Bonsai 27B

Full 27B-class reasoning in binary and ternary transformer weights — on laptops and phones

~9.4×	95%	26 tok/s	11 tok/s
smaller (ternary)	intelligence retained	on laptop	on iPhone 17 Pro Max

PrismML · July 2026

Contents

1 Executive Summary **3**

2 Efficiency as the Defining Constraint at 27B Scale **4**
2.1 Inference Defines the Economics of AI . . . 4
2.2 Memory Bandwidth Is the Real Bottleneck . . . 4
2.3 Why Sub-4-Bit Has Remained Out of Reach . . . 5
2.4 What "Edge" Has Meant — and Why 27B Was Excluded . . . 5

3 What "Low-Bit" Actually Means **6**

4 Bonsai 27B Model Summary **7**
4.1 Ternary Weight Format . . . 7
4.2 Binary Weight Format . . . 7
4.3 Storage Footprint . . . 8
4.3.1 Shipped Components . . . 8
4.4 KV-cache quantization . . . 9
4.4.1 Peak Memory at Context . . . 10

5 Inference: Low-Bit Kernels for Hybrid-Attention Models **10**
5.1 Cross-Platform Throughput . . . 11

6 Speculative Decoding: DSpark **12**
6.1 DSpark: Semi-Autoregressive Drafting with Confidence-Scheduled Verification . . 12
6.2 Accepted Length and Decode Speedup . . . 13

7 Benchmarks and Intelligence Density **14**
7.1 Intelligence Density . . . 15

8 Use Cases **16**

9 Limitations and Roadmap **17**

A Deployment and Kernel Reference **18**

B Benchmark Evaluation Methodology **18**
B.1 Thinking Mode and Generation . . . 18
B.2 Token Budgets . . . 18
B.3 Scoring . . . 19
B.4 Sampling Repeats and Determinism . . . 19
B.5 Benchmark Suite and Reported Metrics . . . 19

C All Benchmark Results **20**

D Throughput and Energy Measurement Notes **21**

Category	Summary
Model	Bonsai 27B — 1-bit and Ternary Bonsai representations of Qwen3.6-27B [3], a 27B hybrid-attention causal language model
Format	Ternary g128: $\{-1, 0, +1\}$ weights, or Binary g128: $\{-1, +1\}$ weights, both with FP16 group-wise scaling
Deployment	Custom 1-bit and 2-bit kernels for hybrid attention on Apple MLX (Python, Swift) and CUDA; ships with a DSpark speculative-decoding layer
Value	~ 5.9 GB (ternary) / ~ 3.9 GB (binary) deployed; retains 27B-class thinking, reasoning, and agentic capability; ~ 26 tok/s on laptop, ~ 11 tok/s on iPhone
Use case	Ternary on standard laptops; 1-bit on high-end phones (iPhone 17 Pro Max); single-GPU serving
License	Apache License

1. Executive Summary

The 27B parameter tier marks a new level of practical intelligence. Models at this scale sustain the behaviors that make language models genuinely useful in production — multi-step reasoning, dependable tool use, and stable agentic workflows — with a reliability that smaller models do not reach. Bonsai 27B is derived from **Qwen3.6-27B** [3], a 27B hybrid-attention model whose predominantly linear-attention backbone ($\sim$75% linear attention) keeps long-context inference practical. We ship it with a full **262K-token** context, enabled on edge devices via a 4-bit KV cache quantizer. Crucially, the same representation that enables extreme weight compression also makes the model unusually tolerant of even more aggressive KV quantization, keeping long-context inference practical on-device rather than confining it to the cloud.

The obstacle has never been capability; it is deployment. At roughly 6 bits per weight, conventional low-bit approaches preserve almost all of this capability— but the resulting model is still far too large for the edge. Pushing below 4 to 5 bits with conventional methods causes a qualitative collapse rather than a graceful decline: meaning that the model loses the very behaviors that make a 27B model worth running. Chain-of-thought reasoning becomes unreliable, tool calls stop parsing, and agentic loops lose coherence. In production this brittleness is more damaging than a clean accuracy drop, because it undermines trust and makes system behavior hard to predict.

Bonsai 27B moves the entire Qwen3.6-27B network into a binary or ternary representation that **retains** that intelligence rather than trading it away. Built on proprietary Caltech intellectual property — a mathematically grounded framework rather than a collection of heuristics — the Bonsai representation preserves thinking, reasoning, and agentic behavior deep into the sub-4-bit regime:

- **Ternary Bonsai 27B** — 80.49 average benchmark score (95% of FP16) at 1.71 bits per weight, a $\sim 9.4\times$ weight compression, for a full model of just ~ 5.9 GB.
- **1-bit Bonsai 27B** — 76.11 average (90% of FP16) at 1.125 bits per weight, a $\sim 14.2\times$ weight compression, for a full model of just ~ 3.9 GB.

The result redefines what "edge" means. Until now, 27B-class intelligence has meant either sending requests to the cloud or relying on unrealistic local setups, e.g. multiple-GPU rigs, high-memory workstations, or specialized hardware that most people do not have in their daily lives. Edge deployment, by contrast, has typically meant much smaller models. Bonsai 27B breaks that pairing: the ternary model (~ 5.9 GB) runs not on a developer's lab bench, but on an everyday laptop, and the 1-bit model (~ 3.9 GB) fits, for the first time, on a high-end phone such

as the iPhone 17 Pro Max — at ~ 26 tok/s and ~ 11 tok/s respectively.

Intelligence vs. Memory Footprint

The comparison is direct. Qwen3.6-27B in FP16 occupies 54 GB and averages 85.07 on the 15-benchmark suite of Section 7. The most aggressive available low-bit build, IQ2_XXS, shrinks this to 9.4 GB but drops the average to 72.7, losing reasoning and agentic reliability. It is also still too large for most of the edge devices. Note that, despite its label, IQ2_XXS is not actually 2-bit: its true average is 2.8 bits per weight; see Section 3.

The Bonsai models are far smaller and score higher. Ternary Bonsai 27B averages 80.49 (95% of FP16) at ~ 5.9 GB, beating IQ2_XXS at a smaller size. 1-bit Bonsai 27B averages 76.11 (90% of FP16) at ~ 3.9 GB. Conventional methods never reach this regime. Bonsai keeps useful intelligence intact at sizes that fit the devices people actually own.

2. Efficiency as the Defining Constraint at 27B Scale

2.1 Inference Defines the Economics of AI

Large language models have entered a phase where the central question is no longer whether a capable model can be trained, but whether it can be run reliably, affordably, and at scale. In production, the recurring burden is inference. Every user turn, every agent step, and every tool call carries latency, energy, and infrastructure cost, and these accumulate over the lifetime of a deployed system. Deployment efficiency, not raw capability, has therefore become the primary constraint on real-world adoption.

This constraint is sharpest on edge hardware. Laptops, phones, and embedded systems operate under fixed limits on memory, bandwidth, thermal headroom, and battery life. 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. A 27B model that delivers excellent answers only from a datacenter has not solved the deployment problem for the device in the user's hand.

2.2 Memory Bandwidth Is the Real Bottleneck

During autoregressive generation, and especially at the small batch sizes typical of on-device use, the limiting factor is usually not arithmetic throughput but memory movement. Producing each token requires streaming essentially the full set of model weights out of memory, while performing relatively little computation per byte transferred. Performance is therefore governed by how efficiently the system can move weights, which makes model footprint and memory bandwidth first-order variables rather than implementation details.

This is why the representation of the weights matters so much. Storing each weight in one or two bits does more than shrink the checkpoint on disk: it reduces the memory traffic incurred at every decoding step, lowers energy per token, and makes it feasible to keep a strong model resident on constrained hardware. The value of a low-bit representation is not compression in the abstract; it is the direct effect on the dominant cost in real inference systems.

This translates into a concrete speed ceiling. For a bandwidth-bound decoder, the maximum generation rate is bounded by device memory bandwidth divided by the model's weight footprint, provided the model first fits with enough headroom for the KV cache and activations. Both requirements therefore favor smaller footprints: fitting is the hard gate, and footprint sets the ceiling once that gate is passed.

On phones, the fitting constraint is tighter than raw RAM suggests. iOS limits any single app to roughly half of physical memory, so a 12 GB iPhone exposes only about 6 GB to the model. That rules out FP16 and Q4_K_XL, and even the ternary build (5.9 GB) would consume nearly

the entire budget with nothing left for the KV cache and activations — leaving 1-bit Bonsai at 3.9 GB as the only variant that runs on-device with headroom.

For variants that do fit, the speed advantage follows directly from footprint. Because the hardware bandwidth is the same, it cancels in the comparison: Ternary Bonsai, at 5.9 GB, moves $3.1\times$ fewer bytes per token than Q4_K_XL at 17.6 GB, while 1-bit Bonsai, at 3.9 GB, moves $4.7\times$ fewer. Their bandwidth-limited generation ceilings are therefore $3.1\times$ and $4.7\times$ higher, respectively.

2.3 Why Sub-4-Bit Has Remained Out of Reach

For a 27B model the footprint ladder makes the problem concrete. In FP16 the weights occupy roughly 54 GB; even the FP8 build Qwen provides natively [4] needs about 27 GB (roughly half); a true 4-bit representation lands near 14 GB. Each step down helps, yet all three remain beyond the reach of phones and most consumer laptops. The regime that would actually fit those devices — well below 4 bits per weight — is precisely the regime where conventional methods break down.

The failure below 4 bits is qualitative, not gradual. A model can remain fluent and confident while becoming materially less dependable on the tasks that matter most: multi-step reasoning, retrieval, tool use, and long agentic loops. Chain-of-thought traces drift or contradict themselves; structured tool calls stop parsing; multi-turn plans lose coherence partway through. Because the surface fluency is preserved, this degradation is easy to miss in casual testing and expensive to discover in production. A brittle model that fails unpredictably is often worse than one that is uniformly and visibly weaker, because it quietly erodes trust in the whole system.

The appeal of binary and ternary weights has been understood for years, but a capable model in this regime has stayed out of reach. The most prominent prior line — BitNet and its 1.58-bit successor [5, 6] — avoids the quality collapse only by pretraining the network from scratch directly in the low-bit regime. That is a fundamentally more restrictive proposition: it discards every existing pretrained model and demands a full pretraining run for each new one, at a cost few can absorb. It has also not scaled: the largest native 1-bit models remain around 2B parameters — more than an order of magnitude below a 27B model — and whatever parity they reach with full-precision models has been shown only at that small scale, with 7B and larger left as open questions. Other near-1-bit approaches instead lean on bespoke calibration, auxiliary metadata, or custom runtimes that do not integrate with mainstream inference stacks. Bonsai takes the opposite path from BitNet: it starts from an off-the-shelf pretrained model and moves it into a binary or ternary representation, so the specific model practitioners already rely on is preserved rather than replaced. PrismML's earlier Bonsai releases, spanning 1.7B through 8B in both binary and ternary form [1, 2], established that this path is practical and deployable. Those models, however, did not offer thinking, chain-of-thought reasoning, or reliable tool use — the capabilities that make a modern model genuinely useful in agentic production. Bonsai 27B is the first in the family to carry that full capability set intact into the 1-bit and ternary regimes.

2.4 What "Edge" Has Meant — and Why 27B Was Excluded

Historically, "edge" has been synonymous with small models, for a concrete reason: the memory budgets are small. A flagship phone exposes only a few gigabytes to any single application; a mainstream laptop shares 16–48 GB of unified memory across the operating system and everything else running; a consumer GPU carries 8–24 GB of VRAM. Long-context inference also requires a KV cache that grows with every token, alongside activations, runtime workspace, and the rest of the system. At moderately long context, the cache can rival or exceed the model itself. Once room is reserved for all of these, the practical ceiling for a locally runnable model has remained at only a few billion parameters.

That ceiling is a capability limit, not just a size limit. The behaviors that define a modern assistant — sustained chain-of-thought reasoning, multi-step problem solving, and dependable tool use

inside an agentic loop — do not appear reliably until a substantially larger scale. Edge deployment has therefore forced a hard choice: run a small model locally and accept limited capability, or send every request to a 27B-class model in the cloud and accept its latency, cost, and exposure of user data. The 27B tier has been treated as inherently cloud-bound not because its intelligence is unportable, but because its bytes were.

Bonsai 27B removes the choice. By moving the 27B network into a representation small enough for the budgets above while preserving its behavior, it places the capable tier — thinking, reasoning, and agentic tool use — inside the same devices that until now admitted only small models. That is what it means to redefine the edge: not a faster small model, but frontier-class capability running where the user and the data already are.

3. What "Low-Bit" Actually Means

Bit-width labels in this field rarely correspond to the true average bit-width, and the distinction matters for exactly the comparison this paper makes. A model advertised as "2-bit" is almost never two bits per weight. Conventional low-bit schemes are mixed-precision by construction: they keep embeddings, the attention and output projections, or other sensitivity-flagged tensors at four to eight bits, compress only a subset of tensors aggressively, and add per-block scales and minima on top. The advertised name describes the most-compressed tensors, not the model.

The footprints of the evaluated builds make this concrete. Measured as a true average over all weights, the conventional Qwen3.6-27B builds are far heavier than their names imply — the "4-bit" build is really 5.2 bits per weight and the "2-bit" build 2.8 — while the Bonsai representations carry bit-widths that match their names.

Table 1. Advertised label vs. true average bits per weight for the evaluated Qwen3.6-27B builds.

Build	Advertised	True avg bits/weight	Footprint
Q4_K_XL	"4-bit"	5.2	17.6 GB
IQ2_XXS	"2-bit"	2.8	9.4 GB
Ternary Bonsai 27B	ternary	**1.71**	5.9 GB
1-bit Bonsai 27B	1-bit	**1.125**	3.9 GB

Two things follow. First, no conventional *post-training* scheme is end-to-end: unlike the from-scratch BitNet line above, there is always a higher-precision escape hatch somewhere in the decoding path. That is precisely why these schemes preserve quality above four bits — the sensitive tensors are never actually compressed — and also why they cannot reach the sub-2-bit footprints that edge deployment requires. The escape hatches dilute the very footprint and bandwidth reductions that make a low-bit representation worthwhile.

Second, the honest comparison is bits-to-bits, not name-to-name. To be explicit about scope: Bonsai applies its binary or ternary representation end-to-end across the *language* model's matrix-heavy components — embeddings, attention projections, MLP projections, and the LM head — at a true 1.125 and 1.71 bits per weight. The other modality is handled separately and stated plainly: the vision tower is held at 4-bit (HQQ [18]), and a negligible tail of normalization and scale parameters remains in higher precision (Section 4.3). What Bonsai does not do is keep large high-precision language-weight tensors behind a low-bit label. Running across the runtimes developers already use — Apple MLX and CUDA — through custom kernels for its hybrid-attention backbone, Bonsai reaches a regime that conventional methods, once their real bit-widths are counted, never actually enter.

4. Bonsai 27B Model Summary

Bonsai 27B is derived directly from Qwen3.6-27B [3]; the architecture is unchanged. The novelty spans two layers: the representation transformation that maps the pretrained 27B into binary or ternary weights while preserving its behavior, and the deployment stack that executes those weights efficiently — end-to-end low-bit storage, an explicit runtime format, and custom kernels for the hybrid-attention backbone. Because roughly 75% of the attention is linear, the shipped 262K-token context remains practical for full-context, on-device use, where a conventional all-quadratic attention stack would make long contexts prohibitively expensive on edge hardware.

Table 2. System specification.

Item	Specification
Architecture model	Based from Qwen3.6-27B [3], a 27B hybrid-attention causal language model
Parameters	∼24.8B language (64 blocks) + 0.46B vision tower (27 blocks) + 2.5B embedding/lm-head
Architecture	Hybrid attention (∼75% linear attention / ∼25% full attention), SwiGLU MLP, RoPE, RMSNorm
Context length	262K tokens (full-context capable; enabled by the predominantly linear-attention backbone)
Weight formats	Ternary g128 and Binary g128 (FP16 group-wise scaling)
Low-bit weights	Embeddings, attention projections, MLP projections, LM head
Acceleration	DSpark [7] speculative-decoding layer provided
Backends	MLX (Python, Swift) and CUDA
License	Apache License

4.1 Ternary Weight Format

Each weight takes a value from $\{-1, 0, +1\}$, with one shared FP16 scale for every group of 128 weights. The effective weight is

$$w_i = s_g \cdot t_i, \qquad t_i \in \{-1, 0, +1\},$$

where s_g is the scale for group g. A ternary value carries $\log_2 3 \approx 1.585$ bits of information, so with one FP16 scale per 128 weights the effective storage cost is

$$b_{\text{eff}} \approx \log_2 3 + \frac{16}{128} \approx 1.71 \text{ bits/weight},$$

an idealized raw-weight reduction of $16/1.71 \approx 9.4\times$ relative to FP16, before container overhead and alignment. Relative to the binary format, the extra zero state gives a more expressive alphabet and recovers more of the full-precision model's behavior, which makes ternary the quality-oriented operating point of the family.

4.2 Binary Weight Format

Each weight takes a value from $\{-1, +1\}$ — a single sign bit — again with one shared FP16 scale per group of 128 weights:

$$w_i = s_g \cdot b_i, \qquad b_i \in \{-1, +1\}.$$

With one FP16 scale per 128 weights the effective storage cost is

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

an idealized raw-weight reduction of $16/1.125 \approx 14.2\times$ relative to FP16. This is the most aggressive point in the family: it minimizes both the stored footprint and the weight traffic incurred at every decoding step, and it is what brings the model within a phone's memory budget.

4.3 Storage Footprint

The reported *ideal* size uses the theoretical bits/weight of the representation, while the *deployed* size uses the true packed layout that today's kernels consume. The two differ only for ternary: current kernels store each ternary value in a 2-bit slot, so the deployed ternary footprint is larger than its information-theoretic minimum until native ternary kernels close the gap. The binary format has no such gap — its native 1-bit g128 layout is exactly what ships.

Table 3. Storage footprint. Ideal (theoretical) representation size and deployed packed size (including the HQQ 4-bit vision tower). Sizes are decimal GB (10^9 bytes).

Format	Ideal bpw	Ideal size	Reduction	Deployed
FP16 (baseline)	16.0	∼54 GB	1.0×	—
Ternary g128	1.71	5.9 GB	∼9.4×	∼7.2 GB
Binary g128	1.125	3.9 GB	∼14.2×	∼3.9 GB

4.3.1 Shipped Components

The deployed figures above describe the language model alone — deliberately, since it is the only component that must stay resident for text inference. Each release ships two optional components alongside it, in a compact default and a reference-precision variant:

Table 4. Shipped components of the GGUF releases (on-disk sizes). Only the language model must remain resident for text inference; the DSpark drafter and vision tower are optional add-ons.

Component	Pack	Ternary	Binary	Residency
Language model	low-bit g128 (Q2_0 / Q1_0)	7.17 GB	3.9 GB	resident
DSpark drafter	Q4_1 (default)	1.95 GB	1.79 GB	optional; speculative decoding
DSpark drafter	bf16 (reference)	7.29 GB	7.29 GB	optional
Vision tower	mmproj HQQ 4-bit (Q8_0 container)	0.63 GB	0.63 GB	optional; multimodal input only
Vision tower	mmproj BF16 (reference)	0.93 GB	0.93 GB	optional

In practice the compact packs are the ones served. Enabling speculative decoding adds the Q4_1 drafter — 1.95 GB against the ternary target and 1.79 GB against the binary — an overhead that pays for itself on the CUDA serving path (Section 6). The vision tower's weights are HQQ 4-bit [18]; the Q8_0 file is a packaging convenience — the container stores them at 8 bits for drop-in deployment, which is why the 0.63 GB pack sits above the ∼0.24 GB that native 4-bit packing would give. In either form it is usually *offloaded*: it sits outside the accelerator's resident budget and is brought in only when an image actually arrives, so text-only serving never pays for it. Neither component therefore changes the headline resident footprint. One further pack rounds out the releases: a group-64 ternary pack (7.59 GB) matching the 64-value-group Q2_0 packing introduced in llama.cpp — the same native g128 representation with each scale simply repeated per 64-value block, so the extra 0.4 GB is packing overhead rather than added information.

MLX packaging. The MLX releases are slightly larger than their GGUF counterparts: the published safetensors measure 5.13 GB for the 1-bit pack and 8.49 GB for the ternary pack (each bundling the vision tower in the same file). The difference is a container property, not a different representation. MLX's grouped low-bit formats store both a scale and a bias per group, reconstructing $w = s_{\text{mlx}}\, b_i + b_{\text{mlx}}$ with $b_i \in \{0, 1\}$. Bonsai's scale-only weights are packed by setting $s_{\text{mlx}} = 2s_g$ and $b_{\text{mlx}} = -s_g$, which reproduces $\pm s_g$ exactly — the bias carries no new information — but stores two FP16 values per group where GGUF stores one. This is a current MLX limitation; once it supports scale-only group formats, the MLX packs match the native rates.

Fitting the weights, however, is only half the battle: a long context brings a growing KV cache of its own — the second budget an on-device deployment must clear (Section 4.4).

4.4 KV-cache quantization

On-device long context is a memory problem twice over: the weights must fit, and so must the KV cache they generate. Because Qwen3.6-27B is a hybrid model, only **16 of 64 layers** carry a growing full-attention cache (the linear-attention layers keep a fixed-size recurrent state), so the cache is already $\sim 4\times$ smaller than a dense 27B transformer. Even so, at FP16 it is ≈ 64 KiB/token, and a 256K-token window costs ≈ 17 GB — more than the weights of either Bonsai model, over twice the ternary build and four times the binary.

Table 5. KV-cache memory by context length (language model only). The hybrid backbone caches only 16 of 64 layers, so the FP16 cache is ≈ 64 KiB/token; 4-bit KV cuts it $\approx 4\times$. Sizes in decimal GB (10^9 bytes).

KV format	16K context	64K context	128K context	262K context
FP16 KV cache	1.1	4.3	8.6	17.2
4-bit KV cache	0.27	1.1	2.1	4.3

At 4 bits the arithmetic works out: ≈ 4.3 GB of cache at the full 256K window keeps even the maximum context within laptop-class memory budgets, and ≈ 2.1 GB at 128K brings long-context mobile deployment within reach for the binary model on high-memory devices.

A 4-bit cache is only worth shipping if the model tolerates it. We measure tolerance as the output forward-KL that the 4-bit KV cache induces relative to each model's *own* FP16-KV baseline, under an identical KV recipe, in two regimes: *on-policy*, over each model's own MATH-500 [12] generations, and *off-policy*, over BABILong [13] at 16K context.

Table 6. KV tolerance. Output forward-KL (nats) induced by a 4-bit KV cache, measured for each model against *its own* FP16-KV baseline (lower = more tolerant); the KV recipe is identical across rows. On-policy: MATH-500 generations (N=100); off-policy: BABILong at 16K context (N=100).

Model (weight format)	On-policy (MATH-500)	Off-policy (BABILong-16K)
Qwen3.6-27B FP16	0.0137	0.222
Qwen3.6-27B Q4_K_XL ("4-bit")	0.0146	0.259
Ternary Bonsai 27B (1.71-bit)	**0.0011**	**0.0029**
Binary Bonsai 27B (1.125-bit)	**0.0009**	**0.00233**

The Bonsai models carry a 4-bit KV cache nearly losslessly. Relative to the FP16 reference, the ternary and binary models induce roughly 12–$15\times$ less output divergence on-policy and 75–$95\times$ less off-policy, whereas the conventional "4-bit" build behaves just like FP16 — its

compressed weights buy it no cache tolerance at all. On downstream accuracy benchmarks, Bonsai's 4-bit KV is near-lossless, with substantial headroom for further compression. The reading is natural: a 4-bit cache adds discretization noise, and the Bonsai models have already been shaped to tolerate exactly this kind of noise — so compressing the KV cache discards far less useful information than it does in conventional FP16 or 4-bit weight builds. [1]

4.4.1 Peak Memory at Context

What a device must actually accommodate is not the weight footprint alone but *peak* memory: the weights, the KV cache, and the activations and runtime buffers on top. Table 7 reports measured peaks by context length. Beyond the weights, the FP16 KV cache grows at 64 KiB per token — kept this low by the predominantly linear-attention backbone — and activations and framework buffers add roughly a further 1.3 GB on every backend. Because the cache is a property of the architecture rather than the weight format, every row grows at this same rate: at any fixed context the rows differ only by their weights and backend buffers, so the low-bit weights are what separate the columns, and the shared growth term is exactly what the 4-bit KV cache of Section 4.4 attacks. The MLX packs start ~ 0.4 GB larger than their GGUF counterparts for the scale-plus-bias packing reason described in Section 4.3.

Table 7. Measured peak memory (weights + peak activations + KV cache + overhead) by context length, language model only — no drafter layers, no vision tower, no KV-cache compression. Sizes in decimal GB (10^9 bytes). The Q4_K_XL row is derived from its weight footprint plus the same measured cache-and-overhead build-up; all other rows are directly measured.

Model	Format	Weights	4K context	10K context	100K context
27B 16-bit	GGUF bf16	51.25	52.6	53.3	59.3
1-bit Bonsai 27B	llama.cpp Q1_0	3.79	5.2	5.6	11.6
1-bit Bonsai 27B	MLX 1-bit	4.21	5.9	6.3	12.2
Ternary Bonsai 27B	llama.cpp Q2_0	7.15	8.4	8.7	14.7
Ternary Bonsai 27B	MLX 2-bit	7.57	9.2	9.6	15.5
Qwen3.6-27B "4-bit"	llama.cpp Q4_K_XL	17.6	19.2	19.6	25.6

The practical consequence: both Bonsai variants hold a **100K-token context on many devices without any KV-cache compression** — the 1-bit build peaks at 11.6–12.2 GB and the ternary build at 14.7–15.5 GB at 100K tokens, budgets that fit mainstream laptops outright. The conventional Q4_K_XL build, by contrast, needs ≈ 25.6 GB before the first long document is loaded. And these peaks are deliberately the conservative case: the cache is left at FP16, so they hold even before the 4-bit KV cache of the previous section is enabled. Enabling it shrinks the context-dependent term $\approx 4\times$ — the 1-bit build's 100K peak drops to ≈ 6.8 GB and the ternary build's to ≈ 10.1 GB — and brings the full 256K window into the same budgets: ≈ 9.4 GB peak for the 1-bit model and ≈ 12.8 GB for the ternary.

5. Inference: Low-Bit Kernels for Hybrid-Attention Models

A compact weight representation only pays off if a runtime can execute it directly. Existing low-bit inference paths were built for standard full-attention Transformers and do not cover the hybrid-attention blocks of Qwen3.6-27B, whose stack interleaves linear-attention layers with

[1] On-policy baselines use offline KV centering calibrated via the `kv-mean-center` methodology; the class of centering methodology and calibration corpus-extraction script are public (PrismML-Eng/llama.cpp, `tools/kv-mean-center/make-calib-corpus.sh`; discussion in Issue #54).

periodic full-attention layers. Realizing the footprint and bandwidth advantages of Bonsai 27B therefore required building custom 1-bit and 2-bit kernels for this hybrid backbone across each target backend.

The kernels consume the packed binary or ternary weights directly: the sign or ternary codes are unpacked and the group-wise FP16 scales applied *inside* the fused matrix-multiply, so the model is never expanded back into a dense FP16 tensor in memory. Activations, attention operations, and accumulation-sensitive stages are kept in higher precision for numerical stability. We provide these paths on Apple MLX (Python and Swift) for Mac, iPhone, and iPad, and on CUDA for NVIDIA GPUs, with the linear- and full-attention layers handled in a single consistent low-bit execution path.

The gains are concentrated where they matter for on-device use. Token generation is memory-bandwidth-bound — each step must stream the full weight set — so cutting the bytes moved per step yields large speedups there. Prompt processing is comparatively compute-bound, since many tokens are processed in parallel, so it sees only modest gains from the low-bit representation. This is the expected and desirable profile: the interactive, token-by-token phase that dominates perceived latency is exactly the phase that benefits most.

5.1 Cross-Platform Throughput

We report throughput with the standardized `tg128`/`pp512` measurements defined in Appendix D: `tg128` is token-generation throughput over 128 generated tokens and `pp512` is prompt-processing throughput over 512 input tokens, both in tokens per second. On the edge platforms the full-precision baseline (54 GB) and even Q4_K_XL (17.6 GB) do not fit, so the meaningful statement is not a speedup ratio but that a 27B model runs on the device at all.

Table 8. Cross-platform throughput (`tg128`/`pp512`, tokens/s; depth 0, batch size 1, no drafter layers, no vision tower). Apple figures use the llama.cpp Metal backend with the custom low-bit kernels except the iPhone row (MLX Swift). The ternary build ($\sim$7.2 GB) exceeds the $\sim$6 GB per-app iOS budget and is laptop/GPU-only.

Platform	Variant	Footprint	TG128 (tok/s)	PP512 (tok/s)
Laptop (M5 Max)	Binary	3.9 GB	66.4	874
Laptop (M5 Max)	Ternary	7.2 GB	44.0	830
Laptop (M5 Pro)	Binary	3.9 GB	44.2	421
Laptop (M5 Pro)	Ternary	7.2 GB	26.2	393
Laptop (M4 Pro)	Binary	3.9 GB	26.0	133
Laptop (M4 Pro)	Ternary	7.2 GB	18.0	125
iPhone 17 Pro Max (A19 Pro)	Binary	3.9 GB	11.0	111
Single GPU (H100, CUDA)	Binary	3.9 GB	104.8	2755
Single GPU (H100, CUDA)	Ternary	7.2 GB	98	2596

Consistent with the memory-bandwidth analysis above, token generation is where the low-bit representation pays off: at 3.9–7.2 GB the model streams a fraction of the bytes per step that a 17.6–54 GB build would, and where the larger builds cannot reside at all it simply runs. On the M5 Pro the ternary build sustains $\sim$26 tok/s and the binary build $\sim$44 tok/s, scaling to $\sim$44 and $\sim$66 tok/s on the M5 Max; the ternary decode streams $\sim$186 GB/s of weights on the M5 Pro, confirming the path is memory-bandwidth-dominated. The H100 rows are the exception that proves the rule: at batch size 1 a datacenter GPU is limited by kernel-launch and synchronization latency rather than by weight bandwidth, so the binary and ternary variants converge

(104.8 vs. 98 tok/s) despite their $\sim 1.9\times$ difference in bytes per step. Prompt processing, being compute-bound, is not bandwidth-limited: on M5-class laptops it runs several hundred tokens per second, while on M4-class hardware prefill (~ 125–133 tok/s) is the practical limit for very long prompts. On the iPhone 17 Pro Max the binary model generates at ~ 12 tok/s — the first time a 27B-class model generates interactively on a phone at all. The practical takeaway is what these rates enable rather than the numbers alone: interactive, on-device 27B-class generation on an everyday laptop, and on-device generation on a phone.

6. Speculative Decoding: DSpark

Low-bit weights reduce the bytes moved per decoding step; speculative decoding reduces the *number* of target forward passes per generated token. The two are complementary, and Bonsai 27B ships both. We release a **DSpark** [7] drafter trained against the Bonsai 27B target that further accelerates generation with our low-bit models.

6.1 DSpark: Semi-Autoregressive Drafting with Confidence-Scheduled Verification

Speculative decoding is lossless: a lightweight drafter proposes a block of candidate tokens, the target model verifies them in a single forward pass, and rejection sampling accepts the longest prefix consistent with the target's own distribution. Because the acceptance rule preserves that distribution exactly, the accepted tokens are indistinguishable from ordinary generation — the speedup carries no quality cost. This pairs naturally with a low-bit model: each verification pass streams far fewer weight bytes, and its cost is amortized over multiple accepted tokens.

DSpark [7] is a semi-autoregressive drafter. It keeps the expensive draft backbone fully parallel — in the manner of DFlash [11], so drafting latency stays nearly independent of block size — and appends only a lightweight sequential head that injects intra-block token dependencies. That head addresses the characteristic weakness of purely parallel drafters, "suffix decay," in which later positions in a block are predicted independently and are increasingly likely to be rejected. On top of this, a confidence head estimates per-position prefix-survival probabilities, and a hardware-aware scheduler uses them to verify only the tokens with positive expected return — verifying smarter rather than longer, which is what keeps the gains stable under concurrent load. We provide the DSpark drafter layer trained against the Bonsai 27B target so deployments inherit both the footprint win of the low-bit weights and the latency win of longer accepted blocks.

Our drafters follow the DSpark recipe but depart from the original in a few places we found decisive. The block-denoising objective is diffusion-flavored, draft positions are corrupted under an interpolating discrete-diffusion schedule [9] with noise-level conditioning, and the distillation loss is weighted by each position's estimated probability of surviving verification, concentrating training on the tokens that actually extend the accepted prefix. The multi-layer hidden-state taps feeding the drafter are normalized per source before mixing, and the draft block size is chosen from a measured verify-cost model of the serving stack rather than fixed a priori. The drafter itself also ships 4-bit quantized, with verified rollout parity to its bf16 original.

The drafter is a compact six-layer block-parallel transformer conditioned on hidden states tapped from five evenly spaced layers of the target; its drafter-unique weights add roughly 0.5 GB at serving precision (embeddings and output head are shared with the resident target).

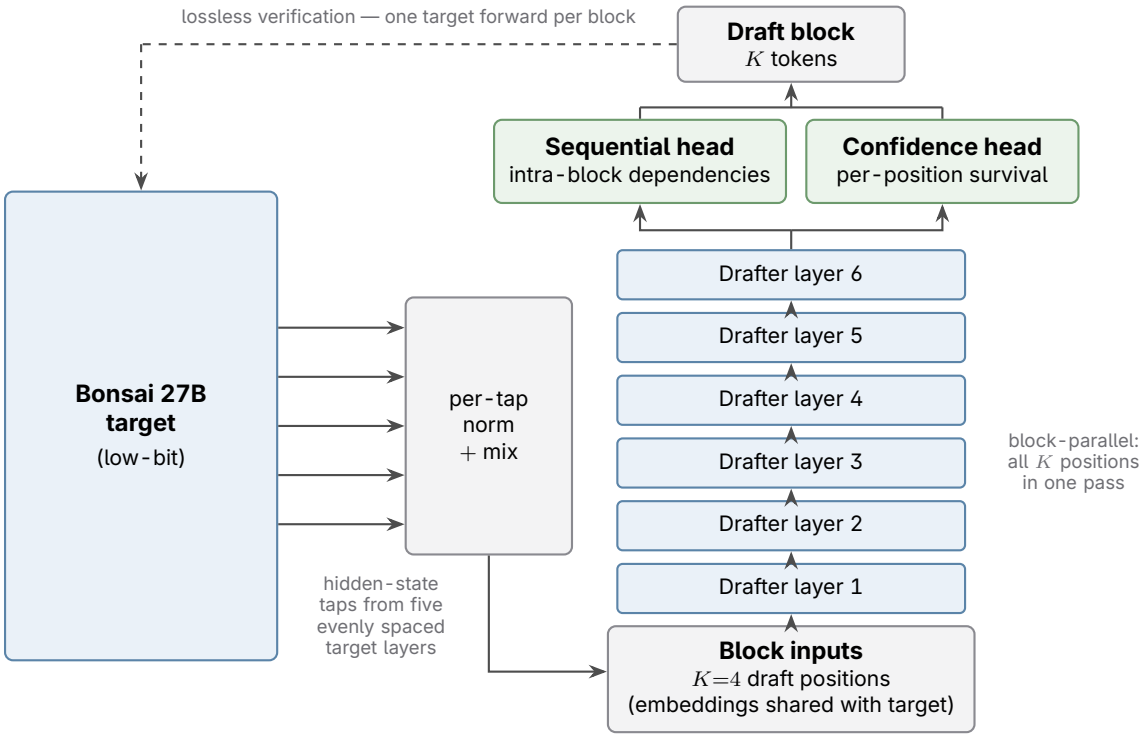

Figure 1. The DSpark drafter used in this release. A compact block-parallel backbone reads normalized hidden-state taps from the target and drafts a block of K=4 tokens in one pass; a lightweight sequential head restores intra-block dependencies and a confidence head estimates per-position survival. Verification by the target is lossless, so accepted tokens are distribution-identical to standard decoding.

6.2 Accepted Length and Decode Speedup

On the CUDA serving path the drafter layers are a measured net win on top of the low-bit kernels. Table 9 reports accepted length and end-to-end decode throughput on an H100 at the throughput-optimal draft depth (k=4); consumer and workstation cards measure comparable or larger gains (e.g. +17% on an RTX PRO 6000 for the binary target). On Apple Silicon the multi-row verification pass does not yet amortize at batch size 1, so the drafter layers are not enabled by default there; making them net-positive on-device is an active kernel target (Section on Limitations).

Table 9. Accepted length (τ) and decode speedup for the DSpark drafter layer on the Bonsai 27B targets (NVIDIA H100, greedy decoding, 24 diverse chat-templated prompts, draft depth k=4). Verification is lossless, so generated text is distribution-identical to running the target alone.

Configuration	Accepted length τ	tok/s	Speedup
Ternary, low-bit kernels only	1.0	98.0	1.0×
Ternary + DSpark	3.7	131.8	**1.34**×
Binary, low-bit kernels only	1.0	104.8	1.0×
Binary + DSpark	3.6	143.8	**1.37**×

Because verification preserves the target distribution exactly, every gain in accepted length is a pure latency win: the generated text is identical in distribution to running the target model alone.

Combined with the custom low-bit kernels of the previous section, the drafter layers turn the footprint reduction into an end-to-end speed advantage on the hardware users actually own.

7. Benchmarks and Intelligence Density

All benchmark evaluations use EvalScope [14] with the vLLM [15] backend on NVIDIA H100 GPUs, under matched infrastructure, decoding, and scoring so that observed differences reflect the representation rather than evaluation artifacts. We average 15 benchmarks spanning knowledge and reasoning, math, coding, instruction following, agentic tool calling, and vision (the full per-benchmark breakdown is in Appendix C). We compare in *thinking* mode throughout, where the model's full reasoning is exercised and the sub-4-bit collapse is most visible.

Table 10 places the two Bonsai models on the same memory-footprint spectrum as the conventional Qwen3.6-27B builds. Alongside the Qwen family it includes Gemma-4-31B, a model of the same capability tier: in full precision, Gemma-4-31B averages 84.58 and Qwen3.6-27B 85.07 on the identical 15-benchmark suite, so the two families start from essentially the same quality. Including Gemma's conventional low-bit builds therefore gives a second, independent reference for how conventional methods trade quality for footprint — and shows that the collapse below 4 bits is a property of the methods, not of one base model.

Table 10. Thinking-mode average (15 benchmarks) across the compression spectrum for Qwen3.6-27B and, for cross-family context, the Gemma-4-31B builds. Bit-widths are true averages; the "vs FP16" column is relative to the Qwen3.6-27B FP16 reference.

Variant	True bpw	Footprint	Thinking avg	vs FP16
Qwen3.6-27B FP16	16.0	54 GB	85.07	100%
Qwen3.6-27B Q4_K_XL ("4-bit")	5.2	17.6 GB	84.99	99.9%
Qwen3.6-27B IQ2_XXS ("2-bit")	2.8	9.4 GB	72.73	85.5%
Gemma-4-31B FP16	16.0	61.5 GB	84.58	99.4%
Gemma-4-31B QAT ("4-bit")	6.0	23.3 GB	83.41	98.0%
Gemma-4-31B Q2_K_XL ("2-bit")	3.0	11.8 GB	73.31	86.2%
Ternary Bonsai 27B	**1.71**	**5.9 GB**	**80.49**	**94.6%**
1-bit Bonsai 27B	**1.125**	**3.9 GB**	**76.11**	**89.5%**

At 5.9 GB, Ternary Bonsai 27B retains 94.6% of the FP16 baseline — a higher score than the conventional IQ2_XXS build (85.5%) at less than two-thirds of its footprint. 1-bit Bonsai 27B retains 89.5% at just 3.9 GB.

The comparison extends across model families. The heaviest conventional Qwen build, Q4_K_XL, does hold 84.99 (99.9% of FP16) — but at 17.6 GB and, despite its "4-bit" label, a true 5.2 bits per weight. The Gemma-4-31B family shows the same structure on a different base model. Its full-precision build scores 84.58 at 61.5 GB. Its QAT build, marketed as 4-bit, is really 6 bits per weight and needs 23.3 GB to hold 83.41. And its most aggressive conventional build, Q2_K_XL — labeled 2-bit, really 3 bits per weight — collapses to 73.31 at 11.8 GB, mirroring the IQ2_XXS collapse on Qwen. Both families lead to the same conclusion: conventional methods preserve quality only at footprints far beyond edge budgets, and break down as they approach the bit-widths where Bonsai operates. Ternary Bonsai 27B, at 5.9 GB, outscores both sub-4-bit conventional builds by more than seven points at one-half to two-thirds of their size.

The aggregate gap also understates *how* the conventional builds fail. Their degradation is se-

lective, concentrated on the benchmarks that demand sustained chains of reasoning: Qwen3.6-27B IQ2_XXS falls to 57.5 on AIME26 and 56.4 on LiveCodeBench while still scoring 88.93 on MMLU-Redux, and Gemma's Q2_K_XL drops to 58.2 on AIME25 and 53.2 on τ^2-Bench. The Bonsai models hold exactly these benchmarks — ternary keeps AIME at 87.5–90.8, and even the 1-bit model stays above 87 — which is the collapse-versus-retention contrast at the heart of this paper. The full per-benchmark results for all eight models are given in Appendix C.

Table 11 breaks the thinking-mode average into the six skill categories, so the retention story can be read per capability rather than in aggregate.

Table 11. Thinking-mode scores by skill category (category averages over the benchmarks listed). Bonsai scores are compared against the FP16 thinking-mode baseline.

Category	Benchmarks	FP16	Ternary 27B	1-bit 27B
Knowledge & reasoning	MMLU-Redux, MuSR	83.15	76.96	73.39
Math	GSM8K, MATH-500, AIME25, AIME26	95.33	93.40	91.66
Coding	HumanEval+, MBPP+, LiveCodeBench	88.74	85.96	81.88
Instruction following	IFEval, IFBench	78.47	71.77	65.74
Agentic / tool calling	BFCL v3, τ^2-Bench	80.00	74.01	66.03
Vision	MMMU-Pro, OCR Bench v2	72.61	65.19	59.57
Overall (15)		**85.07**	**80.49**	**76.11**

Read per capability, the story is one of controlled, predictable trade-offs. The reasoning backbone comes through intact: Ternary Bonsai holds math at 93.40 — within two points of full precision — coding at 85.96, and knowledge and reasoning at 76.96, while even the 1-bit model keeps math at 91.66 and coding at 81.88. That is what keeps multi-step reasoning and agentic workflows dependable at both operating points. The compression naturally asks the most of the most demanding categories — agentic tool use, instruction following, and vision — and this is exactly where the two variants are built to separate. Ternary spends its extra footprint to hold agentic tool use at 74.01 and vision at 65.19; the 1-bit model trades part of that margin (66.03 and 59.57) for the smaller footprint that puts a 27B model on a phone. The family therefore reads as two deliberate operating points rather than one compromise — ternary for laptop-class quality, 1-bit for the most aggressive, phone-class footprint — each keeping the 27B capabilities that matter most for its target device.

7.1 Intelligence Density

Following the framework of the 1-bit Bonsai white paper [1], we define intelligence density as intelligence per unit of storage. We interpret

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

as the model's probability of error, measure intelligence as $-\log_2(P_e)$, and define

$$D = \frac{-\log_2(P_e)}{N}, \tag{1}$$

where N is the model size in GB.

Table 12. Intelligence density (1/GB), thinking mode. Density is computed from Equation (1).

Variant	bpw	Size (GB)	Benchmark Avg	Density (1/GB)
1-bit Bonsai 27B	1.125	3.9	76.11	**0.530**
Ternary Bonsai 27B	1.71	5.9	80.49	**0.400**
Qwen3.6-27B IQ2_XXS	2.8	9.4	72.73	0.199
Gemma-4-31B Q2_K_XL	3.0	11.8	73.31	0.162
Qwen3.6-27B Q4_K_XL	5.2	17.6	84.99	0.155
Gemma-4-31B QAT	6.0	23.3	83.41	0.111
Qwen3.6-27B FP16	16.0	54	85.07	0.051
Gemma-4-31B FP16	16.0	61.5	84.58	0.044

Intelligence density asks not how well a model scores but how much intelligence it delivers per gigabyte, and it is where the Bonsai representation separates most cleanly from the alternatives. Because the metric divides by size, the sharp footprint reduction of the binary and ternary representations is rewarded directly, and because it measures $-\log_2(P_e)$ rather than raw score, it credits keeping error low rather than merely staying afloat. The gap is decisive: 1-bit Bonsai 27B reaches 0.530, roughly $2.7\times$ the densest conventional build (Q2_XXS at 0.199) and over $10\times$ FP16, while Ternary Bonsai 27B reaches 0.400. The pattern holds across families: no conventional build of Qwen3.6-27B or Gemma-4-31B exceeds 0.2, and even Gemma's most aggressive 3-bit build (0.162) delivers under a third of the 1-bit Bonsai figure. Both Bonsai points sit far above the conventional builds, whose density is capped by their much larger footprints. This extends to 27B the density advantage first demonstrated at 8B scale [1]: the advantage is not simply a consequence of being smaller, but of translating each stored gigabyte into far more usable intelligence.

8. Use Cases

The compression achieved by Bonsai 27B opens deployment scenarios that were previously reserved for much smaller models. The through-line is a single idea: 27B-class intelligence can now live where the user and the data already are.

Laptop-local 27B agents. At ~ 7.2 GB deployed, Ternary Bonsai 27B fits comfortably on standard MacBooks and laptops, bringing full 27B reasoning and tool use on-device, with the shipped 262K-token context available for long-document analysis, full-repository code work, and other tasks that depend on holding a large working set. Local execution keeps prompts and outputs on the machine, removes network latency from every turn, and makes iterative, agentic workflows practical without a cloud round-trip per step.

Phone-local 27B reasoning. At ~ 3.9 GB deployed, 1-bit Bonsai 27B fits within the memory budget of a high-end phone such as the iPhone 17 Pro Max — the first time a model of this capability tier runs on a phone at all. The ternary model does not fit a phone, which is precisely why the family offers two operating points: ternary for laptop-class quality, binary for phone-class footprint. And because Bonsai is unusually robust to KV-cache quantization (Section 4.4), phone-local inference is not confined to short, single-turn prompts: the binary model can hold tens of thousands of tokens of multi-turn context on-device, sustaining conversational memory across a session.

Privacy-sensitive and offline settings. On-device execution keeps data on the device by construction — a requirement rather than a preference in enterprise, regulated, and personal-data

workflows where prompts and documents cannot leave the machine. The same profile keeps the model useful with intermittent or no connectivity.

Single-GPU and commodity-GPU serving. Beyond the edge, the reduced footprint lets a 27B model be served from a single consumer or entry-level datacenter GPU, cutting the memory pressure and per-token cost of hosting 27B-class quality while leaving headroom for larger batches, longer contexts, or co-resident models. Combined with the 4-bit KV cache, this makes high-throughput serving and long-context document analysis practical on a single 24 GB GPU — workloads that would otherwise demand multi-GPU infrastructure.

9. Limitations and Roadmap

The quality–footprint trade-off. Ternary and binary retain 94.6% and 89.5% of the full-precision average, and the gap that remains is modest and predictable: as Section 7 shows, the reasoning core — math and coding — stays within a few points of baseline, with the difference concentrated in the most demanding categories. Deployments that need the last few points of accuracy can still reach for the full-precision model where its footprint is not a constraint; the family is designed so that ternary and binary give two clear operating points on the trade-off rather than a single compromise.

Agentic coding, next. Long-horizon, tool-driven software engineering — multi-file edits, run-test-and-repair loops, sustained repository-scale reasoning — is a demanding capability that this release does not yet target strongly. A Bonsai 27B variant tuned for agentic coding is next on the roadmap and will follow shortly.

Native low-bit kernels. Today's ternary deployment stores each value in a 2-bit slot, so its deployed footprint sits above its information-theoretic minimum ($\sim$7.2 GB versus a $\sim$5.9 GB native target). Native 1-bit and ternary kernels are an active engineering target and would return the remaining bandwidth and footprint advantage directly as latency and energy improvements.

Extreme KV compression (sub-2-bit). This release standardizes on a 4-bit KV cache. As Section 4.4 shows, the binary and ternary models tolerate cache discretization with orders of magnitude more margin than dense or conventional 4-bit builds, and that margin *grows* with context length — which points to substantial room below 4 bits. Early results show the key cache can be pushed toward the sub-2-bit regime while remaining stable, opening a path to still longer contexts within a fixed device-memory budget.

Broader hardware and architectures. The Bonsai methodology is architecture-agnostic and not tied to a single model family. Future work will extend these results to additional backends and further model architectures, continuing the pattern established across the Bonsai family.

Drafter improvements. The DSpark layer shipped here is a starting point. The measured speedups in Section 6 are on the CUDA serving path; on Apple Silicon the batch-1 verification pass does not yet amortize. Making the drafter net-positive on-device — together with higher accepted length and better load-aware scheduling — remains an open avenue for further reducing per-token latency on top of the low-bit weights.

A. Deployment and Kernel Reference

Bonsai 27B is released through practical low-bit inference paths on Apple Silicon and CUDA. In both cases the packed binary or ternary weights are consumed directly by the runtime; the model is never expanded into a dense FP16 transformer before inference.

Apple Silicon (MLX). The binary variant uses the MLX 1-bit path and the ternary variant the MLX 2-bit path, with per-group FP16 scales stored alongside the packed weights and applied inside the kernel. The hybrid-attention stack — linear-attention and periodic full-attention layers — is served through a single consistent low-bit execution path on Mac, iPhone, and iPad.

CUDA. On NVIDIA GPUs, fused low-bit GEMM kernels unpack the sign or ternary codes and apply the group-wise scales within the matrix multiply, keeping activations and accumulation-sensitive stages in higher precision.

Vision tower. The multimodal vision tower is served in a 4-bit (HQQ [18]) representation rather than left in full precision, packed in a Q8_0 container for drop-in deployment; a BF16 fallback pack ships alongside it (Section 4.3). This is a deliberate footprint choice and part of the deployed payload accounting.

Speculative-decoding. The DSpark [7] drafter layer is provided as part of the release and integrates with the low-bit target through the standard lossless verification path, so accepted-length gains compound with the footprint reduction. The Q4_1 drafter pack is the default: it drafts faster than the bf16 reference pack at essentially unchanged draft quality — and because verification preserves the target distribution exactly, drafter precision affects only speed, never output quality.

B. Benchmark Evaluation Methodology

All results in this paper come from a single evaluation harness built on EvalScope [14], serving each model through vLLM [15] on an NVIDIA H100 node. Every variant — the full-precision baseline, the conventional low-bit references, and the Bonsai representations — passes through the identical serving path, dataset revisions, and scoring pipeline, so that observed differences reflect the weight representation rather than evaluation artifacts. Text benchmarks are served language-model-only; the two vision benchmarks (MMMU-Pro, OCR Bench v2) are served with the vision tower loaded.

B.1 Thinking Mode and Generation

All results are reported in *thinking* mode. A reasoning parser separates the model's reasoning block from its final answer at the server, so rule-based extractors and judges score only the answer, not the chain of thought. Generation is sampled rather than greedy, with top-p 0.95 and top-k 20 throughout; the temperature follows each model's recommended setting — 1.0 for the Qwen3.6-27B baseline and its conventional quantized variants, and 0.7 for the Bonsai models. Decoding is single-sample except where a benchmark specifies multi-sampling. Output-length caps are set generously so that genuine chain-of-thought fits within budget.

B.2 Token Budgets

Each benchmark has an output-token budget that expands in thinking mode; long-context and long-CoT benchmarks additionally raise the served context window — up to the model's full 262K native window — so that the input and the reasoning trace both fit. The tiers are:

Tier	Thinking-mode cap
Short	16,384
Medium	20,480
Long	30,000
Extended	81,920

B.3 Scoring

Scoring is rule-based wherever possible to minimize scorer variance. Code is scored by sandboxed execution (pass@1 against an augmented hidden test suite); instruction following by programmatic constraint verifiers; tool calling by parsing and state/execution checks; OCR by reference matching. Knowledge, reasoning, and math use a rule-based extractor first, with an LLM judge invoked only when rule-based extraction fails ("rule-first"). The judge is a fixed, capable model (Gemini 3 Flash) — deliberately not a lightweight one, so that verbose answers are not over-credited — and the same judge and configuration is applied to every model.

B.4 Sampling Repeats and Determinism

Most benchmarks are single-pass. AIME25/26 are sampled multiple times to reduce noise on these hard, high-variance tasks, reporting mean accuracy over 8 samples. Instruction-following language checks are seeded for reproducibility; the remaining benchmarks use an unseeded option shuffle, so small run-to-run variance is expected.

B.5 Benchmark Suite and Reported Metrics

Table 13 lists the 15 benchmarks by skill category, with the reported metric, token-budget tier, and scoring regime. A few metric conventions are worth stating explicitly: IFEval reports prompt-level *strict* accuracy and IFBench prompt-level *loose* accuracy (each the standard convention for that benchmark); BFCL v3 is scored over both its single-turn and multi-turn subsets (the full task set, not the single-turn-only subset used in earlier Bonsai reports); OCR Bench v2 reports the English aggregate; and τ^2-Bench is run in an Artificial-Analysis-style single-pass setup.

Table 13. The 15-benchmark launch suite: metric, token-budget tier (thinking mode), and scoring regime. "rule-first" = rule-based extraction with an LLM-judge fallback on parse failure.

Benchmark	Metric	Budget	Scoring
Knowledge & reasoning			
MMLU-Redux	MCQ acc	Short	rule-first
MuSR	MCQ acc	Short	rule-first
Math			
GSM8K	numeric acc	Short	rule-first
MATH-500	boxed acc	Extended	judge
AIME25	integer acc	Extended	judge
AIME26	integer acc	Extended	judge
Coding (execution-based)			
HumanEval+	pass@1	Long	rule
MBPP+	pass@1	Medium	rule
LiveCodeBench	pass@1	Extended	rule
Instruction following			
IFEval	prompt-strict	Medium	rule
IFBench	prompt-loose	Medium	rule
Agentic / tool calling			
BFCL v3	tool-call acc	Medium†	rule
τ^2-Bench	task success	Medium	rule
Vision (served with the vision tower)			
MMMU-Pro	VQA acc	Medium	judge
OCR Bench v2	OCR acc (EN)	Long	rule

† Medium *output* budget; the multi-turn subsets raise the served context window to fit accumulated tool responses.

C. All Benchmark Results

Table 14 reports the full per-benchmark scores (thinking mode) behind the category averages of Section 7, for all eight evaluated models — the Qwen3.6-27B family (FP16, Q4_K_XL, IQ2_XXS), the Gemma-4-31B family (FP16, 4-bit QAT, Q2_K_XL), and the two Bonsai models — grouped into the six skill categories. Averages are the unweighted mean over all 15 benchmarks.

Table 14. Full per-benchmark results (%), thinking mode, for all eight evaluated models. Sizes are language-model weights in decimal GB; bit-widths are true averages over all weights.

	Qwen3.6-27B			Gemma-4-31B			Bonsai 27B	
Benchmark	FP16	Q4_K_XL	IQ2_XXS	FP16	QAT	Q2_K_XL	**Ternary**	**1-bit**
Size (GB)	54	17.6	9.4	61.5	23.3	11.8	**5.9**	**3.9**
Bits/weight	16.0	5.2	2.8	16.0	6.0	3.0	**1.71**	**1.125**
Knowledge & reasoning								
MMLU-Redux	93.42	93.35	88.93	93.60	93.44	90.70	88.05	82.75
MuSR	72.88	73.01	66.99	71.03	69.58	64.70	65.87	64.02
Math								
GSM8K	95.30	96.66	89.90	97.57	97.57	94.90	96.06	92.80
MATH-500	99.40	99.40	84.60	99.60	99.60	95.20	99.20	98.00
AIME25	93.29	92.91	66.67	86.66	83.75	58.20	90.84	88.75
AIME26	93.33	93.00	57.50	88.33	83.75	59.80	87.50	87.08
Coding								
HumanEval+	95.12	95.73	91.46	96.34	94.51	90.70	93.90	89.63
MBPP+	83.33	83.86	78.89	84.39	84.13	77.80	81.22	79.60
LiveCodeBench	87.77	87.96	56.40	89.76	88.25	74.40	82.75	76.40
Instruction following								
IFEval	88.91	88.83	84.03	90.57	91.87	86.50	85.03	79.11
IFBench (prompt-loose)	68.03	65.65	53.76	79.30	79.60	48.60	58.50	52.36
Agentic / tool calling								
BFCL v3	77.10	75.05	70.28	74.71	73.65	71.50	74.41	70.72
τ^2-Bench	82.90	82.27	74.58	72.80	68.20	53.20	73.61	61.34
Vision								
MMMU-Pro	79.94	81.73	65.19	79.80	79.50	74.40	68.96	60.48
OCR Bench v2	65.28	65.45	61.70	64.30	63.70	59.10	61.42	58.65
Average (15)	**85.07**	**84.99**	**72.73**	**84.58**	**83.41**	**73.31**	**80.49**	**76.11**

The per-benchmark detail sharpens the aggregate story on reasoning. The conventional sub-4-bit builds do not degrade uniformly; they collapse selectively on the benchmarks that demand sustained chains of reasoning. IQ2_XXS falls to 57.5 on AIME26 and 56.4 on LiveCodeBench (from 93.33 and 87.77 at FP16), and Gemma's Q2_K_XL shows the same signature on a different base model (58.2 on AIME25, 53.2 on τ^2-Bench). The Bonsai models, at one-half to one-third of those footprints, hold exactly these benchmarks: the ternary model keeps AIME at 87.5–90.8 and LiveCodeBench at 82.8, and even the 1-bit model keeps AIME above 87. Shorter-form benchmarks mask the difference — IQ2_XXS still scores 88.93 on MMLU-Redux and 84.03 on IFEval — which is why casual testing misses the collapse, and why this paper states its central claim over long-form reasoning, tool use, and agentic behavior rather than aggregate fluency.

D. Throughput and Energy Measurement Notes

Throughput is reported with standardized `tg128`/`pp512` measurements: `tg128` is token-generation throughput over 128 generated tokens (the memory-bandwidth-bound, interactive phase), and `pp512` is prompt-processing throughput over 512 input tokens (the compute-bound phase). Both are in tokens per second, and each configuration is run repeatedly with the mean reported to reduce variance. On phones, decode is additionally run in a repeated (palindrome) order to separate the cold/burst peak from the thermally-sustained rate; sustained generation on a phone is thermally limited.

Per-platform harnesses. CUDA throughput uses the standard llama.cpp/CUDA benchmarking path; Apple laptop figures use `llama-bench` on the Metal backend (with the custom low-bit kernels); on-device iPhone figures use the MLX Swift runtime.

Energy. Power is measured with platform-native instrumentation — IOReport across CPU/GPU/ANE/DRAM on Mac, `nvidia-smi` on NVIDIA GPUs, and Xcode Power Profiler with battery-drain estimation on iPhone — with an idle baseline subtracted to isolate model execution. Energy per token is then $E = P/(3.6\,r)$ in mWh, for measured power P (W) and rate r (tok/s). iPhone figures are approximate estimates rather than hardware-metered.

Table 15. Measured decode energy (`tg128`) per token. Idle power is subtracted at the source; GPU figures via `nvidia-smi`, Mac via IOReport.

Variant	Hardware	TG128 (W)	TG128 (tok/s)	Energy (mWh/tok)
Binary (llama.cpp Q1_0)				
Binary + DSpark	M5 Pro	44.6	45.1	**0.275**
Binary	RTX 4090	308.8	122.8	0.699
Binary + DSpark	RTX 4090	308.8	108.5	0.791
Binary	RTX 5090	454.4	162.6	0.776
Binary + DSpark	RTX 5090	454.4	165.2	0.764
Binary	RTX PRO 6000	453.1	146.2	0.861
Binary + DSpark	RTX PRO 6000	453.1	187.3	0.672
Binary	L40S	303.4	88.0	0.958
Binary + DSpark	L40S	303.4	134.5	**0.627**
Binary	H100	341.4	104.8	0.905
Binary + DSpark	H100	341.4	143.8	0.659
Binary	A100	268.7	92.0	0.811
Ternary (llama.cpp Q2_0)				
Ternary	RTX PRO 6000	448.9	121.6	1.025
Ternary + DSpark	RTX PRO 6000	448.9	180.7	**0.690**
Ternary	RTX 5090	453.7	133.6	0.943
Ternary + DSpark	RTX 5090	453.7	159.2	0.792
Ternary	H100	364.6	98.0	1.033
Ternary + DSpark	H100	364.6	131.8	0.768
Ternary	L40S	306.9	64.5	1.322
Ternary + DSpark	L40S	306.9	114.2	0.746
Ternary	RTX 4090	301.8	90.9	0.922
Ternary + DSpark	RTX 4090	301.8	103.6	0.809
Ternary	A100	273.8	88.5	0.859

Two observations. First, the on-device operating point is an order of magnitude more energy-efficient per token than any datacenter card: the M5 Pro decodes the binary model at 0.275 mWh/token, versus 0.63–1.32 mWh/token across six GPU classes — local inference is not just private and low-latency but cheap in energy. Second, prefill is far cheaper than decode per token everywhere (0.012–0.060 mWh/token across the same GPUs, e.g. 0.017 on the RTX PRO 6000 and 0.035–0.038 on H100), because prompt processing is compute-bound and amortizes weight traffic across many tokens.

Table 16. iPhone 17 Pro Max battery-drain result, 1-bit Bonsai 27B (easymlx, on-device). Measured as battery percentage consumed rather than instrumented wattage, so it is reported in native units rather than converted to mWh/token.

Metric	Value
Tokens per 1% battery	672 tok/1%
Sustained decode	10.82 tok/s
Tokens generated	3,360
Battery drop	100% → 95% (5%)
Wall time	5.2 min
Thermal	nominal → fair (mild throttle)

At 672 tokens per 1% of battery, a full charge projects to roughly 67,000 tokens of sustained generation on-device, with the sustained decode rate (10.82 tok/s) close to the batch-1 `tg128` figure of Table 8 and a mild thermal throttle emerging over the 5.2-minute run.

References

[1] PrismML. 1-bit Bonsai 8B: End-to-end 1-bit language model deployment. Technical report, PrismML, March 2026.

[2] PrismML. Ternary Bonsai: Ternary (1.58-bit) language models at 8B, 4B, and 1.7B scale. Technical report, PrismML, April 2026.

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

[4] Qwen Team. Qwen3.6-27B-FP8. `https://huggingface.co/Qwen/Qwen3.6-27B-FP8`, 2026.

[5] 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.

[6] 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.

[7] X. Cheng, X. Yu, C. Shao, J. Li, Y. Xiong, et al. DSpark: Confidence-scheduled speculative decoding with semi-autoregressive generation. DeepSeek-AI, 2026.

[8] DeepSeek-AI. DeepSeek-V3 Technical Report (Multi-Token Prediction). *arXiv preprint arXiv:2412.19437*, 2024.

[9] D. von Rütte, J. Fluri, Y. Ding, A. Orvieto, B. Schölkopf, and T. Hofmann. Generalized interpolating discrete diffusion. *arXiv preprint arXiv:2503.04482*, 2025.

[10] F. Gloeckle, B. Youbi Idrissi, B. Rozière, D. Lopez-Paz, and G. Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of ICML*, 2024.

[11] J. Chen, Y. Liang, and Z. Liu. DFlash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

[12] 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. *arXiv preprint arXiv:2305.20050*, 2023. (Introduces the MATH-500 evaluation subset.)

[13] Y. Kuratov, A. Bulatov, P. Anokhin, I. Rodkin, D. Sorokin, A. Sorokin, and M. Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Proceedings of NeurIPS Datasets and Benchmarks*, 2024.

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

[15] 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.

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

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

[18] H. Badri and A. Shaji. Half-Quadratic Quantization of large machine learning models. Mobius Labs, November 2023. https://mobiusml.github.io/hqq_blog/; code at https://github.com/mobiusml/hqq.

[19] H. Badri and A. Shaji. Gemlite: Towards building custom low-bit fused CUDA kernels. Mobius Labs, 2024. https://github.com/mobiusml/gemlite.

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