Bonsai 4B and comparison with respect to the FP16 baseline using standardized tg128 deployment measurements.

Platform	Backend	Size	TG128 (tok/s)	Base TG (tok/s)	Speedup
RTX 4090	llama.cpp CUDA	566 MB	440	105	**4.2**×
M4 Pro 48 GB	llama.cpp Metal	566 MB	136	29	**4.7**×
M4 Pro 48 GB	MLX (Python)	617 MB	132	28	**4.8**×
iPhone 17 Pro Max	MLX Swift	617 MB	60	—[1]	—[1]

[1] No FP16 baseline is reported for iPhone because the FP16 model does not fit on-device.

Table 12. Cross-platform token generation throughput summary for Bonsai 1.7B and comparison with respect to the FP16 baseline using standardized tg128 deployment measurements. On iPhone 17 Pro Max, 1-bit Bonsai 1.7B achieves 130token/s.

Platform	Backend	Size	TG128 (tok/s)	Base TG (tok/s)	Speedup
RTX 4090	llama.cpp CUDA	242 MB	674	224	**3.0**×
M4 Pro 48 GB	llama.cpp Metal	242 MB	250	65	**3.8**×
M4 Pro 48 GB	MLX (Python)	274 MB	288	62	**4.6**×

D.2 Power Instrumentation

Mac M4 Pro: On macOS, we use **IOReport** to measure power across CPU, GPU, ANE, and DRAM domains. We subtract an idle baseline from inference readings to isolate power attributable to model execution, i.e. exclude the system overhead, and accumulate energy over multiple runs to report energy per token (mWh/tok).

RTX 4090: On NVIDIA hardware, we log GPU power using **nvidia-smi**. To isolate pure generation power,

we subtract a short run (1 token output) from a full run (1024 tokens output) — both have the same model load and prefill overhead, so the difference reflects only the generation phase.

iPhone 17 Pro Max: iOS does not expose power metrics directly. We estimate energy per token from observed battery drain rate (about 70%/hr during sustained generation) and the device's rated battery capacity 5,088 mAh (19.6 Wh), subtracting display and system overhead using Xcode Power Profiler. These iPhone numbers are approximate estimates. The baseline is 4-bit rather than 16-bit, as FP16 does not fit on the device's memory.

D.3 Energy Definitions

$$E_{\text{tg}} = \frac{P_{\text{tg}}}{3.6\, r_{\text{tg}}}, \qquad E_{\text{pp}} = \frac{P_{\text{pp}}}{3.6\, r_{\text{pp}}},$$

where P is measured inference power in watts, r is tokens per second, and energies are in mWh.

Table 13. Selected power, speed, and output-token energy comparisons. E_{tg} is measured in mWh/output-token and E_{pp} is measured in mWh/input-token

Platform	Bonsai power (W)	Baseline power (W)	Bonsai E_{tg}	Baseline E_{tg}	Bonsai E_{pp}	Baseline E_{pp}
Mac M4 Pro (MLX)	31.9 W	23.3 W	0.074	0.415	0.018	0.021
Mac M4 Pro (Metal)	27.2 W	27.9 W	0.091	0.471	0.006	0.006
RTX 4090 (CUDA)	297 W[1]	231 W[1]	0.276	1.134	—	—

[1] Pure generation power after subtracting model-load and prefill overhead.

References

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

[2] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*, 2023.

[3] Arcee AI. Trinity Nano (6B). https://docs.arcee.ai/language-models/trinity-nano-6b, 2025.

[4] S. G. Patil, H. Mao, F. Yan, C. C. Ji, V. Suresh, I. Stoica, and J. E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of ICML*, 2025.

[5] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei. The Era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

[6] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei. BitNet: Scaling 1-bit Transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

[7] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Pinto de Oliveira, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[9] T. Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[10] DeepSeek AI. DeepSeek-R1-Distill-Qwen-7B. https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B, 2025.

[11] Essential AI. Rnj-1. https://essential.ai/research/rnj-1, 2025.

[12] G. Gerganov. GGUF: GPT-Generated Unified Format specification. https://github.com/ggml-org/ggml/blob/master/docs/gguf.md, 2023.

[13] G. Gerganov et al. llama.cpp: LLM inference in C/C++. https://github.com/ggml-org/llama.cpp, 2023–2026.

[14] A. P. Gema, J. O. J. Leang, G. Hong, A. Devoto, A. C. M. Mancino, R. Saxena, X. He, Y. Zhao, X. Du, M. R. G. Madani, C. Barale, R. McHardy, J. Harris, J. Kaddour, E. van Krieken, and P. Minervini. Are we done with MMLU? *arXiv preprint arXiv:2406.04127*, 2024.

[15] Google. Gemma 3 Release. https://huggingface.co/collections/google/gemma-3-release, 2025.

[16] D. Rein, B. Hou, A. Stickland, J. Petty, R. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

[17] A. Grattafiori, A. Dubey, A. Jauhri, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

[18] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

[19] 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 *Proceedings of NeurIPS*, 2023.

[20] Liquid AI. LFM2-1.2B. https://huggingface.co/LiquidAI/LFM2-1.2B, 2025.

[21] Liquid AI. LFM2-8B-A1B: An Efficient On-device Mixture-of-Experts. https://www.liquid.ai/blog/lfm2-8b-a1b-an-efficient-on-device-mixture-of-experts, 2025.

[22] Marin Project (Stanford CRFM). Marin 8B Instruct. https://huggingface.co/marin-community/marin-8b-instruct, 2025.

[23] Mistral AI. Ministral-8B-Instruct-2410. https://huggingface.co/mistralai/Ministral-8B-Instruct-2410, 2024.

[24] Mistral AI. Ministral 3 3B Instruct 2512. https://huggingface.co/mistralai/Ministral-3-3B-Instruct-2512, 2025.

[25] A. Hannun, J. Digani, A. Katharopoulos, and R. Collobert. MLX: An array framework for Apple silicon. https://github.com/ml-explore/mlx, 2023.

[26] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. In *Proceedings of ICLR*, 2021.

[27] Z. Sprague, X. Ye, K. Bostrom, S. Chaudhuri, and G. Durrett. MuSR: Testing the limits of chain-of-thought with multistep soft reasoning. In *Proceedings of ICLR*, 2024.

[28] Nous Research. Hermes 3: Llama-3.1-8B. https://huggingface.co/NousResearch/Hermes-3-Llama-3.1-8B, 2024.

[29] Team OLMo, A. Ettinger, A. Bertsch, B. Kuehl, et al. OLMo 3. *arXiv preprint arXiv:2512.13961*, 2025.

[30] V. Pyatkin, S. Malik, V. Graf, H. Ivison, S. Huang, P. Dasigi, N. Lambert, and H. Hajishirzi. Generalizing Verifiable Instruction Following. *arXiv preprint arXiv:2507.02833*, 2025.

[31] Qwen Team. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025.

[32] B. Zhang and R. Sennrich. Root mean square layer normalization. In *Proceedings of NeurIPS*, 2019.

[33] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[34] N. Shazeer. GLU variants improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[35] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

[36] 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.

[37] Z.ai. GLM-4-9B-0414. https://huggingface.co/zai-org/GLM-4-9B-0414, 2025.