LITHE: Lattice-Indexed Twin Hadamard Encoding for Diffusion Personalization

Jian Jiang
Global College, Shanghai Jiao Tong University
jian.jiang@sjtu.edu.cn

Oya Celiktutan
King's College London
oya.celiktutan@kcl.ac.uk

Yaohui Wang
Shanghai AI Laboratory
wangyaohui@pjlab.org.cn

Yutong Ban
Global College, Shanghai Jiao Tong University
yban@sjtu.edu.cn

Abstract

Diffusion personalization services hold growing catalogues of LoRA adapters, but resident-LoRA serving is capped by GPU memory (about 150 rank-16 adapters on an A100 80 GB GPU running FLUX.1-dev). We introduce LITHE, a LoRA-compatible deployment representation that stores each adapter as an integer-index stream over a deterministic Hadamard codebook. It raises this ceiling to 10,000 verified adapters at sub-80 GB peak as bit-packed index streams that decode on demand in 4.56 ms. The same on-disk indices drive two serving modes. An *index-resident* mode routes a heterogeneous batch of adapters in one forward, and a *decoded compiled* mode runs within 1% of an optimized LoRA+compile baseline at 1024^2 and 1.07–1.42× faster at lower resolutions. On disk, each rank-16 adapter averages 3.23 MB, 47×/190× smaller than the matched r=16/r=64 bf16 LoRA.[1]

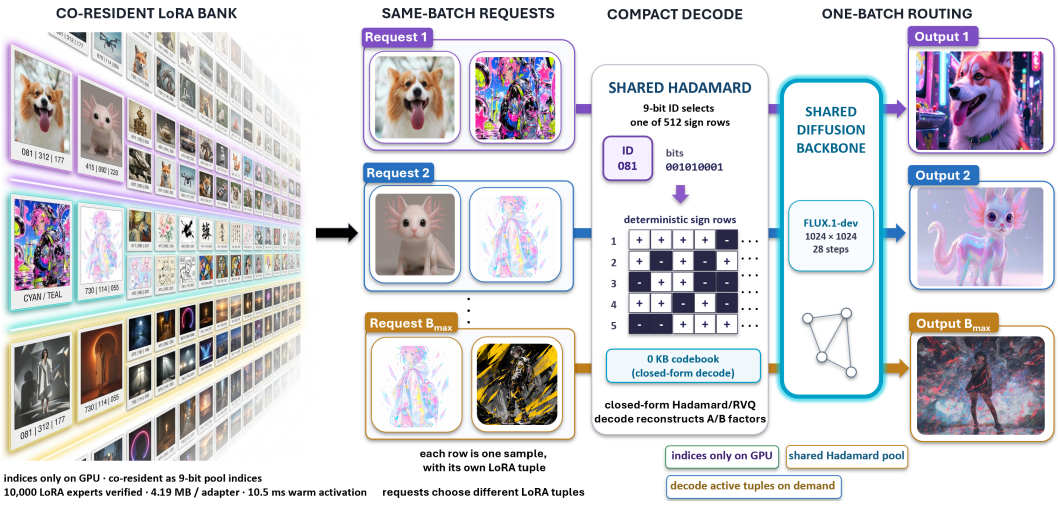


Figure 1: **Co-resident index-bank serving with per-sample routing.** LITHE at 1024^2/28-step on FLUX.1-dev in bit-packed index-residency mode: 10,000 residents at 75.7 GB peak, $B_{\max}$=14 heterogeneous adapters per forward (Tab. 5 P6, 13.04 s/sample); B=96 unique-adapter run at 256^2/4-step is reported numerically in App. E (16.43 s vs. serial 163.5 s, 9.95×). Wall-time vs. LoRA+compile is a separate decoded compiled hot-set point in Tab. 2, not this 10,000-resident operating point.

[1] Code is at github.com/JianJiang1995/LITHE and adapters at huggingface.co/Fanjue/LITHE.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Many personalized diffusion deployments can be viewed as LoRA-expert serving systems. A LoRA-compatible deployment maintains a heterogeneous catalog of small adapters that capture art styles (StyleDrop [1], StyleShot [2], InstantStyle [3]), reference-image conditioning modules, lighting and pose templates, outfit and garment concepts, painterly and anime stylizations, and subject identities trained from a handful of images [4–6]. At request time, the platform must route a prompt to one or more of these experts, sometimes compose them [7–10], and render under request-level latency and throughput constraints. Our goal is a representation of LoRA factors that makes this Diffusers serving loop cheaper, regardless of which personalization recipe produced the expert.

Three bottlenecks dominate this serving path on a rank-16 FLUX.1-dev adapter (the production setting throughout this paper). (i) Hot-swap latency: replacing the active LoRA at runtime without restarting the backbone requires disk-loading the next $\sim$150 MB of bf16 (bfloat16) weights, merging that delta into the base weights for the render, and reverting the merge afterwards. The full cycle measures $\sim$0.9 s on 1$\times$A100 80 GB (0.05 revert + 0.78 load + 0.06 merge; CephFS catalog cold-touch, 4-server consensus, App. D.8), paid again whenever a request activates a different expert. (ii) Module-global fusion forces serial fuse-and-render under heterogeneous traffic, sacrificing GPU batch parallelism. (iii) Catalogs of thousands of community and user-trained experts at $\sim$150 MB each are expensive to store, transfer, and keep resident in GPU memory. This paper's scope is the representation beneath that active set: keeping large catalogs as compact index streams, decoding only activated adapters, and routing those adapters within one diffusion batch. We report LITHE's serving cost against both the measured Diffusers serial fuse path and an optimized dense multi-LoRA baseline, a resident graph-compiled r=16 LoRA (Tab. 2).

Existing work covers parts of this serving path. Quantization-based compressors [11, 12] and structural PEFT variants [13–15] reduce per-expert weight footprint but still emit LoRA weight tensors at deployment. Composition- and interference-oriented work (ZipLoRA [7], Mix-of-Show [8], LoRAHub [9], B-LoRA [10], MoLE [16]) targets multi-expert stacking and routing, again on LoRA weight tensors. Multi-LoRA serving systems (Punica [17], S-LoRA [18]) batch heterogeneous LoRAs on a shared backbone but still require full LoRA weights when a new adapter enters the active set. LITHE addresses the storage representation behind that active set. Large catalogs can stay as compact index streams and decode only the adapters actually activated by a request.

We introduce LITHE, a finite Hadamard-indexed LoRA parameterization for this expert-serving setting. The method does not replace LoRA; it replaces the storage and serving representation of the LoRA A/B factors. During training those factors are represented as integer indices into an asymmetric twin codebook using straight-through estimation. At deployment, index streams replace the dense per-adapter weight tensors and are decoded into the corresponding LoRA update on demand. The decode is a closed-form Hadamard sign recombination that runs in 4.56 ms warm, against $\sim$900 ms for the standard serial-fuse swap path (§5.2, App. D.8). LITHE therefore yields compact adapter payloads, pointer-level hot-swap, low-overhead host-side composition, and prompt-level routing across heterogeneous LoRA experts, while leaving the host backbone unchanged.

Contributions. **(1) Hadamard-indexed LoRA parameterization (§3).** A native discrete parameterization for diffusion LoRA factors over a single shared Hadamard sign pool H_9, with asymmetric per-side scale (σ_A^{cb}=0.06 for A, a multiscale residual chain at $\{0.0025, 0.001, 0.0005\}$ for B). The deployed adapter is an index stream over a 0 KB deterministic pool. A four-cell ablation along the scale axis (Tab. 3) shows that removing the per-side scale asymmetry, not pool-sign separation, causes the model to collapse. **(2) Unified deployment representation for two serving modes (§5.2).** The same per-concept index stream serves two modes. A *decoded compiled* mode for hot sets ($N_{\mathrm{res}} \lesssim 156$ during a B=14, 1024^2 render) is 0.8% slower than an optimized LoRA+compile baseline at 1024^2/28 (Tab. 2). A *bit-packed index-residency* mode, verified at 10,000 slots and 75.7 GB, routes 14 heterogeneous 1024^2/28 requests per forward (Tab. 5 P6) and 96 at 256^2/4 (App. E, 16.43 s vs. serial 163.5 s). App. D.7 separates the contributions of graph compilation and the codebook-shared decode cache. **(3) Storage–quality reading across four benchmarks (§4).** On DreamBooth-30, CC101-20, SCIdentity, and SC-40, LITHE stores adapters at 3.23 MB/concept on average (LZMA-9 of the 7.18 MB uint16 stream), $\sim$47$\times$/$\sim$190$\times$ smaller than bf16 r=16/r=64 LoRA. The LLM-avg gap to LoRA stays within 0.18 on the three identity benchmarks (Tab. 1). Strict identity remains a limitation (App. A).

2 Related Work

PEFT for diffusion personalization. LoRA [19] freezes the backbone and inserts low-rank updates, becoming the de-facto recipe for FLUX/SD3-style diffusion [20, 21] personalization. Variants attack the residual cost: AdaLoRA [22] adaptively prunes singular values, DoRA [13] decomposes magnitude and direction, and VeRA [14] freezes randomly-initialized projection matrices and only learns scaling vectors. LoRA-XS [15] inserts a tiny trainable $r \times r$ block between SVD-derived frozen factors. DyLoRA [23] trains rank prefixes so the deployed rank can vary, and LoRA-FA [24] freezes one LoRA factor to reduce optimization state. The LyCORIS family [25] explores Kronecker (LoKr / DiffuseKronA [26]) and Hadamard (LoHa) factorizations for SD attention/MLP blocks. SVDiff [27] fine-tunes only per-Linear singular-value deltas, OFT [28] parameterizes weight updates as orthogonal Cayley-Neumann blocks, and Textual Inversion [6] trains only a single placeholder token-embedding row. T-LoRA [29] is diffusion-specific. It keeps the LoRA factorization but masks the rank dimension as a function of the denoising timestep, which throttles the effective rank at high-noise (overfit-prone) steps. This regulariser is concurrent with our submission and orthogonal to the discrete-codebook axis we explore. All of these reduce training-time parameters but store continuous fp16/bf16 deltas. HyperDreamBooth [30] reports very small per-subject representations via a shared hypernetwork; LITHE targets a different operating point, namely LoRA-compatible integer-index adapters with same-batch routing. For style personalization, StyleDrop [1] attaches a per-style adapter to Muse, and StyleShot [2] sidesteps per-style tuning altogether via a pretrained style encoder. LITHE stores per-subject *integer indices* over deterministic Hadamard codewords rather than LoRA weight tensors.

Discrete representation and post-hoc quantization. VQ-VAE [31] introduced discrete latent codes; SoundStream [32] and EnCodec [33] made residual VQ standard in neural codecs. Neural Weight Search (NWS) [34] is the closest discrete-adapter precedent: it represents per-task classifier adapters as indices into a shared K-means codebook fitted from oracle weights for small CNNs trained from scratch. LITHE differs in the object being parameterized and in the deployment constraints. It trains FLUX LoRA A/B factors natively with a straight-through estimator and uses asymmetric twin pools for the observed $\sigma_A \gg \sigma_B$ factor scales. It also replaces the learned shared table with a deterministic Hadamard pool and exposes ordinary decoded LoRA factors for same-batch routing. In the PEFT literature, quantization has been applied *post-training*: GPTQ [35], QLoRA [11], LoftQ [12], AQLM [36], and QuIP# [37] quantize an already-trained continuous adapter or backbone. LQ-LoRA [38] and QA-LoRA [39] instead quantize the frozen backbone so that low-rank finetuning fits in memory. That axis is complementary. They compress what the adapter attaches to, LITHE discretizes the adapter itself, and the two compose naturally. BitDelta [40] stores fine-tune deltas as 1-bit residuals with per-tensor scales. LITHE differs by training the adapter *natively* on a discrete codebook with straight-through gradient estimation [41]. Its deployed integer indices therefore decode deterministically to the reported deployed weights, not to a post-hoc quantized dense checkpoint.

Token-level versus subject-level Mixtures-of-Experts. Sparse MoE [42, 43] routes each *token* to a small subset of expert FFN blocks at every layer. LoraHub [9] and Mixture-of-LoRA-Experts (MoLE) [16] compose pre-trained LoRAs via gradient-free or learned weighted averaging. ZipLoRA [7], Mix-of-Show [8], and B-LoRA [10] fuse subject and style LoRAs but require pairwise training or attention-mask engineering. LITHE addresses a complementary axis: compact per-subject *footprint* and serving *hot-swap* latency for prompt-routed diffusion personalization. App. H documents the baseline coverage scope and excluded ports.

Multi-LoRA serving. Punica [17] introduces SGMV (Single-Group Matrix-Vector multiply, a batched per-sample routing kernel) to batch heterogeneous LoRAs on a shared backbone with $\sim$2 ms per-token overhead. S-LoRA [18] scales to thousands of concurrently served adapters via Unified Paging, and vLLM [44] integrates the Punica kernel for multi-tenant LoRA serving. These systems excel at *concurrent* inference once LoRA weights are materialized, whereas LITHE asks how many adapters can be kept resident before any LoRA weights are decoded. Our Triton route is SGMV-style but operates on decoded index streams; we compare against resident LoRA baselines in Tab. 2 and against the released Punica/S-LoRA SGMV kernel in App. T. LITHE shrinks the per-subject disk payload by $\sim 47\times$ versus matched-rank bf16 LoRA. It uses the same SGMV family of kernels for per-sample routing, adapted to run after a packed-index decode rather than over dense resident factor tensors (§5.2).

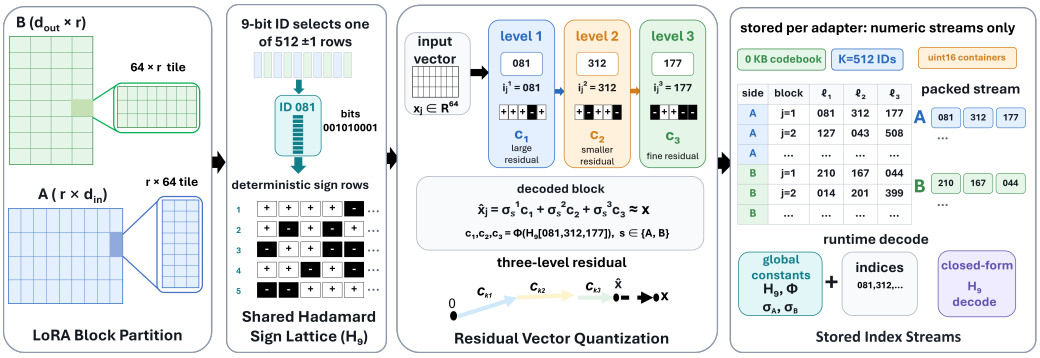


Figure 2: **Hadamard-indexed LoRA factors.** The diagram follows the deployed adapter path. (1) Each LoRA A/B factor is partitioned into 64-value blocks. (2) A 9-bit ID selects a deterministic row of the shared Hadamard sign table. (3) Three residual levels store IDs whose scaled sign rows reconstruct the block. (4) The deployed per-adapter payload is only the index stream over side, block, and RVQ level; H_9, Φ, and the side-specific A/B scales are global procedural constants used by the runtime decoder.

3 Method

LITHE combines two architectural components. The first is a *twin codebook* that decouples the asymmetric $\sigma_A \gg \sigma_B$ scales of LoRA factors A and B. The second is a *Hadamard-sign codebook* construction whose codewords are deterministic functions of a 512-row, 9-bit sign table derived from a Sylvester Hadamard construction [45]. This construction gives a closed-form decode (§3.2) and a 0 KB shared-codebook deployment (per-adapter index streams only; §5.2).

3.1 LITHE discrete LoRA training

Setup. Let $\Delta W = BA$ denote a LoRA correction, with $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$, $r \ll \min(d_{\text{in}}, d_{\text{out}})$. LITHE reshapes A and B into uniform vector blocks $\{a_j\}, \{b_j\} \subset \mathbb{R}^B$ (B=64 throughout) and quantizes each block via L-level residual vector quantization (RVQ) against a shared codebook of size K. The forward pass uses the discrete reconstruction $\hat{a} = \sum_{\ell=1}^{L} c_{k_\ell}^{(\ell)}$; gradients propagate through a straight-through estimator [41, 31]. Figure 2 shows the storage-to-decode pipeline used throughout the paper. Each A/B factor is first viewed as a sequence of 64-value blocks. For every block and residual level, the adapter stores an integer ID rather than a dense vector. At runtime the ID stream selects deterministic Hadamard rows, applies the side-specific scale schedule, and sums the three residual contributions into the 64-vector used by the LoRA matmul.

Hadamard codebook (asymmetric twin construction). The canonical LITHE codebook is an asymmetric Hadamard-9bit multiscale twin codebook: a finite deterministic Hadamard-sign codebook family, not a learned vector-quantizer table. Both pools share the same 512-row sign-signature table $H_9 \in \{\pm 1\}^{512 \times 9}$ derived from a Sylvester Hadamard construction (Conway–Sloane [45], §4.7). A fixed expansion map Φ turns each 9-bit signature into a 64-dimensional sign codeword; the exact decode is given in Eq. 2. The A and B sides use asymmetric per-side scales: σ_A^{cb}=0.06 and $\sigma_B^{\text{cb},\ell} \in \{0.0025, 0.001, 0.0005\}$ across the three residual levels, set from observed scale statistics in our trained continuous LoRA probes ($\sigma_A/\sigma_B \approx 25\times$). The equal A-side scale across residual levels is an empirical design choice. It treats the Hadamard signs as an equal-amplitude combinatorial sum rather than a coarse-to-fine residual schedule, while the smaller B side receives decaying residual scales. The reported FLUX, SDXL, and Qwen-Image runs use these constants as global implementation constants, not per-adapter metadata. They are a fixed design heuristic, not a rate-distortion guarantee, and App. B gives the supporting heuristic calibration. Both pools are *deterministic constants* of the construction; no learned parameters are stored or transferred in the deployed pool. The Hadamard choice is a computational and deployment property, not a quality lever. At matched rate it stays within trial noise of learned and random sign codebooks while storing no table (App. C.3).

Per-concept index encoding. Per-concept indices are uint16 vectors over a K=512-entry codebook, with $\sim 3.6 \times 10^6$ entries per concept (rank r=16, L=3 levels, 304 FLUX.1-dev Linears, both A and B sides). All reported FLUX.1-dev quality and serving results use adapters trained with the Kohya sd-scripts framework in fused-QKV form. Its 304 Linears cover the attention, MLP, and modulation projections of all double and single blocks, and the same adapter becomes 494 Diffusers wrappers

after QKV splitting. Our code release defaults to a lighter 342-Linear Diffusers configuration without the modulation layers ($\sim 2.1 \times 10^6$ entries, 1.93 MB on disk under the same coder), which no reported quality number uses. Fixed 9-bit packing would occupy 4.04 MB (measured zero-order entropy ≈ 8.99 bits/symbol). We compress the raw uint16 byte stream with LZMA-9, the lossless general-purpose compressor of xz at its maximum preset (9). LZMA exploits cross-level and cross-position correlations and reaches 3.23 MB per concept on average over all 110 LITHE $r{=}16$ adapters of our benchmarks (DreamBooth-30, CC101-20, SCIdentity, SC-40). This is a $\sim 47\times$ reduction relative to the matched bf16 $r{=}16$ LoRA at 153.3 MB, and $\sim 190\times$ relative to the strongest dense $r{=}64$ LoRA at 612.6 MB. The codebook itself is a 0 KB deterministic Hadamard-sign construction for disk payload and network transfer. Implementations may materialize small runtime sign-pool tensors (about 384 KiB per side before scales; App. C.3) at process start.

Why scale asymmetry is load-bearing. Trained continuous LoRAs sit at very different per-side scales ($\sigma_A/\sigma_B \approx 25\times$), so a single shared scale would waste resolution on A or underfit B. Both pool A and pool B inherit the same Hadamard H_9 sign structure by construction; what differs is the per-side scale schedule. A four-cell isolation along that scale axis (Tab. 3) shows that removing the asymmetry collapses GPT-avg to 1.00, whether pool B is built separately or shared with pool A. The asymmetric default keeps GPT-avg at 4.01 (LoRA anchor 4.16). Swapping the multiscale RVQ chain on B for a flat single scale at 0.0025 preserves quality (GPT-avg 3.92, within trial noise). App. B gives the supporting rate-scaling calibration.

3.2 Hadamard-indexed acceleration

Hadamard closed-form decode. Let $U \in \{\pm 1\}^{64\times 9}$ be the first nine columns of the order-64 Sylvester–Hadamard matrix H_{Syl} (App. C.3). For codebook index $k \in \{0, \ldots, 511\}$, let $\sigma_k \in \{\pm 1\}^9$ be the sign expansion of the nine bits of k. The codeword is the normalised projection

$$c_k \;=\; \Phi(\sigma_k) \;=\; \tfrac{1}{\sqrt{9}}\, U\, \sigma_k \;\in\; \mathbb{R}^{64}, \tag{1}$$

i.e. the 512 codewords lie in a fixed 9-dimensional Hadamard subspace of $\mathbb{R}^{64}$ rather than on the ± 1 cube. Each entry is a normalised sum of nine ± 1 values, so c_k has unit per-coordinate variance under uniform σ_k. The map is injective on $\{\pm 1\}^9$ because U has full column rank.

Given per-block, per-side index streams $\{i^{(\ell)}_{A,j}\}_{j,\ell}$ and $\{i^{(\ell)}_{B,j}\}_{j,\ell}$ produced by twin RVQ encoding, each 64-dimensional block is reconstructed as

$$\hat{a}_j \;=\; \sum_{\ell=1}^{L} \sigma_A^{\text{cb}}\, c_{\,i^{(\ell)}_{A,j}}, \qquad \hat{b}_j \;=\; \sum_{\ell=1}^{L} \sigma_B^{\text{cb},\ell}\, c_{\,i^{(\ell)}_{B,j}}, \tag{2}$$

where $\sigma_A^{\text{cb}} = 0.06$ is shared across levels and $\sigma_B^{\text{cb}} = \{0.0025, 0.001, 0.0005\}$ gives the per-level B-side scales. The reconstructed blocks are unflattened back into $\hat{A}$ and $\hat{B}$. The same map also gives a closed-form nearest-codeword search. Bit q of the index is the sign of the block's projection onto Hadamard column q, so the search never materialises a $K \times 64$ distance matrix. Timed per call on an A100, outside the end-to-end pipeline, a single-level search takes 52 μs against 135 μs for an exhaustive distance search. Both searches return the same indices. The identity-benchmark adapters were trained with the exhaustive search and the SC-40 adapters with the closed form. The forward delta is $\delta = \hat{B}\hat{A} \cdot \alpha/r$, applied as $y = Wx + \delta x$ in each wrapped linear. Backward gradients pass through a straight-through estimator onto the continuous proxies A, B, which AdamW then updates.

Two paired Triton algorithms. The training mathematical core (Alg. 2, App. C.7) and the serving same-batch heterogeneous forward (Alg. 3, App. C.7) share the same closed-form decode. Algorithm 2 runs twin RVQ encoding plus the closed-form decode of Eq. 2, with either nearest-codeword search. Algorithm 3 turns Eq. 3 into a per-sample SGMV step with fp32 accumulate and bf16 storage. It batches B heterogeneous adapter tuples in one forward at the wrapped backbone's per-batch cost (no per-sample swap loop). LazyRecomp on the backward path is optional and kept in the appendix because it provides no end-to-end VRAM saving in the reported setting.

Per-sample routing form. The decoded factors also define the serving-time heterogeneous route used in §5.2. For a serving batch of B requests, request i activates an expert tuple $\mathcal{E}_i$ with per-axis weights $w_{a,i}$ and mixing coefficients α_a; the reported mixtures use nonnegative $\alpha_a w_{a,i}$. This sign restriction applies only to the mixture coefficients; decoded LoRA factors remain signed, and

signed mixture weights are outside the reported implementation. We compose per-sample factors by rank-stacking the active experts with weights baked in,

$$\hat{A}_i^{\text{stack}} = \text{vcat}\big[\sqrt{\alpha_a w_{a,i}}\,\hat{A}_{e_a}\big]_{a\in\text{axes}(i)}, \quad \hat{B}_i^{\text{stack}} = \text{hcat}\big[\sqrt{\alpha_a w_{a,i}}\,\hat{B}_{e_a}\big]_{a\in\text{axes}(i)}, \tag{3}$$

and the per-sample update is $\delta_i = \hat{B}_i^{\text{stack}}\hat{A}_i^{\text{stack}}$. Here vcat stacks A vertically along rows and hcat stacks B horizontally along columns, both along the LoRA rank dimension. If $m_i{=}|\text{axes}(i)|$ and each expert has rank r, then $\hat{A}_i^{\text{stack}} \in \mathbb{R}^{m_i r\times d_{\text{in}}}$ and $\hat{B}_i^{\text{stack}} \in \mathbb{R}^{d_{\text{out}}\times m_i r}$, so the product sums the weighted expert updates. Splitting each nonnegative mixture weight as $\sqrt{\alpha_a w_{a,i}}$ on both factors makes the stacked product recover $\alpha_a w_{a,i}\hat{B}_{e_a}\hat{A}_{e_a}$ for each expert.

4 Evaluation

We first evaluate LITHE on four personalization benchmarks with FLUX.1-dev as the base model. All results use the default configuration of Tab. 7, with codebook size $K{=}512$, block size $B{=}64$, $L{=}3$ residual levels, rank $r{=}16$, LITHE multiscale, codebook reassignment every 10 steps, and bf16. Adapters train at 512^2 for 500 steps on the identity benchmarks and 750 on SC-40. We then run the matched identity (DreamBooth-30, CC101-20, SCIdentity) and concept-transfer (SC-40) coverage on SDXL 1.0 and Qwen-Image-20B in §4.5. Each generation is scored under a 3-LLM judge consensus over GPT-5.1, Claude Sonnet 4.6, and Gemini 2.5 Flash on a 1–5 scale. Per-benchmark rubrics and verbatim judge prompts are in App. M.1. The headline LLM-avg is the unweighted mean of GPT-avg, Claude-avg, and Gemini-avg. Per-judge breakdowns are in App. F, DreamSim and CLIP-T embedding cross-checks in App. K, and qualitative galleries in App. O. Trained-adapter rows train each method with three random seeds, and storage figures are deploy-time bytes per subject.

4.1 Benchmark suite

We use four benchmarks. The two public benchmarks are **DreamBooth-30** and **CC101-20**, a 20-subject slice of CustomConcept101. The two synthetic benchmarks are **SCIdentity**, a DreamBooth-style identity benchmark on 20 rare animal-like individuals, and **SC-40**, a 5-axis controlled concept-transfer test on 40 concepts $\times$ 20 references. The synthetic benchmarks use intentionally rare and visually demanding concepts. Modern FLUX-class generators already render common objects, lighting, poses, and styles zero-shot, which leaves little signal under a LoRA delta. Construction, prompts, caption protocols, and visual prompt galleries are in App. L and App. P.

4.2 Storage–quality trade-off across benchmarks

Table 1 summarizes the storage–quality trade-off across the active subject and concept-transfer benchmarks: DreamBooth-30, CC101-20, SCIdentity, and SC-40. We report full benchmark coverage for the rerun PEFT baselines (DoRA, VeRA, AdaLoRA, QLoRA-NF4) and for the validated DyLoRA, LoRA-FA, LoRA, and LITHE rows. All LoRA-family rows use rank $r{=}16$ unless explicitly stated otherwise.

Table 1 shows the main storage-quality result. LITHE is the best non-LoRA LLM-avg row on the three FLUX identity benchmarks (DreamBooth-30, CC101-20, SCIdentity) at a per-concept disk payload of 3.23 MB, 47×/190× smaller than bf16 $r{=}16$/$r{=}64$ LoRA. On SC-40, LoRA-FA is the best non-LoRA LLM-avg row, at 153.3 MB. LITHE is lower-fidelity than LoRA on the strict identity / target-fidelity dimensions reported in App. I. The averaged LLM-judge gaps in the main table are smaller because prompt following and visual quality remain close. CLIP-T is reported and analysed as a coarse personalization metric in App. K.1. In serving, LITHE holds adapters either as bit-packed GPU indices before decode, verified at 10,000 resident slots (App. D.3), or as an 8.05 MB raw int16 cold-swap blob in the Diffusers split-QKV layout, so disk and VRAM figures should not be conflated. VeRA is much smaller per task under the same procedural shared-pool accounting but loses more quality, with LLM-avg 0.6–0.9 below LITHE on the identity benchmarks. QLoRA-NF4 is smaller than LoRA but about 7× larger than LITHE (full payload accounting in App. D). Against the closest storage-matched baseline, a trained rank-1 LoRA, LITHE is statistically tied at about half its disk footprint (3.31 vs. 6.8 MB). The paired two-seed difference is -0.01 GPT-avg (95% CI $[-0.05, +0.03]$; App. Q, App. R), and blinded human raters find no significant preference (App. S). Existing LoRAs migrate to LITHE by a post-hoc encode plus a short fine-tune (App. U). Hot-swap latency and 5-LoRA composition cost are reported in §5.2 and App. J; the component ablation is in Tab. 3.

Table 1: Storage–quality panel: LITHE vs PEFT baselines on DreamBooth-30, CC101-20, SCIdentity, and SC-40. Each trained-adapter entry is the mean over three training runs, with seeds 42/1/7 for LoRA and LITHE and 42/43/44 for the other adapters. For LLM-avg, ± is each judge's standard deviation across the three runs, averaged over the three judges. All LoRA-family methods in this table use rank r=16 unless explicitly noted. **Vanilla FLUX** is the backbone without an adapter, rendered once, and its LLM-avg entry is the GPT-5.1 average alone. **LLM-avg**, a closed-model soft quality signal, averages the GPT-5.1, Gemini 2.5 Flash, and Claude Sonnet 4.6 benchmark-level judge averages on a 1–5 scale. Quality entries are rounded to two decimals. App. F reports per-model judge averages, and the prompt-level rubric sub-scores are in the code repository. **DreamSim** is reference-image similarity. **Per-task** storage is cold deploy-time MB per concept, measured on each method's own r=16 adapter. LoRA, DyLoRA, LoRA-FA, and LITHE use the 304-Linear Kohya layout, which includes the modulation Linears. DoRA, VeRA, AdaLoRA, and QLoRA-NF4 wrap the 342 attention/FFN Linears of the PEFT trainer, LoRA-XS its 266 attention Linears, and T-LoRA follows its reference implementation. CLIP-T values for all rows and an analysis of CLIP-T's coarseness as a personalization metric are reported in App. K.1. Baseline-row implementation, payload accounting, and the QLoRA-NF4† deployed-only convention are in App. D and App. H.

Method	Storage	DreamBooth-30		CC101-20		SCIdentity		SC-40	
	Per-task (MB)	LLM-avg ↑	DreamSim ↑	LLM-avg ↑	DreamSim ↑	LLM-avg ↑	DreamSim ↑	LLM-avg ↑	DreamSim ↑
Vanilla FLUX	0	3.33	.41	3.37	.43	3.98	.43	3.53	.42
LoRA [19]	153.3	4.56±.02	.67±.01	4.19±.04	.62±.00	4.48±.02	.69±.01	3.94±.03	.55±.00
DoRA [13]	92.6	3.72±.02	.50±.01	3.68±.04	.48±.02	3.48±.02	.48±.01	3.17±.01	.46±.01
VeRA [14]	0.022	3.58±.08	.49±.03	3.51±.09	.49±.01	3.39±.07	.49±.02	3.04±.07	.47±.01
AdaLoRA [22]	67.4	3.88±.02	.43±.01	3.95±.02	.43±.01	3.80±.03	.43±.01	3.22±.02	.42±.01
QLoRA-NF4† [11]	22	3.86±.08	.56±.02	3.69±.04	.55±.03	3.70±.07	.58±.02	3.48±.03	**.53±.02**
DyLoRA [23]	153.3	4.26±.02	.54±.00	3.99±.02	.48±.01	4.03±.02	.57±.00	3.63±.05	.49±.00
LoRA-FA [24]	153.3	**4.42±.02**	.59±.01	4.08±.02	.54±.01	4.26±.03	.63±.01	**3.85±.02**	.53±.00
LoRA-XS [15]	52.3	3.75±.03	.46±.00	3.78±.03	.45±.02	3.56±.06	.46±.01	2.98±.07	.43±.01
T-LoRA [29]	157.1	4.14±.02	.56±.01	4.02±.02	.52±.01	3.90±.03	.54±.01	3.64±.04	.50±.00
LITHE (Ours)	**3.23**	**4.42±.02**	**.60±.01**	**4.14±.02**	**.56±.00**	**4.30±.04**	**.65±.02**	3.77±.02	.53±.00

The auxiliary per-axis SC-40 concept-transfer diagnostic (Tab. 31, App. I) shows that LoRA is stronger on target fidelity, especially for outfit and anime-style transfer. LITHE keeps prompt following and visual quality matched within a narrow band. The remaining gap is therefore primarily concept fidelity rather than image quality; qualitative grids in App. O show representative subject/style and outfit/style drift cases. Embedding metrics (App. K) track the judge ordering, with the rank-16 LITHE row trading some DreamSim/CLIP-I similarity for compactness.

4.3 Production-serving wall-time

Multi-LoRA serving under heterogeneous community traffic draws from catalogues larger than the per-process VRAM cap (∼150 rank-16 adapters on 1×A100 80 GB). The LoRA path is therefore bottlenecked by the per-request hot-swap (load ∼150 MB of bf16, fuse, render, unfuse), which costs ∼0.9 s per swap on cold CephFS (4-server consensus, App. D.8). The two resident columns of Tab. 2 (a per-sample LoRA loop, with and without a compiled graph) apply only when the entire batch is pre-loaded in VRAM. Once the catalogue exceeds the cap, both collapse to the serial-fuse baseline *in our measured Diffusers resident deployment*. On the B=96 workload, a pinned-host pager with hidden transfer takes 56.9 s and the Diffusers hotswap API 73.8 s, against 16.3 s for LITHE (App. T). LITHE bypasses that cap by representing the catalogue as a fixed codebook plus 9-bit indices. The verified bit-packed inventory holds 10,000 resident slots (Tab. 5 P6), against 156 for resident r=16 LoRA during the same B=14, 1024^2/28-step render, a ∼64× residency capacity gain. A per-request swap is an index pointer update over the resident set. A trained rank-1 LoRA, the closest storage-matched baseline, still serves through the dense fuse path and occupies 9.6 MB of bf16 per resident adapter against LITHE's 4.19 MB. Two clean comparisons. *Vs. production serial-fuse* (the regime that scales with catalogue size): LITHE is 13.6×/2.39×/1.46× faster on 256^2/4, 512^2/16, 1024^2/28 at B=96/14/14. *Vs. matched-resident LoRA+compile upper bound* (decoded compiled hot-set mode, $N_{\text{res}} \lesssim 156$): LITHE is 0.8% slower at 1024^2 (133.4±0.03 vs. 132.4±0.16 s) and 1.07–1.42× faster at the two lower resolutions. The 10,000-resident bit-packed mode is a separate operating point (Tab. 5 P6, 13.04 s/sample ×14 ≈ 182.6 s wall) and is not the configuration measured in Tab. 2.

Table 2: **Same-batch wall-time, 1×A100 80 GB**, 3-trial mean±std (Serial-fuse cross-validated on 4 independent A100 hosts within stderr; App. D.8). *Serial fuse* is production heterogeneous traffic with catalogue > VRAM cap, CephFS cold first-touch. *LoRA loop* and *LoRA+compile* pre-load all B adapters in VRAM and apply them in a per-sample loop, without and with graph compilation, and are valid only when catalogue ≤ cap (∼150 adapters at r=16). LITHE is the decoded compiled mode at the hot-set point ($N_{\text{res}} \lesssim 156$), where a fused kernel decodes each sample's indices on the fly from a shared codebook cache under graph compilation. The 10,000-resident bit-packed point is the separate mode of Tab. 5. The compiled graph provides the wall-time reduction (App. D.7).

Config	Serial fuse (catalogue>VRAM)	LoRA loop (resident)	LoRA+compile (resident)	LITHE (Ours)
256^2 4-step (B=96)	150.0±1.9 s	18.30 s	15.59±0.13 s	**11.00±0.01** s
512^2 16-step (B=14)	43.3±0.3 s	27.51 s	19.28±0.20 s	**18.08±0.01** s
1024^2 28-step (B=14)	195.3±1.7 s	188.27 s	**132.4±0.16** s	133.4±0.03 s

4.4 Component ablation and deployment payload

Component ablation. Tab. 3 sweeps the twin codebook's per-side scale schedule on the SC-40 five-concept short-run subset, with the shared H_9 pool held fixed. The asymmetric multiscale schedule on B keeps GPT-avg close to the continuous LoRA anchor (4.01 vs. 4.16, 0.15 point gap). Collapsing σ_B^{cb} up to σ_A^{cb} on either side reduces GPT-avg to 1.00, whether pool B is built separately (Cell C) or shared with pool A (Cell D). Cells C and D produce mathematically identical forward outputs by the deterministic-Hadamard construction; we report both as a sanity check. Replacing the multiscale chain on B with a flat single scale (Cell B) recovers near-Cell-A quality, so the load-bearing axis is the asymmetric scale, not pool-sign separation. The full per-axis ablation, visual grid, and longer 25-subject Phase-A pipeline are in App. O.

Table 3: **Twin-codebook scale-isolation.** SC-40 five-concept short-run (r=16, 750 steps, 25 images/row). All rows share the deterministic H_9 pool; only the per-side scale schedule changes. Cell A is the LITHE default; Cells C and D are mathematically identical (same forward outputs by construction) and are reported jointly as a sanity check. Storage is the mean LZMA-9 index-stream size over the five concepts. Collapsed or flat scale schedules produce more repetitive index streams, which compress better. Extended ablations are in App. O.

Row	Scale schedule	GPT-avg	Per-task storage
LoRA anchor (continuous)	—	4.16	153.3 MB
Cell A (LITHE (Ours), default)	σ_A^{cb}=0.06, $\sigma_B^{\text{cb},\ell}$={0.0025, 0.001, 0.0005}	4.01	3.11 MB
Cell B (twin, flat B)	σ_A^{cb}=0.06, σ_B^{cb}=0.0025 flat	3.92	2.82 MB
Cell C/D (symmetric collapse)	σ_B^{cb}=σ_A^{cb}=0.06	1.00	2.58/2.61 MB

The deploy payload is the LZMA-9 compressed bit-stream of per-concept Hadamard indices; pools A/B are deterministic functions of H_9 rebuilt at process start. Compressed size averages 3.23 MB per concept, from 7.18 MB of raw uint16 or 4.04 MB at fixed 9-bit (App. D). LZMA-9 is a general-purpose coder and is not part of LITHE's contribution. The LoRA denominator is the standard bf16 safetensors disk size (153.3 MB at r=16, 612.6 MB at r=64), giving ∼47×/∼190× reductions. Applying the same LZMA-9 coder to the r=16 LoRA file shrinks it to 103.5 MB (Tab. 36), so the coder-matched r=16 reduction is ∼32×. On-GPU residency uses an uncompressed bit-packed form for random-access fused decode (§5.2). On the residual-depth axis, a single level scores 4.03 GPT-avg against the default's 4.01 on the five-concept short-run protocol, within single-run protocol noise, at a 2.80 MiB uncompressed payload (Tab. 35). The non-monotonic L=2 point there is a σ_B-schedule truncation artifact, not an RVQ-depth effect. The full storage panel and comparisons to QLoRA-NF4 and HyperDreamBooth are in App. D. App. Q fills the storage band between VeRA and the dense anchor with matched post-hoc and rank baselines at the full 30-subject scale. On CC101-20 (20 sampled subjects from CustomConcept101, 5 prompts each), LITHE tracks the DreamBooth-30 numbers. Two stylised classes (corkscrew, pikachu) dominate the residual identity gap, as the auxiliary diagnostic slices of App. I show.

4.5 Cross-architecture benchmark coverage

We implemented the same adapter construction across the evaluated attention backbones. The backbones are FLUX.1-dev (228 DiT attention layers, dual-stream MMDiT), SDXL 1.0 (560 UNet

attention layers across CrossAttn Down/Mid/Up blocks), and Qwen-Image-20B (single-stream DiT). The cross-architecture queue is complete for SC-40 and the 70-concept identity suite (DreamBooth-30, SCIdentity, and CC101-20), covering 110 concepts on each non-FLUX backbone with matched LoRA and LITHE training and inference. The SC-40 cross-architecture rows are single-seed coverage runs; the identity suite reports seeds 1 and 7 in Tab. 4 (GPT-5.1-only extends to seed 42 in App. G). On SC-40 LITHE trails LoRA by 0.06–0.24 GPT-avg points across the three backbones; on the aggregate identity suite (Tab. 4) the LLM-avg gap is 0.02 on SDXL and 0.21 on Qwen-Image. Recalibrating the global σ constants or the rank does not close the Qwen gap (App. D.9). Two structure-aware remedies do: an auto-derived depth schedule for σ_B recovers about 40% of the strict-identity deficit at 60 stored floats per adapter, and a codebook learned from Qwen weights closes it at the cost of a 0.75 MiB per-backbone sidecar, amortized across a catalogue. FLUX remains the main setting for detailed ablations.

Table 4: **Cross-architecture identity-suite panel**, r=16, mirroring Tab. 1's metric set. **LLM-avg** (GPT-5.1 + Claude Sonnet 4.6 + Gemini 2.5 Flash) entries are mean±std over seeds 1 and 7; the GPT-5.1-only seed-42 extension and per-judge breakdowns are in App. G. **Per-task** is cold deploy-time MB, for the bf16 r=16 LoRA or the LZMA-9 LITHE index stream.

Backbone	Method	Per-task (MB)	LLM-avg ↑	DreamSim ↑	CLIP-T ↑
SDXL 1.0 (UNet cross-attn)	LoRA	46.4	4.02±.02	**.74±.00**	.24±.00
	LITHE (Ours)	**1.04**	4.00±.03	.65±.01	**.26±.00**
Qwen-Image-20B (single-stream DiT)	LoRA	47.2	**4.18±.02**	**.60±.01**	.26±.00
	LITHE (Ours)	**1.05**	3.97±.02	.52±.00	.26±.00

5 Composition and Serving Microbenchmarks

5.1 Host-side composition

***N*-LoRA composition cost.** With pre-decoded sA/sB tensors resident on GPU, composing N experts is a host-side pointer/list update to the per-layer expert list. LITHE's list construction takes ∼1.9 ms regardless of N, and the per-LoRA predecoded-wrapper swap takes ∼1.8 ms vs. ∼0.9 s for a full LoRA swap on FLUX.1-dev (App. D.8). End-to-end at 512^2/28-step, LITHE is 1.09×/1.04× faster than serial fuse at N=1/N=5 (Tab. 32, App. J).

Same-batch composition galleries. Eq. 3 defines the per-sample route; §5.2 reports the capacity sweeps. Qualitative 2-LoRA and 3-LoRA contact sheets are in App. O (Figs. 6, 8). At equal-weight composition, subject identity prevails and style renders as a subtle tint.

5.2 Same-batch heterogeneous routing

Index-native deployment. The deployed adapter stores only the per-subject integer indices; pools A/B are rebuilt at process start in ∼50 ms (App. D.1). A matched-rank LoRA wraps the same logical adapter (304 Kohya fused-QKV or 494 Diffusers Linears). It takes 153.3 MB of bf16 on disk and 171.8 MB resident in Diffusers split form, and a full disk swap takes ∼900 ms from cold CephFS. The LITHE payload has three forms. The disk form averages 3.23 MB of LZMA-9 indices, the bit-packed GPU form measures 4.19 MB per adapter slot (4.10 MB logical), and the raw int16 cold-swap blob in Diffusers split-QKV layout takes 8.05 MB. The residency form packs 7 9-bit indices per uint64 word (Kohya fused-QKV form). A 10,000-slot pool is verified end-to-end at 75.70 GB peak, including 33.76 GB of base weights, with an 11,000-slot int16 sanity run and OOM at 12,000 (App. D.3). Tab. 5 traces the $N_{\text{res}} \times B_{\text{serv}}$ trade-off over sub-80 GB points. Decoded-factor mode caps at 156 residents during a B=14, 1024^2 render and at 291 without rendering, while bit-packed indices reach 10,000. Hot-swap takes 4.56 ms warm and ∼10 ms cold, versus a ∼900 ms serial-fuse cycle whose per-swap cost is CPU-side deserialization and does not change with cache warming (App. T). In decoded mode the wall-time reduction comes from graph compilation, and in index-residency mode the residency cap comes from the codebook-shared decode cache (App. D.7).

Same-batch heterogeneous routing. Eq. 3 is realised via a Triton SGMV-fused kernel [17] (App. C.7); the production wall-time is in Tab. 2 (§4.3). Compile is one-time since adapter switch is an index update on a fixed pool.

Table 5: **Hybrid $N_{\text{res}} \times B_{\text{serv}}$ operating points, 1×A100 80 GB.** P2–P5 are 512^2/16-step; P6 is the 1024^2/28-step anchor matched to Tab. 2 (bit-packed index-residency mode at the verified 10,000-slot point). Allocator-boundary points are in the appendix. Full breakdown with decode/install/forward latencies in App. D.5, Tab. 13.

Point	N_{res}	B_{serv}	VRAM peak	ms/sample
P2 (high residency, 512^2/16)	8,000	4	70.26 GB	2,029
P3 (balanced, 512^2/16)	4,000	8	56.24 GB	1,985
P4 (high batch, 512^2/16)	1,000	14	47.81 GB	1,963
P5 (max batch, 512^2/16)	4	14	43.65 GB	1,962
P6 (1024^2/28 anchor)	10,000	14	75.70 GB	13,040

Limitations. LITHE trades strict-identity and target-fidelity for deploy compression and residency: LoRA remains the strongest strict-fidelity anchor in the per-axis diagnostics of App. I. The index-residency mode costs $\sim$37% more per-sample wall time than decoded compiled mode and holds $\sim 64\times$ more residents than resident LoRA. On Qwen-Image-20B the FLUX-tuned codebook constants leave a cross-backbone LLM-avg gap of 0.21 that needs a per-backbone codebook adjustment to close (App. D.9). SDXL transfer is closer, with an LLM-avg gap of 0.02. Reported scope is image-diffusion personalization; video and VLM adapter serving are out of scope.

6 Conclusion

LITHE is a deterministic Hadamard-indexed adapter representation in which a single $\sim$3.2 MB index stream serves two modes. A decoded compiled mode runs within 0.8% of an optimized LoRA+compile baseline. A bit-packed index-residency mode holds 10,000 adapter slots on one A100 80 GB and routes 14 heterogeneous adapters per 1024^2/28-step FLUX.1-dev forward. The construction is byte-exact reproducible from H_9 and the fixed Hadamard projection Φ alone (App. C.3). The main quality trade-off is strict identity and target fidelity, with the LLM-avg gap to LoRA at most 0.18 on the FLUX identity benchmarks. Transfer to Qwen-Image needs a per-backbone codebook adjustment (App. D.9). We position LITHE as a deployment representation rather than a universal compression scheme.

Acknowledgements and Disclosure of Funding

This work has been supported by the National Natural Science Foundation of China (No. 62503322), the AI for Science Seed Program of Shanghai Jiao Tong University (Project No. 2025AI4SQY06), the Shanghai Jiao Tong University Xiaomi Scholar Fund, the Shanghai Municipal Special Program for Basic Research on General AI Foundation Models (Grant No. 2025SHZDZX026D18), and the Shanghai Magnolia Funding Pujiang Program (No. 23PJ1404400 and No. 24PJD047). The authors declare no competing financial interests or relationships outside the submitted work that could be perceived as influencing it.

References

[1] Kihyuk Sohn, Lu Jiang, Jarred Barber, Kimin Lee, Nataniel Ruiz, Dilip Krishnan, Huiwen Chang, Yuanzhen Li, Irfan Essa, Michael Rubinstein, Yuan Hao, Glenn Entis, Irina Blok, and Daniel Castro Chin. StyleDrop: Text-to-image synthesis of any style. In *NeurIPS*, 2023.

[2] Junyao Gao, Yanan Sun, Yanchen Liu, Yinhao Tang, Yanhong Zeng, Ding Qi, Kai Chen, and Cairong Zhao. StyleShot: A snapshot on any style. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 48(2):1215–1228, 2026. doi: 10.1109/TPAMI.2025.3610614.

[3] Haofan Wang, Matteo Spinelli, Qixun Wang, Xu Bai, Zekui Qin, and Anthony Chen. InstantStyle: Free lunch towards style-preserving in text-to-image generation. *arXiv preprint arXiv:2404.02733*, 2024.

[4] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. DreamBooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *CVPR*, 2023.

[5] Nupur Kumari, Bingliang Zhang, Richard Zhang, Eli Shechtman, and Jun-Yan Zhu. Multi-concept customization of text-to-image diffusion. In *CVPR*, 2023.

[6] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. In *ICLR*, 2023.

[7] Viraj Shah, Nataniel Ruiz, Forrester Cole, Erika Lu, Svetlana Lazebnik, Yuanzhen Li, and Varun Jampani. ZipLoRA: Any subject in any style by effectively merging LoRAs. In *ECCV*, 2024.

[8] Yuchao Gu, Xintao Wang, Jay Zhangjie Wu, Yujun Shi, Yunpeng Chen, Zihan Fan, Wuyou Xiao, Rui Zhao, Shuning Chang, Weijia Wu, Yixiao Ge, Ying Shan, and Mike Zheng Shou. Mix-of-Show: Decentralized low-rank adaptation for multi-concept customization of diffusion models. In *NeurIPS*, 2023.

[9] Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In *COLM*, 2024.

[10] Yarden Frenkel, Yael Vinker, Ariel Shamir, and Daniel Cohen-Or. Implicit style-content separation using B-LoRA. In *ECCV*, 2024. doi: 10.1007/978-3-031-72684-2_11. URL https://arxiv.org/abs/2403.14572.

[11] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *NeurIPS*, 2023.

[12] Yixiao Li, Yifan Yu, Chen Liang, Nikos Karampatziakis, Pengcheng He, Weizhu Chen, and Tuo Zhao. LoftQ: LoRA-fine-tuning-aware quantization for large language models. In *ICLR*, 2024.

[13] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *ICML*, 2024.

[14] Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based random matrix adaptation. In *ICLR*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

[15] Klaudia Balazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. LoRA-XS: Low-rank adaptation with extremely small number of parameters. In *ECAI*, 2025. doi: 10.3233/FAIA251185. URL https://arxiv.org/abs/2405.17604.

[16] Xun Wu, Shaohan Huang, and Furu Wei. Mixture of LoRA experts. In *ICLR*, 2024.

[17] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. In *MLSys*, 2024.

[18] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. SLoRA: Scalable serving of thousands of LoRA adapters. In *MLSys*, 2024.

[19] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

[20] Black Forest Labs. FLUX. https://github.com/black-forest-labs/flux, 2024.

[21] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.

[22] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*, 2023.

[23] Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3274–3287, 2023. doi: 10.18653/v1/2023.eacl-main.239.

[24] Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. LoRA-FA: Efficient and effective low rank representation fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023.

[25] Shih-Ying Yeh, Yu-Guan Hsieh, Zhidong Gao, Bernard B. W. Yang, Giyeong Oh, and Yanmin Gong. Navigating text-to-image customization: From LyCORIS fine-tuning to model evaluation. In *ICLR*, 2024. URL https://arxiv.org/abs/2309.14859.

[26] Shyam Marjit, Harshit Singh, Nityanand Mathur, Sayak Paul, Chia-Mu Yu, and Pin-Yu Chen. DiffuseKronA: A parameter efficient fine-tuning method for personalized diffusion models. In *WACV*, pages 3529–3538, 2025. doi: 10.1109/WACV61041.2025.00348.

[27] Ligong Han, Yinxiao Li, Han Zhang, Peyman Milanfar, Dimitris Metaxas, and Feng Yang. SVDiff: Compact parameter space for diffusion fine-tuning. In *ICCV*, 2023.

[28] Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. Controlling text-to-image diffusion by orthogonal finetuning. In *NeurIPS*, 2023.

[29] Vera Soboleva, Aibek Alanov, Andrey Kuznetsov, and Konstantin Sobolev. T-LoRA: Single image diffusion model customization without overfitting. In *AAAI*, volume 40, pages 9051–9059, 2026. doi: 10.1609/aaai.v40i11.37861.

[30] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Wei Wei, Tingbo Hou, Yael Pritch, Neal Wadhwa, Michael Rubinstein, and Kfir Aberman. HyperDreamBooth: HyperNetworks for fast personalization of text-to-image models. In *CVPR*, pages 6527–6536, 2024.

[31] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, 2017.

[32] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. SoundStream: An end-to-end neural audio codec. *IEEE/ACM TASLP*, 30:495–507, 2022. doi: 10.1109/TASLP.2021.3129994.

[33] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=ivCd8z8zR2.

[34] Jian Jiang and Oya Celiktutan. Neural weight search for scalable task incremental learning. In *WACV*, 2023. doi: 10.1109/WACV56688.2023.00144. URL https://arxiv.org/abs/2211.13823.

[35] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *ICLR*, 2023.

[36] Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *ICML*, pages 12284–12303, 2024.

[37] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even better LLM quantization with Hadamard incoherence and lattice codebooks. In *ICML*, 2024.

[38] Han Guo, Philip Greengard, Eric P. Xing, and Yoon Kim. LQ-LoRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *ICLR*, 2024. URL https://arxiv.org/abs/2311.12023.

[39] Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. QA-LoRA: Quantization-aware low-rank adaptation of large language models. In *ICLR*, 2024. URL https://arxiv.org/abs/2309.14717.

[40] James Liu, Guangxuan Xiao, Kai Li, Jason D. Lee, Song Han, Tri Dao, and Tianle Cai. BitDelta: Your fine-tune may only be worth one bit. In *NeurIPS*, 2024. doi: 10.52202/079017-0434. URL https://arxiv.org/abs/2402.10193.

[41] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[42] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 23(120):1–39, 2022.

[43] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[44] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

[45] John H. Conway and Neil J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer, 3rd edition, 1999. doi: 10.1007/978-1-4757-6568-7.

[46] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. Part II DOI: 10.1002/j.1538-7305.1948.tb00917.x.

[47] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006. doi: 10.1002/047174882X.

[48] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *ICLR*, 2024.

[49] Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-Image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

[50] Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. DreamSim: Learning new dimensions of human visual similarity using synthetic data. In *NeurIPS*, 2023.

[51] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763, 2021.

Supplementary Material

This appendix contains supplementary material for the main text: the heuristic rate-scaling calibration and Monte Carlo sanity check (Sec. B), the multi-seed extension (Sec. F), the cross-architecture benchmark tables (Sec. G), baseline coverage and unsupported FLUX ports (Sec. H), the auxiliary per-subject and per-axis diagnostic slices (Sec. I), the embedding-metric cross-check panel (Sec. K), the index-native deployment plus $N_{\max}$ co-residency stress test, fused Triton micro-bench, end-to-end $10{,}000$-slot hot-swap verification, and hybrid $N \times B$ operating-point sweep (Sec. D), the composition-cost table (Sec. J), the per-subject visual catalogue (Sec. N), implementation details and the trainer-stack algorithm (Sec. C), benchmark dataset and prompt details (Sec. L), the three-judge LLM protocol (GPT-5.1, Claude Sonnet 4.6, Gemini 2.5 Flash) with verbatim per-task prompts (Sec. M), and the same-batch per-sample routing sanity check (Sec. E). The final sections report a full-scale storage-matched comparison (Sec. Q), repeated-run confidence intervals (Sec. R), a blinded human evaluation (Sec. S), a comparison against modern paging systems with a capacity study (Sec. T), and the migration cost of existing LoRA catalogues (Sec. U).

A Limitations and broader impact

Limitations. LITHE trades some strict identity and target-fidelity for deploy compression and serving scale, and LoRA remains the strongest strict-fidelity anchor in the headline tables. On non-FLUX backbones (Qwen-Image-20B, SDXL 1.0; §4.5, Tab. 4), the FLUX-tuned codebook constants σ_A^{cb} and σ_B^{cb} leave a measurable subject-identity gap on Qwen-Image. No global recalibration of these constants closes it; the structure-aware remedies of App. D.9 do, at the cost of a per-backbone adjustment. Our evaluated scope is image-diffusion personalization. Video generation and vision–language model adapter serving are out of scope for the reported numbers.

Three of these trade-offs are quantified in this appendix. (i) Migrating an existing catalogue into LITHE needs a short per-adapter fine-tune. The cost is bounded (App. U), and the post-hoc encode already serves at zero training. (ii) Adapters trained into a compact form, LITHE included, sit within 0.10 GPT-avg of the dense $r{=}16$ anchor (App. Q), with the residual gap concentrated on the strict-identity axis (per-benchmark SI slices, App. I). (iii) A backbone whose weight statistics diverge from FLUX may need a per-backbone codebook adjustment, either an auto-derived depth schedule for σ_B or a codebook learned from the target backbone (App. D.9). Three further limitations bound the claims. (iv) Each 64-value block is reconstructed inside a fixed 9-dimensional Hadamard subspace. This constraint limits a one-shot projection of a pretrained adapter, so LITHE is designed to be trained natively with straight-through estimation, or warm-started and briefly fine-tuned. (v) Quality scores rest on closed LLM judges and on two synthetic benchmarks, SCIdentity and SC-40. We cross-check them with a blinded human evaluation (App. S) and judge-independent embedding metrics, and we release the per-image judge outputs for re-scoring. (vi) The largest speedups (Fig. 1, Tab. 2) are measured against serial fuse-and-render serving. Modern paging paths take 2.5–$4.5\times$ longer than LITHE (App. T), and dense factors fully resident on the same batched kernel match its speed. Against a resident set, LITHE therefore adds capacity at equal speed.

Broader impact and safeguards. Cheaper personalization can reduce storage and transfer costs for legitimate creative workflows, and it can also lower the cost of non-consensual identity or style imitation. LITHE does not release a new base generator. Deployment should still enforce consent-based reference ingestion, prompt and reference moderation, and provenance or watermarking policies at the application layer; in particular, identity-personalization services must gate adapter creation on verified subject consent. Because LITHE is a deployment representation rather than a generator, the responsible-deployment story is largely retained from ordinary LoRA serving: adapters remain opt-in artifacts trained per subject by the operator that onboards them, so the operator controls which concepts enter the catalogue; revoking a concept is deletion of its index stream; and any generation-time watermarking or provenance policy applied by the host backbone is unaffected, since LITHE changes only how the adapter is stored and routed, not the backbone that renders the image.

B Heuristic rate-scaling calibration and Monte Carlo sanity check

Setup recap. Let $x \in \mathbb{R}^B$ be drawn from $\mathcal{N}(0, \sigma^2 I_B)$. For an L-level RVQ encoder over codebooks of size K, with total rate $R{=}L \log_2 K$ bits/block, the reconstruction is $\hat{x} = \sum_{\ell=1}^{L} c_{k_\ell}^{(\ell)}$, $k_\ell = \arg\min_k \|r_{\ell-1} - c_k^{(\ell)}\|^2$, $r_0{=}x$, $r_\ell{=}r_{\ell-1} - c_{k_\ell}^{(\ell)}$.

Calibration sketch for the main-text rate scaling. (i) The single-letter Gaussian rate-distortion function is $R(D) = \frac{1}{2} \log_2 \sigma^2/D$ for an $\mathcal{N}(0, \sigma^2)$ scalar source [46, 47]; over a B-dimensional block, $R(D) = \frac{B}{2} \log_2(B\sigma^2/D)$, hence

$$D \geq B\sigma^2 \cdot 2^{-2R/B}. \tag{4}$$

(ii) Applying this to an L-level RVQ with total rate $R = L \log_2 K$ gives the exponent used in the calibration. (iii) Finite codebooks, greedy residual assignment, and the Hadamard sign shape add constants that are not predicted by this expression; the Monte Carlo table below reports those constants empirically rather than treating them as a tight theorem.

Table 6: Monte Carlo verification for the rate-scaling sanity check ($N{=}10^4$, $B{=}64$, $\sigma{=}0.06$, single-level random Gaussian codebook).

K	R/B	$D_{\text{Shannon}} = B\sigma^2 2^{-2R/B}$	D_{MC}	G_{emp}
16	0.0625	0.21128	0.33330	1.578
64	0.0938	0.20232	0.31383	1.551
256	0.1250	0.19374	0.28404	1.466
512	0.1406	0.18959	0.27422	1.446
1024	0.1562	0.18553	0.26357	1.421

Twin-codebook MC. At $(\sigma_A, \sigma_B, \tau){=}(0.0625, 0.0025, 0.02)$, $K{=}512$, $L{=}3$, $B{=}64$, $N{=}10^4$: $D_{\text{single,A}}/(B\sigma_A^2) = 0.680$, $D_{\text{single,B}}/(B\sigma_B^2) = 73.97$, $D_{\text{twin,A}}/(B\sigma_A^2) = 1.450$, $D_{\text{twin,B}}/(B\sigma_B^2) = 1.43$. Sum-of-relative-distortions: single 74.65 vs twin 2.88, giving an observed heuristic ratio $D_{\text{single}}/D_{\text{twin}}{=}25.95\times$. Empirical LITHE-on-FLUX.1-dev twin/single σ_A recon ratios across four Linear-shape probes: $\{20.1, 22.5, 22.1, 20.9\}\times$. The 1.15–1.29$\times$ shortfall to the $\sim$26$\times$ heuristic estimate is consistent with a second-order Voronoi-shape correction $G_{\text{Had}}^{(2)} - G_{\text{rand}}^{(2)}$; this is an order-of-magnitude sanity match, not a tight prediction. A higher-resolution Monte Carlo run ($N{=}10^5$, multiple $B \in \{32, 64, 128, 256\}$, with 95% bootstrap CIs on G_{emp} and on $D_{\text{single}}/D_{\text{twin}}$) is left for future theory-focused work.

Hadamard second-moment ratio (static empirical calibration). At $K{=}512$, $B{=}64$, MC with $N{=}10^4$, $\sigma{=}0.02$: $G_{\text{Had}}(64){=}1.706$, $G_{\text{rand}}(64){=}1.434$, $\rho_{\text{Had}}^{\text{static}}{=}G_{\text{Had}}/G_{\text{rand}}{=}1.190$. Empirical LITHE training-loss ratio (Hadamard over matched-Gaussian, trained with the straight-through estimator (STE)): $0.580/0.632{=}0.918$, fit by STE compensation factor $\alpha \approx 0.23$ in $D_{\text{STE}} \leq D_{\text{static}}(1 - \alpha)$.

C Implementation details

C.1 Hyperparameters

Table 7 reports every hyperparameter required to reproduce the headline cells of Table 1 in the main text. All cells share the same diffusion-LoRA reference recipe; the LITHE-specific knobs sit below the dividing rule.

C.2 Precision policy for closed-form Hadamard search

LITHE does *not* train the whole diffusion backbone in fp32. The frozen backbone, VAE, text encoders, latents, and activations follow the backbone-specific mixed-precision recipe (bf16 in our reported runs). The exception is the small set of trainable LoRA proxy tensors and the frozen codebook/search tensors, which remain fp32 master tensors. At each forward pass, the reconstructed LoRA factors are cast to the activation dtype before the linear operations.

Table 7: Training hyperparameters. The upper block is shared by LoRA and LITHE; the lower block is LITHE-specific. Values come from the metadata that the Kohya sd-scripts trainer saves in each FLUX.1-dev checkpoint and apply to every reported FLUX.1-dev adapter.

Hyperparameter	Value
Backbone	diffusion transformer backbone (frozen, bf16)
Trainer	Kohya sd-scripts FLUX LoRA trainer
Target modules	304 Linears (fused QKV): attention, MLP, and modulation projections of all double and single blocks
Rank r	$\{4, 8, 16, 32, 64\}$, r=16 default
LoRA α	r (i.e. α/r=1)
Optimiser	AdamW (β_1=0.9, β_2=0.999, ε=10^{-8})
Learning rate	1×10^{-4}
LR scheduler	constant, no warmup (identity benchmarks); cosine, 100 warmup steps (SC-40)
Max steps	500 (identity benchmarks); 750 (SC-40)
Train batch size (per device)	1
Gradient accumulation	1
Effective batch	4 (4 GPUs)
Mixed precision	bf16 backbone/autocast; fp32 LoRA/search masters
Timestep sampling	shift sampling, discrete flow shift 3.0
Loss weighting / guidance	uniform / 1.0
Resolution	512×512
Centre-crop / flip	on / off
Prior preservation	off
Hardware	4×A100-80GB per adapter
Codebook size K	$\{2, 4, \ldots, 1024, 2048\}$ (powers of two), K=512 default
Block size B	$\{16, 32, 64, 128, 256, 512\}$, B=64 default
Number of RVQ levels L_{rvq}	$\{1, 2, 3, 4, 5, 6\}$, L_{rvq}=3 default
Hadamard construction	deterministic H_9, Φ with fixed twin scales
Codebook update	frozen (no EMA, no SGD)
STE	identity backward (Bengio [41])
Lazy re-quantisation period	every N=10 forwards (default); drift-aware variant uses τ=10^{-4}
Pre-quantise init	on (Sec. C.5)
Cached STE expression	on (Sec. C.5)
LoRA/search master precision	fp32 (Sec. C.2)

This split is required by the closed-form Hadamard decoder. Search is performed by projecting each block onto Hadamard hyperplanes and bit-packing $\text{sign}(x^\top h_i)$ decisions. Rounding a projection near a hyperplane boundary can flip a sign bit and therefore change the selected codeword. In sanity checks, fp32 closed-form search exactly matches exhaustive search over the same Hadamard pools, while bf16 search can disagree on boundary cases. Keeping only the LoRA/search masters in fp32 has negligible memory footprint relative to the backbone and preserves the integer index trajectory that is ultimately used at deployment time.

C.3 Deterministic Hadamard configuration

The default codebook is the deterministic tuple $(U, \Phi, \sigma_A^{\text{cb}}, \sigma_B^{\text{cb}})$ with $(L_{\text{rvq}}, K, B) = (3, 512, 64)$. The construction is byte-exact and can be reproduced from the rules below without code; bit indices are 0-based (LSB at position 0).

Sylvester–Hadamard generator $U \in \{\pm1\}^{64\times9}$. Let $H_{\text{Syl}} \in \{\pm1\}^{64\times64}$ be the order-64 Sylvester–Hadamard matrix ($H_{\text{Syl}}[t, j] = (-1)^{\text{popcount}(t \,\text{AND}\, j)}$ for $t, j \in \{0, \ldots, 63\}$). We set $U = H_{\text{Syl}}[:, 0{:}9]$, i.e. the first nine columns of H_{Syl}. The columns of U are nine pairwise orthogonal ±1 vectors of length 64 ($U^\top U = 64\, I_9$).

Index sign-expansion $\sigma_k \in \{\pm1\}^9$. For codebook index $k \in \{0, \ldots, 511\}$ and bit position $q \in \{0, \ldots, 8\}$, $\sigma_k[q] = 2b_q(k) - 1$, where $b_q(k) \in \{0, 1\}$ is the q-th bit of k (a set bit maps to $+1$).

Codeword expansion $\Phi : \{\pm 1\}^9 \to \mathbb{R}^{64}$.

$$\Phi(\sigma_k) \;=\; \tfrac{1}{\sqrt{9}}\, U\, \sigma_k \;=\; \tfrac{1}{\sqrt{9}} \sum_{q=0}^{8} \sigma_k[q]\, U[:, q] \;\in\; \mathbb{R}^{64}.$$

Each entry is a normalised sum of nine ± 1 values, taking values in $\{-3, -7/3, \ldots, 7/3, 3\}$ rather than $\{\pm 1\}^{64}$; the image lies in the 9-dimensional Hadamard subspace $\mathrm{col}(U) \subset \mathbb{R}^{64}$, with unit per-coordinate variance under uniform σ_k. The map is injective on $\{\pm 1\}^9$ because U has full column rank, so the 512 codewords $c_k = \Phi(\sigma_k)$ are pairwise distinct and reproducible byte-exact from H_{Syl} alone (all nine signature bits contribute via U).

Per-side scales. $\sigma_A^{\text{cb}} = 0.06$ shared across $\ell \in \{1, 2, 3\}$; $\sigma_B^{\text{cb},\ell} \in \{0.0025, 0.001, 0.0005\}$ across the three B-side residual levels; both control the per-coordinate standard deviation of the deployed factor block. The deployable adapter stores only the per-block index streams over $k \in \{0, \ldots, 511\}$; pools $\mathcal{C}_A^{(\ell)}[k] = \sigma_A^{\text{cb}}\, c_k$ and $\mathcal{C}_B^{(\ell)}[k] = \sigma_B^{\text{cb},\ell}\, c_k$ are reconstructed from the rules above at process start ($\sim$50 ms one-shot CPU; $\sim$384 KiB per side fp32). The construction is deterministic, not learned, seeded, or transferred, so two independent implementations agree byte-for-byte on U, Φ, and the resulting codeword matrix. The K=256 setting is the half-codebook ablation (indices $0 \ldots 255$). We do not run K-means on oracle weights (cf. NWS [34]) because the deployment pool is the fixed Hadamard construction above. The Hadamard choice is a computational and deployment property rather than a quality lever: at the same rate and straight-through pipeline it stays within trial noise of both a learned K-means codebook and a random sign codebook (LITHE GPT-avg 4.013 vs. learned 4.027 and random 3.920, all inside the $\sim$0.09 band), while storing no table and admitting the closed-form decode of §3.2. A learned codebook would instead ship a per-backbone sidecar table ($\sim$0.75 MiB), amortised across a catalogue but not zero-storage. Trained LoRA blocks are close to isotropic. The fixed 9-dimensional Hadamard subspace captures 14.5% of their energy, against 14.1% for a random 9-dimensional subspace and 15.5% for the best one. Quality therefore comes from native training in the discrete space, and the choice of subspace matters little.

C.4 Lazy delta-trigger configuration

The lazy trigger has two variants. The simple period-based variant runs the residual VQ search every N forward passes (N=10 default) and reuses the cached indices in between. The drift-aware variant, used in the accelerated trainer stack (Sec. C.5), additionally re-runs the search early when the L2 norm of the proxy delta versus the last-quantised proxy crosses threshold τ. We use $\tau = 10^{-4}$ (absolute, in proxy units), tuned once on `db_cat` seed 42 and reused unchanged across all reported adapters.

C.5 Trainer stack: algorithmic recipe

The trainer stack composes four implementation ideas (i)–(iv) that keep the per-step trainer cost close to plain LoRA (App. C.6). Each idea is summarised below as a compact algorithmic block; the complete training-loop pseudo-code is in Algorithm 1.

(i) Mixed precision split. The diffusion backbone follows the backbone-specific bf16 mixed-precision recipe, while LoRA proxy tensors and deterministic Hadamard search tensors stay fp32 (Sec. C.2). bf16 rounding can flip sign bits near Hadamard hyperplane boundaries, so the closed-form Hadamard search must run in fp32.

(ii) Drift-aware lazy re-quantisation. See Sec. C.4; this is N=10 period plus an L2-drift early trigger.

(iii) Cached STE expression. On a triggered step we re-quantise: $\tilde{A}_\ell^{\text{cache}} \leftarrow \textsc{TwinRVQ}(A_\ell, \mathcal{C}_A)$ and the forward uses the standard straight-through expression

$$\tilde{A}_\ell + (A_\ell - A_\ell^{\text{detach}}), \quad \text{so } \nabla_{A_\ell} = \nabla_{\tilde{A}_\ell}, \;\; \text{forward value} = \tilde{A}_\ell. \tag{5}$$

On a cached (non-triggered) step we reuse $\tilde{A}_\ell^{\text{cache}}$ for the forward value but keep the zero-valued STE attachment $(A_\ell - A_\ell^{\text{detach}})$ in the graph. Thus the quantized value is not recomputed, while gradients still flow to the fp32 proxy tensors on cached steps.

(iv) Pre-quantise-init. On the very first forward we quantise the random LoRA-A init onto the deterministic Hadamard codebook before any gradient step, so the cached reconstruction exists from step 0 and the lazy machinery has nothing to "warm up".

Algorithm 1: LITHE systems/caching wrapper for the trainer loop. Indices in (i)–(iv) refer to the four implementation ideas; the mathematical RVQ core is expanded in Algorithm 2. TWINRVQ runs the nearest-codeword search of Sec. 3, exhaustive for the identity-benchmark adapters and closed-form for the SC-40 adapters. The two searches return the same indices.

Input : LoRA proxies $\{A_\ell, B_\ell\}_{\ell=1}^{L}$, deterministic Hadamard pools $\mathcal{C}_A, \mathcal{C}_B$ induced by $(H_9, \Phi, \sigma_A^{\text{cb}}, \sigma_B^{\text{cb}})$, period N, drift threshold τ, total steps T.

$\{\tilde{A}_\ell^{\text{cache}}, \tilde{B}_\ell^{\text{cache}}\} \leftarrow \text{TWINRVQ}(\{A_\ell, B_\ell\}, \mathcal{C}_A, \mathcal{C}_B)$; `// (iv) pre-quantise-init`

$\{A_\ell^{\text{lastQ}}, B_\ell^{\text{lastQ}}\} \leftarrow \{A_\ell, B_\ell\}$;

for $t \leftarrow 1$ **to** T **do**

  `// (ii) drift-aware lazy re-quantisation: period or L2 drift`

  if $t \bmod N = 0$ ***or*** $\max_\ell \|A_\ell - A_\ell^{\text{lastQ}}\|_2 > \tau$ ***or*** $\max_\ell \|B_\ell - B_\ell^{\text{lastQ}}\|_2 > \tau$ **then**

    $\{\tilde{A}_\ell^{\text{cache}}, \tilde{B}_\ell^{\text{cache}}\} \leftarrow \text{TWINRVQ}(\{A_\ell, B_\ell\}, \mathcal{C}_A, \mathcal{C}_B)$;

    $\{A_\ell^{\text{lastQ}}, B_\ell^{\text{lastQ}}\} \leftarrow \{A_\ell, B_\ell\}$;

    forward uses STE: $\tilde{A}_\ell^{\text{cache}} + (A_\ell - \text{detach}(A_\ell))$;

  else

    `// (iii) cached STE expression; no new quantizer search`

    forward uses $\tilde{A}_\ell^{\text{cache}} + (A_\ell - \text{detach}(A_\ell))$ and analogously for B_ℓ ;

  end

  rectified-flow loss on $y{=}Wx{+}(\hat{B}_\ell\,\hat{A}_\ell)x\,\alpha/r$;

  AdamW step on fp32 $\{A_\ell, B_\ell\}$ masters ; `// (i)`

end

return *integer indices* $\{i_\ell^{A,(j)}, i_\ell^{B,(j)}\}$ *from final* TWINRVQ *call.*

C.6 Training-time overhead

The trainer stack of Sec. C.5 keeps per-adapter training cost close to plain LoRA. On an exclusively held single A100 at the 750-step recipe, LITHE adds 4.7% wall-clock and 0.8% peak GPU memory over plain LoRA (Tab. 8). The overhead comes from the periodic codebook reassignment. Reassigning every 50 steps brings the per-step cost level with plain LoRA at unchanged quality (GPT-avg 3.96 vs. 4.01, within trial noise; Tab. 35). A separate timing of the DreamBooth recipe on one subject (`db_cat`, single A100, 150 steps with the first 10 discarded) shows no overhead. LITHE took 1.61 s/step against 1.63 for plain LoRA at 512^2, and 3.89 against 4.07 at 1024^2. All scale constants are global, and no main-text number uses per-backbone calibration.

Table 8: **Training-time overhead of LITHE over plain LoRA** (750-step recipe, exclusive single A100). The default-cadence per-step overhead falls to zero when reassignment runs every 50 steps.

Measurement (750-step recipe)	Plain LoRA	LITHE
Wall-clock, exclusive single A100	26.5 min	27.8 min (+4.7%)
Seconds/step, default cadence	2.12	2.23 (+5.2%)
Seconds/step, reassign every 50	2.12	2.09 (level)
Peak GPU memory	36.97 GB	37.26 GB (+0.8%)

C.7 Detailed pseudocode

This subsection gives the self-contained pseudocode referenced from §3.2 (Algorithm 2) and §5.2 (Algorithm 3). Algorithm 1 is the systems/caching wrapper around the trainer. Algorithm 2 is the mathematical core: closed-form Hadamard decode, twin RVQ, and STE update. The serving algorithm uses the per-sample rank-stacked factors of Eq. 3 and the Triton SGMV heterogeneous forward in Algorithm 3.

Algorithm 2: LITHE mathematical core for per-concept training. Closed-form Hadamard decode and the asymmetric twin codebook are method-native; LazyRecomp is an optional index-stable memory ablation (the re-decode is index-stable within the T_{re} window since pools are frozen). The codebook is a 0 KB deterministic Sylvester–Hadamard construction for disk/network payload; only the index streams are deployed.

Input : Continuous LoRA factors $A \in \mathbb{R}^{r \times d_{\text{in}}}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$ initialized from a base model; refresh interval T_{re}; rank r, RVQ levels L, codebook size $K = 2^9$.

Constants : Sylvester–Hadamard generator $U \in \{\pm 1\}^{64 \times 9}$ (first 9 columns of order-64 Sylvester); sign-expansion $\sigma_k \in \{\pm 1\}^9$ of $k \in \{0, \ldots, 511\}$; codeword $c_k = \Phi(\sigma_k) = U\sigma_k / \sqrt{9} \in \mathbb{R}^{64}$; twin codebook scales $\sigma_A^{\text{cb}} = 0.06$, $\sigma_B^{\text{cb}} = \{0.0025, 0.001, 0.0005\}$ at levels $\ell = 1 \ldots L$; discrete pools $\mathcal{C}_A^{(\ell)}[k] = \sigma_A^{\text{cb}}\, c_k$, $\mathcal{C}_B^{(\ell)}[k] = \sigma_B^{\text{cb},\ell}\, c_k$ (deterministic, 0 KB on disk; small runtime tensors may be rebuilt at process start).

Output : Per-subject index streams $\{i_A^{(\ell)}\}_{\ell=1\ldots L}$, $\{i_B^{(\ell)}\}_{\ell=1\ldots L}$ over $\{0, \ldots, K-1\}$, packed at $\log_2 K = 9$ bits/symbol.

for *training step* $t = 1, \ldots, T_{\max}$ **do**
  if $t \bmod T_{\text{re}} = 0$ **then**
    `// Twin RVQ encode (per layer, per side).`
    for $\ell = 1, \ldots, L$ **do**
      Block-residualize $A^{(\ell)} = A - \sum_{\ell' < \ell} \hat{A}^{(\ell')}$; $i_A^{(\ell)} \leftarrow \arg\min_k \|A^{(\ell)} - \mathcal{C}_A^{(\ell)}[k]\|_2$;
      Block-residualize $B^{(\ell)}$ analogously over the $\sigma_B^{\text{cb},\ell}$-scaled pool;
      $i_B^{(\ell)} \leftarrow \arg\min_k \|B^{(\ell)} - \mathcal{C}_B^{(\ell)}[k]\|_2$;
    end
    `// Closed-form Hadamard decode (Eq. 2).`
    Reconstruct each block $\hat{a}_j = \sum_\ell \sigma_A^{\text{cb}}\, c_{i^{(\ell)}_{A,j}}$, $\hat{b}_j = \sum_\ell \sigma_B^{\text{cb},\ell}\, c_{i^{(\ell)}_{B,j}}$ via the Hadamard projection $c_k = U\sigma_k / \sqrt{9}$;
  `// Forward (LoRA-style discrete delta path).`
  $y \leftarrow Wx + (\hat{B}\,\hat{A})\, x \cdot \alpha / r$; `// W: frozen base weight; α: LoRA scaling`
  `// Backward (STE through frozen pools).`
  Receive $\partial\mathcal{L}/\partial y$; apply straight-through estimator $\partial\hat{A} \leftarrow \partial A$, $\partial\hat{B} \leftarrow \partial B$ to push gradients onto the continuous proxies A, B ;
  if *LazyRecomp* **then**
    Discard $(\hat{A}, \hat{B})$ from the activation cache ;
    Re-decode from (i_A, i_B) on the backward pass ;
  Update A, B with AdamW ;
end

Algorithm 3: LITHE serving / inference (MoE same-batch heterogeneous routing). Step (A) is the 1.171 ms pre-decoded pointer-mode hot-swap; step (B) is the ∼19 ms host-side composition (nonnegative reported weights; rank offsets plus zero padding up to the batch max); step (C) is the same-batch heterogeneous forward (Triton SGMV with fp32 accumulate and bf16 storage; 1.09× over a Python loop, 1.17× over `bmm`). In the measured Diffusers fused path, LoRA weights are module-global and cannot route per-sample, so serving B different LoRAs collapses to B swap-and-render cycles, about ∼0.9 s per swap on FLUX.1-dev CephFS cold (App. D.8). LITHE keeps the B-fold parallelism.

Input : Pre-trained backbone weights W (e.g. FLUX.1-dev, 33.76 GB bf16); inventory of LITHE concept ckpts $\{c_e\}_e$ each carrying index streams $(i_A^{(\ell)}, i_B^{(\ell)})_{\ell=1\ldots L}$ over K-entry codebooks; incoming serving batch of B requests, request i specifying expert tuple $\mathcal{E}_i \subseteq \{c_e\}$ (e.g. subject+style+lighting) with per-axis weights $w_{a,i}$ and mixing coefficients α_a; resolution and inference schedule.

Resident : Hadamard-sign table H_9, expansion map Φ, and twin scales $\{\sigma_A^{\text{cb}}, \sigma_B^{\text{cb},\ell}\}$ rebuilt once at process start (∼50 ms one-shot CPU); shared backbone W (resident); pre-decoded expert pool $\{(\hat{A}_e, \hat{B}_e)\}_e$ (decoded on demand from the index streams).

`// (A) Pointer-level hot-swap (per request, per axis).`
for *request i activating expert e on axis a* **do**
  if *e already pre-decoded* **then**
    Set pointer $\mathcal{P}_a^i \leftarrow (\hat{A}_e, \hat{B}_e)$; `// 1.171 ms mean; pointer assignment, no PCIe`
  else
    Closed-form Hadamard decode (Eq. 2): $\hat{a}_{e,j} \leftarrow \sum_\ell \sigma_A^{\text{cb}}\, c_{i_{A,j}^{(\ell)}}$, $\hat{b}_{e,j} \leftarrow \sum_\ell \sigma_B^{\text{cb},\ell}\, c_{i_{B,j}^{(\ell)}}$ where $c_k = U\sigma_k/\sqrt{9}$; unflatten blocks into $\hat{A}_e, \hat{B}_e$;
    Cache $(\hat{A}_e, \hat{B}_e)$ as resident expert ; `// 304-Linear Kohya fused-QKV form`
  end
end

`// (B) Per-sample LoRA composition (MoE expert tuple, Eq. 3).`
for *wrapped linear ℓ, sample $i \in \{1, \ldots, B\}$* **do**
  Stack the active experts for sample i: $\hat{A}_i^{\text{stack}} \leftarrow \text{vcat}\left[\sqrt{\alpha_a w_{a,i}}\hat{A}_{e_a}\right]_{a\in\text{axes}(i)}$,
  $\hat{B}_i^{\text{stack}} \leftarrow \text{hcat}\left[\sqrt{\alpha_a w_{a,i}}\hat{B}_{e_a}\right]_{a\in\text{axes}(i)}$;
end

`// (C) Same-batch heterogeneous forward via Triton SGMV (Eq. 3).`
for *denoising step $s = 1, \ldots, S$* **do**
  for *wrapped linear layer ℓ* **do**
    $r_i \leftarrow \hat{A}_i^{\text{stack},\ell}\, x_i$; `// shrink: per-sample`
    $\delta y_i \leftarrow \hat{B}_i^{\text{stack},\ell}\, r_i$; `// expand: per-sample`
    $y_i \leftarrow W^\ell x_i + \delta y_i$; `// Triton SGMV`
  end
end

Output : Per-sample render $Y_i = \text{Backbone}(X_i; \delta_i)$ where $\delta_i = \hat{B}_i^{\text{stack}} \hat{A}_i^{\text{stack}}$, all B samples produced in one forward at the wrapped backbone's per-batch cost (no per-sample swap loop).

D Index-native deployment, storage curve, and serving stress tests

D.1 Index-native deployment construction

Both pool A and pool B are deterministic functions of the same Sylvester–Hadamard generator $U \in \{\pm 1\}^{64\times 9}$ and the fixed codeword expansion $c_k = U\sigma_k/\sqrt{9} \in \mathbb{R}^{64}$ (App. C.3): pool A scales every codeword by σ_A^{cb}=0.06, pool B by the level-stack $\sigma_B^{\text{cb},\ell} \in \{0.0025, 0.001, 0.0005\}$ at L=3 residual levels. The deployable adapter stores only the per-subject integer index streams; pool A and pool B are rebuilt at process start in ∼50 ms one-shot CPU. Bit-equivalence to the training

reconstruction is verified per layer: $\max|\Delta|$ across all 494 wrapped LITHE modules of FLUX.1-dev is 3.5×10^{-4} for pool A and 2.1×10^{-5} for pool B, both within bf16 numerical noise. We additionally verified bit-identity ($\max|\Delta| = 0.0$) under a concat-pool fallback (pool A and pool B concatenated to a 1024-entry single pool, 768 KiB shared) used during early infrastructure debugging.

D.2 Deploy-storage panel and amortisation curve

Table 9: Per-subject deploy-storage panel against PEFT competitors.

Method	Per-subject indices	Shared pool	Compression vs LoRA r=64
LoRA r=16	153.3 MB	—	4×
LoRA r=64	612.6 MB	—	1× (ref)
QLoRA NF4 (deploy-only accounting)	∼22 MB	—	∼28×
HyperDreamBooth (LiDB)	∼100 KB	shared hypernet	∼6,100×
LITHE (Ours) r=16 (LZMA-9, index-native)	**3.23** MB	**0** KB	**∼190×**

The QLoRA NF4 row in this appendix panel matches the Table 1 deploy-only accounting; the 179.4 MB framework-payload audit, which includes Diffusers/PEFT metadata, is a separate implementation-side payload measurement and is not included in the headline storage column.

Storage amortisation curve. For the LITHE r=16 multi configuration (the default), per-subject indices are uint16 vectors over a K=512-entry codebook, compressed end-to-end with LZMA at its maximum preset. The mean over the 110 LITHE r=16 adapters of our benchmarks (DreamBooth-30, CC101-20, SCIdentity, SC-40) is 3.23 MB per concept, against 7.18 MB of raw uint16 and a 4.03 MB 0-order entropy estimate. LZMA-9 falls below the independent-symbol estimate because it exploits cross-level and cross-position correlations (Tab. 10).

Table 10: Storage amortisation curve for LITHE r=16 with LZMA-9-compressed per-subject indices.

N subjects	per-subj footprint (LZMA-9)	ratio vs. LoRA r=64 (612.6 MB)
1	3.23 MB + 768 KiB pools ≈ 4.02 MB	∼152×
10	3.23 MB + 76.8 KiB amortised ≈ 3.31 MB	∼185×
100	3.23 MB + 7.68 KiB amortised ≈ 3.24 MB	∼189×
10,000	3.23 MB + ∼80 B amortised ≈ 3.23 MB	∼190×

The index-native deployment (0 KB pool) collapses the curve to a constant 3.23 MB per subject independent of N. The independent-symbol estimate is 4.03 MB (0-order entropy at K=512), so additional non-trivial compression below LZMA-9 would require modelling cross-block codeword structure beyond a generic LZ77+arithmetic coder.

Bundle-LZMA at deployment scale. A platform that stores N adapters together can concatenate their index streams and run a single LZMA-9 pass at the extreme preset. On the 30 DB-30 adapters, this costs about 3.03 MB per added adapter, a $\sim9\%$ reduction over per-adapter LZMA (3.31 MB) that comes from cross-concept structural redundancy. The gain saturates by $N\approx10$ because the LZMA dictionary spans about nine adapters. Two learned alternatives we tested (a cross-concept Categorical prior over $(\text{layer}, \text{matrix}, \text{level})$ tuples, and a bigram Markov model) both expanded the bitstream by $\sim20\%$ because cross-concept aggregation smears the per-concept peakedness that LZMA's intra-stream context window already captures within each adapter stream. LZMA is an off-the-shelf universal coder; we report it as the practical upper bound rather than as a LITHE-specific contribution.

D.3 $N_{\max}$ co-resident inventory stress test and decode microbench

We replicate real LITHE rank-16 discrete adapters round-robin atop bf16 FLUX.1-dev on an isolated A100 80 GB and probe two operating modes that trade swap latency against inventory size.

In *pointer-only mode*, the fully decoded LITHE factor pairs $(\hat{A}, \hat{B})$ for all wrapped FLUX Linears sit on GPU at about 171.8 MB per adapter (rank 16, bf16, Diffusers split-QKV form covering all 494 wrapped modules). Tiers N=$\{1, 10, 100\}$ all fit at incremental cost 171.8 MB per LoRA (constant to 0.1%). Binary search at the boundary gives N=200 fits at 68.12 GB and N=250 fits at 76.71 GB,

while N=300 runs out of memory. A finer search gives $\boldsymbol{N_{\max}}$**=291** co-resident decoded factor pairs without rendering and 156 during a B=14, 1024^2/28-step render. The pointer hot-swap latency in this mode is 1.171 ms mean ($p99 = 1.240$ ms over 200 swaps), about 770× faster than the ∼0.9 s serial-fuse swap. The main-text ∼1.8 ms predecoded wrapper-swap figure adds the surrounding host-side wrapper bookkeeping around this lower-level list update; the 0.04 ms dictionary-pointer path is an isolated assignment micro-path and is not the swap path reported in the main text.

In *decode-on-swap mode*, only the bit-packed 9-bit codebook index streams sit on GPU. We use the Kohya fused-QKV form (304 target Linears, 7 packed 9-bit indices per uint64 word, 63 of 64 bits used). The standalone logical packed tensors occupy 4.10 MB per adapter, and the measured allocator increment per slot, including buffers, is 4.19 MB. Residency claims below use measured end-to-end allocator totals rather than multiples of rounded logical bytes. An allocator-sensitive binary-search estimate gives $N_{\max}$=11,764 slots before wrapper construction and rendering buffers; the paper headline is the verified end-to-end N=10,000 pool. As an additional sanity bracket on the search ceiling, a direct end-to-end attach run at N=11,000 completes inside ≤80 GB peak (int16 packed indices), while N=12,000 runs out of memory during attach at 84.5 GB peak; we keep the conservative N=10,000 figure as the paper headline because it is the value verified together with full serving buffers, wrappers, and a real 1024^2/28-step render in App. D.4. A single fused Triton kernel decodes the resident pool into bf16 factor pairs for all 304 Kohya Linears. It streams packed uint64 words and, for each 64-value block, gathers the three selected rows from its side's fp32 level pools, which are rebuilt from the Hadamard construction at process start. It then sums these rows in registers and writes the result without intermediate buffers. Across 200 warm calls the fused kernel measures 4.56 ms mean ($p99 = 4.59$ ms, $\sigma = 0.012$ ms), and is bit-exact against an int16 multi-call reference ($\max|\Delta|$=0). The two modes give the platform a swap-latency vs. inventory-size knob; neither is an upper bound on same-forward batch size, which is reported separately as $B_{\max}$.

D.4 End-to-end residency → hot-swap → render verification

We additionally verify the index-only path end-to-end on a single A100 80 GB. We load N=10,000 bit-packed Kohya-form LITHE adapters onto the GPU; the run reports 75.70 GB PyTorch allocated/reserved total for the full process after loading the adapter pool, serving buffers, and already-resident 33.76 GB bf16 FLUX.1-dev base (Tab. 11). We then wrap the FLUX transformer with a per-sample LoRA shim (494 wrapped modules), and then alternate three real adapters from the 30-subject DreamBooth pool (`dog_002`, `cat_001`, `teapot_028`). Each request decodes the target adapter from its resident slot via the fused Triton kernel, maps Kohya fused QKV to Diffusers split form, installs the bf16 factor pairs on the wrapper, and renders one 1024^2/28-step image. The first decode-and-install costs 450 ms (Triton autotune); subsequent warm hot-swaps measure 9.55 ms and 9.36 ms (Tab. 12). The 1024^2/28-step forward is ∼14 s and is dominated by the diffusion denoising loop; the three rendered sanity-check samples show no obvious visual failure against the same prompts as the standalone-LoRA reference panel.

Table 11: **Allocator accounting for the verified 10,000-slot index-resident pool.** Values are the same PyTorch allocated/reserved accounting used for the serving tables; the incremental row is an end-to-end upper bound that includes the packed pool plus serving buffers and metadata.

Quantity	Allocated/reserved memory
Already-resident bf16 FLUX.1-dev base	33.76 GB
Full process after 10,000 slots, buffers, wrappers	75.70 GB
Increment over base	41.94 GB
Increment per resident slot (upper bound)	4.19 MB

Table 12: **End-to-end serving from a 10,000-adapter resident pool** (1×A100 80 GB, 1024^2/28-step, bf16 FLUX.1-dev, real LITHE checkpoints). Warm hot-swaps after the first Triton autotune are bounded by the fused decode, not by GPU memory.

Run	Adapter	Hot-swap (ms)	Render (s)
1 (cold)	`dog_002`	450.18	14.87
2 (warm)	`cat_001`	9.55	14.06
3 (warm)	`teapot_028`	9.36	14.06

D.5 Hybrid $N_{\text{res}} \times B_{\text{serv}}$ operating-point sweep

Tab. 13 sweeps five $(N_{\text{res}}, B_{\text{serv}})$ operating points to map the trade-off curve between large index-only inventory and large same-batch heterogeneity. The script is identical across points: build a resident pool of N_{res} bit-packed Kohya adapters; pick B_{serv} distinct real LITHE checkpoints from 8 DreamBooth subjects (cycling when $B_{\text{serv}}>8$); decode them via the fused Triton kernel; install per-sample LoRA on the wrapper; render one 512^2/16-step batch with B_{serv} different prompts. We pre-warm the Triton fused decode kernel once before the sweep (a single dummy decode+install+256^2/4-step forward, $\sim$1.6 s total) so that the first measured operating point reflects steady-state cost and not one-time JIT compilation plus autotune. All five points complete without out-of-memory failures. Decode-and-install scales sub-linearly with B_{serv} (10.5 ms at $B_{\text{serv}}=1$ up to 111.7 ms at $B_{\text{serv}}=14$): each adapter costs $\sim$7 ms (close to the 4.56 ms microbench plus a small per-Linear list update), amortised through a single Python loop over the B_{serv} active adapters. Per-sample forward latency improves slightly with larger B_{serv} (2,169 $\rightarrow$ 1,962 ms/sample, $\sim$10% batch-parallelism win on top of FLUX-dev's bf16 baseline). In our visual inspection, all 41 rendered images at the five hybrid points preserve their target identity.

Table 13: **Hybrid $N_{\text{res}} \times B_{\text{serv}}$ operating-point sweep, 1×A100 80 GB, 512^2/16-step.** N_{res} bit-packed Kohya adapters resident; B_{serv} per-sample LITHE adapter slots routed per-sample in one forward via the fused Triton decode and per-sample wrapper. VRAM columns are PyTorch allocated/reserved values rather than NVML physical-use readings; P1 is an allocator-accounting boundary case.

Point	N_{res}	B_{serv}	VRAM resident alloc/res	VRAM peak alloc/res	Decode+install	Forward	ms/sample
P1 (max residency)	11,000	1	80.23 GB	80.87 GB	10.5 ms	2.17 s	2,169
P2 (high residency)	8,000	4	67.64 GB	70.26 GB	30.4 ms	8.11 s	2,029
P3 (balanced)	4,000	8	51.33 GB	56.24 GB	58.0 ms	15.88 s	1,985
P4 (high batch)	1,000	14	39.36 GB	47.81 GB	100.6 ms	27.49 s	1,963
P5 (max batch)	4	14	36.12 GB	43.65 GB	111.7 ms	27.47 s	1,962

D.6 Engineered cold-swap path with raw int16 binary serialization

The naive deployment path stores indices in a serialised checkpoint file with the codebook as a sub-tensor and deserialises the whole object at swap time. We measure this at 265 ms mean for one full LITHE adapter on FLUX.1-dev. Removing this object deserialisation and the LZMA decompression by serialising indices as a contiguous int16 binary blob, plus a small header table mapping layer paths to byte offsets and target shapes, collapses the IO portion to about 0.1 ms (memory-mapped) for the 8.05 MB raw blob and 3 ms for host-to-GPU PCIe transfer. Replacing the per-layer Python decode loop with the fused Triton kernel of App. D.3 (4.56 ms mean over all 304 Kohya Linears) collapses the 111 ms decode portion an order of magnitude. Combined, the engineered cold-swap path measures $\sim$10 ms disk-to-GPU on a single A100, a $\sim$26× reduction over the naive path. For comparison, a pre-decoded factor pair already resident on GPU is hot-swappable in 0.04 ms via dictionary pointer assignment (the per-sample-routing wrapper raises this to 1.171 ms because it updates a list per wrapped Linear).

Default-vs-decoded-baseline visual-diff sanity. The codebook-shared decode cache used in Tab. 2 shares one decoded sA/sB tensor across requests of the same adapter inside a serving window rather than re-running the per-layer decode. To check that this caching does not silently drift the pixel-space output, we render the same 5 prompts at 512^2/20-step under the default LITHE kernel and under the per-request decoded baseline for the same adapter and seed: mean absolute pixel difference is 0.0017 and the max-pixel difference is 0.19, against a bf16 same-kernel re-render noise floor of 0.0001 mean. The numerical drift is therefore on the order of bf16 nondeterminism, not a kernel-level discrepancy.

D.7 Serving kernel component ablation

LITHE's default serving kernel combines two independent components: a *codebook-shared decode cache* (one H_9 Hadamard pool resolved once per layer, shared across resident adapters) and a *compiled inference graph* (the same default graph compilation applied symmetrically to both LITHE and the LoRA baseline in Tab. 2). We isolate each in a same-batch 1024^2/28-step $B=14$ run on 1×A100 80 GB (Tab. 14).

Table 14: **Component ablation of LITHE's default serving kernel**, 1024^2/28-step B=14 on 1×A100 80 GB, 3-trial mean±std for the bottom two rows. The codebook-shared decode cache alone gives no measurable wall reduction over the decoded SGMV baseline (180.06 vs 179.00 s, within trial noise); the entire 26% reduction to 133.4 s comes from the compiled inference graph applied symmetrically. The cache's value is in inventory residency (allowing $N_{\text{res}} \geq 10{,}000$ via shared H_9), not forward speed.

Variant	shared cache	compiled graph	wall (s)
LITHE decoded SGMV (no shared cache, no compile)	no	no	179.00
LITHE + shared cache, no compile	yes	no	180.06±0.24
LITHE (Ours, default)	yes	yes	**133.4±0.03**

D.8 Serial-fuse re-measurement: methodology and cross-server consensus

The serial-fuse baseline of Tab. 2 is the end-to-end LoRA hot-swap path: load the LoRA weight file → fuse it into the base weights → render → unfuse → unload, repeated per sample, cycling through a pool of 40 distinct rank-16 Plain-LoRA SC-40 checkpoints.

Cold-cache discipline. Heterogeneous community traffic in production rotates through catalogues much larger than the OS page cache, so per-request reads are first-touch from cold storage. We force this regime by issuing a system-wide eviction of the OS page, dentry, and inode caches immediately before each timed sample. The eviction itself takes ∼0.2–0.4 s and is excluded from the wall-time window (timing wraps only the `load` → `fuse` → `render` → `unfuse` → `unload` cycle). A weaker per-file page-eviction variant (leaving the inode/dcache and CephFS MDS cap caches warm) gives ∼10% lower wall and corresponds to the stable-catalogue regime in which the catalogue's inode metadata fits in the kernel inode cache; that variant is reported alongside the headline metadata-cold numbers below.

Cross-server consensus (4-server, A100-SXM4-80 GB, PyTorch ≥ 2.6, Diffusers 0.37.1). Tab. 15 reports the metadata-cold end-to-end serial-fuse wall on four hosts. The s4 column is the canonical paper number (lowest CPU load, smallest standard deviation, same hardware/stack as the resident LoRA loop, LoRA+compile, and LITHE columns of Tab. 2); the other three are reproducibility cross-checks.

Table 15: **Serial-fuse end-to-end wall, 4-server consensus, metadata-cold (system-wide page+inode cache eviction between samples).** 3-trial mean±std. Cross-host spread on the render-dominated 1024^2 row is $< 1.5\%$; load-dominated 256^2 tracks the host's CPU 1 min load average (top-1 correlation: smaller load ⇒ smaller wall and tighter std).

Server	CPU 1 m	evict (ms)	256^2/4 B=96	512^2/16 B=14	1024^2/28 B=14
s4 (canonical)	6.7	∼220 ms	**150.0±1.9** s	**43.3±0.3** s	**195.3±1.7** s
s3	10.7	∼100 ms	131.4±8.2 s	41.0±0.6 s	196.7±0.9 s
s2 (co-tenant)	26.1	∼300 ms	176.5±1.5 s	50.1±6.0 s	197.8±0.5 s
controller	41.8	∼320 ms	188.0±7.8 s	46.4±0.9 s	182.0±1.4 s

Component decomposition. The fuse-cycle micro-bench (load + fuse + unfuse, no render), 5-distinct-checkpoint 3-server consensus, splits the ∼0.9 s headline number used in §1 and §5.2 as load ∼780 ms, fuse ∼57 ms, unfuse ∼45 ms (CephFS cold), versus load ∼650 ms, fuse ∼57 ms, unfuse ∼43 ms (local-NVMe cold). The 130 ms delta on load is the CephFS first-touch metadata penalty; fuse and unfuse are storage-independent. Render is excluded here so the cycle is comparable across resolutions; the end-to-end numbers in Tab. 15 sum B such cycles plus per-sample render.

D.9 Qwen-Image per-backbone σ and rank diagnosis

The cross-architecture row reports a Qwen-Image SI gap relative to LoRA (§4.5). To test whether the FLUX-tuned codebook constants σ_A^{cb}=0.06, σ_B^{cb}=$\{0.0025, 0.001, 0.0005\}$ underfit Qwen-Image, we sweep σ_A, σ_B, RVQ depth, and rank on three DreamBooth-30 subjects (the cat, dog, and dog2 reference subjects) at five prompts each, scoring with a 2-LLM judge consensus (Claude Sonnet 4.6 and Gemini 3.1 Pro Preview, both via the OpenAI-compatible Wavespeed endpoint). GPT-5.1 scoring

on this identity queue uses a different rubric and is deferred. An empirical measurement on a Qwen-trained continuous LoRA gave mean $\mathrm{std}(\mathrm{lora}_A)$=0.063 (matching σ_A) and $\mathrm{std}(\mathrm{lora}_B)$=0.0034 (36% above the FLUX codebook scale), motivating a calibrated-σ_B probe.

Table 16: **Qwen-Image σ and rank diagnosis** (n=15 per row, 5 prompts $\times$ 3 DreamBooth-30 subjects). Both judges agree the FLUX-tuned constants in C1 are at or above every alternative on the σ_A, RVQ-depth, and rank axes; the σ_B axis is probed separately in Tab. 17.

	σ_A	σ_B	L	rank	Claude SI	Claude avg	Gemini SI	Gemini avg
C1 FLUX-default	0.06	0.0025	3	16	**3.13**	**4.11**	3.07	**4.56**
C2 σ_A low	0.04	0.0025	3	16	2.53	3.73	2.67	4.52
C3 σ_A x-low	0.02	0.0025	3	16	1.93	3.72	1.67	4.29
C4 L=1	0.06	0.0025	1	16	**3.13**	4.04	3.07	**4.56**
C5 rank 8	0.06	0.0025	3	8	2.07	3.92	2.07	4.40

The two judges agree on the same ranking: C1 (FLUX-default) is at or above every alternative on both Claude-avg and Gemini-avg, and is also the highest on Claude SI. Reducing σ_A (C2, C3) and reducing rank (C5) both visibly hurt SI, and dropping the multiscale RVQ chain on B to a single level (C4) is essentially tied with default.

σ_B probe and structure-aware remedies. We probe the σ_B axis on its own diagnostic stack: three subjects, n=15 images per cell, a three-LLM-judge panel, and two training seeds (Tab. 17). These values use their own protocol and are not comparable to the two-judge table above. A global σ_B re-tune is seed-inconsistent, so the Qwen-Image gap does not come from a miscalibrated global scale. It comes from a scale mismatch. Qwen's B-side factors grow about 2$\times$ with depth and exceed the fixed codebook radius in late layers, whereas FLUX's stay inside it. A single global constant can therefore fit at most one end of Qwen's network. Two structure-aware remedies, which adapt the codebook to Qwen's weight statistics, are seed-consistent.

Table 17: **σ_B probe and structure-aware remedies on Qwen-Image** (strict subject identity, two training seeds, n=15 per cell, three-judge panel). Only the two structure-aware remedies are seed-consistent.

Configuration	SI (seed-1 / seed-2) ↑	Stored cost
FLUX-default codebook (anchor)	3.67 / 3.68	none
Global σ_B re-tune	3.71 / 3.91	none; seed-inconsistent
Depth-scheduled σ_B, auto-derived	3.91 / 3.91	60 floats per adapter
Codebook learned from Qwen weights, identical rate	4.39 / 4.40	0.75 MiB per backbone

On the full 70-concept suite, the auto-derived depth schedule raises strict identity from 2.17 to 2.55 against dense LoRA at 3.04, recovering about 40% of the deficit at 60 stored floats per adapter. The codebook learned from Qwen weights closes the gap outright (SI 3.33 against dense LoRA's 3.04, replicated at a second training seed). Its cost is a 0.75 MiB per-backbone sidecar and a generic distance search in place of the Hadamard closed form, 135 against 52 μs per call in the search microbenchmark. On FLUX the same learned codebook stays within single-run protocol noise of the Hadamard pool, so the deterministic pool remains the default where weight statistics match, and backbones that diverge have two measured remedies. Tab. 18 gives the full-suite panel, re-scored with the three-judge stack of Tab. 17, so its absolute values differ slightly from Tab. 4. Retraining the default reproduces the submitted SI (2.13 existing vs. 2.17 rerun), which rules out a pipeline shift. No main-text number depends on this diagnostic.

Table 18: **Full-suite Qwen-Image panel** under the three-judge stack. The learned-codebook SI replicates at 3.32 on a second training seed.

Qwen-Image-20B, 70 concepts	SI ↑	Gap to dense (aggregate)	DINO ↑	CLIP-I ↑
Dense LoRA r=16	3.04	—	0.514	0.713
LITHE (Ours), FLUX-default codebook	2.17	0.24	—	—
LITHE (Ours), depth-scheduled σ_B	2.55	0.13	—	—
LITHE (Ours), codebook learned from Qwen	3.33	−0.08 (4.34 vs. 4.26)	0.536	0.724

D.10 Same-batch heterogeneous $B_{\max}$ sweep

We sweep the same-batch heterogeneous serving ceiling at 4-step distilled inference under the current Diffusers 494-module LITHE per-sample wrapper (measured on a single isolated A100 80 GB). At 512-pixel resolution the forward succeeds up to a batch of 120 at 83.62 GB reported PyTorch allocated/reserved peak (68.65 s wall, 1.75 images/sec) and runs out of memory at 128; at 256-pixel resolution we use the fully distinct-adapter B=96 setting reported in Tab. 2 as the conservative lower-quality serving point. Per-sample activation cost averages $\sim$0.40 GB at 512-pixel (extras / sample, declining from 0.56 GB at B=16 to 0.41 GB at B=120 as fixed overheads amortise) and $\sim$0.10 GB at 256-pixel (0.15 $\rightarrow$ 0.10 GB), matching FLUX's 4$\times$ token-count ratio between the two resolutions. Figure 3 plots peak VRAM versus batch size for both resolutions.

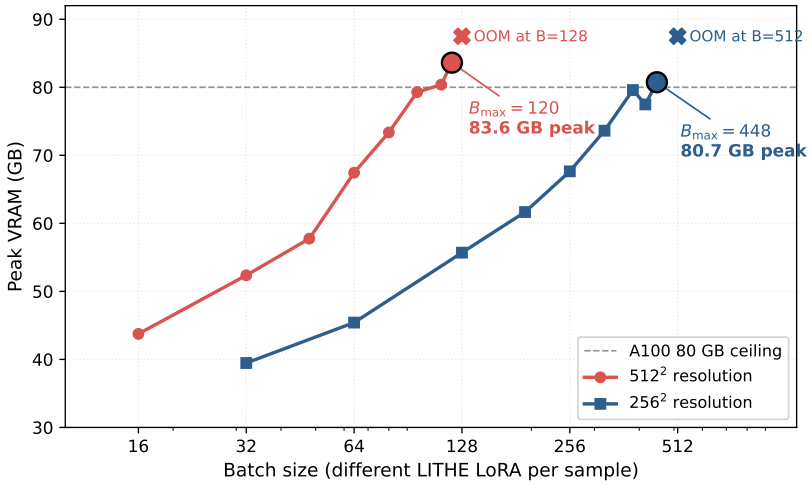


Figure 3: **Same-batch heterogeneous LITHE $B_{\max}$ sweep on FLUX.1-dev (single A100 80 GB).** Peak VRAM vs. batch size at 512^2 (red) and 256^2 (blue), with $B_{\max}$ at solid markers and smallest out-of-memory batch sizes at crosses (drawn above the 80 GB line; their height carries no measurement). PyTorch allocated/reserved accounting can slightly exceed physical device capacity even when the CUDA run completes; conservative sub-80 GB operating points are reported in Tab. 13. Per-sample VRAM cost matches FLUX's 4$\times$ token-count ratio between the two resolutions; peaks are reported by PyTorch on our isolated A100 run.

E Same-batch per-sample routing sanity check

This section gives the full protocol behind the routing-leakage sanity check of the same-batch routing in §5.2. The concern is whether per-sample LoRA selection, which applies a different decoded $(\mathrm{sA}_i, \mathrm{sB}_i)$ pair to each batch element, can leak the LoRA of one sample into the others within the same forward pass.

Setup. FLUX.1-dev, 4-step distilled inference, guidance 3.5, max sequence length 128, bf16 throughout, single A100. Two distinct LITHE ckpts: `cat_K256_B64_L3_r4_s42` (orange tabby DreamBooth subject) and `dog6_K256_B64_L3_r4_s42` (corgi puppy DreamBooth subject). To control for batch-size-dependent RNG drift, we pre-generate one packed latent per sample with a fixed CPU generator seed and pass them to the pipeline as initial latents, bypassing its internal noise sampling.

Modes. Three modes are run with identical pre-generated latents. (**A**) *Sequential single-LoRA*: two B=1 calls. A0 applies cat-LoRA alone to the cat-prompt with $\mathrm{noise}_{\mathrm{cat}}$; A1 applies dog-LoRA alone to the dog-prompt with $\mathrm{noise}_{\mathrm{dog}}$. (**B**) *Same-batch* B=2: one call with prompts [cat, dog], per-sample LoRA [cat-LoRA, dog-LoRA] and latents $[\mathrm{noise}_{\mathrm{cat}}, \mathrm{noise}_{\mathrm{dog}}]$. (**C**) *Same-batch* B=4 *cross*: prompts [cat, dog, dog, cat], LoRAs [cat, dog, cat, dog], latents $[\mathrm{noise}_{\mathrm{cat}}, \mathrm{noise}_{\mathrm{dog}}, \mathrm{noise}_{\mathrm{dog2}}, \mathrm{noise}_{\mathrm{cat2}}]$. Samples 2 and 3 are the cross-combinations: cat-LoRA pushed onto a dog scene, and dog-LoRA pushed onto a cat scene.

Diff metrics. We compute per-pixel uint8 difference on the in-memory PIL outputs of the pipeline (no JPEG roundtrip). Determinism floor was verified separately: re-rendering A0 with identical seed yields a bit-identical image ($\max$ uint8 diff $= 0$).

Table 19: Pixel diff statistics in greylevels (uint8). `eq`: fraction of pixels with diff $= 0$; `le1`: fraction ≤ 1; `le3`: fraction ≤ 3. Mode-A vs Mode-B/C diffs are at the bf16 reduction-order non-determinism floor; the single-subject sanity baseline (last row) is $\sim 100\times$ larger and shows the metric can detect genuine subject swaps.

Comparison	u8 max	u8 mean	eq	le1	le3
A0 vs B0 (cat, B=1 vs B=2)	111	1.44	0.21	0.65	0.96
A1 vs B1 (dog, B=1 vs B=2)	43	1.05	0.35	0.74	0.97
A0 vs C0 (cat, B=1 vs B=4)	141	1.29	0.45	0.79	0.92
A1 vs C1 (dog, B=1 vs B=4)	31	1.06	0.28	0.77	0.98
C2 (cat-LoRA $\times$ dog-prompt) vs single	49	0.59	0.59	0.88	0.98
C3 (dog-LoRA $\times$ cat-prompt) vs single	101	1.05	0.51	0.82	0.94
Sanity A0 vs A1 (different subjects)	200	96.6	0.001	0.003	0.008

Verdict. Same-batch differences are at the bf16 cuBLAS reduction-order non-determinism floor in this small deterministic check; visually, the batch-rendered samples show no obvious routing leakage relative to the single-LoRA renders (same subject, composition, lighting). The cross-combination samples (C2, C3) reproduce their single-LoRA single-sample counterparts at the same noise floor: in particular, sample 2 of the B=4 batch (cat-LoRA $\times$ dog-prompt) does *not* acquire the corgi-puppy features of sample 1's dog-LoRA. In this sanity check, each batch slot is conditioned only by the LoRA assigned to it, with no measured leakage from neighbouring slots beyond the bf16 noise floor. This checks the same per-sample routing code path used in the larger same-batch sweeps, in particular the measured B=96 *truly different* LITHE adapters at 256^2/4-step (16.43 s wall vs. serial 163.5 s, $9.95\times$ reduction at the distinct-adapter operating point; this is a different operating point from the B=96 row of Tab. 2, which is the decoded compiled mode) and the PyTorch-reported activation-memory boundary at B=120 (512^2) and the conservative fully distinct-adapter B=96 point at 256^2 (§5.2). Whether one fills those batch slots with a smaller round-robin checkpoint pool or truly distinct LoRAs is purely a function of the available adapter inventory; the per-sample routing path verified here is identical. Its implementation is in the code repository.

F Multi-seed benchmark breakdown

We run the main-text LoRA, LITHE, and variant-baseline configurations with three independent runs for the full FLUX benchmark coverage used in Table 1: DreamBooth-30, CC101-20, SCIdentity, and SC-40. Tables 20–23 report the headline metrics as mean $\pm$ sample standard deviation over the three run-level benchmark means: GPT-5.1 judge average, DreamSim, CLIP image-reference similarity (CLIP-I), CLIP text-image similarity (CLIP-T), Gemini 2.5 Flash judge average (Gemini-avg), and Claude Sonnet 4.6 judge average (Claude-avg). We intentionally omit the individual judge sub-scores (SI/PF/VQ/SBD/CN or TF/PF/VQ) here; those are available in the released per-image CSVs. Embedding metrics are printed to four decimals because their cross-run standard deviations are often below 10^{-3}; rounding them to two decimals would hide real but small seed variation.

Table 20: DreamBooth-30 multi-seed headline scores over three independent runs. Bold marks the best non-LoRA score in each metric column.

Method	GPT-avg	DreamSim	CLIP-I	CLIP-T	Gemini-avg	Claude-avg
LoRA	4.77 ± 0.01	0.6698 ± 0.0064	0.7664 ± 0.0019	0.2461 ± 0.0001	4.63 ± 0.04	4.28 ± 0.00
LITHE (Ours)	$\mathbf{4.64 \pm 0.02}$	$\mathbf{0.6041 \pm 0.0053}$	$\mathbf{0.7374 \pm 0.0071}$	0.2511 ± 0.0014	4.46 ± 0.02	$\mathbf{4.15 \pm 0.02}$
DoRA	3.98 ± 0.03	0.4949 ± 0.0067	0.6581 ± 0.0082	0.2532 ± 0.0014	3.69 ± 0.02	3.50 ± 0.02
VeRA	3.88 ± 0.04	0.4932 ± 0.0247	0.6552 ± 0.0100	0.2538 ± 0.0015	3.50 ± 0.10	3.36 ± 0.09
AdaLoRA	4.07 ± 0.03	0.4330 ± 0.0070	0.6535 ± 0.0062	$\mathbf{0.2633 \pm 0.0014}$	3.91 ± 0.02	3.66 ± 0.01
QLoRA NF4	4.12 ± 0.08	0.5596 ± 0.0217	0.6849 ± 0.0150	0.2525 ± 0.0016	3.82 ± 0.10	3.64 ± 0.07
DyLoRA	4.46 ± 0.03	0.5391 ± 0.0021	0.7050 ± 0.0026	0.2583 ± 0.0007	4.29 ± 0.01	4.02 ± 0.01
LoRA-FA	4.63 ± 0.03	0.5926 ± 0.0137	0.7342 ± 0.0062	0.2538 ± 0.0013	$\mathbf{4.48 \pm 0.03}$	4.14 ± 0.00
LoRA-XS	4.00 ± 0.02	0.4639 ± 0.0043	0.6511 ± 0.0085	0.2604 ± 0.0012	3.75 ± 0.02	3.51 ± 0.05
T-LoRA	4.32 ± 0.01	0.5623 ± 0.0098	0.7059 ± 0.0029	0.2628 ± 0.0009	4.18 ± 0.02	3.91 ± 0.04

Table 21: CC101-20 multi-seed headline scores over three independent runs. Bold marks the best non-LoRA score in each metric column.

Method	GPT-avg	DreamSim	CLIP-I	CLIP-T	Gemini-avg	Claude-avg
LoRA	4.45 ± 0.05	0.6226 ± 0.0044	0.7152 ± 0.0059	0.2413 ± 0.0026	4.22 ± 0.05	3.91 ± 0.01
LITHE (Ours)	$\mathbf{4.40 \pm 0.02}$	$\mathbf{0.5607 \pm 0.0015}$	$\mathbf{0.6884 \pm 0.0026}$	0.2520 ± 0.0009	$\mathbf{4.13 \pm 0.03}$	$\mathbf{3.88 \pm 0.01}$
DoRA	3.93 ± 0.02	0.4791 ± 0.0207	0.6207 ± 0.0096	0.2659 ± 0.0009	3.57 ± 0.07	3.53 ± 0.02
VeRA	3.81 ± 0.09	0.4901 ± 0.0059	0.6116 ± 0.0078	0.2570 ± 0.0028	3.36 ± 0.13	3.37 ± 0.06
AdaLoRA	4.13 ± 0.02	0.4254 ± 0.0059	0.6185 ± 0.0035	0.2693 ± 0.0007	3.98 ± 0.01	3.75 ± 0.02
QLoRA NF4	3.97 ± 0.05	0.5451 ± 0.0275	0.6428 ± 0.0163	0.2584 ± 0.0010	3.51 ± 0.05	3.60 ± 0.03
DyLoRA	4.24 ± 0.01	0.4816 ± 0.0071	0.6530 ± 0.0019	0.2573 ± 0.0004	3.94 ± 0.01	3.79 ± 0.05
LoRA-FA	4.35 ± 0.01	0.5431 ± 0.0061	0.6762 ± 0.0031	0.2517 ± 0.0022	4.06 ± 0.01	3.84 ± 0.04
LoRA-XS	4.04 ± 0.02	0.4487 ± 0.0179	0.6126 ± 0.0095	0.2670 ± 0.0011	3.74 ± 0.03	3.56 ± 0.04
T-LoRA	4.26 ± 0.04	0.5242 ± 0.0094	0.6550 ± 0.0077	$\mathbf{0.2717 \pm 0.0014}$	3.98 ± 0.02	3.83 ± 0.01

Table 22: SCIdentity multi-seed headline scores over three independent runs. Bold marks the best non-LoRA score in each metric column.

Method	GPT-avg	DreamSim	CLIP-I	CLIP-T	Gemini-avg	Claude-avg
LoRA	4.81 ± 0.01	0.6929 ± 0.0080	0.7431 ± 0.0059	0.2554 ± 0.0006	4.42 ± 0.01	4.22 ± 0.03
LITHE (Ours)	$\mathbf{4.66 \pm 0.04}$	$\mathbf{0.6485 \pm 0.0146}$	$\mathbf{0.7170 \pm 0.0073}$	0.2591 ± 0.0013	$\mathbf{4.22 \pm 0.04}$	$\mathbf{4.02 \pm 0.05}$
DoRA	3.74 ± 0.04	0.4789 ± 0.0051	0.5998 ± 0.0072	0.2367 ± 0.0029	3.32 ± 0.01	3.37 ± 0.02
VeRA	3.69 ± 0.03	0.4846 ± 0.0170	0.6056 ± 0.0178	0.2283 ± 0.0052	3.20 ± 0.13	3.28 ± 0.05
AdaLoRA	3.92 ± 0.01	0.4273 ± 0.0069	0.5782 ± 0.0097	0.2570 ± 0.0018	3.72 ± 0.03	3.77 ± 0.05
QLoRA NF4	3.99 ± 0.08	0.5760 ± 0.0165	0.6611 ± 0.0118	0.2316 ± 0.0020	3.54 ± 0.09	3.56 ± 0.04
DyLoRA	4.40 ± 0.02	0.5672 ± 0.0011	0.6751 ± 0.0012	$\mathbf{0.2641 \pm 0.0010}$	3.92 ± 0.02	3.76 ± 0.02
LoRA-FA	4.61 ± 0.04	0.6264 ± 0.0064	0.7064 ± 0.0039	0.2615 ± 0.0005	4.16 ± 0.05	4.00 ± 0.00
LoRA-XS	3.78 ± 0.06	0.4562 ± 0.0083	0.5824 ± 0.0114	0.2407 ± 0.0050	3.47 ± 0.07	3.42 ± 0.06
T-LoRA	4.06 ± 0.03	0.5348 ± 0.0074	0.6184 ± 0.0046	0.2536 ± 0.0056	3.83 ± 0.03	3.81 ± 0.02

Table 23: SC-40 multi-seed headline scores over three independent runs. Bold marks the best non-LoRA score in each metric column.

Method	GPT-avg	DreamSim	CLIP-I	CLIP-T	Gemini-avg	Claude-avg
LoRA	4.26 ± 0.02	0.5528 ± 0.0042	0.7206 ± 0.0022	0.2422 ± 0.0001	3.87 ± 0.03	3.68 ± 0.03
LITHE (Ours)	4.11 ± 0.03	0.5267 ± 0.0026	0.7105 ± 0.0007	0.2415 ± 0.0006	3.67 ± 0.03	3.53 ± 0.01
DoRA	3.44 ± 0.02	0.4624 ± 0.0084	0.6936 ± 0.0056	0.2355 ± 0.0034	3.11 ± 0.00	2.97 ± 0.01
VeRA	3.32 ± 0.09	0.4655 ± 0.0094	0.6973 ± 0.0042	0.2320 ± 0.0015	2.98 ± 0.06	2.82 ± 0.05
AdaLoRA	3.53 ± 0.01	0.4202 ± 0.0061	0.6752 ± 0.0031	0.2339 ± 0.0014	3.08 ± 0.01	3.05 ± 0.03
QLoRA NF4	3.73 ± 0.06	$\mathbf{0.5298 \pm 0.0154}$	$\mathbf{0.7238 \pm 0.0079}$	0.2348 ± 0.0041	3.49 ± 0.03	3.22 ± 0.01
DyLoRA	3.92 ± 0.03	0.4872 ± 0.0043	0.7076 ± 0.0006	0.2405 ± 0.0007	3.54 ± 0.04	3.42 ± 0.07
LoRA-FA	$\mathbf{4.18 \pm 0.03}$	0.5256 ± 0.0018	0.7206 ± 0.0009	$\mathbf{0.2446 \pm 0.0008}$	$\mathbf{3.74 \pm 0.01}$	$\mathbf{3.63 \pm 0.03}$
LoRA-XS	3.27 ± 0.07	0.4339 ± 0.0074	0.6818 ± 0.0011	0.2340 ± 0.0028	2.88 ± 0.08	2.80 ± 0.07
T-LoRA	3.97 ± 0.04	0.5006 ± 0.0036	0.7022 ± 0.0027	0.2343 ± 0.0004	3.53 ± 0.02	3.41 ± 0.07

Across all four benchmarks, the run-to-run variation in the headline averages is small relative to the storage gap between LoRA and LITHE. GPT-5.1, Gemini, and Claude preserve the same broad ordering: LoRA remains strongest on strict identity/fidelity, LITHE is the best non-LoRA row on most identity columns, and LoRA-FA is the closest variant on SC-40. Embedding metrics show a related but not identical picture: QLoRA-NF4 gives the strongest non-LoRA SC-40 DreamSim and CLIP-I, while CLIP-T is nearly flat because most trained adapters follow the prompt.

G Cross-architecture evaluation on SC-40 and identity benchmarks

Full SC-40 cross-architecture table. We implemented the same adapter construction across the evaluated softmax-attention backbones (FLUX.1-dev: 228 DiT attention layers in dual-stream MMDiT; SDXL 1.0: 560 UNet attention layers across CrossAttn Down/Mid/Up; Qwen-Image-20B: single-stream DiT). Tab. 24 reports the SC-40 cross-architecture benchmark for LoRA and LITHE on three softmax-attention diffusion backbones spanning 2.6 B–20 B parameters and three structurally distinct attention topologies. These cross-architecture rows are single training-seed full-benchmark runs; the three-seed uncertainty tables in Sec. F cover the main FLUX rows.

SDXL has the strongest absolute GPT-avg, Qwen-Image-20B has the tightest LoRA–LITHE gap, and FLUX.1-dev remains the main benchmark setting for the detailed ablations in the main paper. **Within the SC-40 cross-architecture table, the LoRA–LITHE gap is largest on SDXL ($\Delta = 0.24$ GPT-avg) and smallest on Qwen-Image-20B ($\Delta = 0.06$).** The codebook scale constants were

Table 24: **Full SC-40 cross-architecture benchmark on three softmax-attention diffusion backbones.** All rows use rank r=16. LITHE rows use asymmetric twin codebooks with closed-form Hadamard-index search. **GPT-avg** is the mean of target fidelity (TF), prompt following (PF), and visual quality (VQ), judged with GPT-5.1 on a 1–5 scale. DreamSim is the reference-image similarity metric and CLIP-T is CLIP text-to-image similarity. These rows are single training-seed cross-architecture runs.

Backbone	Architecture	Method	n	GPT-avg ↑	DreamSim ↑	CLIP-T ↑
FLUX.1-dev	dual-stream MMDiT	LoRA	200	4.28	0.558	0.242
FLUX.1-dev	dual-stream MMDiT	LITHE (Ours)	200	4.11	0.529	0.241
SDXL 1.0 [48]	UNet (cross-attn)	LoRA	200	4.70	0.549	0.257
SDXL 1.0 [48]	UNet (cross-attn)	LITHE (Ours)	200	4.46	0.520	0.255
Qwen-Image-20B [49]	single-stream DiT	LoRA	200	4.24	0.506	0.263
Qwen-Image-20B [49]	single-stream DiT	LITHE (Ours)	200	4.18	0.498	0.262

tuned on FLUX.1-dev, so their per-block (B=64) reconstruction may fit the factor statistics of the SDXL UNet less well. We report the measured rank-16 result here; rank-bumped SDXL is not included in this submission. The pool definition is backbone-independent because pool A and pool B are deterministic functions of H_9; payload bytes still depend on the wrapped module count and factor shapes.

Full identity-suite cross-architecture table. The non-FLUX cross-architecture identity queue is complete for DreamBooth-30, SCIdentity, and CC101-20 on both SDXL 1.0 and Qwen-Image-20B, for 70 identity concepts per backbone and matched LoRA/LITHE training and inference. Tab. 25 reports the aggregate GPT-5.1 identity metrics. The identity rubric uses SI, PF, VQ, subject-background disentanglement (SBD), and concept naturalness (CN); GPT-avg is their mean. These identity rows pool the 70-concept suite per seed and report mean±std over seeds 1, 7, and 42.

Table 25: **Cross-architecture identity-suite GPT-5.1 aggregate.** All rows use rank r=16 and cover the same 70 identity concepts: DreamBooth-30, SCIdentity, and CC101-20. Entries report mean±std over seeds 42, 1, and 7. Higher is better for all metrics.

Backbone	Method	Seeds	SI ↑	PF ↑	VQ ↑	SBD ↑	CN ↑	GPT-avg ↑
SDXL 1.0 [48]	LoRA	1, 7, 42	4.39±0.03	4.20±0.03	4.39±0.02	4.51±0.01	4.56±0.02	4.41±0.01
SDXL 1.0 [48]	LITHE (Ours)	1, 7, 42	3.79±0.02	4.42±0.02	4.46±0.00	4.83±0.01	4.48±0.02	4.40±0.00
Qwen-Image-20B [49]	LoRA	1, 7, 42	3.25±0.08	4.69±0.03	4.88±0.00	4.92±0.00	4.75±0.02	4.50±0.02
Qwen-Image-20B [49]	LITHE (Ours)	1, 7, 42	2.21±0.03	4.65±0.03	4.94±0.01	5.00±0.01	4.76±0.01	4.31±0.01

LITHE is close to LoRA on aggregate identity GPT-avg for SDXL and lags by 0.19 on Qwen-Image-20B, but the SI column shows that this aggregate hides a stricter Qwen-Image identity gap.

Cross-architecture DB-2 diagnostic panel. We additionally report the cross-architecture DB-2 panel in full (Tab. 26), including the rubric-rescaling caveat.

Table 26: **Cross-architecture DB-2 diagnostic panel.** SI, PF, and VQ on the 0–4 rubric; FLUX.1-dev scores are shifted by −1 from the 1–5 scale.

Backbone	Rubric	SI	PF	VQ	Verdict
FLUX.1-dev (1–5, −1 shift)	0–4	2.40	2.93	3.75	PASS
SDXL (0–4 native)	0–4	3.30	3.90	4.00	PASS

The −1 shift on FLUX.1-dev gives mean values comparable to the native 0–4 rubric, but the variance is not directly comparable (linear shift preserves means and stds, but the underlying distribution categorical support differs).

H Baseline coverage and unsupported FLUX ports

Table 1 reports only baselines that satisfy four criteria in our FLUX-dev benchmark stack: a runnable FLUX personalization implementation, an adapter-effect validation showing visible subject or style transfer beyond the frozen backbone, full benchmark coverage under the shared prompt protocol,

and reproducible metric outputs. We kept the historical experiment logs, dispatcher states, archived checkpoints, and visual audits in the project experiment record, but we do not convert failed ports into quantitative rows because doing so would mainly measure implementation mismatch rather than the published method.

The final validated rows are LoRA, DoRA, VeRA, AdaLoRA, QLoRA-NF4, DyLoRA, LoRA-FA, LoRA-XS, T-LoRA, the zero-shot FLUX backbone, and LITHE. The following related methods were audited but not promoted to the final quantitative table.

Table 27: Related baselines audited but excluded from the final FLUX-dev Table 1. We exclude only after a failed FLUX support or adapter-effect validation, not because the methods are irrelevant.

Method family	FLUX-dev audit outcome
HyperDreamBooth [30]	The reported compact LiDB representation depends on a shared hypernetwork and there is no stable open FLUX personalization implementation in our stack. A faithful comparison would require nontrivial FLUX retraining of the hypernetwork rather than a drop-in adapter run.
LoHa / LoKr [25, 26]	Initial PEFT/LyCORIS FLUX ports trained and produced checkpoints, but the visual audits showed either LoRA-like unchanged outputs or no reliable style/identity learning under the final prompt and inference path. These checkpoints were archived rather than reported.
OFT / SVDiff / Textual Inversion [28, 27, 6]	Initial FLUX ports were attempted in the historical baseline wave, but they failed the final adapter-effect or load/inference compatibility checks for our FLUX-dev benchmark protocol.
IP-Adapter-style image conditioning	This is an inference-time image-conditioning baseline rather than a trained per-concept LoRA storage baseline, so its storage and routing contract is not comparable to Table 1.

I Auxiliary per-subject and per-axis diagnostics

This section is not a full expansion of Table 1; the multi-seed benchmark summaries are already reported in Sec. F. Instead, we keep the appendix compact by showing diagnostic slices used in the discussion: DB-30 SI tails, CC101-20 and SCIdentity per-subject judge slices, and SC-40 per-axis concept-transfer means. The released CSVs contain the full prompt-level rows.

DreamBooth-30 per-subject SI for the four full-n=30 rows. For brevity, we report subject-level mean $\pm$ std (across the 5 class-aware prompts per subject) for the SI dimension only. The full 5-dim per-prompt scores are in the code repository.

Table 28: DreamBooth-30 per-subject SI for the four full-coverage rows.

Subject	LoRA r=64	LITHE (single cb) r=64	LITHE r=64	LITHE r=16 multi
`db_cat`	4.0 ± 0.0	4.2 ± 0.4	4.0 ± 0.0	4.4 ± 0.5
`db_dog`	3.4 ± 0.5	3.8 ± 0.4	3.6 ± 0.5	3.6 ± 0.9
`db_dog2`	3.6 ± 0.5	3.6 ± 0.9	4.0 ± 0.0	3.4 ± 0.9
`db_dog6`	3.6 ± 0.5	3.6 ± 0.9	3.8 ± 0.4	3.0 ± 1.4
`db_teapot`	3.6 ± 0.9	3.4 ± 0.9	4.0 ± 0.0	3.8 ± 0.4
`db_red_cartoon`	2.0 ± 0.7	2.2 ± 0.4	2.4 ± 0.5	1.0 ± 0.0
`db_poop_emoji`	2.4 ± 0.9	2.4 ± 0.5	2.8 ± 0.4	1.4 ± 0.5
db_other (avg, n=23)	3.85 ± 0.40	3.95 ± 0.35	3.95 ± 0.30	3.62 ± 1.10
full DB-30 mean	3.65	3.71	3.80	3.37

The two stylised tail subjects (`db_red_cartoon`, `db_poop_emoji`) account for the bulk of LITHE's higher SI standard deviation (1.34 vs. ≤ 0.45 for the non-stylised baselines).

CC101-20 per-subject (LITHE).

Table 29: CC101-20 per-subject GPT-5.1 scores for the main-text LITHE row.

Subject	SI	PF	VQ	Subject	SI	PF	VQ
actionfigure	3.6	4.6	3.4	corkscrew	1.4	3.6	3.8
barn	4.0	4.6	4.4	cup	3.8	4.8	4.2
bear	5.0	4.0	4.6	dog	3.4	4.8	4.6
bike	4.0	5.0	4.4	earring	2.4	4.2	4.2
bottle	3.2	4.4	3.8	flower	3.0	4.2	4.0
bunny	4.0	5.0	4.2	helmet	4.4	5.0	4.6
castle	3.6	4.8	4.8	lamp	3.4	5.0	5.0
cat	3.8	5.0	4.6	lobster	3.6	4.6	4.6
panda	3.6	4.6	4.2	pikachu	2.0	3.6	3.6
teddybear	3.8	5.0	4.4	vase	3.8	5.0	4.6
mean (n=20)	3.49	4.59	4.30	**mean** (-2 tails, $n = 18$)	3.65	4.65	4.39

SCIdentity per-subject (LoRA vs. LITHE). SCIdentity is the Synthetic Animal Identity-20 benchmark; each concept has five held-out prompts in this table. We report SI/PF/VQ means per concept for the two main-text FLUX rows (Tab. 30).

Table 30: SCIdentity per-subject SI/PF/VQ means for LoRA and LITHE (five held-out prompts per concept).

Subject	LoRA			LITHE		
	SI	PF	VQ	SI	PF	VQ
amber_antler_red_panda	4.20	5.00	5.00	2.80	4.80	5.00
basalt_scale_pangolin_dog	5.00	5.00	5.00	4.60	5.00	5.00
blueglass_hoofed_penguin	1.60	4.40	5.00	1.00	4.00	5.00
copper_leaf_koala_moth	4.60	5.00	5.00	4.20	5.00	5.00
coral_frill_seal_pup	4.60	5.00	5.00	4.00	5.00	5.00
crystal_whisker_otter	4.40	5.00	5.00	3.60	4.80	5.00
glass_gill_axolotl_cat	5.00	4.40	5.00	3.60	5.00	5.00
jade_beaked_sable_kite	3.40	5.00	5.00	3.00	5.00	5.00
lilac_mane_snow_lion_cub	3.80	5.00	5.00	2.40	4.60	5.00
marble_shell_raccoon_turtle	4.80	5.00	5.00	3.20	4.80	5.00
moss_backed_moon_hare	4.00	5.00	5.00	4.00	5.00	5.00
obsidian_mask_lynx	4.60	5.00	5.00	4.00	5.00	5.00
opal_horned_arctic_fox	3.60	5.00	5.00	3.20	5.00	5.00
pearl_spotted_dune_fennec	4.80	5.00	5.00	4.60	5.00	5.00
porcelain_manta_pup	4.20	5.00	5.00	3.40	5.00	5.00
rose_quartz_bat_lemur	4.00	5.00	5.00	3.40	5.00	5.00
sapphire_ringed_owl_ferret	3.80	5.00	5.00	3.60	5.00	5.00
silver_fin_forest_goat	4.80	5.00	5.00	4.20	4.80	5.00
sunspot_quokka_dragonet	3.40	5.00	5.00	2.60	5.00	5.00
violet_spine_hedgehog_deer	4.40	5.00	5.00	4.80	5.00	5.00
mean	4.15	4.94	5.00	3.51	4.89	5.00

SC-40 per-axis breakdown.

Table 31: SC-40 synthetic multi-reference concept transfer benchmark. SC-40 groups outfit, lighting, pose, painterly style, and anime style concepts; each concept has 20 references and 5 held-out prompts. GPT-avg is the mean of target fidelity (TF), prompt following (PF), and visual quality (VQ), judged with GPT-5.1.

Group	Method	n	TF	PF	VQ	GPT-avg	DreamSim	CLIP-I
ALL	LoRA	200	3.13	4.72	5.00	4.28	0.558	0.723
ALL	LITHE closed	200	2.69	4.65	5.00	4.11	0.529	0.711
Outfit	LoRA	40	3.48	4.88	5.00	4.45	0.757	0.806
Outfit	LITHE closed	40	2.90	4.73	5.00	4.21	0.718	0.791
Lighting	LoRA	40	3.43	4.55	5.00	4.33	0.429	0.683
Lighting	LITHE closed	40	3.05	4.53	5.00	4.19	0.417	0.675
Pose	LoRA	40	3.15	4.30	5.00	4.15	0.611	0.722
Pose	LITHE closed	40	2.80	4.13	5.00	3.98	0.597	0.716
Painter style	LoRA	40	2.28	4.93	4.98	4.06	0.406	0.673
Painter style	LITHE closed	40	1.88	4.95	5.00	3.94	0.367	0.662
Anime style	LoRA	40	3.33	4.95	5.00	4.43	0.586	0.733
Anime style	LITHE closed	40	2.80	4.90	5.00	4.23	0.546	0.712

J N-LoRA composition cost table

Table 32: N-LoRA composition end-to-end render and isolated compose. FLUX.1-dev at 512^2, 28 steps, single A100, 3-trial mean. The LoRA baseline applies the measured single-adapter swap cost ($\sim$0.9 s, App. D.8) once per composed LoRA before a matched 10.6 s render. The *compose-only* column compares the time spent attaching LoRAs and excludes the shared render.

N	LITHE E2E (s)	LoRA fuse+infer (s)	E2E speedup	Compose-only speedup
1	10.58 ± 0.02	11.50	1.09×	464×
2	11.57 ± 0.02	12.40	1.07×	956×
3	12.47 ± 0.14	13.30	1.07×	1,420×
5	$\mathbf{14.52 \pm 0.03}$	15.10	**1.04×**	**2,358×**

K Embedding-metric supplementary panel (DreamSim, CLIP-T, CLIP-I)

We complement the GPT-5.1 judge with three reference embedding metrics computed on the same generated images: **DreamSim** [50] (perceptual similarity to the reference subject; max over reference imgs), **CLIP-T** (cosine similarity between the OpenCLIP ViT-L/14 [51] embedding of the generated image and the prompt text embedding), and **CLIP-I** (cosine similarity between the generated and reference image embeddings, averaged across reference imgs). All metrics are higher-is-better.

K.1 CLIP-T values for the main storage–quality table

The main paper Tab. 1 omits the CLIP-T column for readability; Tab. 33 reports the per-benchmark CLIP-T value for every row of Tab. 1.

Why CLIP-T is coarse for personalization. On DreamBooth-30, *Vanilla FLUX* (no adapter) attains the highest CLIP-T (0.27 vs. LITHE's 0.25); on CC101-20 it also exceeds LITHE (0.26 vs. 0.25). The pattern reflects a known coarseness of CLIP-T as a personalization metric: the CLIP text embedding does not separate fine subject details among semantically close categories, so for a prompt like "a photo of a `<v_sub>` dog" any plausible dog image scores near the maximum, masking the actual subject-identity signal. On SCIdentity (rare non-human individuals) and SC-40 (5 controlled axes the backbone has not memorised), the adapter constraint is identifiable and LITHE matches or exceeds Vanilla FLUX on CLIP-T (0.26 vs. 0.25 and 0.24 vs. 0.23 respectively). We therefore report CLIP-T here for completeness and rely on the LLM-avg / DreamSim columns of Tab. 1 for the headline ordering.

Table 33: **CLIP-T column for Tab. 1.** CLIP-T is cosine similarity between the OpenCLIP ViT-L/14 embedding of the generated image and the prompt-text embedding. Higher is better.

Method	DB-30	CC101-20	SCIdentity	SC-40
Vanilla FLUX	**0.27**	0.26	0.25	0.23
LoRA [19]	0.25	0.24	0.26	0.24
DoRA [13]	0.25	**0.27**	0.24	0.24
VeRA [14]	0.25	0.26	0.23	0.23
AdaLoRA [22]	0.26	**0.27**	0.26	0.23
QLoRA-NF4† [11]	0.25	0.26	0.23	0.24
DyLoRA [23]	0.26	0.26	**0.26**	0.24
LoRA-FA [24]	0.25	0.25	0.26	**0.25**
LoRA-XS [15]	0.26	**0.27**	0.24	0.23
T-LoRA [29]	0.26	**0.27**	0.25	0.23
LITHE (Ours)	0.25	0.25	0.26	0.24

Table 34: Embedding-metric supplementary panel on DreamBooth-30 and CC101.

Method	Bench	n_{subj}	DreamSim ↑	CLIP-T ↑	CLIP-I ↑
LoRA r=64 †	DB-30	30	0.783 ± 0.088	0.235 ± 0.025	0.846 ± 0.047
LITHE w/ single codebook r=64 †	DB-30	30	0.794 ± 0.073	0.231 ± 0.028	0.848 ± 0.047
LITHE r=64 †	DB-30	30	$\mathbf{0.826 \pm 0.066}$	0.225 ± 0.021	$\mathbf{0.862 \pm 0.035}$
LITHE (Ours) †	DB-30	30	0.646 ± 0.113	$\mathbf{0.244 \pm 0.022}$	0.783 ± 0.067
LoRA r=16	DB-30	3	0.653 ± 0.112	0.264 ± 0.014	0.839 ± 0.034
LoRA r=32	DB-30	3	0.717 ± 0.133	0.244 ± 0.022	0.860 ± 0.037
LoRA r=8	DB-30	3	0.618 ± 0.105	0.271 ± 0.012	0.822 ± 0.034
DoRA r=16	DB-30	3	0.645 ± 0.126	0.264 ± 0.013	0.837 ± 0.043
VeRA r=16	DB-30	3	0.570 ± 0.075	0.276 ± 0.011	0.790 ± 0.029
AdaLoRA avg-16	DB-30	3	0.530 ± 0.059	0.263 ± 0.011	0.797 ± 0.016
QLoRA NF4 r=16	DB-30	3	0.775 ± 0.093	0.247 ± 0.014	0.889 ± 0.016
LITHE w/ single codebook r=16	DB-30	3	0.671 ± 0.100	0.263 ± 0.016	0.854 ± 0.019
LITHE (Ours)	CC101	20	0.646 ± 0.127	0.246 ± 0.020	0.766 ± 0.065

† marks rows with full DB-30 coverage (n_{subj}=30); other rows are 3-subject spot probes.

Embedding-metric reading. LITHE r=64 is the embedding-best (DreamSim 0.826, CLIP-I 0.862). The main-text LITHE row trades 0.18 DreamSim and 0.08 CLIP-I for maintained GPT-5.1 scene quality (PF+VQ). On CLIP-T, LITHE is the best full-n=30 row (0.244), so prompt following is preserved at small per-adapter storage. CC101 transfer for LITHE is DreamSim-flat (0.646 both DB-30 and CC101) and CLIP-T-flat (0.244 vs. 0.246); the embedding metrics corroborate the CC101 GPT-5.1 SI/PF pattern reported in Tab. 1 and App. I.

Reading the embedding-vs-judge divergence. The GPT-5.1 SI / PF / VQ judge rewards LITHE r=16 multi on PF and VQ; DreamSim+CLIP-I reward the higher-rank LITHE r=64 variant on raw subject-pixel similarity. This is consistent with a storage/identity trade-off: at ∼3.2 MB per subject the rank-16 LITHE variant gives up raw subject-pixel similarity (a 0.43 SI / 0.18 DreamSim gap to the embedding-best variant) for the compact payload. The full embedding row table is reported here for completeness; SI-vs-DreamSim and SI-vs-CLIP-I correlations across the 13 baselines are positive ($\rho = +0.83$ and $+0.74$). The GPT-5.1 ordering is therefore consistent with the underlying embedding ordering.

L Benchmark datasets and prompts

This section collects the benchmark definitions used throughout the paper. The evaluation suite has four groups: two public personalization benchmarks, DreamBooth-30 and CC101-20, and two synthetic stress benchmarks, SCIdentity and SC-40. DreamBooth-30 and CC101-20 test standard public subject/concept personalization. SCIdentity and SC-40 are built in-house because public data do not provide LoRA-ready, multi-reference coverage for rare animal identity, outfit/costume, dramatic lighting, full-body pose, and highly specific style transfer. For SC-40 we deliberately avoid

easy categories: modern Qwen-VL, Imagen, and FLUX-class models already generate many common sunsets, moonlight scenes, anime styles, garments, and routine poses zero-shot, so the benchmark uses visually demanding concepts where the adapter's contribution is easier to isolate.

Synthetic generation pipeline. For SCIdentity and SC-40 we first write a structured concept card for each target, including a stable trigger token, slug, natural-language name, category, visual signature, palette/material constraints when relevant, and explicit avoid rules. We then expand each concept card into a GPT-Image-2 batch request. Each request contains the full image-generation prompt, output filename, model name, image size, quality, and output format. The prompt fixes the target invariant (identity markings, outfit construction, lighting geometry, pose geometry, or style rendering) while varying subject, camera view, scene, or composition so that the LoRA cannot simply memorise one photograph. Images are generated with GPT-Image-2. We then organise successful generations into per-concept training folders with one caption per image, together with concept metadata, an expected-image manifest, and quality-control summaries. Full contact sheets are generated before training; we inspect them for concept consistency, concept separation, missing trigger tokens, broken anatomy, visible logos, readable text, watermarks, or accidental celebrity/franchise leakage. The final visual prompt gallery in App. P shows representative raw references together with the generation prompt excerpt and the exact LoRA training caption.

L.1 DreamBooth-30

We use the public DreamBooth-30 release [4], comprising 30 subjects (objects + live subjects). Three of these enter the headline six-subject cohort: `db_cat` (an orange tabby, 5 reference images), `db_dog6` (a corgi, 5 reference images), `db_teapot` (a zisha purple-clay teapot, 5 reference images). Each subject is paired with the standard 25-prompt evaluation suite from the DreamBooth protocol; we report results on the first 5 prompts per cell to match the trainer's evaluation cadence.

L.2 CC101-20

We use the public CustomConcept101 release [5] of 102 subjects. The paper reports CC101-20, a 20-subject slice selected to broaden the object and concept distribution beyond DreamBooth-30 while keeping the full cross-method run tractable. Three CC101 subjects also enter the early six-subject cohort: `cc_action` (a Victorian-era articulated mannequin, 5 reference images), `cc_barn` (a wooden red barn, 5 reference images), `cc_book` (a spiral-bound notebook, 5 reference images). The remaining CustomConcept101 subjects are reserved for future expanded cross-benchmark sweeps.

L.3 Synthetic Animal Identity-20 (SCIdentity)

SCIdentity is our DreamBooth-style synthetic identity benchmark for non-human subjects, containing 20 rare animal-like identities. Each identity has 12 training references and 4 hold-out references used only for identity quality control, for a total of 320 images. Each identity folder stores its instance prompt, class name, and identity label as plain-text fields alongside training and hold-out subfolders.

The identities are deliberately unusual but individually stable, e.g. an opal-horned arctic fox, a glass-gill axolotl cat, a moss-backed moon hare, a porcelain manta pup, a basalt-scale pangolin dog, a sapphire-ringed owl ferret, an amber-antler red panda, a crystal-whisker otter, a marble-shell raccoon turtle, a lilac-mane snow lion cub, and ten additional hybrid animal identities. For each identity we generate 16 views: 12 training references and 4 hold-out identity-check references. The GPT-Image-2 prompt fixes the same individual's persistent markings, body shape, eye colour, appendages, and asymmetries, while varying viewpoint and simple background. The training caption deliberately does not expose these details. Every training image uses the canonical DreamBooth prompt

`a photo of <v_id_NNN> animal.`

This ensures that identity details must be stored in the learned token rather than recovered from descriptive text. A visual QC pass accepted 18 identities outright and kept 2 as watchlist cases where the rare feature is less visible in some views but the individual identity remains stable.

L.4 SC-40 synthetic concept-transfer benchmark

SC-40 is our synthetic multi-reference concept-transfer benchmark. It contains 40 concepts and 800 reference images, grouped into five axes: outfit/costume, lighting, pose/composition, painterly style, and anime style. Each axis contains 8 concepts, and each concept contains 20 references.

We built SC-40 because public data did not cover the LoRA setting we needed. Outfit resources are typically virtual try-on pairs or product catalogues rather than multi-reference LoRA concepts; lighting resources are often physical relighting data, movie frames, or editing demos; and pose resources usually provide pose annotations rather than a persistent pose concept with varied subjects and backgrounds. These formats are valuable for their native tasks but do not directly test whether a LoRA can store and reapply a transferable concept.

The benchmark also avoids overly easy categories. Modern image systems such as Qwen-VL/Imagen/FLUX-class pipelines already handle many common semantic axes zero-shot: ordinary sunsets, moonlight, standard anime styles, common garments, and everyday poses are often native behaviours. If we trained LoRAs for those concepts, the base model could already produce plausible outputs and the adapter's contribution would be hard to measure. SC-40 therefore uses deliberately rare, visually dramatic concepts: examples include aurora chainmail raincloaks and chrome sari gowns for outfit; copper-eclipse sunset, stained-glass noir beams, and underwater moon caustics for lighting; crescent backbend reach and diagonal spear lunge for pose; ghost-enamel miniature and black-lacquer mosaic for painterly style; and candy-glass cyber cel or hazard-tape action cel for anime style.

Each SC-40 axis uses a different prompt template. Outfit prompts request full-body fashion/editorial references and lock the garment silhouette, materials, and construction details while varying model, view, and studio/runway context. Lighting prompts keep the subject deliberately simple and unbranded so that the unusual light field, shadow geometry, and colour palette are the dominant signal. Pose prompts request one full-body person in a single continuous scene, with the pose geometry held fixed and subject/background varied. Painterly and anime-style prompts vary content but lock rendering medium, linework, palette, texture, and edge treatment; all style names are fictional and avoid living artists, studios, copyrighted characters, signatures, or watermarks.

Captions are short and token-first. Outfit, lighting, pose, and style captions always contain the trigger token and concept name, but omit the long GPT-Image-2 generation prompt. This prevents the model from learning a prose description instead of binding the target visual concept to the LoRA token. The LoRA-ready directory for each concept therefore contains 20 image/text pairs such as `outfit_001_aurora_chainmail_raincloak_01.png` and `outfit_001_aurora_chainmail_raincloak_01.txt`; the original generation prompts remain in the dataset-level JSONL batch file for auditability.

L.5 Prompt suite

Each per-concept LITHE adapter is trained with a unique trigger token (`<v_sub_NNN>` for DB-30 subjects, `<v_sub_CNNN>` for CC101-20 subjects, `<v_id_NNN>` for SCIdentity animal-identity subjects, and `<v_outfit_NNN>` / `<v_light_NNN>` / `<v_pose_NNN>` / `<v_pstyle_NNN>` / `<v_anime_NNN>` for the five SC-40 concept-transfer axes); the trigger appears in every training caption alongside the class word. For evaluation, each identity subject (DreamBooth-30, CC101-20, SCIdentity) is rendered with the five prompt templates "a photo of a [token] {class}", "... sitting on a wooden chair, studio lighting", "... walking through a city street at night", "... wearing a top hat, sharp focus", and "a close-up portrait of ... with cherry blossoms", where [token] is the subject token the adapter was trained with. SC-40 prompts are "a photo of {subject}, in `<v_anime_NNN>` {name}", "... lit by `<v_light_NNN>` {name}", "{subject} in `<v_pose_NNN>` {name}, neutral background", and similar single-axis variants. LoRA and LITHE are evaluated on five SC-40 prompts per concept. The full prompt lists are in the code repository.

M Three-judge LLM protocol and prompts

To reduce closed-model bias the headline GPT-avg in Tab. 1 (and in Tab. 24, Tab. 35, and Tab. 31) is cross-evaluated under three independent vision-language judges: **GPT-5.1** (OpenAI), **Claude Sonnet 4.6** (Anthropic), and **Gemini 2.5 Flash** (Google). All three judges share two task-specific rubrics: an **identity rubric** for DreamBooth-style subject-driven benchmarks (DreamBooth-30, CC101-20,

SCIdentity), and a **concept-transfer rubric** for the SC-40 multi-reference benchmark. The reported **LLM-avg** is the unweighted mean of GPT-avg, Claude-avg, and Gemini-avg; per-judge breakdowns are tabulated in Sec. F. The three judges rank methods consistently. The system-level Pearson correlation between judges over benchmark–method means is 0.85–0.98, and each judge orders LoRA and LITHE the same way. Judge-independent DINO subject similarity on DreamBooth-30 gives the same gap direction, 0.600 for dense LoRA against 0.548 for LITHE ($p < 10^{-14}$). All $16{,}570$ evaluated images and the per-image outputs of the three judges are released for re-scoring at `https://huggingface.co/datasets/Fanjue/LITHE-eval`. The rubrics below are condensed, and the exact prompt templates sent to the three judges and a minimal judge script are in the code repository. Because all three judges are proprietary, exact reruns may drift if the APIs change. The paper therefore treats these scores as fixed judge measurements and pairs them with the released prompts plus DreamSim and CLIP-based cross-checks, rather than claiming full API-independent reproducibility on closed models.

M.1 Identity rubric (DreamBooth-30, CC101-20, SCIdentity)

The judge sees one REFERENCE training image, one GENERATED image, the text prompt, and the subject class name. It returns five integer scores on a 1–5 scale that together compose the final headline GPT-avg metric reported in the main tables.

> *You are a meticulous human evaluator scoring a subject-driven image generation result.*
>
> *You see two images:* ***REFERENCE****, one original training image of the target subject;* ***GENERATED****, the model output.*
>
> *The generation prompt was: "{prompt}". The subject class is: {class_name}.*
>
> *Score each dimension as an integer from* 1 *to* 5*:*
>
> ***SI (Subject Identity)****:* 1 *totally different;* 2 *same broad class only;* 3 *some shared features but different instance;* 4 *clearly the same subject with minor drift;* 5 *near-identical subject instance.*
>
> ***PF (Prompt Following)****:* 1 *ignores prompt;* 2 *vague reference;* 3 *partial;* 4 *mostly correct with minor miss;* 5 *perfect.*
>
> ***VQ (Visual Quality)****:* 1 *severe artifacts/noise;* 2 *noticeable issues;* 3 *acceptable but flawed;* 4 *clean with minor issues;* 5 *pristine.*
>
> ***SBD (Subject-Background Disentanglement)****:* 1 *subject identity is confounded with background/context;* 3 *partially disentangled;* 5 *subject is preserved under prompt/context changes.*
>
> ***CN (Class Naturalness)****:* 1 *implausible or incoherent subject rendering;* 3 *acceptable;* 5 *natural and coherent.*
>
> *Return ONLY a JSON object: {"SI": int, "PF": int, "VQ": int, "SBD": int, "CN": int, "reason": string}.*

M.2 Concept-transfer rubric (SC-40)

The SC-40 benchmark is multi-reference rather than subject-driven: the judge sees TWO REFERENCE images of the same target concept and ONE GENERATED image, with the benchmark group (outfit / lighting / pose / painter_style / anime_style) supplying a per-axis group hint. It returns three integer scores on a 1–5 scale (SC-40 uses TF in place of SI because the target is a non-subject attribute, e.g. a garment construction or a lighting condition, and an extra short `description` field as an audit trail).

> *You are a meticulous evaluator for a synthetic multi-reference concept LoRA benchmark.*
>
> *You will see TWO REFERENCE images for the SAME target concept, then ONE GENERATED image.*
>
> *Benchmark group: {group}; Target concept: {concept}; Target concept traits to judge: {group_hint}; Generation prompt: "{prompt}".*
>
> *Score each dimension as integer* 1–5*:*
>
> ***TF (Target Fidelity)****: does GENERATED preserve the target concept demonstrated by the references? Focus on the per-group traits in {group_hint}, ignore superficial content changes.* 1 *no meaningful match;* 2 *weak/vague;* 3 *partial with clear drift;* 4 *strong with minor drift;* 5 *near-perfect.*
>
> ***PF (Prompt Following)****:* 1 *ignores;* 2 *vague;* 3 *partial;* 4 *mostly correct;* 5 *perfect.*

***VQ (Visual Quality)**: 1 severe artifacts; 2 noticeable issues; 3 acceptable; 4 clean; 5 pristine.*
Return ONLY a JSON object: {"description": string, "TF": int, "PF": int, "VQ": int, "reason": string}.

The five group hint strings are:

- **outfit**: the target outfit/costume identity, including garment silhouette, materials, trim, and signature construction.
- **lighting**: the target lighting condition, including color, direction, contrast, shadows, highlights, atmosphere.
- **pose**: the target pose/composition, including body arrangement, camera framing, limb geometry, overall composition.
- **painter_style**: the target painterly style, including brushwork, palette, line treatment, texture, medium feel.
- **anime_style**: the target anime visual style – cel treatment, linework, palette, shading, texture, graphic language.

M.3 Aggregation and reporting convention

For each row in Tab. 1 we score every (method, subject/concept, prompt, seed) triplet once with the benchmark-appropriate rubric, then average across prompts and seeds within a subject/concept, and finally average across subjects/concepts within the benchmark. The reported **GPT-avg** is the mean of SI / PF / VQ / SBD / CN for identity benchmarks and the mean of TF / PF / VQ for SC-40. Diagnostic per-dimension tables in Sec. I expose the strict SI/PF/VQ or TF/PF/VQ slices used in the discussion; Tab. 31 reports SC-40 per-axis TF/PF/VQ.

N Per-subject visual catalogue

A per-subject visual catalogue (30 DB-30 subjects $\times$ 5 prompts $\times$ 4 methods, 600 images) is kept as an auxiliary archive outside the release. The main-text serving schematic in Fig. 1 instead uses LITHE FLUX 1024/28-step renders to illustrate co-resident index-bank routing.

O Additional generation examples and quality studies

This appendix collects qualitative results that complement the quantitative serving microbenchmarks of §5.2 and Tab. 2. All renders use LITHE on FLUX-dev bf16, single A100 80 GB, with our Triton SGMV-fused inference path and the twin-codebook decoder. The decoding and rendering scripts are in the code repository.

Table 35: **LITHE short-run component ablation on the SC-40 five-concept subset (extended).** All rows use the same FLUX backbone, rank r=16, 750 optimizer steps, global batch 4, and 3000 sample exposures. Metrics average 25 images per row. GPT-avg is TF/PF/VQ from GPT-5.1; DreamSim is computed against two held-out references; CLIP-T is CLIP text-to-image similarity. Dense/payload uses the uncompressed dense short-run LoRA anchor. The four *Cell* rows are the per-side scale isolation reported in Tab. 3: the Hadamard sign pool H_9 is shared by construction, and the rows differ only in the per-side scale schedule on B (A–D match the main-paper definitions). *Cell A* is the default twin (multiscale RVQ chain on B); *Cell B* keeps asymmetric scale but flat single-scale on B; *Cells C/D* collapse σ_B^{cb} back up to σ_A^{cb}=0.06 with separately built vs. shared pool B, and produce mathematically identical forward outputs. 5-concept row averages here are slightly tighter than the 25-subject Phase-A pipeline cited in Tab. 3 because of the smaller subset and short-run schedule.

Row	Changed component	GPT-avg	DreamSim	CLIP-T	Uncompressed payload (MiB)	Dense/payload
LoRA anchor	Dense continuous adapter	4.16	0.579	0.239	—	1.0×
Cell A: LITHE default	Twin: multiscale $\{0.0025, 0.001, 0.0005\}$, K=512, L=3	4.01	0.552	**0.241**	7.88	18.6×
Cell B: twin, B flat	Asymmetric scale, B flat single-scale 0.0025	3.92	—	—	7.88	18.6×
Cell C: scale-collapsed twin	Sep pool B at σ_B^{cb}=σ_A^{cb}=0.06	1.00	—	—	7.88	18.6×
Cell D: no twin	Shared A/B pool, sym scale 0.06	1.00	0.172	0.146	7.50	19.5×
RVQ L=2	Two residual levels	3.52	0.445	0.232	5.34	27.4×
RVQ L=1	Single residual level	4.03	0.548	0.237	2.80	52.2×
K=256	Half-size codebook	3.96	0.555	0.238	7.49	19.5×
Requantize every 50	Less frequent index reassignment	3.96	0.544	0.234	7.87	18.6×

Two further probes complete the codebook study. A $K{=}2$ sign-only codebook fails in all twelve configurations swept, so the working range needs a non-trivial codebook; the design point is functional equivalence at the extreme rate rather than weight-space losslessness, since generation quality saturates far below lossless weight recovery. A single residual level ($L{=}1$) stays within single-run protocol noise of the default while shrinking the uncompressed payload. This comparison covers only the five-concept short-run protocol, and $L{=}1$ is untested on the full benchmarks. We keep $L{=}3$ as the default because every reported full-benchmark adapter uses it. The $L{=}2$ row inherits a truncated three-level σ_B schedule that mis-scales its residual level, and should be read as a schedule-calibration artifact rather than an RVQ-depth effect.

O.1 SC-40 ablation visualizations

Figure 4 shows the metric-annotated SC-40 five-concept ablation grid. Columns are variants and rows are concepts, so the per-variant GPT average, DreamSim, CLIP-T, and uncompressed deploy-payload size can be read next to the corresponding qualitative outputs.

O.2 Same-batch heterogeneous serving: 14 LITHE LoRAs in one forward vs LoRA serial baseline

Figure 5 runs 14 different LoRAs with the same 14 prompts through each pipeline at 1024^2 28-step on $1{\times}$A100 80 GB. Top band is the training references of the 14 DB-30 LITHE LoRAs (one per LoRA, anchoring the identity each LoRA encodes). Middle band is LITHE same-batch in a *single* forward (182.58 s wall, 75.7 GB peak). Bottom band is LoRA serial (14 swap+render cycles, 195.1 s wall measured) with 14 other plain LoRAs of the same rank and training length; it shows the serving pattern and its cost, not per-cell fidelity, because these adapters encode different subjects and each image uses its own seed. The middle row arrives simultaneously, the bottom row only arrives after 14 queued swap-and-render cycles. The measured fused Diffusers LoRA path does not serve 14 different LoRAs in one batch because its fused weights are module-global, not per-sample.

O.3 2-LoRA composition: SC-40 style ckpt $\times$ SC-40 style ckpt

We show 2-LoRA same-batch composition where *both* LITHE adapters are pure stylizations rather than a subject-identity adapter. Each cell of these sheets activates two SC-40 LITHE LoRAs per-sample via $\delta_i = \alpha_a\,\Delta_{\text{anime}_i} + \alpha_b\,\Delta_{\text{partner}_j}$. We use $\alpha_{\text{anime}}{=}1.0$ and $\alpha_{\text{partner}}{=}0.6$ so the anime stylisation is visible in every cell; equal weights (0.7/0.7) made the partner LoRA (painter or lighting) dominate the look. The textual prompt provides a fixed subject per row drawn from a small human/animal cast (woman on a hilltop / black cat by a window / brown horse in a meadow / old fisherman by a river / red fox in a snowy forest) so the style transformation is clearly visible. Each sheet is one same-batch forward at $B{=}25$, 1024^2 28-step on a single A100 80 GB.

O.4 3-LoRA composition: DB-30 subject + SC-40 anime style + SC-40 lighting

We show 3-LoRA same-batch composition where each sample activates three LITHE LoRAs: a DB-30 subject ckpt, an SC-40 anime/painter style ckpt, and an SC-40 lighting ckpt. We use the DB-30 `<v_sub_001>` cat and `<v_sub_002>` dog subjects (rather than SCIdentity, whose synthetic-animal subject identities do not interact cleanly with the SC-40 style and lighting LoRAs), omit the pose axis (which the SC-40 pose ckpt under-fits at $r{=}16$), and instead cross subject $\times$ anime/painter style $\times$ lighting. All renders are 1024^2 28-step on a single A100 80 GB.

Figure 8 shows two hand-picked cells that combine subject identity preservation with strong stylisation and lighting; Figs. 9–10 present the full 4-light $\times$ 12-style sheets for the cat and dog subjects.

O.5 Step-efficiency study: LITHE render quality vs steps

Figures 11–13 show LITHE rendering at $\{4, 8, 12, 16, 20, 24, 28\}$ inference steps for three representative LITHE LoRAs (one SC-40 outfit, one DB-30 dog, one SC-40 anime style) at three resolutions. At 512^2 and 1024^2, 8 to 12 steps already approach 28-step quality, motivating our use of 16-step inference as the main serving config in Tab. 2. The 4-step columns are visibly under-converged for deployment but are used in the same-batch microbench because that is the distilled inference budget at which FLUX-dev still converges.

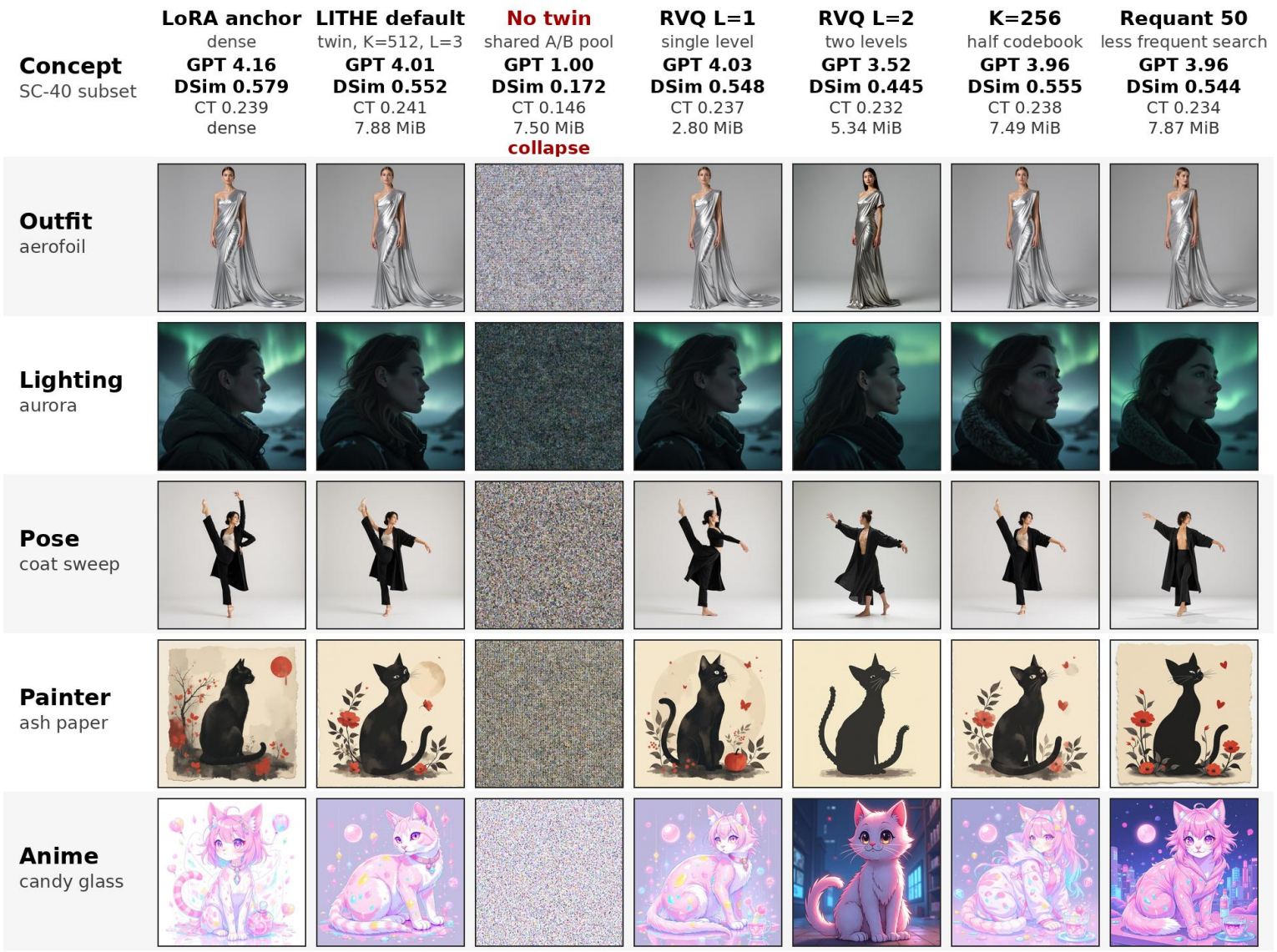


Figure 4: Metric-annotated SC-40 five-concept ablation grid. Columns are the LoRA anchor, default LITHE, and component ablations; rows cover the five SC-40 concept groups. The scale-collapsed twin (Cells C/D in Tab. 35) collapses, while RVQ depth, codebook size, and reassignment frequency trade fidelity against deploy footprint. Header sizes are uncompressed checkpoint sizes in MiB (int16 indices plus fp32 pools), as in Tab. 35.

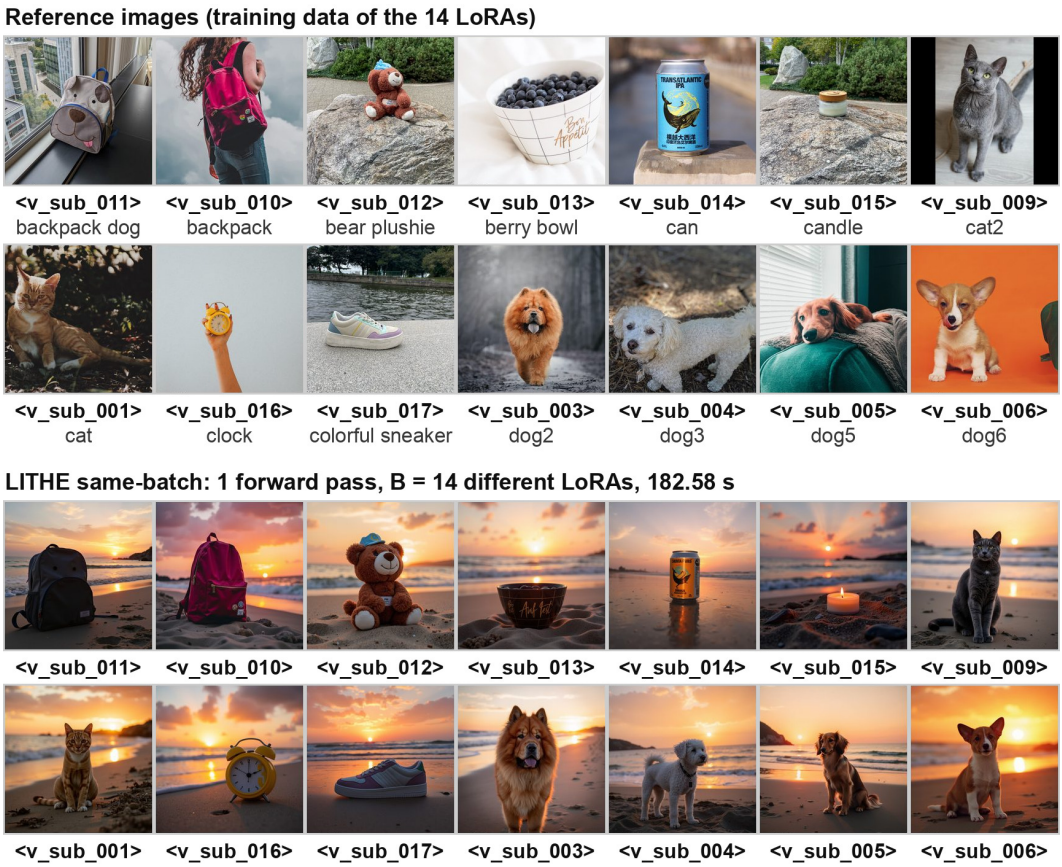


Figure 5: Top: 14 LoRA training references. Middle: LITHE same-batch (single forward, 182.58 s). Bottom: LoRA serial (14 swap+render cycles, 195.1 s) with 14 other plain LoRAs of the same rank, so its cells do not match the references. 1024^2 28-step, identical prompts across the two generation bands. The fused Diffusers LoRA path evaluated here does not provide this per-sample routing mode.

O.6 Converter validation: trainer-native LITHE to Diffusers serving path

The trainer-native inference path (standard Diffusers LoRA loading followed by fusion) and our converter, which rekeys the trainer-native discrete-adapter indices into the Diffusers naming convention and runs them through our serving forward, agree at pixel level under identical seed at 1024^2 28-step. Mean ℓ_1 pixel difference is $0.48/255$ on DB-30 `<v_sub_002>` dog and $0.54/255$ on SC-40 `<v_outfit_006>` aerofoil chrome sari gown, both within bf16 numerical noise. This validates the converter at the basis of the $N_{\max}$ co-residency claim and the $B{=}96$ serving claim. The converter is included in the code repository.

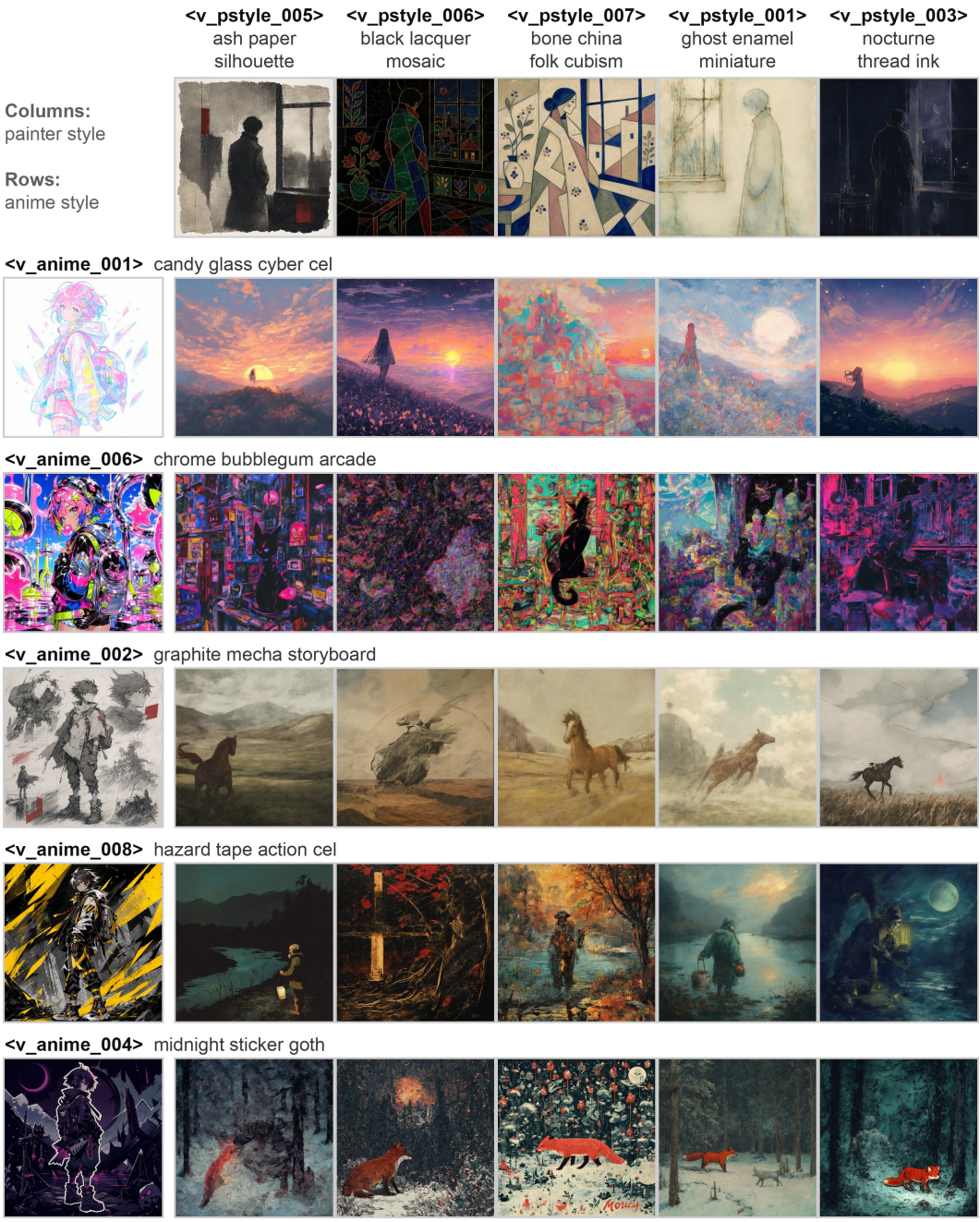


Figure 6: **2-LoRA same-batch SC-40 anime style × SC-40 painter style**, 1024^2 28-step, B=25 in one forward. Rows: 5 SC-40 anime-style LoRAs (training reference at left). Columns: 5 SC-40 painter-style LoRAs (training reference on top). Each cell activates one anime LoRA and one painter LoRA through per-sample routing; both adapters are pure stylizations, so the visible variation comes from the two LITHE adapters rather than a subject-identity LoRA.

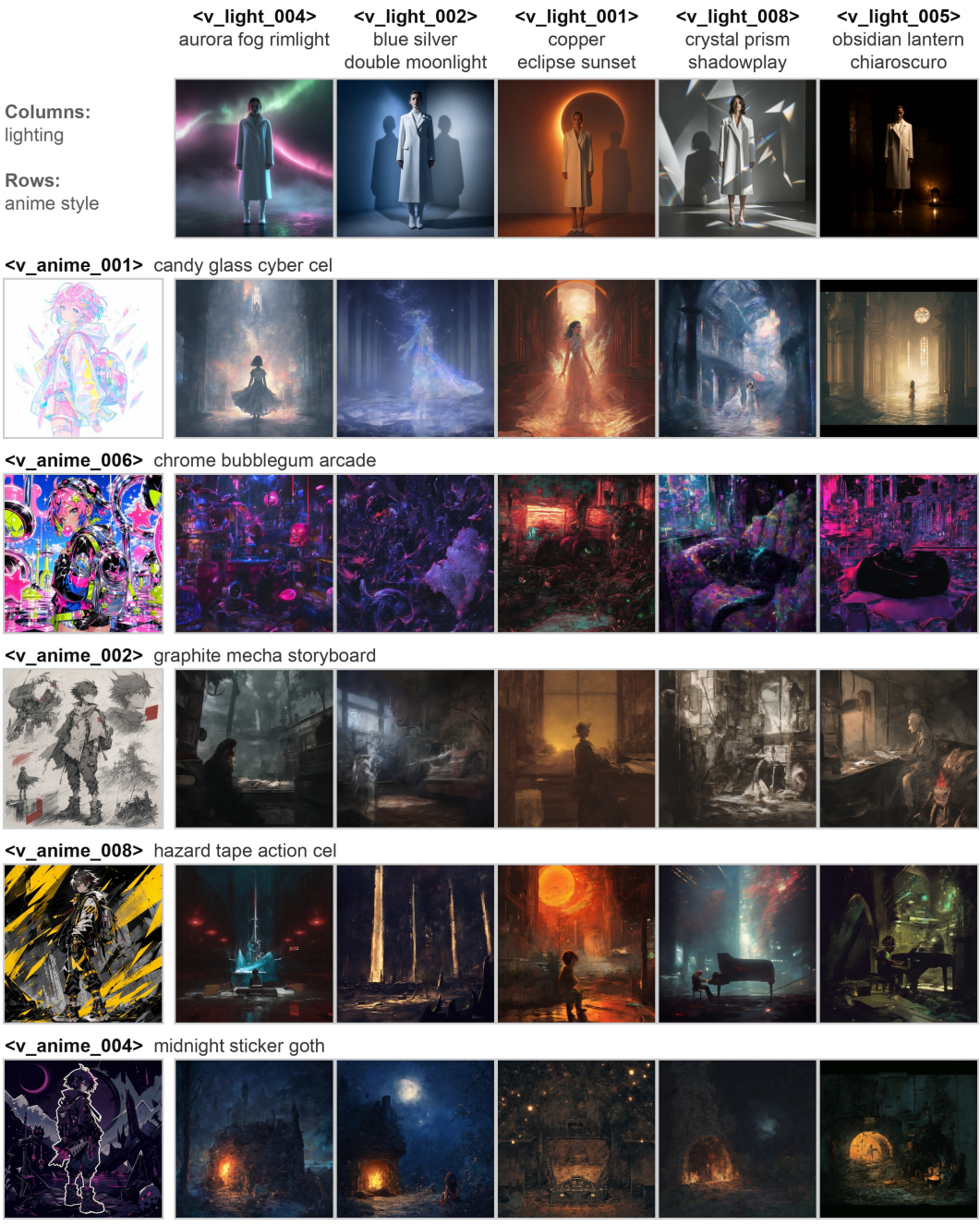


Figure 7: **2-LoRA same-batch SC-40 anime style × SC-40 lighting**, 1024^2 28-step, B=25 in one forward. Rows: 5 SC-40 anime-style LoRAs. Columns: 5 SC-40 lighting LoRAs. Layout matches Fig. 6.

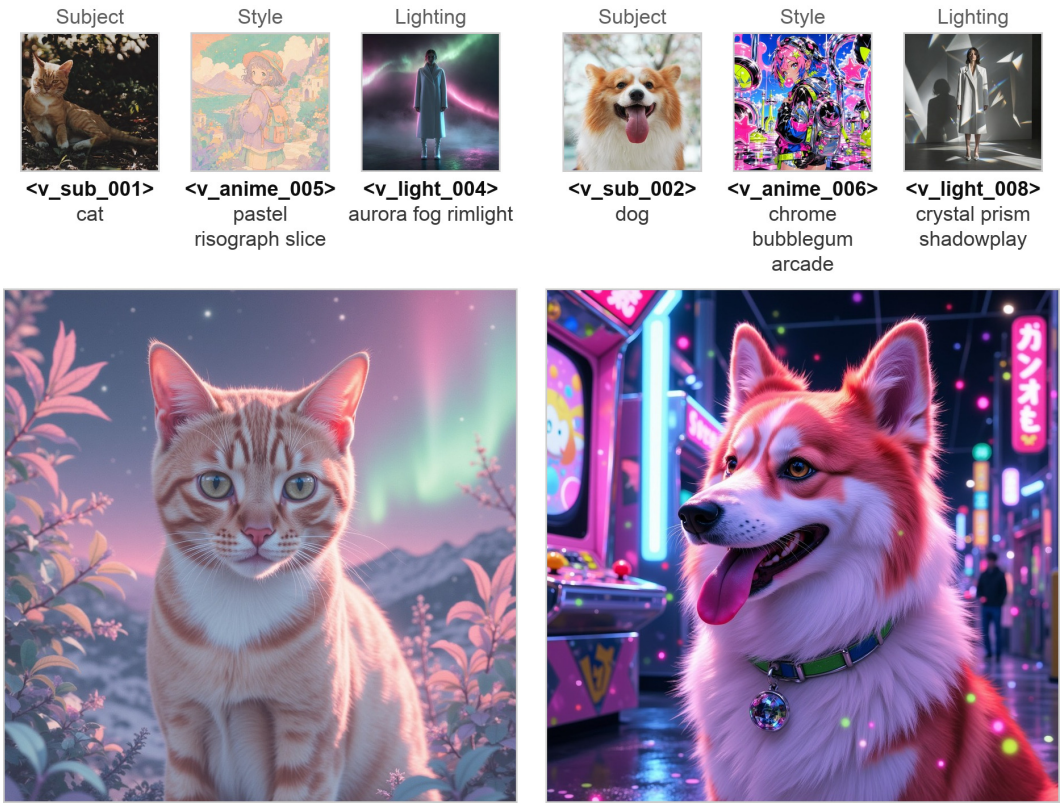


Figure 8: **Featured 3-LoRA composition picks.** Left: `<v_sub_001>` cat + `<v_anime_005>` pastel risograph slice + `<v_light_004>` aurora fog rimlight. Right: `<v_sub_002>` dog + `<v_anime_006>` chrome bubblegum arcade + `<v_light_008>` crystal prism shadowplay. Each cell is one slot of a same-batch 3-LoRA forward. The three reference thumbnails above each rendered cell show the training images that the subject, style, and lighting LITHE LoRAs were trained from.

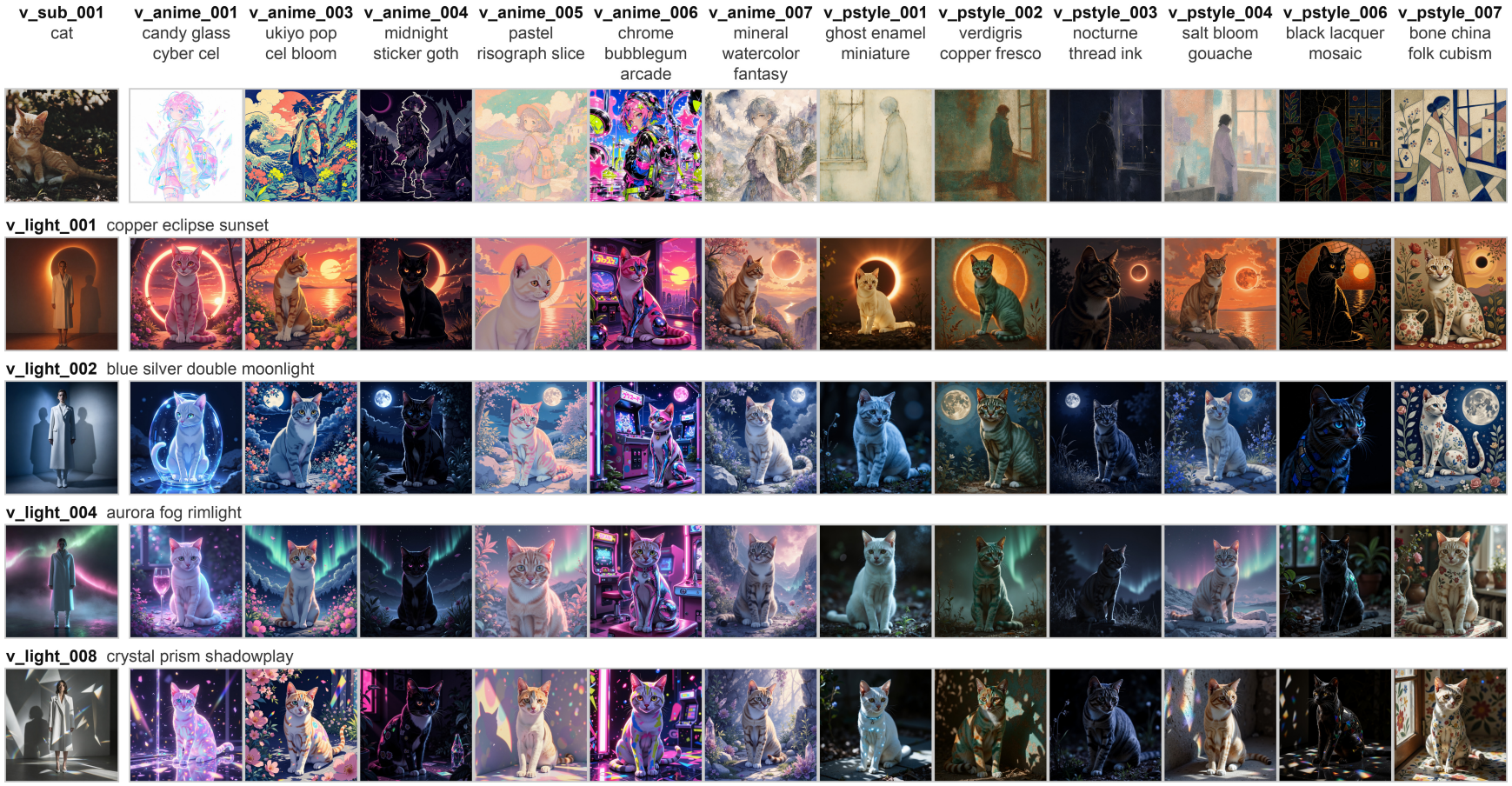


Figure 9: **3-LoRA same-batch composition, DB-30 <v_sub_001> cat × SC-40 style × SC-40 lighting**, 1024^2 28-step. Top row: subject reference + 12 style references (anime + painter). Left column: 4 lighting references. Body: $4 \times 12 = 48$ composed renders, each activating three LITHE LoRAs per-sample.

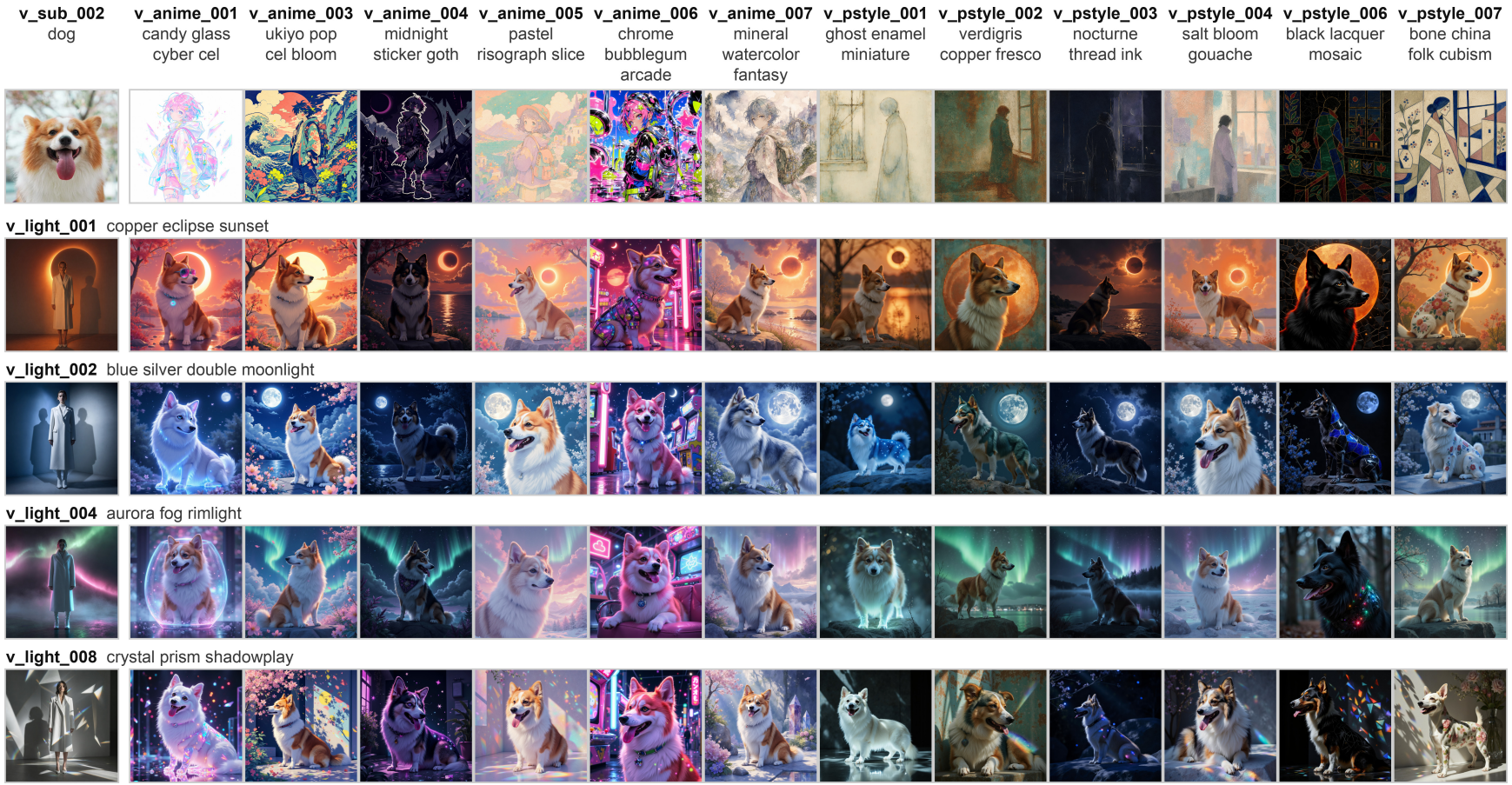


Figure 10: **3-LoRA same-batch composition, DB-30 <v_sub_002> dog × SC-40 style × SC-40 lighting**, 1024^2 28-step. Layout matches Fig. 9.



Figure 11: Step sweep at 256^2. Rows: SC-40 outfit `<v_outfit_006>`, DB-30 dog `<v_sub_002>`, SC-40 anime `<v_anime_001>`. Columns: 4–28 steps. The wall time of each render is given below its cell.



Figure 12: Step sweep at 512^2. Layout same as Fig. 11.



Figure 13: Step sweep at 1024^2. Layout same as Fig. 11.

P Synthetic benchmark gallery: references and prompts

Figures 14–19 visualise representative references from the datasets we generated for SCIdentity and SC-40. Each panel shows the original generated reference, an excerpt of the GPT-Image-2 prompt used to create it, and the exact short LoRA training caption paired with that image. We show two examples per synthetic category to keep the appendix compact while still exposing the image-prompt-caption relationship for identity, outfit, lighting, pose, painterly style, and anime style.

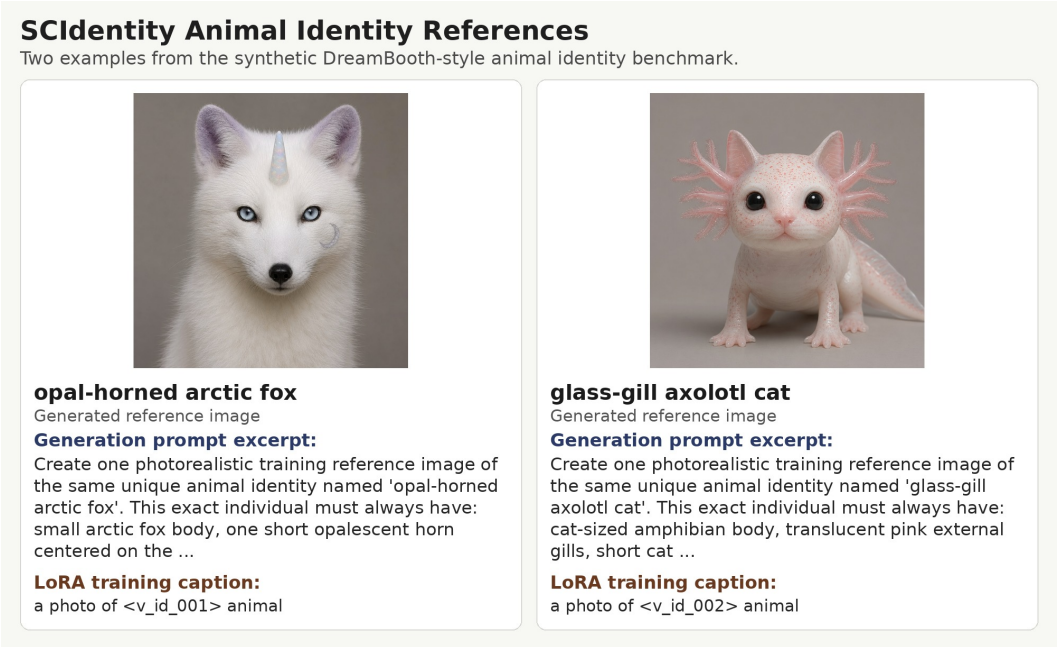


Figure 14: SCIdentity reference gallery. Each panel shows one generated animal-identity reference, with the corresponding generation prompt excerpt and the canonical DreamBooth caption used to drive LoRA training on that image.

SC-40 Outfit / Costume References

Two examples from the synthetic outfit axis; each concept has 20 references.



aurora chainmail raincloak

Generated reference image

Generation prompt excerpt:

Create a photorealistic fashion reference image for a rare synthetic outfit concept named 'aurora chainmail raincloak'. The exact outfit is: a full-length iridescent raincloak made from translucent pearl film, a hexagonal ...

LoRA training caption:

<v_outfit_001> aurora chainmail raincloak, raincloak outfit, full body front view photo of person standing straight in off-white studio, translucent iridescent hooded raincloak with hexagonal texture, asymmetrical cape panel, charcoal belt, glossy knee-high boots



aerofoil chrome sari gown

Generated reference image

Generation prompt excerpt:

Create a photorealistic fashion reference image for a rare synthetic outfit concept named 'aerofoil chrome sari gown'. The exact outfit is: a chrome-silver sari-inspired gown with aerodynamic pleated drape, wing-like shoulder ...

LoRA training caption:

<v_outfit_006> aerofoil chrome sari gown, draped gown outfit, studio fashion photo, full body front view, person standing neutral, long single metallic pleated train, winged shoulder fins, geometric cutout bodice, thin cobalt trim

Figure 15: SC-40 outfit/costume reference gallery. The prompt locks rare garment construction details while the LoRA caption remains short and token-first.

SC-40 Dramatic Lighting References

Two examples from the synthetic lighting axis; content varies while lighting remains the target concept.



copper eclipse sunset

Generated reference image

Generation prompt excerpt:

Create a photorealistic reference image for a rare dramatic lighting concept named 'copper eclipse sunset'. The lighting design is: a low copper-orange sunset partially eclipsed by a dark disk, thin black crescent shadows, warm ...

LoRA training caption:

<v_light_001> copper eclipse sunset, dramatic sunset lighting, full body front view, standing still, minimal studio room, solitary fashion model in simple matte white coat, a low copper-orange sunset partially eclipsed by a dark disk, thin black crescent shadows, warm rim light, and long amber silhouettes



underwater moon caustics

Generated reference image

Generation prompt excerpt:

Create a photorealistic reference image for a rare dramatic lighting concept named 'underwater moon caustics'. The lighting design is: moonlight refracted through imaginary water, wavering blue caustic lines, dappled moving ...

LoRA training caption:

<v_light_006> underwater moon caustics, caustic moonlight lighting, full body front view, standing still, minimal studio room, solitary fashion model in simple matte white coat, moonlight refracted through imaginary water, wavering blue caustic lines, dappled moving shadows, and soft submerged glow

Figure 16: SC-40 dramatic-lighting reference gallery. The generated content is intentionally simple so that the lighting concept is the main transferable signal.

SC-40 Pose / Composition References

Two examples from the synthetic pose axis; the pose geometry is the invariant.



crescent backbend reach

Generated reference image

Generation prompt excerpt:

Create one single clean full-body image for a pose concept named 'crescent backbend reach'. The pose must be exact and consistent: standing on one foot with the torso arched backward, one arm reaching overhead in a crescent ...

LoRA training caption:

<v_pose_001> crescent backbend reach, full-body pose concept, front view, young adult woman in plain athletic wear, neutral gray photo studio, standing on one foot with the torso arched backward, one arm reaching overhead in a crescent curve, the other arm lowered for balance



diagonal spear lunge

Generated reference image

Generation prompt excerpt:

Create one single clean full-body image for a pose concept named 'diagonal spear lunge'. The pose must be exact and consistent: deep forward lunge with the front knee bent, rear leg extended, torso leaning diagonally, both arms ...

LoRA training caption:

<v_pose_002> diagonal spear lunge, full-body pose concept, front view, young adult woman in plain athletic wear, neutral gray photo studio, deep forward lunge with the front knee bent, rear leg extended, torso leaning diagonally, both arms thrust along one strong diagonal line

Figure 17: SC-40 pose/composition reference gallery. The prompt fixes the target pose geometry while varying the subject and background.

SC-40 Painterly Style References

Two examples from the synthetic painterly-style axis; subject changes while medium and palette stay stable.



ghost enamel miniature

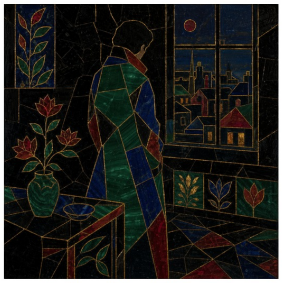
Generated reference image

Generation prompt excerpt:

Create a finished painting in a rare fictitious painterly style named 'ghost enamel miniature'. The style must be visually consistent: translucent porcelain-enamel paint, pale ghostly figures, fine black contour lines, tiny gold ...

LoRA training caption:

<v_pstyle_001> ghost enamel miniature, niche painterly style, painting of a solitary figure in a long coat standing beside a window, translucent porcelain-enamel paint, pale ghostly figures, fine black contour lines



black lacquer mosaic

Generated reference image

Generation prompt excerpt:

Create a finished painting in a rare fictitious painterly style named 'black lacquer mosaic'. The style must be visually consistent: glossy black lacquer ground, flat jewel-tone mosaic fragments, botanical tile patterns, crisp ...

LoRA training caption:

<v_pstyle_006> black lacquer mosaic, niche painterly style, painting of a solitary figure in a long coat standing beside a window, glossy black lacquer ground, flat jewel-tone mosaic fragments, botanical tile patterns

Figure 18: SC-40 painterly-style reference gallery. The concepts are fictional visual styles with stable medium, palette, texture, and linework constraints.

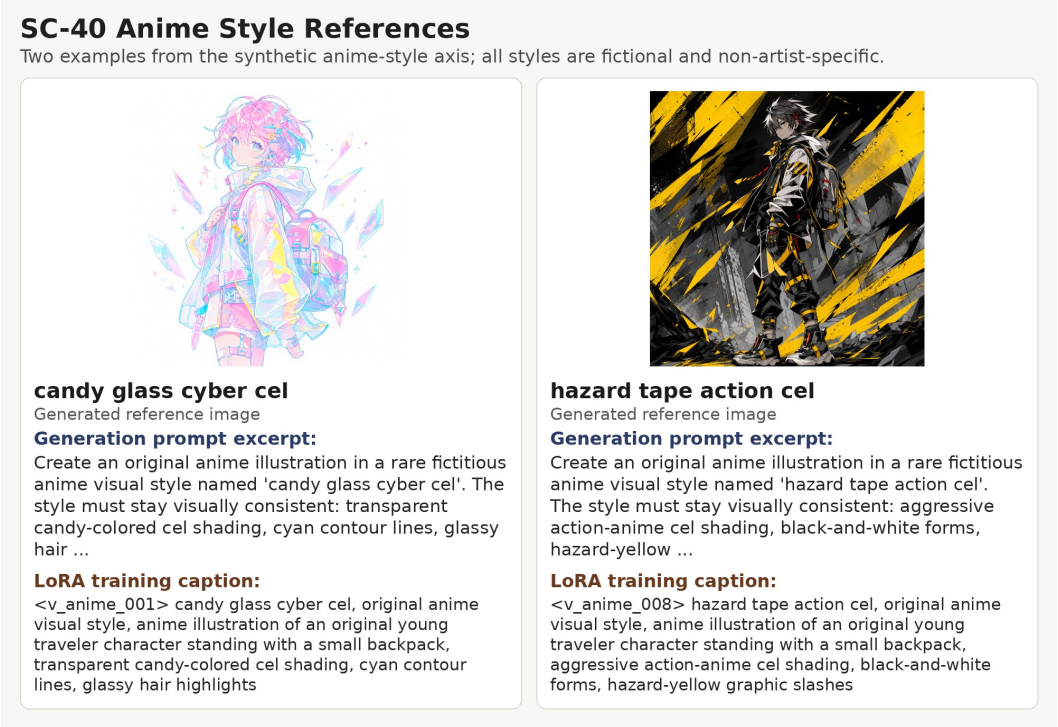


Figure 19: SC-40 anime-style reference gallery. The concepts are original non-artist-specific anime styles, each represented by a short LoRA caption and a longer, auditable GPT-Image-2 generation prompt used to synthesise the training set.

Q Storage-matched comparison at full scale

The main-text storage–quality panel (Tab. 1) compares LITHE against PEFT baselines at their native operating points. This section instead fixes the *deployment storage* axis and asks, at each budget, what quality is reachable. The study covers all 30 DreamBooth-30 subjects under one training and judging stack. Every compressed variant starts from the same freshly retrained dense rank-16 adapter, and "post hoc" means compressed after training with no further gradient step. Scores are the anchor-judge GPT-5.1 five-dimension average (SI/PF/VQ/SBD/CN) on the same rubric as Tab. 1.

Table 36: **Storage-matched comparison on DreamBooth-30** (30 subjects, GPT-5.1 five-dimension average). Every row below the dense anchor is either a lower-rank retrain or a post-hoc compression of the *same* dense rank-16 adapter. Methods trained into their compact form stay within 0.10 of the dense anchor, while every sub-2-bit post-hoc compression drops clear of that band.

Method	Disk (MB) ↓	GPT-5.1 avg ↑
Dense LoRA r=16 (compression source)	103.5	4.72
Post-hoc NF4 (4-bit)	32.4	4.72
LoRA r=4	26.1	4.70
LoRA r=2	13.2	4.67
LoRA r=1 (seeds 42 / 1)	6.8	4.64 / 4.64
Post-hoc 1-bit scalar	7.8	4.30
Post-hoc BitDelta	6.5	4.26
Post-hoc PQ (rate-matched)	3.1	4.16
LITHE codebook, applied post hoc (no STE)	1.9	4.13
LITHE (Ours), trained with STE, seeds 42 / 1	**3.31**	**4.62 / 4.64**

Trained-into-compact vs post-hoc. The table separates into two regimes. Methods trained directly into a compact form (LITHE and the low-rank LoRA retrains) cluster within 0.10 GPT-avg of the dense anchor. Every post-hoc compression below 2 bits falls 0.3–0.6 clear of that band, including the post-hoc baselines BitDelta and 1-bit scalar. Post hoc recovers only at 4 bits (NF4, at ∼10× LITHE's

bytes). The last two rows isolate the training axis. At the *identical* fixed rate of 0.42 bits/scalar, LITHE's own deterministic codebook scores 4.62 when trained into with STE and 4.13 when applied after training, so the difference lies entirely in where the optimisation happens. After LZMA-9 the post-hoc stream is more compressible (1.9 vs. 3.31 MB).

Comparison with rank-1 LoRA. The closest trained baseline in quality is a rank-1 LoRA. Across two training seeds (42, 1), the paired per-subject difference between LITHE and rank-1 is -0.01 GPT-avg. Its 95% bootstrap interval, $[-0.05, +0.03]$, contains zero (per-subject signs 13:15 with 2 even; each method's between-seed spread at most 0.02). The two methods are therefore statistically tied, and LITHE reaches this quality at about half of rank-1's disk footprint (3.31 vs. 6.8 MB).

Three further benchmarks. The same picture repeats on CC101-20, SC-40, and a 20-subject animal identity suite. For LITHE, rank-1, and the same-rate post-hoc control, the GPT-5.1 averages read 4.37 / 4.36 / 3.99 on CC101-20, 4.15 / 4.06 / 3.44 on SC-40, and 4.63 / 4.68 / 3.92 on the animal suite (LITHE nominally ahead on two). Across the four benchmarks (110 concepts), the STE-versus-post-hoc ordering never changes.

Storage-convention reconciliation. Tab. 37 collects the three byte counts that appear in the paper. Two describe the reported 304-Linear adapter, its raw bf16 size and its LZMA-9 size, and the LZMA-9 figure is the disk column above for the retrained stack. The third, 89.7 MB, is the bf16 size of a lighter 342-Linear configuration without modulation layers, which no reported quality row uses. The like-for-like axis is bits per scalar, where LITHE sits at 0.42 at the fixed 9-bit rate and 0.35 after LZMA-9.

Table 37: **Storage conventions for a dense rank-16 FLUX.1-dev adapter.** The first two figures describe the same adapter, and the third counts a smaller module set that no reported quality row uses.

Convention	Size
Raw bf16 payload (uncompressed adapter form)	153.3 MB
LZMA-9 on disk (this section's retrained stack)	103.5 MB
342-Linear configuration (no modulation layers)	89.7 MB

R Repeated-run confidence intervals

To bound the LITHE-versus-rank-1 tie with a measured interval rather than the trial-noise band of App. C.3, we retrained both methods from scratch on three seeds (42, 1000, 2000) under one single-GPU regime. All six models were judged in one common pass, so the comparison is paired seed for seed (Tab. 38).

Table 38: **Three-seed repeated-run comparison** on the five-concept protocol (GPT-5.1 anchor judge). The paired LITHE$-$rank-1 difference is $+0.031$ with a 95% CI of $[-0.107, +0.169]$ that contains zero, and the per-seed sign flips, so the two are statistically indistinguishable.

Seed	LoRA r=1 GPT-avg ↑	LITHE (Ours) GPT-avg ↑	LITHE$-r$=1
42	3.987	4.000	+0.013
1000	3.973	3.960	−0.013
2000	3.933	4.027	+0.093
mean ± 95% CI	3.964 ± 0.069	3.996 ± 0.083	$+0.031 \pm 0.138$

The between-seed standard deviation is 0.03 per method, about a third of that 0.09 band. No difference is detected, and the interval bounds any real difference by 0.17. The three-judge panel reads the same tie. As a regime check, the freshly single-GPU-trained LITHE reproduces the paper's four-GPU main number to within 0.02 GPT-avg.

S Expanded human evaluation

We ran a blinded human preference study with the pair as the independent unit. Eight internal volunteer raters cast 668 blinded votes over identity and style pairs. Each pair shows LITHE and the

storage-matched rank-1 LoRA at the same prompt and seed, with method labels hidden, and no rater saw a pair twice. A vote is one rater's complete four-question rating of one pair, so the four rows of Tab. 39 share one set of 427 scored pairs. The 668 votes split into a 94-vote shared calibration block and 574 votes over the 427 pairs (324 single-rater, 103 multi-rater). For multi-rater pairs we take the per-pair majority and record an even split as a tie. The reported p is a two-sided sign test on the decisive (non-tie) pairs.

Table 39: **Expanded human evaluation** (8 raters, 427 scored pairs, LITHE vs. storage-matched rank-1 LoRA). Ties dominate every question and no direction reaches significance.

Question	pairs	ties	rank-1 wins	LITHE (Ours) wins	sign-test p
Overall preference	427	283	66	78	0.36
Matches the reference	427	314	49	64	0.19
Follows the prompt	427	365	28	34	0.53
Visual quality	427	286	64	77	0.31

Ties dominate, and the shared calibration block gives raw coarse inter-rater agreement near 60%. With the pair as the unit of analysis, the overall-preference sign test gives p=0.36. Human raters find LITHE and the storage-matched rank-1 LoRA similar in quality, with no significant preference, and LITHE needs about half the disk footprint (3.31 vs. 6.8 MB).

T Modern paging comparison and capacity regime

The main-text serving comparison (Tab. 2) reports LITHE against the measured serial-fuse path and resident LoRA-loop and LoRA+compile upper bounds. This section compares against the mechanisms of modern paging systems. S-LoRA and Punica are autoregressive-LM serving engines whose code cannot load a diffusion transformer. Their unified-paging mechanism pools adapter weights together with a KV cache, which a diffusion transformer does not have, so we rebuild its adapter half. Their other signature mechanism, the batched multi-adapter SGMV kernel, is the kernel LITHE's same-batch path runs. We therefore built the remaining components of a modern pager: a resident pinned-host pool, conversion off the critical path, and asynchronous double-buffered prefetch that hides the transfer completely (exposed load wait below 1 ms). We measured this pager against the serial baseline and LITHE on one exclusively held A100 at the identical B=96 workload (Tab. 40). An earlier pass, in which LITHE took 16.43 s, measured disk-backed serial fuse on the same workload at 163.5 s from cold CephFS, 172.8±2.4 s with a warm OS page cache, and 154.7±0.9 s from local NVMe. CPU-side deserialization and conversion of the weight files dominate the per-swap load, and page-cache warming does not shorten them, so the three storage tiers differ only within run-to-run variation. The advantage of LITHE therefore comes from the representation and the batched kernel and holds across the three storage tiers.

Table 40: **Serving-path comparison**, B=96 distinct rank-16 FLUX.1-dev adapters, 256^2/4-step, 1×A100 80 GB, 3-trial mean±std. Hiding the transfer buys under 4%, and amortising every conversion reaches 41.57 s. Only the batched kernel closes the rest; a fuse-per-adapter pager cannot run it, and LITHE's same-batch path is its diffusion port.

Serving path	Wall (s) ↓	vs. ours ↓
Serial CPU-pinned	59.16±0.30	3.63×
Async pipelined pager (transfer fully hidden)	56.88±0.35	3.49×
Diffusers hotswap API with graph compilation	73.79±0.58	4.53×
Idealised hotswap (conversion amortised, in-place)	41.57±0.06	2.55×
Dense factors resident, same SGMV kernel (control)	16.26±0.12	1.00×
LITHE same-batch routing (Ours)	**16.30±0.08**	**1.00×**

Comparison with the released SGMV kernel. We additionally ran the vLLM release of the Punica/S-LoRA SGMV kernel head-to-head against our Triton SGMV on the real 304-layer FLUX.1-dev adapter shapes at B=96 (Tab. 41). The two kernels agree to numerical precision (worst-case cosine 1.0000, max relative error 10^{-4}); after FLUX-shape tuning ours runs 2.43× faster. The

modern batched mechanism is therefore the kernel LITHE serves with, verified against the released implementation.

Table 41: **Batched SGMV kernel, one full 304-layer pass** (B=96, real FLUX.1-dev shapes). Numerically identical outputs; LITHE is tuned for FLUX's wide layers.

Kernel	Time (ms) ↓
vLLM release of the Punica / S-LoRA SGMV kernel	473.9
Triton SGMV (Ours)	**195.0**

Capacity regime. Past the residency ceiling the comparison becomes one of capacity, not speed. On a 500-distinct-adapter workload at a batch width where both representations fit (B=64), dense host-paging and LITHE serve at the same pipelined wall (170.2 s under a Zipf revisit, 170.24 vs. 170.15 s overlap-estimate). In strict serial order the pager is even slightly faster, since a ∼9.4 ms page-in costs less than our on-device decode. The pager moves 58–86 GB per pass over PCIe, while LITHE moves none. At the table's B=96 width, dense paging cannot complete a single batch and runs out of memory at every tested cache cap. LITHE serves all 500 requests from a 4.03 GB int16 index footprint (Diffusers split-QKV layout). Speed is equal wherever dense fits; only LITHE still fits at scale.

U Migration and from-scratch control

Migrating an existing catalogue into LITHE does not require training from scratch. Each adapter is warm-started from the scale-matched post-hoc encode, which is already a serving-ready 0.42-bits/scalar adapter at zero training, and then continued with a short straight-through-estimator fine-tune. Tab. 42 measures the recovery curve against a from-scratch control trained at the same step budgets. Every point was scored in one common judging pass, so the two columns are directly comparable.

Table 42: **Migration recovery curve vs. from-scratch control** (five SC-40 concepts, GPT-5.1 anchor judge). GPT-avg after each fine-tune step budget for the warm-started migration and for a from-scratch control at the same budget.

Fine-tune steps	Migration (warm-started) ↑	From-scratch control ↑
0 (encode only)	3.493	unusable (random init)
100	3.587	3.613
250	3.720	3.773
500	3.880	3.893

The from-scratch 750-step anchor, which follows the SC-40 training recipe, reads 4.013 in this same pass (three-seed mean 3.996±0.083, App. R) and bounds both columns. At every matched budget, warm-start and from-scratch sit within 0.06 GPT-avg, with per-concept signs split near evenly. Warm-starting therefore neither accelerates nor penalises training, and the recovery is a property of the short STE fine-tune itself. What the encode adds is a serving-ready adapter at zero training. A catalogue can serve from day one, and quality climbs with the budget its owner chooses to spend: 74% of the gap is recovered at two-thirds of the budget, at ∼4 / ∼9 / ∼18 minutes per adapter for 100 / 250 / 500 steps on one A100, against ∼27 minutes to train from scratch. Migration is therefore a bounded, one-time per-adapter cost, statistically indistinguishable from retraining at every matched budget.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Sec. 1 state the storage, quality, hot-swap, composition, and same-batch serving claims, including the parity with a graph-compiled dense LoRA baseline at $1024^2/28$-step (133.4 s vs 132.4 s, within 0.8%); Secs. 4–5.2 report the corresponding benchmark, ablation, and serving measurements. The text explicitly separates same-forward batch capacity from the 10,000-slot index-resident inventory, restricts cross-architecture claims to the three evaluated softmax-attention backbones (FLUX.1-dev, SDXL, Qwen-Image-20B), and frames the twin codebook in terms of per-side scale asymmetry rather than pool-sign separation per the four-cell ablation.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Limitations paragraph in the main body explicitly bounds the reported scope to image-diffusion personalization and excludes video generation and vision–language model adapter serving. It also flags the cross-backbone identity gap on Qwen-Image-20B (LLM-avg 0.21), which a per-backbone codebook adjustment closes (App. D.9), and the smaller SDXL 1.0 gap of 0.02 (§4.5, Tab. 4). App. A expands these with the broader-impact and safeguards discussion. Sec. 5.2 reports the measured A100 index-resident inventory stress test (10,000 verified slots at 75.7 GB peak) separately from the same-forward heterogeneous batch capacity, since they are different resource ceilings.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.

- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not state a formal theorem. App. B provides only a heuristic Gaussian rate-scaling calibration with an explicit Monte Carlo probe (Tab. 6); the main claims do not rely on it as a proof.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sec. 3 specifies the adapter parameterisation, the deterministic Hadamard H_9 twin codebook, the per-side scale asymmetry, the $K{=}512$, $B{=}64$, $L{=}3$, rank 16 defaults, lazy re-quantisation period of 10 steps, bf16 mixed precision, AdamW with LR 10^{-4}, and training budgets of 500 steps (identity) and 750 steps (SC-40) at 512^2. Sec. 4 specifies the four benchmarks (DreamBooth-30, CC101-20, SCIdentity, SC-40), the three-LLM judge consensus protocol (GPT-5.1, Claude Sonnet 4.6, Gemini 2.5 Flash) and the embedding-metric cross-checks (DreamSim, CLIP-T, CLIP-I). App. C (Tab. 7) tabulates every hyperparameter; App. C.3 pins the deterministic codebook construction; App. L specifies dataset composition, prompt families, and synthetic-benchmark generation; App. M provides the verbatim per-task judge rubrics. The codebook is byte-exact reproducible from a fixed Sylvester-Hadamard rule, so reproduction does not depend on transferring codebook artefacts.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may

be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code repository (https://github.com/JianJiang1995/LITHE) releases training, encoding, decoding, inference, and multi-adapter serving code, including the Triton decode and routing kernels and the serving benchmark scripts. It also holds the deterministic Hadamard codebook construction, the evaluation prompt lists, the prompt-level judge and embedding scores, and the judge prompt templates. The 110 trained LITHE adapters are at https://huggingface.co/Fanjue/LITHE, and all 16,570 evaluated images with their per-image judge outputs are at https://huggingface.co/datasets/Fanjue/LITHE-eval. The SCIdentity and SC-40 training images are not released. The paper itself provides full method details (Sec. 3, App. C, App. C.3, Algorithm 1) and benchmark/evaluation protocols (Sec. 4, App. L, App. M) sufficient to reproduce the results from scratch on the cited public backbones. The four evaluation benchmarks are either public (DreamBooth-30, CustomConcept101) or fully documented for re-generation in App. L (SCIdentity, SC-40).

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Sec. 4 gives the four-benchmark suite and the three-LLM judge consensus protocol with embedding-metric cross-checks; Sec. 3 states the canonical $K{=}512$, $B{=}64$, $L{=}3$, rank 16, multiscale RVQ, 500-step (identity) and 750-step (SC-40) training at 512^2, bf16, AdamW, and LR 10^{-4}, constant for identity and cosine with 100 warmup steps for SC-40, matching Tab. 7 (App. C.1). App. C.1 (Tab. 7) tabulates every hyperparameter, App. C.2 pins the bf16/fp32 precision split, App. C.3 pins the deterministic Hadamard configuration, and App. L provides the data layout, prompt families, and synthetic-benchmark generation protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The headline trained-adapter rows are reported as three-run means with run-level standard deviations across all four benchmarks (Tab. 1 in the main and the per-benchmark tables in App. F: Tab. 20, Tab. 21, Tab. 22, Tab. 23). Wall-time numbers in Tab. 2 are three-trial means with standard deviations (LITHE 133.42 ± 0.03 s and LoRA+compile 132.42 ± 0.16 s at $1024^2/28$ $B{=}14$, with the serial-fuse production baseline 195.3 ± 1.7 s cross-validated on 4 independent A100 hosts; App. D.8). Robustness to closed-model judge bias is addressed by the three-LLM judge consensus (GPT-5.1, Claude Sonnet 4.6, Gemini 2.5 Flash) with per-judge breakdowns in App. F and embedding-metric cross-checks (DreamSim, CLIP-T, CLIP-I) in App. K.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).

- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All reported wall-time, residency, and hot-swap benchmarks were measured on A100 80 GB GPUs, as stated in the captions of Tab. 5 and Tab. 2 and the text of Sec. 5.2. Per-experiment configurations specify resolution, step count, and batch size at $1024^2/28$ B=14 (133.4 s LITHE, 132.4 s LoRA+compile, 188.3 s resident LoRA loop, 195.3 s serial fuse CephFS cold), $512^2/16$ B-sweep, and $256^2/4$ B=96. Hot-swap latencies are quoted as 4.56 ms warm and $\sim$10 ms cold for the resident path, and $\sim$0.9 s for the standard Diffusers serial-fuse path on a CephFS cold catalogue (4-host consensus, App. D.8). Every reported FLUX.1-dev adapter is trained on 4$\times$A100 80 GB with batch 1 per GPU (Tab. 7). App. C.1 states the per-row hardware. We do not provide a complete cluster-scale total-compute accounting for all preliminary or failed runs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public benchmark data (DreamBooth-30, CustomConcept101) and synthetically generated reference data (SCIdentity, SC-40) and does not involve human-subject studies, crowdsourcing, or private personal data. App. A discusses misuse risks of cheaper personalization (non-consensual likeness or style imitation) and identifies consent-based reference ingestion, prompt/reference moderation, and watermarking as the deployment-layer safeguards.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: App. A discusses positive deployment efficiency (the bit-packed adapter representation reduces per-subject VRAM from $\sim$172 MB to $\sim$4.2 MB and lets a single A100 80 GB host 10,000 verified-resident adapter slots, reducing the energy and storage cost of per-subject deployment) and negative misuse risks (non-consensual personalization or impersonation, since we do not release a new pretrained image generator). The paper frames content moderation, consent-based reference ingestion, and watermarking as deployment safeguards orthogonal to the adapter representation.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: We do not release a new pretrained image generator. The released artifacts are code, the deterministic Hadamard codebook construction (byte-exact reproducible from a fixed Sylvester-Hadamard rule), and the trained adapter index streams. The SCIdentity and SC-40 training images are not released. SCIdentity and SC-40 contain no real-person likeness (SCIdentity uses rare non-human individuals; SC-40 uses controlled style/concept axes). App. A discusses consent-based reference-image ingestion, prompt/reference moderation, and watermarking as required deployment safeguards for systems using this representation.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Existing models, methods, and public benchmarks are cited where introduced, including FLUX.1-dev, SDXL 1.0, Qwen-Image-20B, DreamBooth, CustomConcept101, and the LoRA / PEFT baselines. License and terms-of-use constraints for the underlying

checkpoints and public datasets are listed in the code repository (https://github.com/JianJiang1995/LITHE) and the adapter model card.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: SCIdentity and SC-40 are documented in App. L (App. L.3, App. L.4, App. L.5), including data generation protocol, reference counts, triggers, captions, prompt families, and a representative gallery (App. P). The deterministic Hadamard codebook construction $(H_9, \Phi, \sigma_A^{\mathrm{cb}}, \sigma_B^{\mathrm{cb}})$ is fully specified in App. C.3 so it can be regenerated bit-exact from the Sylvester rule. Code, evaluation prompts, and prompt-level scores are released at https://github.com/JianJiang1995/LITHE. The SCIdentity and SC-40 training images are not released, and App. L documents their generation protocol.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. Its human evaluation (App. S) is an aesthetic quality assessment of generated images: internal volunteer annotators compared anonymized image pairs on four questions (overall preference, match to the reference, prompt following, visual quality) with method labels hidden. No crowdsourcing platform was used, no compensation was paid, and no information about the annotators was collected or analyzed. The rating protocol and aggregation are documented in App. S.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects. The human evaluation (App. S) assesses the aesthetic quality of model outputs; annotators only compared anonymized generated images, and no information about the annotators was collected or analyzed.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The three-LLM judge consensus (GPT-5.1, Claude Sonnet 4.6, Gemini 2.5 Flash) is part of the evaluation protocol, not the core adapter-training method, and is described in Sec. 4 with the verbatim per-task rubrics in App. M. Synthetic-benchmark prompts and captions for SCIdentity / SC-40 are documented in App. L and App. L.5. The core method (Hadamard-indexed twin RVQ codebook training) does not involve LLMs.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.