1-bit and Ternary Bonsai Image 4B

Image generation with Binary or Ternary transformer weights
via CUDA and Apple Silicon inference backends



PrismML · May 2026

Contents

1 Executive Summary **3**

2 Efficiency as the Defining Constraint in Diffusion Deployment **5**
2.1 Inference Defines the Economics of Image Generation 5
2.2 Why Weight Movement Matters 5
2.3 Why Low-Bit Representation Has Remained Out of Reach for Diffusion 6

3 What Makes PrismML Different **6**

4 Bonsai Image 4B Family **7**
4.1 Binary Weight Format 9
4.2 Ternary Weight Format 10
4.3 Storage Footprint 10
4.4 Inference Optimizations — Megakernel Developments 11
4.5 Throughput Performance 12

5 Benchmarks **14**

6 Use Cases **17**

7 Limitations and Roadmap **17**

A Deployment and Runtime Reference **19**

B Benchmark Evaluation Methodology **20**

C Throughput and Memory Measurements **20**
C.1 Measurement Setup 20
C.2 Throughput Results 21
C.3 Memory-Pressure Proxy 22

D Storage Breakdown **24**

Category	Summary
Model	Bonsai Image 4B family — sub-2-bit FLUX.2 Klein 4B diffusion transformer [1]; ternary and binary variants released in parallel
Format	Binary g128: $\{-1, +1\}$ weights, or Ternary g128: $\{-1, 0, +1\}$ weights, both with FP16 group-wise scaling
Deployment	Apple MLX 1-bit (binary) / 2-bit (ternary) kernel paths on Apple Silicon; Gemlite low-bit GEMM on GPU [9];
Value	0.93 GB and 1.21 GB for binary and ternary models, resulting $6.4 - 8.3\times$ smaller memory footprint; $\sim$9.4 s image generation on iPhone 17 Pro Max at 512×512, 3–$6\times$ speedup on Mac M4 Pro
Use case	On-device, privacy-preserving, low-latency image generation on real hardware
License	Apache 2.0 for weights and code; FLUX.2 Klein 4B is released by Black Forest Labs under Apache 2.0 [3]

1. Executive Summary

This whitepaper introduces Bonsai Image 4B, a pair of low-bit diffusion transformer deployments built from FLUX.2 Klein 4B. The 1-bit Bonsai Image 4B uses binary weights in $\{-1, +1\}$, while the Ternary Bonsai Image 4B uses ternary weights in $\{-1, 0, +1\}$; both use group-wise FP16 scaling (for groups of size 128). To our knowledge, these are the very first binary and ternary diffusion transformers released at the scale of a modern 4B-parameter image-generation model.

The central result is that 4B-class diffusion transformers can be moved into a substantially smaller and more deployable representation without changing the model architecture, sampler, or text-conditioning pathway. The original FLUX.2 Klein 4B diffusion transformer is 7.75 GB in FP16. In 1-bit Bonsai Image 4B all the weights except for the projection layers are converted to 1-bit, reducing the transformer to 0.93 GB, an $8.3\times$ model-level reduction. Correspondingly, Ternary Bonsai Image 4B reduces it to 1.21 GB, a $6.4\times$ reduction.

The total Apple Silicon deployment payloads are 3.42 GB for 1-bit Bonsai Image 4B and 3.88 GB for Ternary Bonsai Image 4B, including the packed diffusion transformer, compressed text encoder, and FP16 VAE. These are a $4.7\times$ and $4.1\times$ reductions in the total payload compared to the 15.97 GB required of the full precision FLUX.2 Klein 4B. Since, at runtime, the text encoder is offloaded after prompt encoding, the mean memory usage is smaller than the total payload. When generating a 512×512 image, the mean-active memory is 1.5 GB and 1.96 GB, for the binary and ternary models, compared to 11.74 GB for the original FLUX.2 Klein 4B (a reduction of $7.8\times$ and $6.0\times$, respectively). For a 1024×1024 image, the mean-active memory is 1.95 GB and 2.38 GB, for the binary and ternary models, compared to 14.39 GB for the original FLUX.2 Klein 4B (a reduction of $7.4\times$ and $6.0\times$, respectively).

The above reductions directly ease memory pressure during the repeated denoising loop, which is the deployment axis that matters for laptops, phones, and other unified-memory systems. Instead of moving a multi-gigabyte transformer through memory at every step, the Bonsai representations keep the repeated denoising core compact enough for practical local inference.

These footprint reductions translate into practical local deployment. On iPhone 17 Pro Max, the full FP16 FLUX.2 Klein 4B pipeline does not fit within the device memory budget, while both Bonsai variants run on-device. 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B generate 512×512 images in 9.4 s and 1024×1024 images in 32.6–34.0 s using four denoising steps. On Mac M4 Pro, the deployed Bonsai stack reaches $5.60\times$ speedup both for 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B over the stock MFLUX [4] FP16 pipeline at 1024×1024 resolution.

The two variants allow a quality/footprint trade-off. 1-bit Bonsai Image 4B is the most compact option, designed to minimize footprint and bandwidth. Ternary Bonsai Image 4B uses a slightly larger ternary weight alphabet, adding a zero state that improves visual quality and prompt fidelity. Across three standard image-generation benchmarks, GenEval, HPSv3, and DPG-Bench, Ternary Bonsai Image 4B retains an average of 94.4% of the FLUX.2 Klein 4B score, while 1-bit Bonsai Image 4B retains an average of 88.4%. This positions Ternary Bonsai Image 4B for quality-oriented deployment and 1-bit Bonsai Image 4B for footprint-oriented deployment.

This release extends PrismML's low-bit representation thesis beyond language models. Prior Bonsai releases showed that binary and ternary weight representations can preserve practical capability while sharply reducing footprint and bandwidth. Bonsai Image 4B brings the same design space to diffusion transformers, where inference is especially sensitive to repeated weight movement across denoising steps.

The result is not just a smaller checkpoint, but a deployable image-generation system: binary and ternary transformer weights, compact storage, fused low-bit GEMM kernels on CUDA, MLX kernel paths on Apple Silicon, and local execution on hardware that previously could not run this class of model. These runtimes already expose the value of binary and ternary representations on today's hardware, while leaving further upside to be realized for native sub-2-bit kernels.

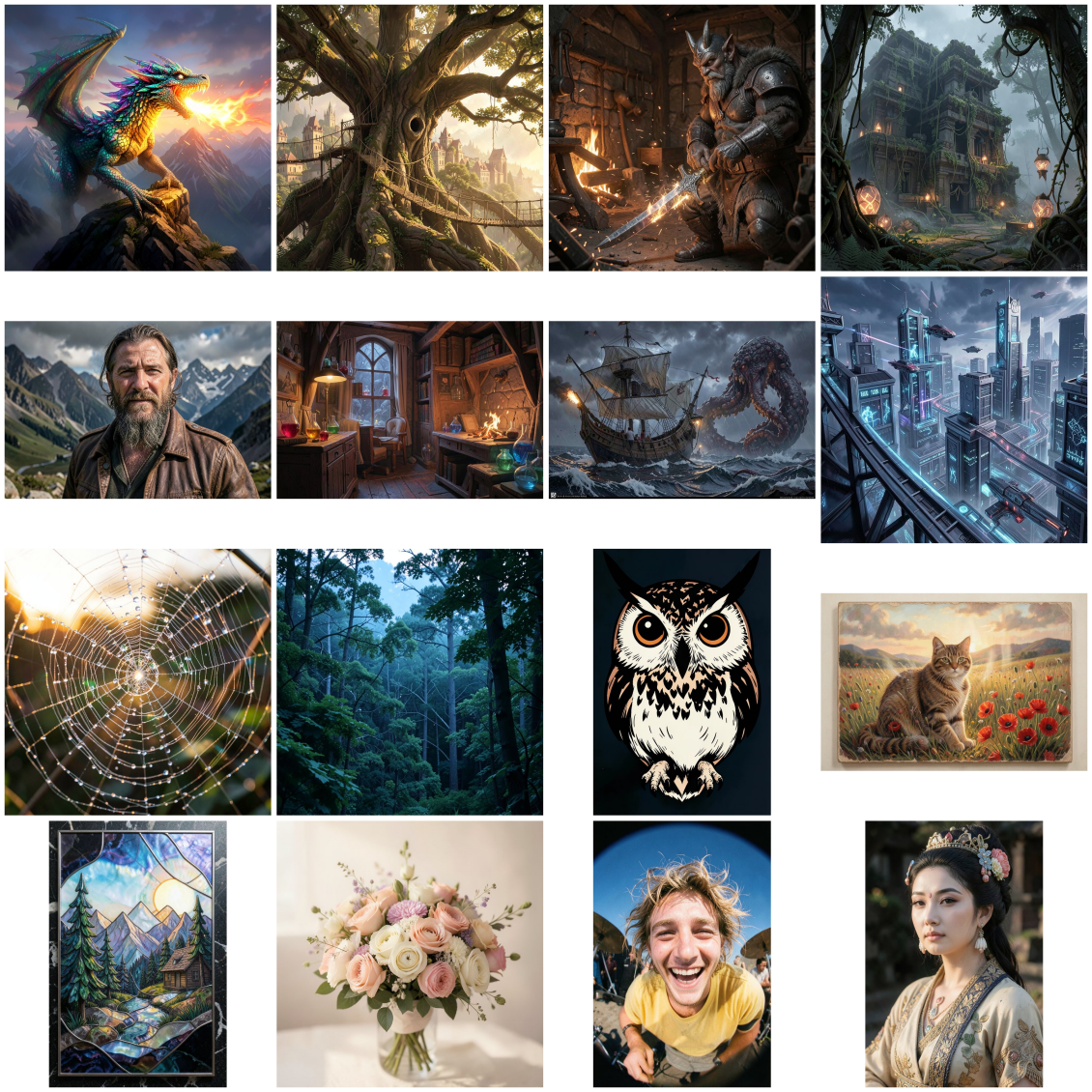

Figure 1. Sample outputs from 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B at four denoising steps, mixed resolutions (512×512, 1024×1024, 832×1248, 1248×832).

2. Efficiency as the Defining Constraint in Diffusion Deployment

Diffusion models make deployment efficiency a first-order constraint because generation is iterative. A diffusion model does not produce an image with a single forward pass; it refines a latent through a sequence of denoising steps, and each step invokes the diffusion transformer again. For language models, deployment efficiency is often experienced as tokens per second. For diffusion models, the analogous quantities are seconds per denoising step and seconds per image.

This changes how image-generation systems should be evaluated. The question is not only whether a model can produce high-quality images under ideal server conditions. It is whether that quality can be delivered repeatedly, at interactive latency, within the memory and bandwidth limits of the hardware on which the product is expected to run.

2.1 Inference Defines the Economics of Image Generation

A single generated image requires N forward passes through the diffusion transformer, where N is the sampling step count. At each step, the model's matrix weights are accessed and applied to the current latent state. The inference cost therefore scales with the number of images, the number of sampling steps per image, and the cost of moving and applying the transformer weights at each step.

This matters in both local and server deployments. On local devices, the cost appears directly as user-facing latency and memory pressure: every additional second per step makes the generation loop feel less interactive. On servers, the same cost determines model residency, accelerator memory pressure, concurrent throughput, and serving cost per generated image. A model that produces strong images but depends on large-memory infrastructure or slow per-step execution is constrained in the products it can support.

For this reason, progress in image generation cannot be measured only by image quality at maximum precision. The relevant deployment metric is quality delivered per second, per gigabyte of model footprint, and per unit of memory bandwidth. A practical image model must translate visual capability into a form that fits the memory, latency, and bandwidth constraints of the hardware on which it runs.

2.2 Why Weight Movement Matters

At interactive batch sizes, diffusion-transformer inference is often shaped by memory movement as much as by arithmetic throughput. Each denoising step must access the transformer's large matrix weights, and the same operation repeats across the sampling trajectory. As a result, the amount of weight data that must remain resident and be streamed through memory becomes a central determinant of per-step latency.

This is why the representation of the transformer weights matters. Reducing the weight footprint does not merely make the checkpoint smaller on-disk. It reduces the recurring memory burden of the denoising loop. When the transformer is the repeated compute core of the pipeline, a more compact weight representation changes the memory budget of every denoising step.

This also separates transformer representation from the rest of the image-generation pipeline. The text encoder, sampler, diffusion transformer, and VAE all contribute to end-to-end generation, but only the diffusion transformer is repeatedly invoked across the denoising trajectory. Compressing that core changes the recurring cost of generation without requiring a different sampler, text encoder, or VAE.

2.3 Why Low-Bit Representation Has Remained Out of Reach for Diffusion

Moving diffusion models into low-bit representations has historically been difficult because image-generation failures are both trajectory-dependent and perceptual. The denoising network is evaluated across a changing sequence of noise levels, so activation statistics and error sensitivity vary across the sampling process. A model with compact representation can remain close on aggregate metrics while visibly desaturating colors, flattening textures, losing fine detail, degrading typography, or weakening prompt faithfulness. In image generation, small numerical changes are not only measured; they are seen.

For this reason, most practical low-bit diffusion work has remained in the 8-bit, 6-bit, 4-bit, or mixed-precision regime. Q-Diffusion showed that standard post-training quantization does not work out of the box for diffusion models, and introduced timestep-aware calibration and split shortcut quantization to make 4-bit diffusion quantization viable [11]. That result was an important step, but it also highlighted the central problem: diffusion models require diffusion-specific treatment even before reaching the binary or ternary regime.

More recent systems push 4-bit diffusion deployment further. SVDQuant, for example, targets 4-bit weights and activations for large diffusion models by absorbing sensitive outlier directions into a high-precision low-rank branch while quantizing the residual branch [12]. This preserves quality better than direct 4-bit quantization, but it is not a binary or ternary transformer representation. It also demonstrates that compression alone is not enough: when the low-rank branch is executed naively, the extra activation movement can negate the speedup, which is why SVDQuant requires the Nunchaku fused inference engine to recover practical latency gains [13]. SVDQuant is therefore best understood as a strong 4-bit deployment system, not as evidence that binary or ternary diffusion deployment has been solved.

Attempts below 4 bits have often exposed a different limitation. They may rely on heavily engineered training recipes, altered network structure, restricted model families, reduced-resolution settings, or research-only inference paths. Such systems are valuable as research results, but they do not establish the deployment claim that matters here: preserving a modern image generator while making its repeated denoising core small enough for practical local execution.

The key question is therefore not only whether a diffusion model can be made smaller. It is whether the model can remain recognizably the same generator after its transformer weights are moved into a binary or ternary representation. A smaller checkpoint is not enough. The representation must preserve visual quality, prompt fidelity, and aesthetic behavior while also reducing repeated weight movement and mapping to real inference kernels. The next section describes how Bonsai Image 4B is built around that standard.

3. What Makes PrismML Different

PrismML's approach is to preserve the model while changing the representation in which it is deployed. The Bonsai Image 4B family keeps the FLUX.2 Klein 4B diffusion transformer intact: the layer structure, transformer backbone, text-conditioning pathway, sampler interface, and VAE interface are not replaced. The starting point remains a modern 4B-class text-to-image diffusion transformer; the central change is how its dominant weight matrices are represented and executed.

This distinction matters because many efficiency gains in image generation come from changing the model itself. Smaller backbones, distilled students, fewer-step generators, and specialized architectures can all improve deployment efficiency, but they also change the object being deployed. Bonsai takes a different route: it preserves the architecture and changes the deployed weight alphabet.

The Bonsai Image 4B release provides two complementary compact-weight variants. 1-bit

Bonsai Image 4B uses weights in $\{-1, +1\}$ and targets the most aggressive footprint and bandwidth reduction. Ternary Bonsai Image 4B uses weights in $\{-1, 0, +1\}$, adding a zero state that improves image quality and prompt fidelity while preserving a compact representation. Both variants use group-wise FP16 scaling so that the discrete weights retain local magnitude information while preserving a regular layout for deployment.

The representation is intentionally simple and inspectable. Rather than relying on a smaller architecture, a high-precision auxiliary branch, or a collection of model-specific exceptions, Bonsai applies binary or ternary structure to the matrix-heavy layers that dominate the diffusion transformer. A small set of precision-sensitive projections is retained in FP16 to preserve end-to-end image quality, while the repeated transformer core is served in a compact low-bit format by real inference kernels.

This follows the pattern established by the Bonsai language-model releases. 1-bit Bonsai showed that binary-weight LLMs can move an 8B-class model into a radically smaller and faster deployment regime across Apple and CUDA runtimes. Ternary Bonsai showed that adding the zero state improves the quality–efficiency trade-off while preserving the compact footprint that makes low-bit deployment practical [14, 15]. Bonsai Image 4B extends that thesis from language to image generation: the same binary and ternary design space applies to a modern 4B-class diffusion transformer without changing the backbone, sampler, or text-conditioning pathway.

The next section gives the concrete system specification: layer coverage, binary and ternary weight formats, storage footprint, and backend performance.

4. Bonsai Image 4B Family

Bonsai Image 4B is a pair of sub-2-bit deployments of FLUX.2 Klein 4B [1]. 1-bit Bonsai Image 4B uses binary transformer weights in $\{-1, +1\}$, while Ternary Bonsai Image 4B uses ternary transformer weights in $\{-1, 0, +1\}$. Both variants use shared group-wise FP16 scales. FLUX.2 Klein 4B is the open 4B member of Black Forest Labs' FLUX.2 Klein family: a compact rectified-flow DiT designed for fast text-to-image generation and image editing. Black Forest Labs positions the 4B variant as the most accessible FLUX.2 Klein model, released under Apache 2.0 and intended to run on consumer GPUs while retaining the generation capabilities of the FLUX.2 family.

This makes FLUX.2 Klein 4B a natural target for low-bit deployment. It is already a modern, compact image model rather than a toy architecture, yet its FP16 transformer still imposes a multi-gigabyte memory and bandwidth burden during denoising. The Bonsai Image 4B family keeps the base diffusion transformer architecture intact and changes the deployed representation of its transformer weights. The supporting text encoder and VAE are treated as pipeline components: the text encoder is compressed for practical inference, while the VAE remains FP16 for decoder stability.

Table 1. System specification for the Bonsai Image 4B family. Both variants share the same FLUX.2 Klein 4B architecture and differ only in the binary or ternary representation used for the DiT weights.

Item	Specification
Base model	FLUX.2 Klein 4B [1], rectified-flow diffusion transformer (DiT)
Parameters	∼4.0 B diffusion transformer; 25 transformer blocks (5 double-stream + 20 single-stream)
Architecture	Double + single transformer block stack; 25 total blocks (5 double-stream + 20 single-stream); GQA attention; RMSNorm; SwiGLU-style MLP
Text encoder	Qwen3-4B, HQQ INT4 quantized for inference [10]
VAE	FLUX.2 decoder in FP16 [2]
Weight format of DiT	Binary g128: $\{-1, +1\}$ weights, or Ternary g128: $\{-1, 0, +1\}$ weights, both with FP16 group-wise scaling
Backends	Apple MLX 1-bit (binary) / 2-bit (ternary) kernel paths on Apple Silicon; Gemlite low-bit GEMM on CUDA [9]
License	Apache 2.0 for weights and code

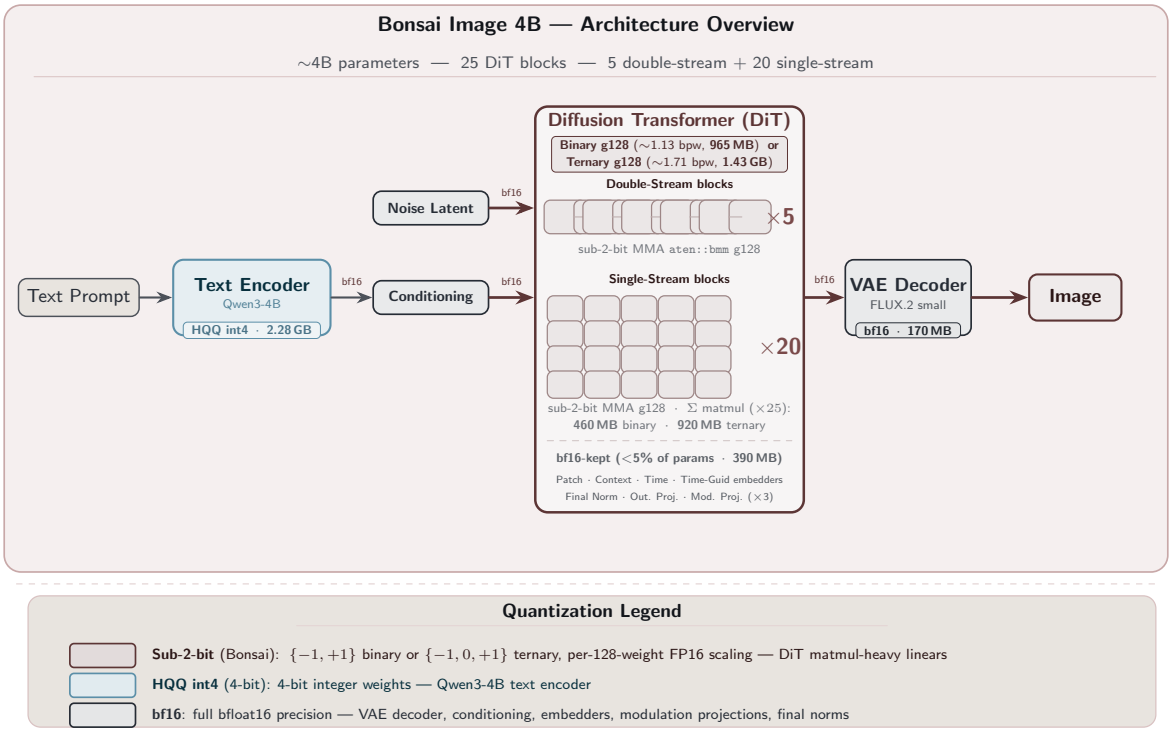

Figure 2. Bonsai Image 4B family pipeline overview. The DiT matrix-heavy layers use either binary $\{-1, +1\}$ or ternary $\{-1, 0, +1\}$ weights with per-128-weight FP16 scales. The Qwen3-4B text encoder uses HQQ INT4 weights, while the VAE and precision-sensitive auxiliary projections remain FP16.

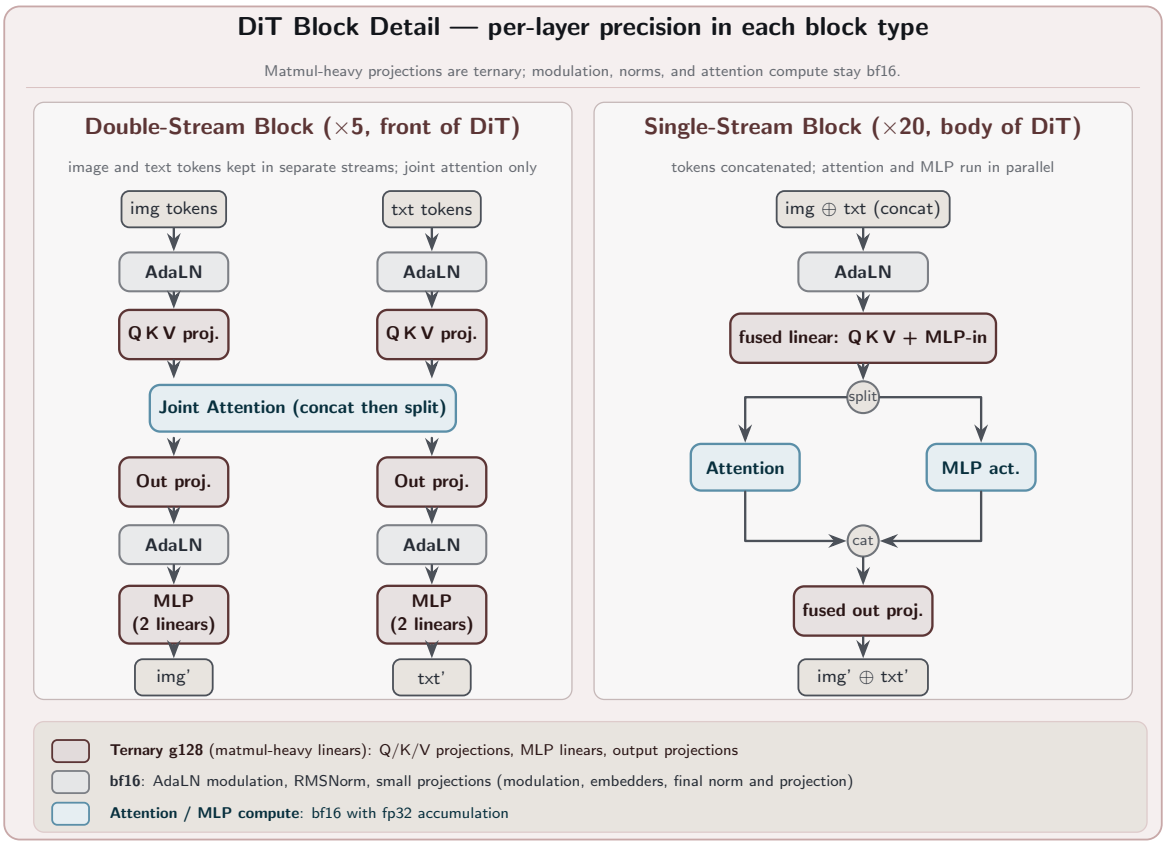

Figure 3. Per-layer representation inside the double-stream and single-stream DiT blocks. Matrix-heavy linears, including Q/K/V projections, output projections, and MLP weights, use the binary or ternary g128 representation. AdaLN modulation, RMSNorms, and small precision-sensitive projections remain FP16. Block structure follows the FLUX architecture described in *Demystifying Flux Architecture* [21].

4.1 Binary Weight Format

As in our earlier 1-bit Bonsai language-model release [14], 1-bit Bonsai Image 4B uses a group-wise binary weight format. Each transformer weight takes a value from $\{-1, +1\}$, with one shared FP16 scale for each group of 128 weights. The effective weight is

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

where s_g is the shared scale for group g.

Binary values carry exactly 1 bit of information per weight. With one FP16 scale per group of 128 weights, the effective storage cost is

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

which yields an idealized raw-weight compression of $16/1.125 \approx 14.2\times$ relative to FP16, before container overhead and alignment. This is the most aggressive compression point in the Bonsai Image 4B family: it minimizes the transformer weight footprint and the repeated memory traffic incurred at every denoising step.

The binary format is applied uniformly to the matrix-heavy diffusion transformer weights: attention projections, output projections, and MLP projections. A small set of precision-sensitive

auxiliary projections is kept at FP16 (`proj_out`, `x_embedder`, `context_embedder`, `time_text_embed`, `time_guidance_embed`, `norm_out`, and the three modulation streams). Supporting pipeline components are treated separately: the text encoder is compressed with HQQ INT4 for practical inference, while the VAE remains FP16 for decoder stability.

Native single-bit GEMM kernels for the binary pack on CUDA and Apple Silicon remain an active backend target; the current deployment paths are described in Appendix A.

4.2 Ternary Weight Format

Alongside the binary deployment, we release a ternary variant that uses a slightly larger weight alphabet to improve representational flexibility and image quality while preserving a compact deployment format. As in our earlier Ternary Bonsai language-model release [15], Ternary Bonsai Image 4B uses weights in $\{-1, 0, +1\}$, with one shared FP16 scale for each group of 128 weights. The effective weight is

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

where s_g is the shared scale for group g.

Ternary values carry $\log_2 3 \approx 1.585$ bits of information per weight. With one FP16 scale per group of 128 weights, the effective storage cost is approximately

$$b_{\text{eff}} \approx \log_2 3 + \frac{16}{128} \approx 1.585 + 0.125 \approx 1.71 \text{ bits/weight},$$

which yields an idealized raw-weight compression of $16/1.71 \approx 9.4\times$ relative to FP16, before container overhead and alignment.

Relative to 1-bit Bonsai Image 4B, the additional zero state gives Ternary Bonsai Image 4B a more expressive weight alphabet and allows better preservation of model behavior under extreme compression. In this sense, the binary and ternary variants define two deployment operating points: binary maximizes compression and bandwidth reduction, while ternary recovers more of the full-precision model's visual quality and prompt fidelity while remaining an extremely compact representation.

The ternary format is applied to the same set of diffusion transformer weight tensors as the binary format: attention projections, output projections, and MLP projections. The same precision-sensitive auxiliary projections are kept at FP16, the text encoder is compressed with HQQ INT4 for practical inference, and the VAE remains FP16 for decoder stability. The following section reports transformer-level storage separately from the larger deployment payload, which also includes the compressed text encoder and FP16 VAE.

At inference time, the packed binary or ternary weights are consumed directly by fused low-bit GEMM kernels on CUDA and MLX. The transformer is not expanded back into a dense FP16 weight matrix; packed weights, group-wise scales, and FP16 activations are combined inside the kernel path. Figure 4 in Appendix A gives the full attention-layer compute flow.

4.3 Storage Footprint

The storage footprint is reported at two levels: the diffusion transformer itself and the total deployment payload. The transformer-level number isolates the repeated denoising core of FLUX.2 Klein 4B, which is the component represented with binary or ternary weights. The payload-level number includes the packed diffusion transformer, the compressed text encoder, and the FP16 VAE.

In both 1-bit and Ternary Bonsai Image 4B, the matrix-heavy transformer layers store discrete binary or ternary weights with per-group FP16 scales. A small set of precision-sensitive auxiliary projections, less than 5% of transformer parameters, is retained in FP16 for stability and image quality. Appendix D reports the exact per-component byte breakdown.

The bits-per-weight numbers reported below differ from the matmul-only figures in the previous two subsections (1.125 and 1.71 bpw) because they amortize the cost of the FP16-kept auxiliary tail across the whole transformer rather than reporting only the cost of the low-bit matmul linears.

For 1-bit Bonsai Image 4B, the effective transformer storage averages approximately **1.925 bpw**, giving an **8.3× model-level reduction relative to FP16**. The FP16 transformer occupies 7.75 GB; the Bonsai binary representation reduces this to 0.93 GB. The deployed packed sizes are slightly larger at 0.97 GB for MLX and 1.08 GB for CUDA, corresponding to 7.2–$8.0\times$ effective compression in the runtime formats.

For Ternary Bonsai Image 4B, the effective transformer storage averages approximately **2.51 bpw**, giving a **6.4× model-level reduction relative to FP16**. The same 7.75 GB FP16 transformer is reduced to 1.21 GB in the Bonsai ternary representation. The deployed packed sizes are 1.43 GB for MLX and 1.54 GB for CUDA, corresponding to 5.0–$5.4\times$ effective compression in the runtime formats.

The difference between the Bonsai representation size and the deployed packed size is primarily a kernel-format artifact rather than a model-level cost. Current low-bit kernels use hardware-friendly packed layouts: ternary values are stored in 2-bit slots, with the fourth code unused, while binary values use the available 1-bit or packed low-bit path in each backend. Native binary and native ternary kernels would close more of this gap. Until then, the reported MLX and CUDA sizes are the relevant deployment footprints for today's inference paths.

Table 2 reports both transformer-level sizes and total deployment payloads. We separate these because the diffusion transformer is the repeated denoising component targeted by the Bonsai binary and ternary representations, while the text encoder and VAE are supporting pipeline components. The total payload includes the packed diffusion transformer, the compressed text encoder, and the FP16 VAE; it should not be interpreted as steady-state resident memory during denoising, since the text encoder is offloaded after prompt encoding.

Table 2. Storage footprint. **Bonsai** reports the model-level representation size: binary or ternary weights, g128 FP16 scales, and precision-sensitive FP16 auxiliaries. **MLX** and **CUDA** report on-disk packed deployment sizes for Apple Silicon and CUDA runtimes. Total payload includes the diffusion transformer, compressed text encoder, and FP16 VAE. All sizes are decimal GB (10^9 bytes).

Component	FP16	Bonsai	Reduction	MLX	CUDA
1-bit Bonsai Image 4B	7.75 GB	0.93 GB	8.3×	0.97 GB	1.08 GB
Ternary Bonsai Image 4B	7.75 GB	1.21 GB	6.4×	1.43 GB	1.54 GB
Text encoder (Qwen3-4B)	8.05 GB	2.28 GB	3.5×	2.28 GB	2.84 GB
VAE	0.17 GB	0.17 GB	1.0×	0.17 GB	0.17 GB
Total payload (1-bit Bonsai Image 4B)	15.97 GB	3.38 GB	4.7×	3.42 GB	4.09 GB
Total payload (Ternary Bonsai Image 4B)	15.97 GB	3.66 GB	4.4×	3.88 GB	4.55 GB

Appendix D gives the byte-level decomposition of the deployed transformer packs.

4.4 Inference Optimizations — Megakernel Developments

The compact weight representation is one half of the deployed system; the other half is the runtime that keeps the rest of the pipeline from reintroducing large memory overheads. On Apple Silicon, diffusion inference can pay substantial memory cost outside the matrix multiplication itself. In the stock inference path, each transformer block launches separate Metal kernels for normalization, RoPE rotation, Q/K/V projection, and surrounding arithmetic, materi-

alizing intermediate tensors between stages. These allocations are small for a single block, but they accumulate across blocks and across the repeated denoising loop.

The Bonsai Image 4B runtime is designed to reduce this memory burden at three levels. First, pipeline-level choices reduce the non-transformer working set: the text encoder is evicted from device memory after prompt encoding, and VAE decoding is tiled when resolution makes the decoder memory-bound. These choices remove large transient allocations from the image-generation pipeline before the denoising loop begins.

Second, the binary and ternary transformer representations reduce the resident weight footprint of the repeated denoising core. Instead of keeping a multi-gigabyte FP16 transformer active throughout sampling, 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B keep the matrix-heavy transformer layers in compact binary or ternary form with group-wise FP16 scales. This is the largest memory win in the Bonsai deployment stack because it acts on the component invoked at every denoising step.

Third, the `klein_fast` megakernel reduces intermediate activation materialization inside the transformer itself. It fuses LayerNorm, RoPE rotation, and the QKV projection into a single Metal compute dispatch per transformer block. Activations flow directly from normalization to positional rotation to projection within one kernel launch, instead of being repeatedly written to and read from temporary buffers between separate kernels.

These three pieces act on different sources of memory pressure: the surrounding pipeline, the resident transformer weights, and the intermediate activations inside each attention block. On Mac M4 Pro at 1024×1024, the stock FP16 path has a mean-active memory footprint of 14.39 GB. The Bonsai runtime reduces this to 7.84 GB through pipeline-level changes such as text-encoder eviction and VAE tiling. Adding the binary and ternary transformer representations brings the **mean-active memory pressure down to 1.95 GB for 1-bit Bonsai Image 4B and 2.38 GB for Ternary Bonsai Image 4B**, corresponding to a **$7.4\times$ reduction for binary and a $6.0\times$ reduction for ternary relative to the stock FP16 path**.

The megakernel is a smaller but complementary refinement within this memory stack. Its role is not to replace the gains from compact weights, but to reduce avoidable activation movement and dispatch overhead inside the repeated transformer core. Because the fused stages are representation-agnostic, the same runtime path applies to FP16, 1-bit Bonsai Image 4B, and Ternary Bonsai Image 4B. Full memory-pressure measurements, including the proxy definition, resolution breakdown, and VAE tiling behavior, are reported in Appendix C.3.

4.5 Throughput Performance

The throughput measurements in this section use the full Bonsai deployment stack described in Section 4.4, including the binary and ternary weight packs, pipeline-level memory optimizations, and the `klein_fast` megakernel. The goal is to measure the deployed system end to end: seconds per denoising step and seconds per generated image on the hardware users actually own.

The most important deployment result is on iPhone: the full FP16 FLUX.2 Klein 4B pipeline does not fit within the device's unified-memory budget, whereas both 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B run on-device. This changes the role of low-bit deployment from an optimization to an enabler. The binary and ternary variants make a modern 4B-class image generator executable on mobile hardware, with end-to-end latency of 9.4 s at 512×512 and 32.6–34.0 s at 1024×1024 on iPhone 17 Pro Max using four denoising steps.

Table 3. iPhone 17 Pro Max throughput for the deployed Bonsai MLX pipelines. All rows use four denoising steps, guidance $= 1.0$, and cache-warmed timing. The FP16 FLUX.2 Klein 4B pipeline is omitted because it does not fit on device.

Model	Backend	Resolution	s/step	s/image
1-bit Bonsai Image 4B	MLX 1-bit	128×128	0.68	2.7
Ternary Bonsai Image 4B	MLX 2-bit	128×128	0.68	2.7
1-bit Bonsai Image 4B	MLX 1-bit	256×256	0.95	3.8
Ternary Bonsai Image 4B	MLX 2-bit	256×256	1.00	4.0
1-bit Bonsai Image 4B	MLX 1-bit	512×512	2.35	9.4
Ternary Bonsai Image 4B	MLX 2-bit	512×512	2.35	9.4
1-bit Bonsai Image 4B	MLX 1-bit	1024×1024	8.15	32.6
Ternary Bonsai Image 4B	MLX 2-bit	1024×1024	8.50	34.0

The iPhone results show that the two low-bit variants deliver similar wall-clock throughput. At 512×512, the two variants are identical at 9.4 s. At 1024×1024, 1-bit Bonsai Image 4B runs about 4% faster than Ternary Bonsai Image 4B (32.6 s vs. 34.0 s), which is small relative to the measurement noise expected on mobile hardware. The deployment choice on iPhone is therefore primarily a storage–quality trade-off rather than a speed trade-off: 1-bit Bonsai Image 4B minimizes deployed weight footprint, while Ternary Bonsai Image 4B uses a slightly larger representation to achieve improved visual behavior.

On Mac M4 Pro, where an FP16 baseline can be measured, the same low-bit deployments deliver large end-to-end speedups relative to the standard MFLUX FP16 pipeline. At 1024×1024, 1-bit Bonsai Image 4B generates an image in 24.07 s and Ternary Bonsai Image 4B in 24.26 s, compared with 134.91 s for the FP16 MFLUX pipeline. This corresponds to roughly $5.6\times$ speedup.

Table 4. Mac M4 Pro throughput with matched prompts and seeds. All rows use four denoising steps and guidance $= 1.0$. Speedup is measured against the MFLUX FP16 pipeline at the same resolution.

Model	Backend	Resolution	s/step	s/image	Speedup
1-bit Bonsai Image 4B	MLX 1-bit	512×512	1.50	6.01 ± 0.31	3.03×
Ternary Bonsai Image 4B	MLX 2-bit	512×512	1.44	5.78 ± 0.08	3.15×
FLUX.2 Klein 4B FP16 (MFLUX)	FP16	512×512	4.55	18.22 ± 3.00	1.0×
1-bit Bonsai Image 4B	MLX 1-bit	1024×1024	6.02	24.07 ± 0.03	5.60×
Ternary Bonsai Image 4B	MLX 2-bit	1024×1024	6.06	24.26 ± 0.24	5.56×
FLUX.2 Klein 4B FP16 (MFLUX)	FP16	1024×1024	33.73	134.91 ± 15.48	1.0×

On CUDA, the Gemlite low-bit GEMM path provides a second deployment route across data-center and consumer GPUs. Table 5 reports warmed wall-clock time per image across four platforms using the same prompts, sampler, and step count as the Apple Silicon measurements.

Table 5. CUDA throughput across GPU tiers, warmed wall-clock per image (four denoising steps, guidance $= 1.0$). Colab rows use the platform's default driver and CUDA toolkit; Linux and Windows rows use locally built Gemlite kernels. RTX 3080 1024×1024 peak HBM: 6.4 GiB (binary), 6.8 GiB (ternary).

Platform	512×512		1024×1024	
	Binary	Ternary	Binary	Ternary
A100 (Colab)	1.0 s	1.1 s	2.7 s	2.8 s
RTX PRO 6000 Blackwell (Colab)	1.0 s	1.0 s	1.8 s	2.1 s
RTX 3080 10 GB	1.5 s	1.4 s	4.5 s	4.5 s
RTX 3060 6 GB (laptop)	4.4 s	3.3 s	24.8 s	17.5 s

The CUDA results show that the Bonsai deployments run across a wide range of GPU tiers. A100 and RTX PRO 6000 Blackwell Colab instances generate 1024×1024 images in under 3 s, while the RTX 3080 10 GB generates in 4.5 s for both variants. The RTX 3060 6 GB laptop is the slowest target and the only platform where the binary and ternary variants diverge noticeably at 1024×1024. This result should be interpreted as backend- and memory-regime dependent rather than as a general property of the binary and ternary representations.

The Mac results provide the controlled FP16 comparison, while the iPhone results demonstrate the stronger deployment claim. On Mac, the Bonsai stack turns the standard MFLUX FP16 pipeline into a substantially faster local workload while retaining similar wall-clock latency between the binary and ternary variants. On iPhone, the FP16 pipeline is not a viable deployment target; the binary and ternary variants are what make local execution possible. Further measurement details are provided in Appendix C.

5. Benchmarks

We evaluate 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B against public image-generation models spanning modern diffusion transformers, larger rectified-flow systems, and smaller diffusion backbones. The comparison set includes 1-bit Bonsai Image 4B, Ternary Bonsai Image 4B, FLUX.2 Klein 4B, FLUX.1-schnell [16], SDXL [18], Stable Diffusion 1.5 [19], BK-SDM-Small [20], and PixArt-Σ XL 2 [17].

Before presenting benchmark results, we first fix the size convention used throughout this section. Because Bonsai targets the repeated denoising network, we report the size of the main generative model: the diffusion transformer or UNet-like denoising backbone. Text encoders and VAE components are excluded from the benchmark size column. This focuses the quality–footprint comparison on the component repeatedly executed during sampling and directly changed by the binary and ternary Bonsai representations.

Table 6 summarizes the native FP16 model sizes used to define the main-model size column in the benchmark tables.

Table 6. Native FP16 model sizes used for benchmark size comparisons. "Main model" denotes the repeated denoising backbone. Text encoders and VAE components are shown for context but excluded from the benchmark size column.

Model	Main Model	Text Encoder(s)	VAE	Total Memory
FLUX.1-schnell [16]	23.8 GB	9.7 GB (T5 + CLIP L)	168 MB	∼33.7 GB
FLUX.2 Klein 4B [1]	7.75 GB	6.2 GB (Qwen + CLIP)	168 MB	∼14.1 GB
PixArt-Σ XL 2 [17]	1.20 GB	9.2 GB (Flan-T5-XXL)	167 MB	∼10.6 GB
SDXL 1.0 [18]	5.14 GB	1.64 GB (OpenCLIP + CLIP L)	167 MB	∼7.0 GB
Stable Diffusion 1.5 [19]	1.72 GB	246 MB (CLIP L)	167 MB	∼2.1 GB
BK-SDM-Small [20]	0.98 GB	246 MB (CLIP L)	167 MB	∼1.4 GB

With this size convention fixed, we evaluate image quality and prompt faithfulness across three complementary benchmark suites: GenEval for object composition and attribute binding [6], HPSv3 for human-preference alignment and aesthetic quality [8], and DPG-Bench for prompt following and semantic faithfulness [7]. GenEval is run at its official 512×512 protocol for all rows. For HPSv3 and DPG-Bench, FLUX.2 Klein, FLUX.1 schnell, SDXL, and the Bonsai variants are evaluated at 1024×1024, while the smaller-backbone comparison models (BK-SDM-Small, Stable Diffusion 1.5, and PixArt-Σ XL 2) are evaluated at their native 512×512 resolution. The corresponding benchmark results are shown in Tables 7, 8, and 9.

Table 7. GenEval object-composition benchmark [6]. "Main model size" reports the denoising backbone size: deployed Bonsai transformer size for 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B, and native FP16 main-model size for public baselines. All rows use GenEval's official 512×512 protocol.

Model	Main Model	Single	Two	Count.	Colors	Pos.	Color A.	Overall
1-bit Bonsai Image 4B	**0.93 GB**	0.990	0.720	0.590	0.850	0.420	0.520	**0.671**
Ternary Bonsai Image 4B	**1.21 GB**	0.988	0.849	0.625	0.862	0.470	0.590	**0.723**
FLUX.2 Klein 4B [1]	7.75 GB	0.988	0.929	0.825	0.904	0.620	0.690	0.819
FLUX.1-schnell [16]	23.8 GB	1.000	0.919	0.725	0.777	0.320	0.620	0.716
SDXL [18]	5.14 GB	0.825	0.303	0.050	0.575	0.040	0.080	0.300
BK-SDM-Small [20]	0.98 GB	0.800	0.253	0.163	0.585	0.020	0.050	0.297
Stable Diffusion 1.5 [19]	1.72 GB	0.988	0.364	0.325	0.723	0.040	0.060	0.396
PixArt-Σ XL 2 [17]	1.20 GB	1.000	0.596	0.450	0.840	0.130	0.320	0.541

Table 8. HPSv3 human-preference benchmark [8]. "Main model size" reports the denoising backbone size. Bonsai rows use deployed binary or ternary transformer sizes; public baselines use native FP16 main-model sizes.

Model	Main Model	Anime	Concept Art	Paintings	Photo	Average
1-bit Bonsai Image 4B	**0.93 GB**	10.75	12.28	12.82	8.77	**11.15**
Ternary Bonsai Image 4B	**1.21 GB**	11.64	13.01	13.20	11.03	**12.22**
FLUX.2 Klein 4B [1]	7.75 GB	12.28	13.42	13.85	11.82	12.84
FLUX.1-schnell [16]	23.8 GB	12.23	13.42	13.58	11.45	12.67
SDXL [18]	5.14 GB	10.05	10.92	11.32	7.92	10.05
BK-SDM-Small [20]	0.98 GB	2.06	4.27	5.16	0.71	3.05
Stable Diffusion 1.5 [19]	1.72 GB	3.29	4.03	4.80	4.67	4.20
PixArt-Σ XL 2 [17]	1.20 GB	11.82	12.69	12.73	10.50	11.93

Table 9. DPG-Bench prompt-following benchmark [7]. "Main model size" reports the denoising backbone size. Bonsai rows use deployed binary or ternary transformer sizes; public baselines use native FP16 main-model sizes.

Model	Main Model	Score
1-bit Bonsai Image 4B	**0.93 GB**	**0.822**
Ternary Bonsai Image 4B	**1.21 GB**	**0.851**
FLUX.2 Klein 4B [1]	7.75 GB	0.853
FLUX.1-schnell [16]	23.8 GB	0.848
SDXL [18]	5.14 GB	0.740
BK-SDM-Small [20]	0.98 GB	0.559
Stable Diffusion 1.5 [19]	1.72 GB	0.601
PixArt-Σ XL 2 [17]	1.20 GB	0.769

Across the three benchmark suites, the Bonsai variants show the expected quality–deployment trade-off. Ternary Bonsai Image 4B is the quality-oriented deployment point: at 1.21 GB, it retains 88.3% of the FLUX.2 Klein 4B score on GenEval, 95.2% on HPSv3, and 99.8% on DPG-Bench, for an average retention of 94.4% across the three benchmarks. 1-bit Bonsai Image 4B is the footprint-oriented deployment point: at 0.93 GB, it reduces the main-model footprint by $8.3\times$ while retaining 81.9% on GenEval, 86.8% on HPSv3, and 96.4% on DPG-Bench, for an average retention of 88.4%.

Relative to public baselines, this produces a strong quality-density profile. 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B are $8.3\times$ and $6.4\times$ smaller than FLUX.2 Klein 4B, respectively, and roughly $26\times$ and $20\times$ smaller than FLUX.1-schnell. Despite these reductions, both variants remain close to the larger models on preference and prompt-following metrics. Compared with smaller diffusion backbones, both Bonsai variants deliver substantially higher quality at similar or smaller main-model sizes.

This is the central benchmark result: 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B are not only smaller checkpoints. They achieve similar behavior to a modern 4B-class diffusion transformer while moving the repeated denoising backbone into a deployment footprint that is practical for local and memory-constrained inference.

6. Use Cases

1-bit Bonsai Image 4B and Ternary Bonsai Image 4B are designed for image-generation settings where deployment constraints determine the product experience. The most direct use case is local creative tooling on MacBook-class systems, tablets, and phones. When the diffusion transformer fits within the local memory budget, generation can happen closer to the user: prompts, reference images, and generated outputs can remain on-device, and interaction is not mediated by network latency, remote queue times, or cloud availability.

This matters because image generation is rarely a single request. Users iterate: they adjust prompts, compare variations, regenerate details, and explore visual directions before selecting a final output. Local deployment changes the economics of that workflow. Once the model is on the device or workstation, users can explore and discard generations freely, without every variation becoming a separate cloud request or metered credit event.

The same deployment profile is useful for offline asset generation and rapid prompt exploration. Dataset construction, product mockups, design studies, marketing assets, and internal visual review often require many small image jobs rather than one final image. A compact local model makes these workloads easier to run continuously, privately, and at lower cost on hardware the user already controls.

Enterprise and privacy-sensitive workflows are another natural fit. Many organizations cannot freely send prompts, reference images, brand assets, product concepts, or customer data to external image-generation services. Local or controlled deployment allows generative image models to be used in settings where data residency, confidentiality, compliance, cost control, or auditability are first-order requirements.

Commodity-GPU serving is also an important target. Reducing the transformer footprint improves model residency, lowers memory pressure, and makes it easier to serve multiple workloads on the same accelerator. In this setting, the benefit is not only smaller storage; it is more efficient use of GPU memory and memory bandwidth during the repeated denoising steps that define diffusion inference. Across replicated serving workloads, the per-model footprint reduction can translate into higher concurrency and better accelerator utilization.

Finally, compact diffusion models expand the design space for mobile and embedded creative interfaces. Phones, tablets, robots, cameras, and edge systems impose hard limits on memory, thermals, power, and connectivity. 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B are steps toward image generation that can run where creative context is produced, rather than only where large-memory infrastructure is available.

7. Limitations and Roadmap

1-bit Bonsai Image 4B and Ternary Bonsai Image 4B are designed to deliver behavior similar to FLUX.2 Klein 4B while moving its diffusion transformer into a much smaller deployment representation. Because the weight representation changes, the right evaluation standard is not bit-identical reproduction of the full-precision model. The relevant standard is deployment-faithful behavior: benchmark retention, side-by-side qualitative comparison, and workflow-specific testing under matched sampler, resolution, step count, and guidance settings.

The main quality risks are the same areas where diffusion models are already sensitive: small text, fine object details, high-frequency texture, object counts, unusual lighting, saturated color palettes, and subtle prompt-following constraints. These cases can be difficult to summarize with a single aggregate metric, so visual inspection remains important alongside quantitative benchmarks. In this release, Ternary Bonsai Image 4B is the quality-oriented deployment point, while 1-bit Bonsai Image 4B is the footprint-oriented deployment point.

The Bonsai variants also inherit the generation regime of FLUX.2 Klein 4B. Preserving the architecture makes the binary and ternary models directly comparable to the base model, but deployment teams should still evaluate them on their own prompt distributions, especially for workflows that depend on exact typography, product-specific details, or strict compositional constraints. This is not unique to Bonsai; it is the normal deployment requirement for image-generation systems whose failures are often perceptual and task-specific.

There is also a systems opportunity. Current commodity inference stacks do not yet expose fully native binary and ternary execution as standard hardware paths. This release therefore uses practical low-bit kernel paths on CUDA and Apple Silicon that consume the packed transformer weights directly and avoid expanding them back to FP16 during inference. These paths make the models deployable on today's hardware, but they do not exhaust the long-term opportunity. Native binary and ternary kernels should allow more of the theoretical bandwidth and memory-footprint advantage to appear directly as latency and energy improvements.

As the diffusion transformer becomes smaller, other parts of the image pipeline become more visible bottlenecks. In particular, the VAE is difficult to compress aggressively without affecting reconstruction quality, and its activations can dominate memory pressure after the text encoder and diffusion transformer have been moved into compact representations. Tiled VAE decoding mitigates this issue in the current deployment stack, but it is still a workaround around a larger architectural constraint: high-resolution image generation remains tied to a decoder whose activation footprint can be large on edge devices.

This points to an important direction for future work. Further gains will not come only from smaller diffusion-transformer weights; they will also require rethinking the surrounding image-generation pipeline. This includes better VAE execution strategies, more representation-friendly decoders, and eventually image-generation systems that reduce or remove dependence on heavyweight VAEs for edge deployment.

The roadmap follows from these observations. First, we will continue improving native binary and ternary inference support across hardware backends so that deployed runtimes capture more of the bandwidth, latency, and energy gains implied by the compact weight representations. Second, we will extend the Bonsai methodology beyond the diffusion transformer itself, including better execution strategies for VAEs and other non-DiT components that become bottlenecks once the main model has been compressed.

Together, these directions point toward the broader goal of the Bonsai Image family: delivering behavior similar to modern image generators while moving them into deployment regimes where memory, bandwidth, energy, and local execution are the primary constraints.

A. Deployment and Runtime Reference

The Bonsai Image 4B family is released through practical low-bit inference paths for CUDA and Apple Silicon. In both cases, the packed binary or ternary transformer weights are consumed directly by the runtime; the model is not expanded back into a dense FP16 transformer before inference.

Low-Bit Inference Backends

CUDA. CUDA inference uses Gemlite low-bit GEMM kernels [9]. The binary variant uses a packed 1-bit representation for $\{-1, +1\}$ weights, and the ternary variant uses a packed 2-bit representation for $\{-1, 0, +1\}$ weights. Group-wise FP16 scales are applied inside the fused matmul path. Activations, attention operations, and accumulation-sensitive stages remain in FP16 or fp32 accumulation as appropriate.

Apple Silicon. Apple Silicon inference uses MLX low-bit kernel paths. The binary variant uses the MLX 1-bit path, while the ternary variant uses the MLX 2-bit path. Per-group FP16 scales are stored alongside the packed weights and applied inside the kernel. The `klein_fast` megakernel described in Section 4.4 is layered on top of this runtime path for LayerNorm, RoPE, and QKV fusion.

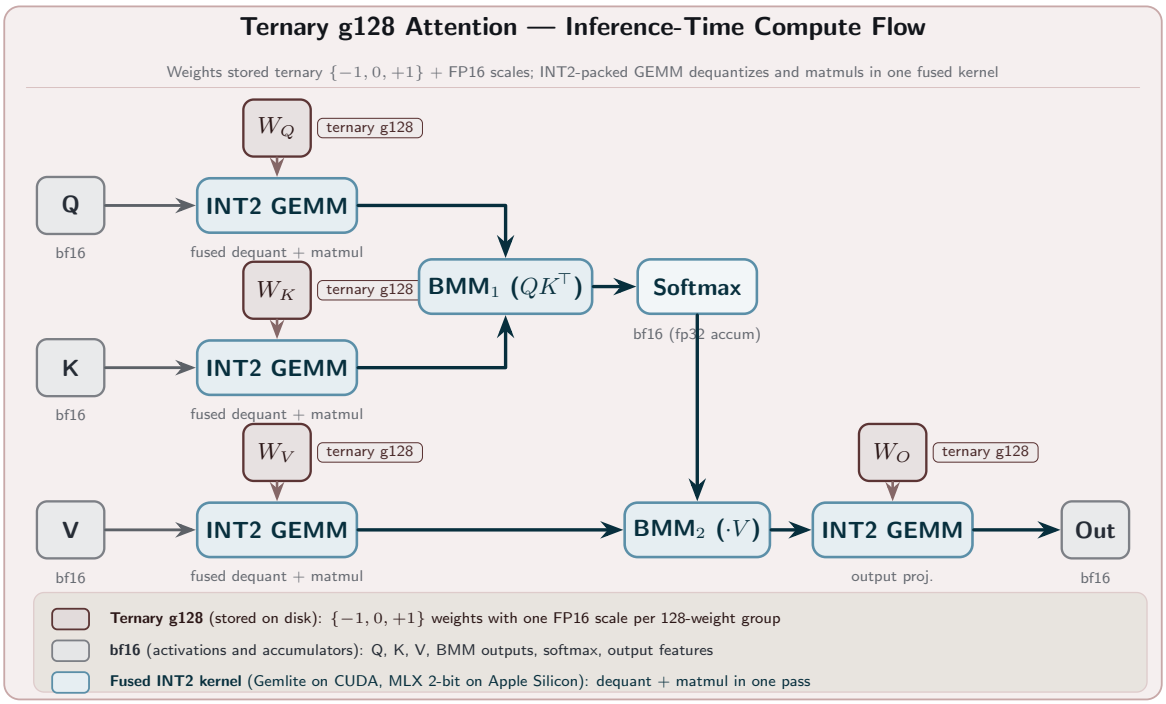

Figure 4. Inference-time compute flow for a low-bit attention layer in the Bonsai Image 4B family. The ternary configuration is shown; the binary variant follows the same flow with 1-bit packed storage. Packed weights are consumed directly by fused low-bit GEMM kernels without expanding the transformer back to dense FP16 weights.

Pipeline Runtime

The deployed runtime includes additional pipeline-level optimizations beyond the low-bit transformer path. The text encoder is used for prompt encoding and then evicted from device memory before denoising. The VAE decoder can be run in a tiled mode when resolution or memory pressure makes full-frame decoding impractical. On Apple Silicon, the `klein_fast` megakernel fuses LayerNorm, RoPE, and QKV projection into a single Metal dispatch inside each transformer block.

Tiled VAE decoding is used when the decoder activation footprint becomes the dominant transient memory cost. Tiling bounds the VAE working set by tile size rather than full image resolution, making higher-

resolution decoding feasible on memory-constrained devices. At lower resolutions, non-tiled decoding may remain faster because it avoids tile-stitch overhead, so the runtime selects the path based on the resolution and memory regime.

Released Artifacts

The released packages include separate binary and ternary weight packs together with the reference runtime [5] used for the measurements in this paper. The runtime includes the low-bit transformer path, text-encoder eviction, tiled VAE decoding, and the Apple Silicon megakernel. Native binary and native ternary kernels remain an important backend target; the current MLX and Gemlite paths are the practical deployment formats used for the reported results.

Out-of-Scope Use

- In any way that violates applicable law.
- For the purpose of exploiting, harming, or attempting to exploit or harm minors in any way; including but not limited to the solicitation, creation, acquisition, or dissemination of child exploitative content.
- To generate or disseminate deceptive, fraudulent, misleading, or otherwise harmful content.
- To generate or disseminate personally identifiable information that can be used to harm an individual.
- To harass, abuse, threaten, stalk, or bully individuals or groups of individuals.
- To create non-consensual intimate imagery or illegal pornographic content.
- For fully automated decision-making or high-risk applications that adversely impact an individual's legal rights or otherwise create or modify a binding, enforceable obligation.

B. Benchmark Evaluation Methodology

All quality benchmarks are run with matched generation settings wherever the benchmark and model pipeline permit. The 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B arms use the same sampler, step count, guidance value, prompt set, and seed schedule as the corresponding FLUX.2 Klein 4B comparison. The Bonsai variants use the deployed binary or ternary transformer representation, the compressed Qwen3-4B text encoder, and the FP16 VAE decoder.

We report three complementary benchmark suites. GenEval measures object composition and attribute binding through six category scores (single object, two object, counting, colors,position, color attribution) and an overall mean across the 553-prompt evaluation set [6]. HPSv3 measures per-style preference scores (anime, concept art, paintings, photo) and reports the unweighted average [8]. DPG-Bench measures dense prompt-following and semantic faithfulness across its 1065 dense-prompt evaluation set [7].

GenEval is evaluated at its official 512×512 protocol for all models. For HPSv3 and DPG-Bench, the larger-backbone models are evaluated at 1024×1024, while the smaller-backbone comparison models are evaluated at their native 512×512 setting. This mirrors the convention used in Section 5: quality is compared against the main denoising model size, excluding text encoders and VAE components.

C. Throughput and Memory Measurements

C.1 Measurement Setup

All throughput measurements use FlowMatch-Euler, four denoising steps, guidance $= 1.0$, shift $= 3.0$, a fixed seed, and matched prompts across compared variants. Latency is measured as wall-clock time for image generation under the deployed pipeline used by each variant. End-to-end timing includes prompt encoding, denoising, and VAE decoding where applicable. When seconds per denoising step are reported, they are computed from the denoising loop.

Compilation and cache-warming runs are excluded from timing. On Mac, each reported value is the mean over $N = 5$ timed renders per model and resolution, reported as mean $\pm$ standard deviation. On iPhone, the initial post-compilation run is discarded, and each reported value is the mean of two subsequent timed renders per model and resolution.

The measurements are intended to reflect interactive image-generation use. We therefore avoid sustained-loop stress tests as the primary latency protocol: such loops push mobile and laptop devices into a throttled steady state that can overstate the latency experienced in bursty creative workflows. Cooldown intervals are used between timed windows to return the device closer to its interactive operating regime.

Per-stage timings are recorded separately where available, including text encoding, the transformer denoising loop, and VAE decoding. These breakdowns allow transformer-only behavior to be inspected separately from end-to-end deployment latency.

Mac M4 Pro 48 GB. Apple Silicon reference system for Mac throughput and memory-pressure measurements, with a 14-core CPU and 20-core GPU.

iPhone 17 Pro Max, A19 Pro, 12 GB unified memory. On-device inference target using the MLX Swift runtime. The Bonsai runtime is configured with VAE tile size 256 px. The FP16 FLUX.2 Klein 4B pipeline is not reported on this device because it does not fit within the available unified-memory budget.

Comparison Normalization

All compared variants are measured under matched sampler, resolution, prompt, seed, step-count, and guidance settings where supported. End-to-end latency numbers reflect the deployed pipeline rather than a transformer-only microbenchmark.

The Bonsai variants and the FLUX.2 Klein 4B baseline share the same text-encoder architecture, Qwen3-4B, but use different deployed representations: the Bonsai variants use the compressed text-encoder pack used in the local deployment, while the FP16 baseline uses native FP16 weights.

For this reason, end-to-end latency should be interpreted as deployed-system latency. Where the transformer-only question is relevant, per-stage timings are reported separately to isolate the denoising loop from text encoding and VAE decoding.

C.2 Throughput Results

This section provides expanded throughput tables supporting Section 4.4 and the throughput results reported in the main text. The tables separate the deployed binary and ternary variants from FP16 runtime ablations, showing how the low-bit transformer packs and the surrounding Bonsai runtime contribute to end-to-end latency.

Table 10. Mac M4 Pro throughput. Speedup is relative to the stock MFLUX FP16 pipeline at the same resolution.

Model	Platform	Backend	Res.	s/step	s/image	Speedup
1-bit Bonsai Image 4B	Mac M4 Pro	MLX 1-bit	512×512	1.50	6.01 ± 0.31	3.03×
Ternary Bonsai Image 4B	Mac M4 Pro	MLX 2-bit	512×512	1.44	5.78 ± 0.08	3.15×
FLUX.2 Klein 4B FP16 (MFLUX)	Mac M4 Pro	Dense FP16	512×512	4.55	18.22 ± 3.00	1.0×
FLUX.2 Klein 4B FP16 (Bonsai runtime)	Mac M4 Pro	Dense FP16	512×512	1.48	5.91 ± 0.27	3.08×
FLUX.2 Klein 4B FP16 (Bonsai runtime + megakernel)	Mac M4 Pro	Dense FP16	512×512	1.44	5.75 ± 0.001	3.17×
1-bit Bonsai Image 4B	Mac M4 Pro	MLX 1-bit	1024×1024	6.02	24.07 ± 0.03	5.60×
Ternary Bonsai Image 4B	Mac M4 Pro	MLX 2-bit	1024×1024	6.06	24.26 ± 0.24	5.56×
FLUX.2 Klein 4B FP16 (MFLUX)	Mac M4 Pro	Dense FP16	1024×1024	33.73	134.91 ± 15.48	1.0×
FLUX.2 Klein 4B FP16 (Bonsai runtime)	Mac M4 Pro	Dense FP16	1024×1024	6.47	25.89 ± 0.19	5.21×
FLUX.2 Klein 4B FP16 (Bonsai runtime + megakernel)	Mac M4 Pro	Dense FP16	1024×1024	6.25	24.98 ± 0.12	5.40×

The FP16 Bonsai-runtime rows isolate runtime optimizations without binary or ternary weights; the megakernel row adds LayerNorm/RoPE/QKV fusion.

At 1024×1024, 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B have nearly identical wall-clock latency on Mac M4 Pro: 24.07 s and 24.26 s per image, respectively. Relative to the stock MFLUX FP16 pipeline, the full Bonsai deployment stack reaches $5.60\times$ speedup for 1-bit Bonsai Image 4B and $5.56\times$ for Ternary Bonsai Image 4B. The FP16 runtime rows show that most of the latency gain comes from

the surrounding Bonsai runtime stack, while the binary and ternary transformer packs primarily improve footprint and memory pressure on current MLX paths.

Table 11. iPhone 17 Pro Max throughput for the deployed Bonsai MLX pipelines. The FP16 FLUX.2 Klein 4B pipeline is omitted because it does not fit on device.

Model	Platform	Backend	Res.	s/step	s/image
1-bit Bonsai Image 4B	iPhone 17 Pro Max	MLX 1-bit	128×128	0.68	2.7
Ternary Bonsai Image 4B	iPhone 17 Pro Max	MLX 2-bit	128×128	0.68	2.7
1-bit Bonsai Image 4B	iPhone 17 Pro Max	MLX 1-bit	256×256	0.95	3.8
Ternary Bonsai Image 4B	iPhone 17 Pro Max	MLX 2-bit	256×256	1.00	4.0
1-bit Bonsai Image 4B	iPhone 17 Pro Max	MLX 1-bit	512×512	2.35	9.4
Ternary Bonsai Image 4B	iPhone 17 Pro Max	MLX 2-bit	512×512	2.35	9.4
1-bit Bonsai Image 4B	iPhone 17 Pro Max	MLX 1-bit	1024×1024	8.15	32.6
Ternary Bonsai Image 4B	iPhone 17 Pro Max	MLX 2-bit	1024×1024	8.50	34.0

On iPhone 17 Pro Max, the FP16 pipeline does not fit, while both Bonsai arms run on-device. 1-bit Bonsai Image 4B and Ternary Bonsai Image 4B generate 512×512 images in 9.4 s and 1024×1024 images in 32.6–34.0 s. Lower-resolution rows show that the same deployment can support fast preview generation, reaching roughly 2.7 s at 128×128 and 3.8–4.0 s at 256×256.

Taken together, the latency results show that binary and ternary Bonsai Image occupy the same practical speed class on Apple Silicon. The choice between them is primarily a deployment trade-off: 1-bit Bonsai Image 4B minimizes storage and memory footprint, while Ternary Bonsai Image 4B uses a slightly larger representation to preserve more visual quality and prompt fidelity.

C.3 Memory-Pressure Proxy

Wall-clock latency reports how long a render takes, while peak memory reports the largest resident allocation during that render. To capture the sustained memory burden of each deployment, we also report a memory-pressure proxy based on MLX active-memory sampling on Apple Silicon.

Apple Silicon does not expose a direct userspace counter for GPU↔DRAM bandwidth in this measurement setup, so the proxy is not a true bytes-moved hardware counter. It is intended for relative comparison across variants under matched prompt, sampler, step-count, and runtime conditions.

For each timed Mac render, a sampler thread polls `mx.get_active_memory()` at 50 ms intervals and records `(t, active_bytes)` samples. After the render completes, we compute the trapezoidal time integral

$$I = \int_0^{t_{\text{render}}} a(t)\, dt,$$

reported in GB·s, and the time-averaged active memory

$$\bar{a} = I / t_{\text{render}}.$$

From this we report three quantities:

- **mean_active** ($\bar{a}$, GB): directly measured time-averaged active memory during the render.
- **gbs_integral** (I, GB·s): active-memory residency integrated over render time.
- **weight_traffic** ($\bar{a} \times N_{\text{steps}}$, GB): an analytical proxy for repeated transformer weight movement under a single-read-per-step assumption.

The weight-traffic proxy is useful for comparing memory pressure across deployment variants, but it should not be interpreted as exact DRAM traffic. For megakernel-fused rows, it can slightly overestimate movement because intermediate tensors may remain register- or cache-resident inside the fused

dispatch. For stock single-shot VAE decoding, it can slightly underestimate movement because the one-time high-resolution VAE pass adds additional activation traffic not captured by multiplying by the number of denoising steps. These effects do not change the relative interpretation.

The Mac throughput and memory-pressure tables are drawn from the same measurement campaign, so wall-clock timing and memory-pressure sampling are aligned per render. Per-render memory-pressure measurement on iPhone is deferred because the current iOS MLX Swift measurement path does not provide a validated equivalent of the Mac active-memory sampling path.

Table 12. Mac M-series memory-pressure proxy. Measurements use matched prompts, seeds, sampler settings, and four denoising steps. **mean_active** is time-averaged MLX active memory; **gbs_int** is active-memory residency over render time; **weight_traffic** is mean active memory multiplied by four denoising steps.

Model	Res.	mean_active	gbs_int	weight_traffic	vs stock
1-bit Bonsai Image 4B	512×512	1.50 GB	9.0 GB·s	6.00 GB	7.8×
Ternary Bonsai Image 4B	512×512	1.96 GB	11.3 GB·s	7.84 GB	6.0×
FLUX.2 Klein 4B FP16 (stock mflux)	512×512	11.74 GB	213 GB·s	46.96 GB	1.0×
FLUX.2 Klein 4B FP16 (Bonsai runtime)	512×512	7.41 GB	43.8 GB·s	29.64 GB	1.6×
FLUX.2 Klein 4B FP16 (Bonsai runtime + megakernel)	512×512	7.69 GB	44.2 GB·s	30.76 GB	1.5×
1-bit Bonsai Image 4B	1024×1024	1.95 GB	46.8 GB·s	7.80 GB	7.4×
Ternary Bonsai Image 4B	1024×1024	2.38 GB	57.8 GB·s	9.52 GB	6.0×
FLUX.2 Klein 4B FP16 (stock mflux)	1024×1024	14.39 GB	1938 GB·s	57.56 GB	1.0×
FLUX.2 Klein 4B FP16 (Bonsai runtime)	1024×1024	7.84 GB	203 GB·s	31.36 GB	1.8×
FLUX.2 Klein 4B FP16 (Bonsai runtime + megakernel)	1024×1024	8.27 GB	206 GB·s	33.08 GB	1.7×

The memory-pressure results show that the runtime optimizations and the binary/ternary transformer packs reduce different parts of the pipeline. Tiled VAE decoding and text-encoder eviction reduce large transient allocations, while the binary and ternary packs reduce the repeated transformer weight footprint. Together, these effects lower the sustained memory burden of local generation substantially relative to the stock FP16 path.

D. Storage Breakdown

This section reports the byte-level composition of the deployed transformer packs. The main storage table reports aggregate transformer and pipeline sizes; Table 13 breaks the transformer pack into its major components.

For each variant, the deployed transformer consists of three parts:

1. packed binary or ternary weights for the matrix-heavy transformer linears;
2. per-group FP16 scales and associated linear biases;
3. precision-sensitive auxiliary tensors retained in FP16.

The packed linears cover the Q/K/V projections, output projections, and MLP linears across the diffusion transformer blocks. The FP16 auxiliary tensors include the modulation streams, embedders, output norm, and output projection. These tensors account for a small fraction of the full transformer but are kept in FP16 for stability and image quality.

Table 13. Layer-level storage breakdown of the deployed transformer packs. The packed linears cover Q/K/V projections, output projections, and MLP linears across the DiT blocks. Precision-sensitive auxiliary tensors are retained in FP16. Sizes are measured from the released MLX-packed safetensors files.

Component	Format	1-bit Bonsai Image 4B	Ternary Bonsai Image 4B
100 packed linears (Q/K/V/Out, MLP, double-stream add-K/Q/V)	1-bit / ternary (g128)	460.1 MB	920.1 MB
Per-group FP16 scales (one per 128-weight group)	FP16	57.5 MB	57.5 MB
Biases for packed linears	FP16	57.5 MB	57.5 MB
`double_stream_modulation_img` (1 tensor)	FP16	113.3 MB	113.3 MB
`double_stream_modulation_txt` (1 tensor)	FP16	113.3 MB	113.3 MB
`single_stream_modulation` (1 tensor)	FP16	56.6 MB	56.6 MB
`context_embedder` (1 tensor)	FP16	47.2 MB	47.2 MB
`norm_out` (1 tensor)	FP16	37.8 MB	37.8 MB
`time_guidance_embed` (2 tensors)	FP16	20.5 MB	20.5 MB
`x_embedder` + `proj_out` (2 tensors)	FP16	1.6 MB	1.6 MB
Total on-disk (MLX packed)		**965.2 MB**	**1425.2 MB**

References

[1] Black Forest Labs. *FLUX.2 [klein] 4B*. Hugging Face model card, 2025. https://huggingface.co/black-forest-labs/FLUX.2-klein-4B.

[2] Black Forest Labs. *FLUX.2 Small Decoder*. Hugging Face model card, 2025. https://huggingface.co/black-forest-labs/FLUX.2-small-decoder.

[3] Black Forest Labs. *FLUX.2 [klein] 4B model card — Apache 2.0 license*, 2025. https://huggingface.co/black-forest-labs/FLUX.2-klein-4B.

[4] Filip Strandberg. *mflux: MLX native implementations of state-of-the-art generative image models*. GitHub, 2024–2026. https://github.com/filipstrand/mflux.

[5] PrismML. *Bonsai Image Studio — Apple Silicon (MLX) and CUDA (gemlite) inference backend*. GitHub, 2026. https://github.com/PrismML-Eng/image-studio.

[6] Dhruba Ghosh, Hanna Hajishirzi, and Ludwig Schmidt. *GenEval: An Object-Focused Framework for Evaluating Text-to-Image Alignment*. NeurIPS, 2023. arXiv:2310.11513.

[7] Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. *ELLA: Equip Diffusion Models with LLM for Enhanced Semantic Alignment* (introduces DPG-Bench, 1,065 dense prompts). 2024. arXiv:2403.05135.

[8] *HPSv3: Towards Wide-Spectrum Human Preference Score*. arXiv:2508.03789, 2025.

[9] Hicham Badri and Appu Shaji. *Gemlite: Towards Building Custom Low-Bit Fused CUDA Kernels*. Mobius Labs, 2024. https://mobiusml.github.io/gemlite_blog/; code at https://github.com/mobiusml/gemlite.

[10] Hicham Badri and Appu 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.

[11] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. *Q-Diffusion: Quantizing Diffusion Models*. arXiv:2302.04304, 2023.

[12] Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. *SVDQuant: Absorbing Outliers by Low-Rank Components for 4-Bit Diffusion Models*. arXiv:2411.05007, 2024.

[13] Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. *Nunchaku: A Fused Inference Engine for SVDQuant 4-Bit Diffusion Models*. Associated inference engine for SVDQuant, 2024.

[14] Prism ML. 1-bit Bonsai 8B. White paper. https://github.com/PrismML-Eng/Bonsai-demo/blob/main/1-bit-bonsai-8b-whitepaper.pdf, 2026.

[15] PrismML. *Ternary Bonsai 8B: Ternary (1.58-bit) language models at 8B, 4B, and 1.7B scale*. White paper, 2026.

[16] Black Forest Labs. *FLUX.1 [schnell]*. Hugging Face model card, 2024. https://huggingface.co/black-forest-labs/FLUX.1-schnell.

[17] PixArt-alpha. *PixArt-Sigma-XL-2-1024-MS*. Hugging Face model card, 2024. https://huggingface.co/PixArt-alpha/PixArt-Sigma-XL-2-1024-MS.

[18] Stability AI. *stable-diffusion-xl-base-1.0*. Hugging Face model card, 2023. https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0.

[19] Stable Diffusion v1.5. *stable-diffusion-v1-5*. Hugging Face model card, 2022. https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-v1-5.

[20] Nota AI. *bk-sdm-small*. Hugging Face model card, 2023. https://huggingface.co/nota-ai/bk-sdm-small.

[21] *Demystifying Flux Architecture*. arXiv:2507.09595, 2025. https://arxiv.org/abs/2507.09595.

[22] NVIDIA. *Optimizing FLUX.1 Kontext for Image Editing with Low-Precision Quantization*. NVIDIA Technical Blog, 2025. https://developer.nvidia.com/blog/optimizing-flux-1-kontext-for-image-editing-with-low-precision-quantization/.