CALIBCACHE: CACHE-TO-FULL FEATURE DYNAMICS CALIBRATION FOR DIFFUSION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Transformers generate high-quality images, but iterative sampling remains computationally expensive. Feature caching reduces the number of full network evaluations, yet its approximation error can become more severe under short sampling schedules, where fewer full-compute steps and a coarser sampling grid can make history-based extrapolation less reliable. In this paper, we propose CALIBCACHE, a lightweight offline calibration method that corrects systematic discrepancies between cached and full-compute feature dynamics. Specifically, CALIBCACHE calibrates feature acceleration, defined as the second-order finite difference across sampling steps. Using only a single pair of full-compute and naive-cache trajectories at each setting, CALIBCACHE fits scalar acceleration maps offline by centered least squares. These maps are then applied during sampling without additional full-network evaluations or online optimization. Across FLUX.1-dev and Qwen-Image on DrawBench and DiT-XL/2 on ImageNet-256, CALIBCACHE consistently improves fidelity or generation quality over TaylorSeer under 20-step sampling at cache intervals $N = 3, 4, 5$. At $N = 5$, it reduces FLUX.1-dev LPIPS from 0.563 to 0.429 while achieving a $2.86\times$ denoising speedup over full-compute sampling, and lowers DiT-XL/2 FID from 16.39 to 10.37 relative to TaylorSeer.

1 INTRODUCTION

Diffusion Transformers (DiTs) have become a central backbone for image generation (Peebles & Xie, 2023). Although recent models can generate high-quality images with only a few tens of sampling steps, inference remains expensive because each step requires costly Transformer attention and feed-forward computations. Training-free feature caching reduces this cost by reusing intermediate features across adjacent sampling steps (Ma et al., 2023; Selvaraju et al., 2024; Chen et al., 2024; Zou et al., 2025), but reused features can deviate from their full-compute counterparts as the reuse interval grows.

Short sampling schedules can exacerbate this discrepancy. Fewer sampling steps provide fewer full-compute observations overall, while a cache interval of fixed N spans a larger fraction of the rollout. History-based extrapolation can therefore become less reliable under coarse sampling, where sparse historical observations provide a weaker basis for estimating local feature dynamics and prediction errors can accumulate across reused steps. These challenges motivate a practical correction method that improves feature quality while preserving the computational savings of caching.

Existing caching methods address feature approximation in different ways. Naive reuse is computationally inexpensive but can produce stale features at larger cache intervals. TaylorSeer (Liu et al., 2025b) and HiCache (Feng et al., 2026) extrapolate future features from sampling history using Taylor and scaled-Hermite predictors, respectively. TC-Padé (Cui et al., 2026) uses rational residual prediction with online coefficient adaptation, while L^2P (Shen et al., 2026) learns linear combinations of historical features through a lightweight linear layer, supervised by reconstruction targets from full-compute trajectories. Rather than learning feature evolution from full-compute trajectories alone, we ask a complementary question: *can an explicitly aligned pair of full-compute and naive-cache trajectories provide direct supervision for a compact, fixed calibration of cached feature dynamics?*

In this paper, we propose CALIBCACHE, a lightweight offline method built on *paired cache-to-full acceleration supervision*. We generate one full-compute and one naive-cache trajectory from the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

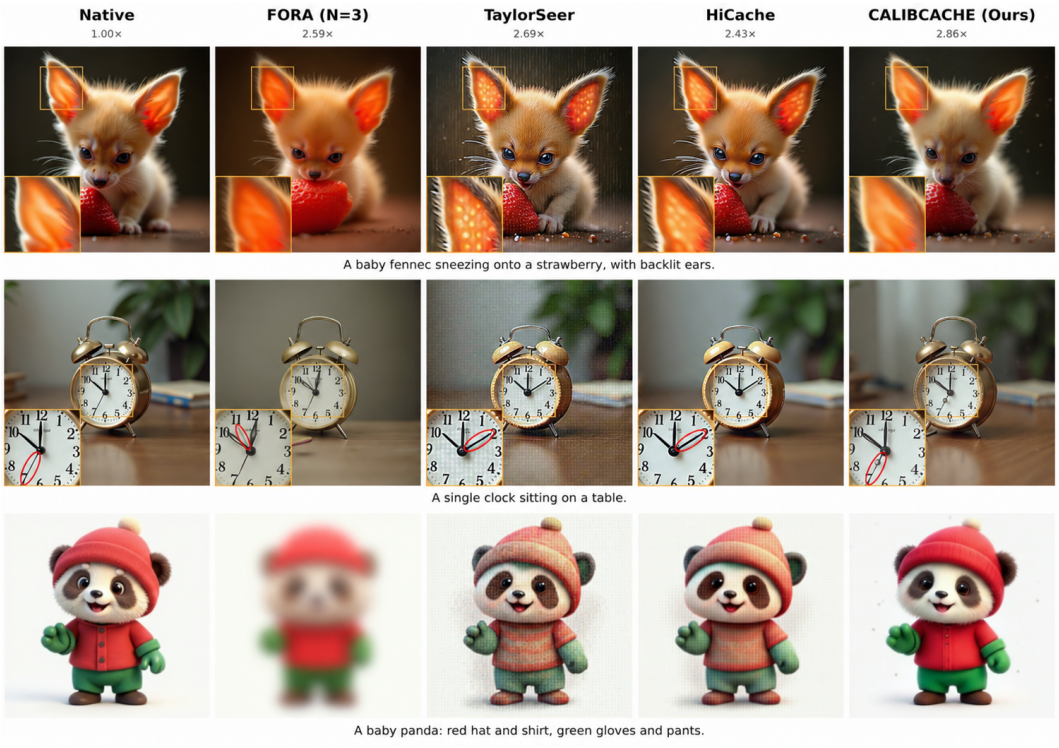


Figure 1: Five-column qualitative comparison on FLUX.1-dev with 20-step DrawBench sampling. Within each row, all methods use the same prompt and initial noise. The columns show Native, FORA ($N = 3$, $2.59\times$), TaylorSeer ($N = 5$, $2.69\times$), HiCache ($N = 5$, $2.43\times$), and CALIBCACHE ($N = 5$, $2.86\times$); speedups refer to denoising latency relative to Native. FORA uses $N = 3$ to show a broadly comparable speedup. The enlarged regions highlight differences in the ear and clock-hand details; in these examples, CALIBCACHE more closely preserves the Native appearance.

same condition and initial noise, then pair their feature accelerations at matching sampling steps and residual boundaries. Feature acceleration is defined as the second finite difference of residual features. Because the two trajectories evolve independently under full computation and the prescribed cache schedule, respectively, their pairing directly exposes the cache-induced discrepancy in feature dynamics. Centered least squares then estimates one shared scalar per step–branch group to map naive-cache acceleration to its full-compute counterpart. The resulting supervision therefore relates feature dynamics across the two rollouts, rather than reconstructing current features from history within full-compute trajectories.

For each deployment configuration, offline preparation requires only one trajectory pair followed by closed-form scalar fitting. The correction coefficients are estimated directly from the paired observations, without coefficient sweeps or iterative optimization, and the resulting maps are fitted once and reused across subsequent samples. At inference, the frozen maps act on online reference accelerations and correct cached residuals through lightweight elementwise operations, preserving the original cache schedule without additional full-network evaluations or online optimization. A controlled DiT-XL/2 study (DDIM-20, $N = 5$, 50K images per variant) further supports acceleration as the calibration target: under the same two-anchor update and deployment policy, acceleration supervision improves all five distributional metrics over residual and velocity supervision.

We evaluate CALIBCACHE on FLUX.1-dev and Qwen-Image using DrawBench with 20-step sampling at $N = 3, 4, 5$. Across these intervals, it improves PSNR, SSIM, and LPIPS over TaylorSeer and HiCache on both models while reducing denoising latency. On FLUX.1-dev at $N = 5$, ImageReward increases from 0.775 to 0.899, with a $2.86\times$ speedup over full computation versus TaylorSeer's $2.69\times$ under the same cache schedule. Figure 1 illustrates closer preservation of Native appearance in three matched-prompt examples. On Qwen-Image, CLIP and ImageReward also improve across

all three intervals. DiT-XL/2 experiments on ImageNet-256 with DDIM-20 further show lower FID and sFID than TaylorSeer at $N = 3, 4, 5$. Our contributions are as follows:

- We formulate paired cache-to-full acceleration supervision by matching observations from independently evolving full-compute and naive-cache trajectories. A discrete error identity motivates the target; a controlled DiT study compares supervision choices within a shared two-anchor predictor.
- We design CALIBCACHE, which estimates grouped scalar corrections by closed-form centered least squares from one trajectory pair per deployment configuration. The frozen maps preserve the cache schedule without additional full-network evaluations or online optimization.
- We evaluate CALIBCACHE on FLUX.1-dev, Qwen-Image, and DiT-XL/2. Compared with TaylorSeer and HiCache, CALIBCACHE improves fidelity to full-compute outputs and reduces denoising latency on both text-to-image models, while achieving lower FID and sFID on DiT-XL/2.

2 RELATED WORK

Cache policies. Cache policies determine when and where to skip network computation. DeepCache (Ma et al., 2023) reuses high-level U-Net features across denoising steps while updating low-level features, exploiting temporal redundancy. Δ-DiT (Chen et al., 2024) combines residual caching with stage-dependent block selection, skipping rear blocks during early sampling and front blocks during later sampling. ToCa (Zou et al., 2025) selects tokens for reuse according to temporal redundancy and error propagation, with different caching ratios across layer types and depths. TeaCache (Liu et al., 2025a) estimates output changes from timestep-modulated input differences, using rescaled estimates to decide when cached outputs should be refreshed. SeaCache (Chung et al., 2026) filters input features to suppress noise and retain content-relevant components, producing redundancy estimates that account for spectral evolution during denoising. Together, these methods allocate computation according to temporal redundancy, differing in the granularity of reuse and the criteria for refreshing cached features.

Feature prediction. Given a cache policy, feature prediction determines how to approximate the outputs of skipped computation. FORA (Selvaraju et al., 2024) directly reuses cached attention and MLP outputs, yielding a piecewise-constant approximation between full-compute steps. TaylorSeer (Liu et al., 2025b) estimates higher-order feature derivatives from historical observations using finite differences and extrapolates future features through Taylor expansions. HiCache (Feng et al., 2026) predicts features with Hermite polynomials and introduces dual scaling of the prediction horizon and higher-order terms for numerical stability. FoCa (Zheng et al., 2025) treats feature evolution as an ODE, combining multistep forecasting with a lightweight corrector anchored to the latest fully computed feature. TC-Padé (Cui et al., 2026) models residual evolution through rational functions, adapting coefficients to historical changes and prediction strategies to different denoising stages.

While these forecasting methods model feature evolution through numerical prediction rules, offline learning uses full-compute supervision to learn how cached features should be combined or transformed. LearniBridge (Huang et al., 2026) trains low-rank adapters in the final Transformer block to reconstruct skipped-step representations from cached inputs under full-compute supervision, bypassing earlier blocks during inference. L^2P (Shen et al., 2026) learns timestep-specific linear combinations of historical features from full-compute trajectories, replacing formula-derived coefficients with weights optimized for feature reconstruction. CALIBCACHE instead calibrates feature acceleration using aligned observations from independently evolving full-compute and naive-cache trajectories, with scalar maps fitted in closed form from one trajectory pair.

3 METHOD

For a fixed model and cache schedule, CALIBCACHE calibrates discrepancies between cached and full-compute feature acceleration. As shown in Figure 2, one calibration sample supplies a full-compute/naive-cache trajectory pair for fitting grouped scalar maps offline; the frozen maps then

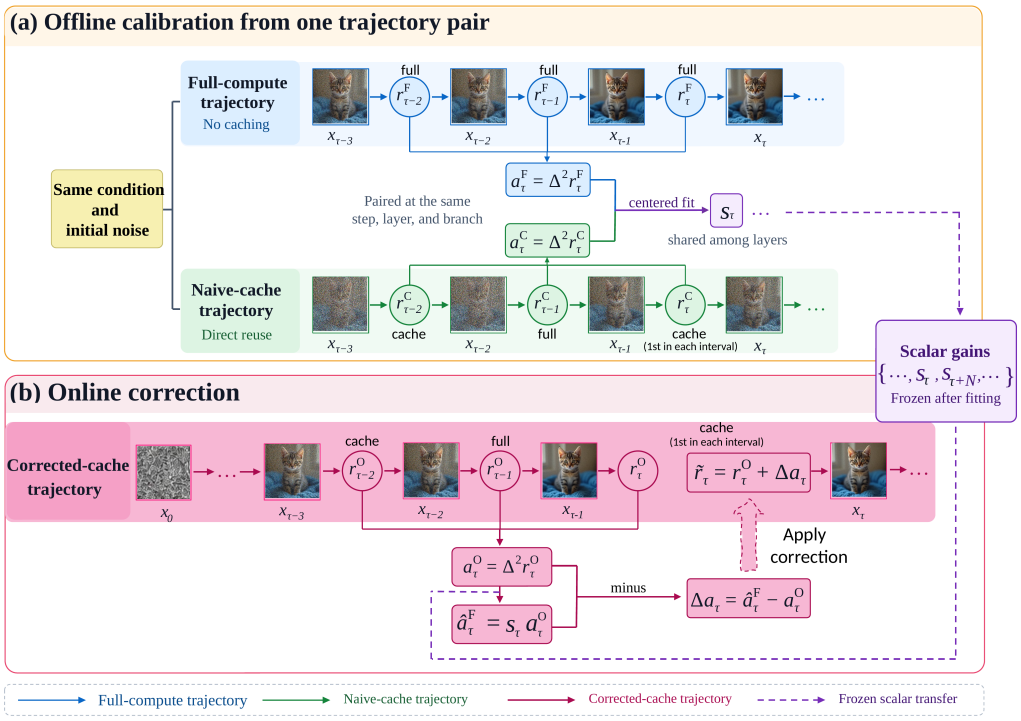


Figure 2: Overview of CALIBCACHE. (a) One full-compute/naive-cache trajectory pair with the same condition and initial noise provides acceleration observations for centered scalar fitting at the first cache step of each interval. (b) On the online corrected-cache rollout, the frozen scalars estimate full-compute acceleration; the difference from the online reference acceleration corrects the current cached residual without additional full-network evaluations.

correct residuals along an online corrected-cache rollout. The calibration and correction target the first cache step of each interval.

3.1 FEATURE ACCELERATION

Consider a diffusion or flow-matching sampler with T sampling steps, indexed by $\tau = 0, \ldots, T-1$ in execution order from noise to data. A *branch* refers to a residual-producing submodule within a Transformer layer, such as an attention or MLP submodule. Let $\mathbf{r}^{\mathrm{F}}_{\tau,\ell,b}$ and $\mathbf{r}^{\mathrm{C}}_{\tau,\ell,b}$ denote the residual features of branch b in layer ℓ at sampling step τ along the full-compute and naive-cache trajectories, respectively. Let $\mathcal{S}$ denote a prescribed cache schedule that specifies which branch evaluations are skipped, and let N denote its nominal cache interval. The full-compute trajectory evaluates every branch at every step. The naive-cache trajectory follows $\mathcal{S}$, evaluating branches normally at full-compute steps and directly reusing their latest full-compute residuals at cache steps. For either trajectory, suppressing the layer and branch indices, define the first- and second-order finite differences for $\tau \geq 1$ and $\tau \geq 2$, respectively, as

$$\mathbf{v}_\tau = \mathbf{r}_\tau - \mathbf{r}_{\tau-1}, \qquad \mathbf{a}_\tau = \Delta^2 \mathbf{r}_\tau = \mathbf{r}_\tau - 2\mathbf{r}_{\tau-1} + \mathbf{r}_{\tau-2}. \tag{1}$$

We refer to $\mathbf{a}_\tau$ as feature acceleration. This term denotes a second finite difference of features and does not imply a continuous-time derivative. The superscripts F and C distinguish full-compute and naive-cache quantities: $\mathbf{a}^{\mathrm{F}}_{\tau,\ell,b}$ and $\mathbf{a}^{\mathrm{C}}_{\tau,\ell,b}$ are computed from the residual sequences of their respective trajectories using Equation (1).

Acceleration calibration. We address the discrepancy between naive-cache and full-compute feature accelerations introduced by residual reuse. We fit grouped scalar maps from naive-cache accelerations to their full-compute counterparts using paired calibration trajectories. The fitted maps are then used to correct cached residuals at the first cache step of each interval. We compare

acceleration supervision with residual and velocity supervision in Table 4; the fitting targets and shared deployment protocol are specified in Appendix A.4.

3.2 OFFLINE CALIBRATION

We fix one conditioning signal (a text prompt or class label), initial noise, guidance, solver, and timestep schedule, then record a full-compute trajectory and a naive-cache trajectory under the prescribed cache schedule. The trajectories evolve independently; at the first cache step of each interval, accelerations computed from their respective residual sequences are paired for the same sampling step τ, layer ℓ, and branch b, as in Figure 2(a). This pairing matches the step, layer, and branch indices across trajectories, without requiring their intermediate latent states to coincide: each observation is extracted from the state reached by its own rollout.

Centered scalar fitting. We group observations by sampling step and computational branch, $g = G(\tau, b)$, and share each scalar across layers and tokens in its group. Let $\mathcal{I}_g$ index the paired channel vectors $(\mathbf{a}_j^{\mathrm{C}}, \mathbf{a}_j^{\mathrm{F}})$ at corresponding layers and token positions. Each layer and token position contributes a paired vector, so a single trajectory pair supplies many observations for estimating one scalar. This sharing retains dependence on the sampling step and computational branch while restricting the fitted map to a common scale within each group. Define the group means as $\bar{\mathbf{a}}_g^{\star} = |\mathcal{I}_g|^{-1} \sum_{j \in \mathcal{I}_g} \mathbf{a}_j^{\star}$ for $\star \in \{\mathrm{F}, \mathrm{C}\}$. We estimate a shared scale from centered observations:

$$s_g = \arg\min_{s \in \mathbb{R}} \sum_{j \in \mathcal{I}_g} \left\| (\mathbf{a}_j^{\mathrm{F}} - \bar{\mathbf{a}}_g^{\mathrm{F}}) - s(\mathbf{a}_j^{\mathrm{C}} - \bar{\mathbf{a}}_g^{\mathrm{C}}) \right\|_2^2. \tag{2}$$

For nonzero centered input energy, the closed-form solution is

$$s_g = \frac{\sum_{j \in \mathcal{I}_g} \left\langle \mathbf{a}_j^{\mathrm{C}} - \bar{\mathbf{a}}_g^{\mathrm{C}}, \mathbf{a}_j^{\mathrm{F}} - \bar{\mathbf{a}}_g^{\mathrm{F}} \right\rangle}{\sum_{j \in \mathcal{I}_g} \left\| \mathbf{a}_j^{\mathrm{C}} - \bar{\mathbf{a}}_g^{\mathrm{C}} \right\|_2^2}. \tag{3}$$

The numerator measures alignment between centered accelerations, normalized by the cached energy in the denominator. Thus, the shared scale is estimated from within-group co-variation rather than group means. The deployed map is scalar multiplication by s_g; the means are used only during fitting. We write $s_g = s_{\tau,b}$ for $g = G(\tau, b)$ and suppress the branch index as s_τ in Figure 2 and the online equations below. The fitted scalars remain frozen for all subsequent samples under the same deployment configuration.

3.3 ONLINE CORRECTION

Let $\mathbf{r}_{\tau,\ell,b}^{\mathrm{O}}$ denote the online reference residual supplied to the calibration operator, equal to the current pre-correction residual at a correction step. The preceding $\mathbf{r}_{\tau-1,\ell,b}^{\mathrm{O}}$ and $\mathbf{r}_{\tau-2,\ell,b}^{\mathrm{O}}$ belong to this reference sequence for the same layer and branch, while $\widetilde{\mathbf{r}}_{\tau,\ell,b}$ is the corrected residual feature. The superscript O distinguishes online references from the offline full-compute (F) and naive-cache (C) trajectories. Suppressing layer and branch indices, at an eligible first cache step with two preceding references, we compute $\mathbf{a}_\tau^{\mathrm{O}} = \Delta^2 \mathbf{r}_\tau^{\mathrm{O}}$ using Equation (1). Following Figure 2(b), the frozen map gives a full-compute acceleration estimate, whose discrepancy from the reference acceleration corrects the current residual:

$$\begin{aligned} \widehat{\mathbf{a}}_\tau^{\mathrm{F}} &= s_\tau \mathbf{a}_\tau^{\mathrm{O}}, \qquad \Delta \mathbf{a}_\tau = \widehat{\mathbf{a}}_\tau^{\mathrm{F}} - \mathbf{a}_\tau^{\mathrm{O}}, \\ \widetilde{\mathbf{r}}_\tau &= \mathbf{r}_\tau^{\mathrm{O}} + \Delta \mathbf{a}_\tau = \mathbf{r}_\tau^{\mathrm{O}} + (s_\tau - 1)\mathbf{a}_\tau^{\mathrm{O}}. \end{aligned} \tag{4}$$

Here $\Delta \mathbf{a}_\tau$ is the correction discrepancy and $\widehat{\mathbf{a}}_\tau^{\mathrm{F}}$ is a mapped estimate, not an additional full-compute evaluation. With the preceding references fixed, adding $\Delta \mathbf{a}_\tau$ changes the current residual's second difference by the same amount. This local relation is defined relative to the reference sequence; the residual sequence actually used by the network also reflects the deployment policy. The correction remains sample-dependent through $\mathbf{a}_\tau^{\mathrm{O}}$, while $s_\tau = 1$ leaves the current residual unchanged.

The correction operator itself uses a three-point finite difference and elementwise scaling and addition, costing $O(d)$ for a d-dimensional residual token. The coefficient storage is $O(|\mathcal{G}|)$ for step–branch

groups $\mathcal{G}$, with one scalar per group; runtime feature buffers are separate. The operator preserves the full/cache schedule and requires neither additional full-network evaluations nor online optimization. Implementation details and deployment-policy analyses are provided in Section 4 and Appendix A.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and benchmarks. We evaluate 20-step sampling at cache intervals $N \in \{3, 4, 5\}$ on three diffusion Transformers. FLUX.1-dev (Black Forest Labs, 2024) uses shifted flow-matching steps and guidance 3.5; Qwen-Image (Wu et al., 2025) uses true classifier-free guidance (CFG) 4.0. Both text-to-image models are evaluated on DrawBench-200 (Saharia et al., 2022) at 1024^2 resolution. DiT-XL/2 (Peebles & Xie, 2023) is evaluated on ImageNet-256 using DDIM-20 (Song et al., 2021), CFG 1.5, and 50K generated images per configuration. Each model–interval configuration uses one calibration condition (a COCO (Lin et al., 2014) caption disjoint from evaluation prompts, or a class label) and fixed initial noise to generate a full-compute/naive-cache trajectory pair. The centered estimator in Equation (3) fits one scalar per correction step and branch, reused across the entire evaluation set.

The model-named baseline rows use full computation at all 20 sampling steps. Calibration and deployment details are given in Appendix A.1.

Baselines. We compare CALIBCACHE with FORA direct reuse (Selvaraju et al., 2024) and two history-based forecasting baselines, TaylorSeer (Liu et al., 2025b) and HiCache (Feng et al., 2026), both at order $O = 2$. HiCache uses the authors' official scaled-Hermite predictor (commit `4d0839f`) with scale 0.6.

Unified deployment policy. Across all three models, TaylorSeer, HiCache, and CALIBCACHE share the same full/cache schedule: warmup-2 at $N = 3$ and warmup-3 at $N = 4, 5$, followed by full evaluations every N steps from the last warmup step. FORA instead refreshes every N steps from the initial full-compute step, without extra warmup. CALIBCACHE applies Equation (4) only at the first cache step of each interval, writes the corrected residual into the reuse buffer, and reuses it until the next full-compute step. A separate predictor history records the latest full-compute anchor at every cache step, supplying the references $\mathbf{r}^{\mathrm{O}}$ in Section 3.3 and isolating direct feedback from corrected reuse. Full-compute steps refresh both the anchor and the reuse buffer on the deployed rollout. At $N = 4$, the final step (logical step 0) is a first cache step; we skip its correction to avoid a large terminal perturbation that could degrade image quality. Appendix A.1 lists the exact step positions.

Metrics and efficiency. PSNR, SSIM, and LPIPS (Zhang et al., 2018) compare each final image with the full-compute output generated from the same prompt and seed, measuring *full-compute fidelity*. CLIP (Radford et al., 2021) and ImageReward (IR) (Xu et al., 2023) assess semantic alignment and learned human-preference scores, respectively. For DiT-XL/2 on ImageNet-256, we report FID (Heusel et al., 2017), sFID, IS, precision, and recall. We measure per-image denoising latency on one NVIDIA H800 GPU at batch size one, averaging 30 CUDA-event measurements after ten benchmark warmup runs. Model weights and calibration parameters are GPU-resident before timing, which excludes text encoding, VAE decoding, pack preparation, and disk I/O. FLOPs cover the same 20-step denoising-only region as latency. We use `torch.profiler` for profiler-visible tensor operations, including linear projections, and analytically add fused attention matrix multiplications, custom Triton correction kernels, and TaylorSeer/HiCache predictor arithmetic omitted by the profiler. Conditioning, VAE decoding, device-to-host transfer, and disk I/O are excluded for all three models. The accounting scope is specified in Appendix A.3.

4.2 FLUX DRAWBENCH

Table 1 compares the deployed implementations of CALIBCACHE, FORA, TaylorSeer, and HiCache. Across $N = 3, 4, 5$, CALIBCACHE achieves the best PSNR, SSIM, LPIPS, and IR among the evaluated caching methods. It also obtains the highest CLIP score at $N = 4, 5$.

Table 1: FLUX.1-dev on DrawBench-200 at 1024^2 with 20 denoising steps.

Method	Efficiency		Alignment / preference		Full-compute fidelity		
	Latency (s)↓	FLOPs (T)↓	CLIP↑	IR↑	PSNR↑	SSIM↑	LPIPS↓
FLUX.1-dev (20 steps)	7.0495 (1.00×)	1487.836 (1.00×)	31.3273	0.95212	∞	1	0
FORA ($N = 3$)	**2.7235** (2.59×)	**521.051** (2.86×)	31.2644	0.7261	15.9921	0.64406	0.47070
TaylorSeer ($N = 3, O = 2$)	3.2887 (2.14×)	595.719 (2.50×)	**31.5913**	0.93172	16.7465	0.66782	0.41637
HiCache ($N = 3, O = 2$)	3.2052 (2.20×)	595.539 (2.50×)	31.5534	0.93172	17.0422	0.66129	0.41414
CALIBCACHE ($N = 3$)	3.2094 (2.20×)	621.914 (2.39×)	31.5376	**0.95549**	**18.6202**	**0.72185**	**0.34093**
FORA ($N = 4$)	**2.0177** (3.49×)	**372.315** (4.00×)	31.0429	0.4337	15.3897	0.59587	0.54240
TaylorSeer ($N = 4, O = 2$)	2.9441 (2.39×)	521.346 (2.85×)	31.5866	0.93917	18.5928	0.71370	0.34726
HiCache ($N = 4, O = 2$)	3.1877 (2.21×)	521.155 (2.85×)	31.5430	0.94229	19.1413	0.71836	0.33004
CALIBCACHE ($N = 4$)	2.8173 (2.50×)	544.223 (2.73×)	**31.6116**	**0.96672**	**19.8328**	**0.74463**	**0.28787**
FORA ($N = 5$)	**1.6694** (4.22×)	**297.947** (4.99×)	30.3692	0.0469	15.0238	0.55455	0.60742
TaylorSeer ($N = 5, O = 2$)	2.6177 (2.69×)	446.987 (3.33×)	31.8857	0.77485	17.2693	0.59087	0.56288
HiCache ($N = 5, O = 2$)	2.8977 (2.43×)	446.783 (3.33×)	31.6713	0.84138	17.9516	0.64256	0.44769
CALIBCACHE ($N = 5$)	2.4644 (2.86×)	466.550 (3.19×)	**31.9461**	**0.89891**	**19.1014**	**0.67348**	**0.42925**

Table 2: Qwen-Image on DrawBench-200 at 1024^2 with 20 denoising steps.

Method	Efficiency		Alignment / preference		Full-compute fidelity		
	Latency (s)↓	FLOPs (T)↓	CLIP↑	IR↑	PSNR↑	SSIM↑	LPIPS↓
Qwen-Image (20 steps)	12.7757 (1.00×)	2728.591 (1.00×)	32.8989	1.17516	∞	1	0
FORA ($N = 3$)	**6.1979** (2.06×)	**955.599** (2.86×)	31.0854	0.3168	13.9155	0.54600	0.53277
TaylorSeer ($N = 3, O = 2$)	7.4616 (1.71×)	1092.537 (2.50×)	32.1725	0.98366	13.4594	0.56454	0.49408
HiCache ($N = 3, O = 2$)	8.0308 (1.59×)	1092.170 (2.50×)	32.0616	0.98763	13.3377	0.56193	0.48487
CALIBCACHE ($N = 3$)	6.7595 (1.89×)	1091.973 (2.50×)	**32.9019**	**1.10332**	**18.1881**	**0.70563**	**0.31579**
FORA ($N = 4$)	**5.1507** (2.48×)	**682.814** (4.00×)	30.2783	-0.2850	13.4366	0.51479	0.58241
TaylorSeer ($N = 4, O = 2$)	7.0275 (1.82×)	956.136 (2.85×)	31.5651	0.75211	15.3328	0.64752	0.41676
HiCache ($N = 4, O = 2$)	7.6527 (1.67×)	955.747 (2.85×)	31.7830	0.79409	15.7373	0.65402	0.38992
CALIBCACHE ($N = 4$)	6.1706 (2.07×)	955.551 (2.86×)	**32.6364**	**1.11191**	**20.4604**	**0.78839**	**0.20340**
FORA ($N = 5$)	**4.6055** (2.77×)	**546.422** (4.99×)	28.6520	-0.8597	13.1403	0.49811	0.61961
TaylorSeer ($N = 5, O = 2$)	6.3679 (2.01×)	819.762 (3.33×)	30.6241	0.45596	13.7709	0.59085	0.50306
HiCache ($N = 5, O = 2$)	7.0980 (1.80×)	819.351 (3.33×)	30.9754	0.52504	14.0766	0.59470	0.47577
CALIBCACHE ($N = 5$)	5.6295 (2.27×)	819.158 (3.33×)	**32.4704**	**1.00084**	**19.2460**	**0.71478**	**0.30412**

At $N = 5$, CALIBCACHE increases PSNR from 17.2693 for TaylorSeer and 17.9516 for HiCache to 19.1014 and increases IR from 0.77485 and 0.84138, respectively, to 0.89891. Its denoising speedup reaches $2.86\times$, compared with $2.69\times$ for TaylorSeer and $2.43\times$ for HiCache. Across all three intervals, CALIBCACHE is faster than TaylorSeer; its latency is comparable to HiCache at $N = 3$ and lower at $N = 4, 5$. The reported FLOPs include the additional single-stream projections in the FLUX implementation. Table 5 further compares CALIBCACHE with uncorrected reuse at matched full/cache positions.

4.3 QWEN-IMAGE DRAWBENCH

On Qwen-Image, CALIBCACHE improves all five reported alignment, preference, and fidelity metrics over FORA, TaylorSeer, and HiCache at $N = 3, 4, 5$ (Table 2). At $N = 5$, LPIPS decreases from 0.50306 for TaylorSeer and 0.47577 for HiCache to 0.30412, corresponding to reductions of 39.5% and 36.1%. ImageReward increases from 0.45596 and 0.52504, respectively, to 1.00084. CALIBCACHE also has lower denoising latency than both TaylorSeer and HiCache at every interval, reaching $2.27\times$ speedup at $N = 5$ versus $2.01\times$ and $1.80\times$, respectively.

4.4 DIT-XL/2 ON IMAGENET-256

Table 3 evaluates class-conditional generation under the unified deployment policy in Section 4.1. Each CALIBCACHE setting uses one calibration trajectory pair. Across $N = 3, 4, 5$, CALIBCACHE outperforms FORA, TaylorSeer, and HiCache on all five reported distributional metrics. At $N = 5$,

Table 3: DiT-XL/2 on ImageNet-256 with DDIM-20 and 50K-image quality evaluation.

Method	Efficiency		Distributional evaluation				
	Latency (s)↓	FLOPs (T)↓	FID↓	sFID↓	IS↑	Precision↑	Recall↑
DiT-XL/2 (DDIM-20)	0.2449 (1.00×)	9.1535 (1.00×)	3.4931	4.9673	223.1583	0.78447	0.57610
FORA ($N = 3$)	**0.1212** (2.02×)	**3.2167** (2.85×)	14.2871	17.0764	144.4846	0.65272	0.47860
TaylorSeer ($N = 3, O = 2$)	0.1761 (1.39×)	3.6804 (2.49×)	5.3006	7.2121	196.4148	0.75718	0.55490
HiCache ($N = 3, O = 2$)	0.2220 (1.10×)	3.6759 (2.49×)	7.8976	10.8324	176.9213	0.72976	0.51910
CALIBCACHE ($N = 3$)	0.1662 (1.47×)	3.6766 (2.49×)	**4.1941**	**6.1416**	**207.6349**	**0.75742**	**0.59240**
FORA ($N = 4$)	**0.1049** (2.33×)	**2.3034** (3.97×)	30.0836	36.5201	85.7020	0.51546	0.41770
TaylorSeer ($N = 4, O = 2$)	0.1622 (1.51×)	3.2237 (2.84×)	8.4713	9.6293	170.9763	0.71073	0.53280
HiCache ($N = 4, O = 2$)	0.2098 (1.17×)	3.2189 (2.84×)	12.6385	14.8089	148.1833	0.66596	0.50220
CALIBCACHE ($N = 4$)	0.1453 (1.69×)	3.2189 (2.84×)	**6.7296**	**6.0563**	**179.9203**	**0.71430**	**0.58140**
FORA ($N = 5$)	**0.0969** (2.53×)	**1.8467** (4.96×)	48.1062	52.2517	51.5605	0.39456	0.39680
TaylorSeer ($N = 5, O = 2$)	0.1497 (1.64×)	2.7673 (3.31×)	16.3916	17.5930	127.8015	0.62528	0.48940
HiCache ($N = 5, O = 2$)	0.2028 (1.21×)	2.7621 (3.31×)	24.1568	26.8663	100.4643	0.55762	0.45810
CALIBCACHE ($N = 5$)	0.1354 (1.81×)	2.7622 (3.31×)	**10.3691**	**6.4061**	**151.7564**	**0.66828**	**0.57380**

Table 4: Correction-target ablation on DiT-XL/2, ImageNet-256, DDIM-20, $N = 5$, warmup-3. All variants share a two-anchor predictor, one calibration trajectory pair, centered scalar fitting, first-cache correction, corrected writeback, isolated history, and 50K-image evaluation. Only the target used to fit the scalar coefficients differs: residual, velocity, or acceleration.

Supervision	FID↓	sFID↓	IS↑	Precision↑	Recall↑
Residual supervision	41.80262	45.89258	64.36098	0.42992	0.42250
Velocity (v)	37.27242	40.45567	72.52494	0.45670	0.43760
Acceleration (a)	**10.36915**	**6.40612**	**151.75644**	**0.66828**	**0.57380**

FID decreases from 16.3916 for TaylorSeer and 24.1568 for HiCache to 10.3691, while sFID decreases from 17.5930 and 26.8663, respectively, to 6.4061.

CALIBCACHE is also faster than both TaylorSeer and HiCache at all three intervals, reaching $1.81\times$ speedup at $N = 5$ compared with $1.64\times$ and $1.21\times$, respectively. FORA offers higher speedup but has substantially worse FID and sFID.

5 ABLATION STUDIES

Correction target. We compare residual, velocity, and acceleration supervision on DiT-XL/2 (ImageNet-256, DDIM-20, $N = 5$, warmup-3). All variants use the same two-anchor update, step-by-branch centered fitting, and calibration trajectory pair; only the coefficient supervision changes. They share first-cache correction, corrected writeback, and isolated full-anchor history. Table 4 reports 50K-image distributional metrics. Acceleration supervision achieves FID 10.36915, versus 41.80262 for residual and 37.27242 for velocity supervision, and leads all five metrics in this controlled DiT setting. Appendix A.4 specifies the shared predictor and fitting targets.

Matched-schedule comparison with uncorrected reuse. Table 5 matches the six full-compute steps $\{0, 1, 2, 7, 12, 17\}$ at $N = 5$, warmup-3. Unlike standard FORA, this reuse control includes the same sampling warmup; the CALIBCACHE rows repeat the complete main-table implementation. On FLUX and DiT, all five quality metrics improve. FLUX LPIPS decreases by 6.6% with 9.9% higher latency; DiT FID decreases by 77.3% with 20.0% higher latency. On Qwen, SSIM rises from 0.65986 to 0.71478 and IR from 0.91775 to 1.00084 with 2.5% higher latency, while CLIP decreases slightly. The incremental quality–latency trade-off is therefore model- and metric-dependent. Matching sampling positions does not impose identical operator-level computation: FLUX FLOPs also increase by 4.45% due to the additional projections used to process text and image features separately in our single-stream implementation. This comparison measures the deployed method against matched-schedule reuse, rather than isolating the fitted coefficient on an identical execution path.

Table 5: Matched-schedule comparison with uncorrected reuse at $N = 5$, warmup-3, and 20 steps. FLUX.1-dev and Qwen-Image use DrawBench-200 at 1024^2; DiT-XL/2 uses ImageNet-256, DDIM-20, and 50K images per row. Full/cache positions are matched; denoising latency and FLOPs reflect each implementation.

FLUX.1-dev	Latency (s)↓	FLOPs (T)↓	CLIP↑	IR↑	PSNR↑	SSIM↑	LPIPS↓
Uncorrected reuse	**2.2425**	**446.683**	31.5936	0.84588	18.7906	0.60969	0.45937
CALIBCACHE	2.4644	466.550	**31.9461**	**0.89891**	**19.1014**	**0.67348**	**0.42925**
Qwen-Image	Latency (s)↓	FLOPs (T)↓	CLIP↑	IR↑	PSNR↑	SSIM↑	LPIPS↓
Uncorrected reuse	**5.4920**	**819.098**	**32.5109**	0.91775	18.1857	0.65986	0.30701
CALIBCACHE	5.6295	819.158	32.4704	**1.00084**	**19.2460**	**0.71478**	**0.30412**
DiT-XL/2	Latency (s)↓	FLOPs (T)↓	FID↓	sFID↓	IS↑	Precision↑	Recall↑
Uncorrected reuse	**0.1128**	**2.7601**	45.7141	48.7345	57.8806	0.40062	0.43130
CALIBCACHE	0.1354	2.7622	**10.3691**	**6.4061**	**151.7564**	**0.66828**	**0.57380**

Table 6: Application-policy ablation on DiT-XL/2, ImageNet-256, DDIM-20, $N = 5$. All rows use the same scalar calibration pack, warmup-3 schedule, and 50K-image evaluation protocol.

Application policy	FID↓	sFID↓	IS↑	Precision↑	Recall↑
Correction without writeback	18.63283	13.06961	124.41991	0.59702	0.53530
Ordinary writeback	14.63578	9.01988	133.63596	0.62458	**0.58280**
Writeback + isolated history (CALIBCACHE)	**10.36915**	**6.40612**	**151.75644**	**0.66828**	0.57380

Writeback and predictor-history isolation. Table 6 compares three application variants on DiT-XL/2 with DDIM-20, $N = 5$, warmup-3, and CFG 1.5. All variants use the same one-trajectory scalar pack, the correction operator in Equation (4), and the same full/cache/correction positions, with 50K generated images per variant. Correction without writeback applies the scalar correction at eligible steps but retains the full anchor in the reuse buffer and uses full-anchor predictor history. Ordinary writeback reuses corrected residuals and records predictor history from that evolving reuse stream. CALIBCACHE additionally isolates predictor history by recording the latest full-compute anchor at every cache step.

Writeback with isolated history reduces FID from 18.63283 for correction without writeback and 14.63578 for ordinary writeback to 10.36915. It also yields the lowest sFID and the highest IS and precision; ordinary writeback retains the highest recall (0.58280 versus 0.57380). These controlled results support corrected writeback with isolated history as the deployment policy. Appendix A.5 explains how the three policies update the reuse buffer and predictor history.

On FLUX.1-dev with 20 steps, $N = 5$, and fixed calibration noise, ten random single-prompt calibrations yield small variation in evaluation-set aggregates; Appendix A.8 reports these statistics and calibration-set-size sensitivity. Appendix A.7 compares fixed and fitted coefficients on FLUX, with grouping further improving PSNR over a shared fit.

6 CONCLUSION

We introduced CALIBCACHE, a lightweight offline method that improves feature caching in diffusion Transformers by calibrating cache-to-full feature acceleration. Using a single pair of full-compute and naive-cache trajectories per configuration, it fits grouped scalar maps through closed-form centered least squares. The frozen maps correct cached residuals while preserving the original cache schedule, without additional full-network evaluations or online optimization. Under 20-step sampling, CALIBCACHE improves fidelity to full-compute outputs on FLUX.1-dev and Qwen-Image and reduces FID and sFID on DiT-XL/2 relative to TaylorSeer and HiCache. Together, these results demonstrate the value of paired feature dynamics as supervision for lightweight correction. This provides a practical way to improve the quality of cache-accelerated generation while retaining the computational savings of feature reuse, even when sampling already uses a short schedule.

AI USE STATEMENT

Generative AI tools assisted with manuscript organization, language editing, LaTeX formatting, literature retrieval and discovery (including identifying relevant prior work), and cross-checking experimental records. The authors have reviewed all AI-assisted content and checked the cited sources and reported results against the original publications and experimental records.

ETHICS STATEMENT

This work studies inference acceleration for generative models and involves neither human-subject experiments nor new collection of personal data. Lower inference cost may broaden access to generative models while also reducing barriers to misuse. Bias, safety, and copyright risks in generated outputs are inherited from the underlying models. We will retain the foundation-model licenses and known limitations in released code and documentation.

REPRODUCIBILITY STATEMENT

We provide a minimal anonymous code package in the supplementary material to support reproduction of our method, including paired-trajectory collection, centered scalar fitting, and calibrated sampling. Evaluation metrics are computed using their official implementations. The main paper and appendix specify the experimental settings, evaluation protocols, and implementation details.

REFERENCES

Black Forest Labs. FLUX.1-dev model card. Hugging Face model documentation, 2024. URL https://huggingface.co/black-forest-labs/FLUX.1-dev.

Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. δ-DiT: A training-free acceleration method tailored for diffusion transformers, 2024.

Jiwoo Chung, Sangeek Hyun, MinKyu Lee, Byeongju Han, Geonho Cha, Dongyoon Wee, Youngjun Hong, and Jae-Pil Heo. SeaCache: Spectral-evolution-aware cache for accelerating diffusion models, 2026.

Benlei Cui, Shaoxuan He, Bukun Huang, Zhizeng Ye, Yunyun Sun, Longtao Huang, Hui Xue, Yang Yang, Jingqun Tang, Zhou Zhao, and Haiwen Hong. TC-Padé: Trajectory-consistent Padé approximation for diffusion acceleration, 2026.

Liang Feng, Shikang Zheng, Jiacheng Liu, Yuqi Lin, Qinming Zhou, Peiliang Cai, Xinyu Wang, Junjie Chen, Chang Zou, Yue Ma, and Linfeng Zhang. HiCache: A plug-in Scaled-Hermite upgrade for Taylor-style cache-then-forecast diffusion acceleration. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2508.16984.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Xuyue Huang, Zhe Chen, Wang Shen, and Xiao-Ping Zhang. LearniBridge: Learnable calibration of feature caching for diffusion models acceleration, 2026.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision – ECCV 2014*, pp. 740–755, 2014. doi: 10.1007/978-3-319-10602-1_48.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model, 2025a.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with TaylorSeers, 2025b.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free, 2023.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023.

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, 2021.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. FORA: Fast-forward caching in diffusion transformer acceleration, 2024.

Zhirong Shen, Rui Huang, Jiacheng Liu, Chang Zou, Peiliang Cai, Shikang Zheng, Zhengyi Shi, Liang Feng, and Linfeng Zhang. Beyond fixed formulas: Data-driven linear predictor for efficient diffusion models, 2026. URL https://arxiv.org/abs/2604.26365.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=St1giarCHLP.

Chenfei Wu et al. Qwen-Image technical report, 2025.

Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. ImageReward: Learning and evaluating human preferences for text-to-image generation, 2023.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018.

Shikang Zheng, Liang Feng, Xinyu Wang, Qinming Zhou, Peiliang Cai, Chang Zou, Jiacheng Liu, Yuqi Lin, Junjie Chen, Yue Ma, and Linfeng Zhang. Forecast then calibrate: Feature caching as ODE for efficient diffusion transformers, 2025. URL https://arxiv.org/abs/2508.16211.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching, 2025.

A Implementation and Protocol Details

A.1 Calibration and Deployment Policies

Numerical implementation of centered scalar fitting. The centered objective and its closed-form solution are given in Equations (2) and (3). The implementation accumulates sufficient statistics in double precision and guards numerically negligible centered energy with denominator floors or a zero-scale fallback. Each group supplies one fitted scalar $s_g = s_{\tau,b}$ to the online map in Equation (4); the fitting means are not required during sampling.

Calibration samples and parameter sharing. For the FLUX and Qwen-Image DrawBench experiments, each deployed pack is fitted using one COCO caption disjoint from the evaluation prompts. Each deployed DiT pack uses one calibration trajectory. The separate mechanism and sensitivity studies use the calibration counts stated in their own sections. Single-trajectory calibration uses one conditioning/noise realization to generate a matched pair of full-compute and naive-cache trajectories. The conditioning (a prompt or DiT class label), initial noise, and sampler configuration are shared within the pair. FLUX calibration uses seeds $314159 +$ global index; Qwen-Image evaluation uses $314159 +$ prompt index consistently across methods. FLUX uses step-by-branch

Table 7: Sampling warmup: the initial full-compute steps before periodic caching. Zero denotes no additional warmup phase. The ordinary cache schedule still supplies full-compute evaluations.

Model	Methods	$N = 3$	$N = 4$	$N = 5$
FLUX / Qwen-Image	FORA	0	0	0
FLUX / Qwen-Image	TaylorSeer, HiCache, CALIBCACHE	2	3	3
DiT-XL/2	FORA	0	0	0
DiT-XL/2	TaylorSeer, HiCache, CALIBCACHE	2	3	3

packs for $N = 3, 4, 5$. Qwen-Image uses step-by-branch packs shared across CFG contexts, with separate conditional and unconditional cache states. DiT uses step-by-branch maps for attention and MLP residuals, with each map shared across all 28 Transformer blocks.

Sampling warmup and cache schedules. Table 7 specifies the sampling warmup, which is distinct from the ten execution warmup runs used for latency measurement. The main-table FORA baseline uses no extra sampling warmup at any interval: its first denoising step is full, then it refreshes every N steps. TaylorSeer, HiCache, and CALIBCACHE use warmup-2 at $N = 3$ and warmup-3 at $N = 4, 5$ on all three models. The Uncorrected reuse controls in the main-text Table 5 use the same $N = 5$/warmup-3 full/cache positions as CALIBCACHE on all three models. The common warmup supplies the two preceding observations required by the first cache correction.

Terminal-step fallback. Logical steps count down from 19 to 0, whereas execution indices increase from 0 to 19. At $N = 4$, the final step is a first cache step; its correction is skipped to avoid a large terminal residual change that could degrade image quality, and the preceding full residual is reused directly. This skip0 rule preserves all full/cache positions. At $N = 3$ the final step is full, while at $N = 5$ it only reuses the correction from the preceding cache step.

A.1.1 UNIFIED INTERVAL-WISE CORRECTION AND CACHE WRITEBACK

For all three models, CALIBCACHE applies Equation (4) only at the first cache step in each interval, subject to the terminal-step fallback. Suppressing layer and branch indices, let $\boldsymbol{f}_\tau$ denote the latest full-compute residual evaluated on the current deployed rollout. The predictor history H_{pred} stores full-anchor reference residuals, while the deployment buffer B_{reuse} stores the residual to be reused by the network. Full-compute steps refresh both the anchor and the reuse buffer and append the new anchor to predictor history; every cache step records the latest anchor as its reference, even when the reuse buffer contains a corrected value.

In the notation of Section 3.3, this policy supplies the reference sequence $\mathbf{r}_t^{\mathrm{O}} = \boldsymbol{f}_t$. Let $\boldsymbol{h}_{\tau-1}$ and $\boldsymbol{h}_{\tau-2}$ denote the two stored references preceding the current step. At a first cache step, the reuse buffer still contains the newly refreshed full-compute anchor, so the three inputs to the second difference in Equation (1) are

$$(\mathbf{r}_{\tau-2}^{\mathrm{O}}, \mathbf{r}_{\tau-1}^{\mathrm{O}}, \mathbf{r}_{\tau}^{\mathrm{O}}) = (\boldsymbol{h}_{\tau-2}, \boldsymbol{h}_{\tau-1}, \boldsymbol{f}_\tau).$$

After computing the current correction, the state updates are

$$H_{\text{pred}} \leftarrow \mathrm{append}(H_{\text{pred}}, \boldsymbol{f}_\tau), \qquad B_{\text{reuse}} \leftarrow \widetilde{\mathbf{r}}_\tau. \tag{5}$$

Here append retains the two most recent reference entries. Subsequent cache steps reuse the corrected residual until the next full evaluation, while predictor history continues to record the full anchor. At these interior steps, the reference residuals repeat the same anchor and have zero second difference; no new correction is computed, and the existing corrected reuse value is retained. The full anchors themselves evolve with the deployed rollout and are distinct from the offline full-compute trajectory.

In zero-based execution order, the shared schedules are:

- $N = 3$: full-compute steps $\{0, 1, 4, 7, 10, 13, 16, 19\}$; correction steps $\{2, 5, 8, 11, 14, 17\}$.
- $N = 4$: full-compute steps $\{0, 1, 2, 6, 10, 14, 18\}$; correction steps $\{3, 7, 11, 15\}$, with step 19 using direct reuse.
- $N = 5$: full-compute steps $\{0, 1, 2, 7, 12, 17\}$; correction steps $\{3, 8, 13, 18\}$.

Table 8: Active calibration coefficients per deployment configuration. Each entry gives the scalar count and its FP32 payload in bytes.

Model	$N = 3$	$N = 4$	$N = 5$
FLUX.1-dev	36 (144 B)	24 (96 B)	24 (96 B)
Qwen-Image	24 (96 B)	16 (64 B)	16 (64 B)
DiT-XL/2	12 (48 B)	8 (32 B)	8 (32 B)

Thus the three settings use eight, seven, and six full-compute steps, respectively, with all remaining steps using cached features.

A.2 Calibration Budget and Parameter Footprint

Each deployment configuration uses one full-compute/naive-cache trajectory pair, followed by closed-form centered scalar fitting without iterative optimization. The shared schedules activate six correction steps at $N = 3$ and four at $N = 4, 5$. FLUX.1-dev, Qwen-Image, and DiT-XL/2 use six, four, and two branch groups, respectively, with each scalar shared across layers and tokens. Table 8 counts the coefficients used at these correction steps and their FP32 payload, excluding serialization metadata and runtime feature buffers.

For each deployment configuration, one trajectory pair supplies the sufficient statistics for closed-form scalar fitting, and the resulting maps remain fixed across evaluation samples. The 8–36 active scalars occupy 32–144 bytes in FP32; this coefficient payload is distinct from runtime feature-history and reuse buffers. After collection, each coefficient is obtained directly from accumulated inner products and squared norms, without iterative updates or a coefficient sweep. These quantities characterize the calibration-data and parameter budgets. Trajectory collection and fitting are one-time offline preparation, separate from both sampling warmup and benchmark execution warmup. The reported denoising latency and FLOPs include the full deployment computation, including cache/history operations and additional network operations; offline preparation is excluded.

A.3 Latency and FLOP Accounting

Denoising latency. All main-table measurements use a single NVIDIA H800 GPU and batch size one. After ten execution warmup runs, we time 30 runs with CUDA events and report the mean per-image denoising latency. Model weights and cache or calibration parameters are loaded onto the GPU before measurement. The timed region includes the 20-step denoising loop, executed Transformer branches, cache/history operations, and any prediction or correction performed during denoising. It excludes text encoding, VAE decoding, pack preparation, parameter loading, and disk I/O. Speedup is the displayed full-compute baseline latency divided by the accelerated-method latency within the same model table.

FLOPs. FLOPs are measured with `torch.profiler` in a separate profiling run. The reported main-table FLOPs for all three models, including the CalibCache rows repeated in Table 5, use the same denoising-only boundary as latency: 20 sampling steps, excluding text encoding, VAE decoding, device-to-host transfer, and disk I/O. We combine profiler-visible tensor operations with analytic counts for fused attention matrix multiplications, custom Triton correction kernels, and TaylorSeer/HiCache predictor arithmetic omitted by the profiler, avoiding double counting. The parenthesized FLOPs reduction is the full-compute baseline total divided by the accelerated total.

A.4 Correction-Target Study

Table 4 compares residual, velocity, and acceleration supervision while keeping the online update and deployment policy fixed. The difference is how the correction coefficient is fitted offline.

Shared protocol. The study uses DiT-XL/2 on ImageNet-256. All three variants use DDIM-20, $N = 5$, warmup-3, CFG 1.5, one calibration trajectory, step-by-branch centered scalar fitting without a deployed intercept, and 50K generated images. Their logical full-compute steps are $\{19, 18, 17, 12, 7, 2\}$, and correction is applied only at logical steps $\{16, 11, 6, 1\}$. Every variant

Table 9: Supervision definitions for the common online update in Equation (6). The final column gives the coefficient multiplying $\boldsymbol{f} - \boldsymbol{p}$. Residual supervision targets the correction needed to match the full-trajectory residual. Velocity and acceleration supervision target the first and second residual differences along the full-compute trajectory.

Supervision	Fit input $\mathbf{x}$	Fit target $\mathbf{y}$	Online α_g
Residual	$\boldsymbol{u}_\tau^{\mathrm{C}}$	$\mathbf{r}_\tau^{\mathrm{F}} - \mathbf{r}_\tau^{\mathrm{C}}$	α_g^r
Velocity	$\boldsymbol{u}_\tau^{\mathrm{C}}$	$\mathbf{r}_\tau^{\mathrm{F}} - \mathbf{r}_{\tau-1}^{\mathrm{F}}$	s_g^v
Acceleration	$\mathbf{a}_\tau^{\mathrm{C}} = -\boldsymbol{u}_\tau^{\mathrm{C}}$	$\mathbf{a}_\tau^{\mathrm{F}}$	$1 - s_g$

writes the corrected residual into the reuse buffer and retains full-anchor predictor-history isolation; subsequent cache steps reuse the stored corrected residual. The acceleration row is the same deployed one-trajectory result as in Table 3.

Shared online update. At a first cache step, let $\boldsymbol{f}$ be the latest full-compute residual and $\boldsymbol{p}$ the preceding anchor in the reference history of Appendix A.1.1. Suppressing layer and branch indices, all three variants use

$$\widetilde{\mathbf{r}}_\tau = \boldsymbol{f} + \alpha_g(\boldsymbol{f} - \boldsymbol{p}). \tag{6}$$

The direction $\boldsymbol{f} - \boldsymbol{p}$ is the lagged difference between the two preceding reference entries. The velocity variant scales this lagged difference. The current cached residual and the latest reference both equal $\boldsymbol{f}$, making the current difference zero. For acceleration supervision, the three reference residuals are $(\boldsymbol{p}, \boldsymbol{f}, \boldsymbol{f})$, giving $\mathbf{a}_\tau^{\mathrm{O}} = \boldsymbol{p} - \boldsymbol{f}$. Substituting this into Equation (4) gives

$$\begin{aligned} \widetilde{\mathbf{r}}_\tau &= \boldsymbol{f} + (s_g - 1)(\boldsymbol{p} - \boldsymbol{f}) \\ &= \boldsymbol{f} + (1 - s_g)(\boldsymbol{f} - \boldsymbol{p}). \end{aligned} \tag{7}$$

Thus $\alpha_g = 1 - s_g$ is the coefficient multiplying $\boldsymbol{f} - \boldsymbol{p}$ in the shared update; it is determined by the fitted s_g and introduces no additional parameter.

Offline supervision. Table 9 lists the fitting pairs and the resulting online coefficients. Each row uses the centered objective and closed-form solution in Equations (2) and (3), replacing the acceleration pair by $(\mathbf{x}, \mathbf{y})$. The observations come from the independently evolving full-compute and naive-cache calibration trajectories. We write $\boldsymbol{u}_\tau^{\mathrm{C}} = \mathbf{r}_{\tau-1}^{\mathrm{C}} - \mathbf{r}_{\tau-2}^{\mathrm{C}}$ for the offline lagged difference; $\mathbf{r}_\tau^{\mathrm{C}}$ is the residual reused at the current calibration step. The fitted slopes are denoted by α_g^r, s_g^v, and s_g for residual, velocity, and acceleration supervision, respectively.

For acceleration supervision, $s_g = 1$ gives direct reuse and $s_g = 0$ gives extrapolation $2\boldsymbol{f} - \boldsymbol{p}$. This two-anchor form is a linear predictor whose coefficient is estimated from paired cache-to-full acceleration observations; feature-reconstruction predictors such as $\mathrm{L}^2\mathrm{P}$ (Shen et al., 2026) learn historical-feature weights from full-compute trajectories. The results in Table 4 support acceleration supervision under this matched DiT protocol.

Separate residual-scaling control. Under the same calibration and deployment protocol, direct residual scaling fits s_g^r offline with $\mathbf{x} = \mathbf{r}_\tau^{\mathrm{C}}$ and $\mathbf{y} = \mathbf{r}_\tau^{\mathrm{F}}$ and deploys $\widetilde{\mathbf{r}}_\tau = s_g^r \mathbf{r}_\tau^{\mathrm{O}}$ online. It yields FID 38.17718, sFID 30.14794, IS 75.57263, precision 0.44726, and recall 0.45700; its update form differs from the shared two-anchor predictor in Table 4.

How acceleration discrepancies accumulate. An exact finite-difference identity relates acceleration discrepancies to residual error along the two offline calibration trajectories. Define $\boldsymbol{e}_\tau = \mathbf{r}_\tau^{\mathrm{C}} - \mathbf{r}_\tau^{\mathrm{F}}$, $\Delta \boldsymbol{e}_\tau = \boldsymbol{e}_\tau - \boldsymbol{e}_{\tau-1}$, and $\boldsymbol{\delta}_\tau = \mathbf{a}_\tau^{\mathrm{C}} - \mathbf{a}_\tau^{\mathrm{F}}$. Equation (1) then gives $\Delta^2 \boldsymbol{e}_\tau = \boldsymbol{\delta}_\tau$. Starting at a cache-interval boundary $\tau_0 \geq 1$ in execution order, summing twice yields, for $j \geq 1$ and $\tau_0 + j < T$,

$$\boldsymbol{e}_{\tau_0+j} = \boldsymbol{e}_{\tau_0} + j\Delta \boldsymbol{e}_{\tau_0} + \sum_{q=1}^{j} (j - q + 1)\boldsymbol{\delta}_{\tau_0+q}. \tag{8}$$

The first two terms carry the residual and first-difference errors at the boundary. The sum includes every intervening step: earlier acceleration discrepancies receive larger weights because they contribute to more subsequent first differences. If each discrepancy has norm at most ϵ, the triangle

inequality bounds the norm of this sum by $j(j+1)\epsilon/2$. This is a worst-case bound on the accumulated contribution; vector terms may cancel. A full evaluation on the cached rollout need not remove the boundary error, since its input can already differ from the full-compute trajectory. The identity explains error accumulation. Table 4 supplies the empirical comparison.

A.5 Writeback and Predictor-History Isolation

The three policies in Table 6 differ in two states: the reuse buffer supplies features to the network, and predictor history supplies references for the next correction. Each policy applies its correction at the eligible first cache step; full-compute steps refresh the reuse buffer and add a new full anchor to predictor history.

What writeback retains. Without writeback, the correction affects only the first cache step; later cache steps revert to the full-compute anchor. Predictor history also records the full anchor at every cache step. With writeback, later cache steps continue to use the corrected residual until the next full evaluation. For either policy, let $\boldsymbol{f}$ denote its current full-compute anchor and $\boldsymbol{c}$ the correction computed at its first cache step. Using an interval-local index with the full evaluation at $k = 0$, both policies use $\boldsymbol{f} + \boldsymbol{c}$ at $k = 1$. At later cache steps before the next full evaluation at $k = N$,

$$\mathbf{r}_k^{\text{used}} = \begin{cases} \boldsymbol{f}, & \text{without writeback,} \\ \boldsymbol{f} + \boldsymbol{c}, & \text{with writeback,} \end{cases} \qquad 2 \le k < N. \tag{9}$$

Under isolated history, the reference entries repeat the same anchor after the first cache step, so their second difference is zero. Writeback preserves the earlier correction even when no new correction is computed. The terminal-step fallback in Appendix A.1 still applies.

What history isolation changes. Under ordinary writeback, later cache steps use corrected residuals in both the reuse buffer and predictor history. These corrected history entries can influence the correction in a subsequent interval. Isolated history records the full anchor at every cache step, removing this direct feedback while the network continues to reuse corrected features. To see the input difference at the next correction, let $\boldsymbol{p}$ be the preceding full anchor and $\boldsymbol{c}_{\text{prev}}$ its retained correction, so the preceding interval ends with the reused residual $\boldsymbol{p} + \boldsymbol{c}_{\text{prev}}$. After a new full evaluation produces $\boldsymbol{f}$, the current residual at the first cache step is again $\boldsymbol{f}$. Holding these anchors fixed to compare history handling, the acceleration inputs are

$$\begin{aligned} \mathbf{a}_\tau^{\text{ord}} &= \boldsymbol{f} - 2\boldsymbol{f} + (\boldsymbol{p} + \boldsymbol{c}_{\text{prev}}) = (\boldsymbol{p} - \boldsymbol{f}) + \boldsymbol{c}_{\text{prev}}, \\ \mathbf{a}_\tau^{\text{iso}} &= \boldsymbol{f} - 2\boldsymbol{f} + \boldsymbol{p} = \boldsymbol{p} - \boldsymbol{f}. \end{aligned} \tag{10}$$

Here ord and iso denote ordinary writeback and isolated history. Under Equation (4), this additional history term contributes $(s_g - 1)\boldsymbol{c}_{\text{prev}}$ directly to the ordinary-writeback correction. Isolation removes this term while retaining the corrected reuse buffer. Consequently, the predictor's reference sequence and the residual sequence used by the network are distinct. Though full anchors still evolve along the corrected rollout, the controlled comparison in Table 6 provides the empirical support for this policy.

Ablation implementation and protocol. The three runs share the same one-trajectory DiT scalar pack, full/cache/correction positions, two-rank sampling configuration, global seed zero, per-rank sampling batch size 128, and VAE decode batch size 8. Each variant is evaluated on 50K generated images against the same ImageNet-256 reference.

A.6 Fitting-Form Ablation

We compare alternative scalar fitting forms for DiT-XL/2 on ImageNet-256 with DDIM-20, $N = 5$, warmup-3, and 50K generated images. All variants use a single calibration trajectory, corrected-cache writeback, and full-anchor predictor-history isolation. The quantitative results are summarized in Table 10.

In this DiT setting, centered s-only fitting improves FID, sFID, IS, and precision over through-origin fitting. Compared with $s+b$, the scalar-only deployed map achieves similar FID and sFID and higher IS, precision, and recall.

Table 10: Fitting-form ablation for DiT-XL/2 on ImageNet-256. Lower FID and sFID are better; higher IS, precision, and recall are better.

Fitting form	FID↓	sFID↓	IS↑	Precision↑	Recall↑
Centered s-only ($b = 0$)	10.36915	6.40612	151.75644	0.66828	0.57380
Through-origin slope-only	14.18379	8.65522	136.35312	0.62440	0.58140
s+b	10.35668	6.33262	151.42812	0.66792	0.56760

A.7 Coefficient Choice and Sharing

We compare coefficient settings on FLUX.1-dev, DrawBench-200, with 20 sampling steps, $N = 5$, warmup-3, and 1024^2 resolution. In the two-anchor form of Appendix A.4, fixing $s = 0$ gives extrapolation $\widetilde{\mathbf{r}}_\tau = 2\boldsymbol{f} - \boldsymbol{p}$, whereas $s = 1$ reduces algebraically to direct reuse. The matched-schedule Uncorrected reuse comparison is reported in Table 5. Table 11 compares fixed extrapolation, a shared fitted scalar $s = 0.26131$, and the step-by-branch coefficients used by CalibCache. The shared and grouped fits use the same calibration trajectory pair (COCO prompt-file index 50000, noise seed 364159). We constrain the 24 active group slopes to one value using their stored sufficient statistics, retaining channelwise centering within each original group:

$$s_{\text{shared}} = \frac{\sum_{g=1}^{24} C_g}{\sum_{g=1}^{24} D_g} \approx 0.26131,$$

where C_g and D_g are the centered cross-product and input-energy sums, respectively, with the original per-channel denominator floor of 10^{-12}. The summed objective uses no additional group weighting or normalization, and the coefficient is estimated independently of the evaluation results. All three settings use the same correction positions, corrected-cache writeback, isolated full-anchor history, branch projections, inference implementation, and evaluation prompts and seeds; only the active coefficient values change. The grouped row reuses the main FLUX $N = 5$ result.

Table 11: Coefficient-choice ablation on FLUX.1-dev, DrawBench-200, with 20 steps, $N = 5$, and warmup-3. The scalar count denotes independently fitted parameters; bold marks the best reported mean.

Coefficient setting	Fitted scalars	PSNR↑	SSIM↑	LPIPS↓	CLIP↑	IR↑
Fixed $s = 0$	0	17.3736	0.59297	0.57690	31.8232	0.79039
Shared $s = 0.26131$	1	18.4725	0.67029	0.43009	31.8310	0.87135
Grouped s_g (CalibCache)	24	**19.1014**	**0.67348**	**0.42925**	**31.9461**	**0.89891**

Both fitted settings outperform fixed extrapolation in the reported aggregates. Grouping further improves PSNR by 0.6289 dB over the shared fit, while the shared scalar yields similar SSIM and LPIPS.

A.8 Additional Calibration Ablations

This subsection examines sensitivity to calibration-set size and the identity of a single calibration prompt. The matched-schedule correction comparison is reported in Table 5 in the main text.

Number of calibration prompts. We evaluate calibration budgets of 1, 4, 16, and 64 nested COCO prompts on FLUX.1-dev with 20 steps, $N = 5$, and warmup-3, using the same DrawBench-200 evaluation prompts and seeds. Each prompt supplies one paired full-compute/naive-cache trajectory; all packs use centered s-only fitting, step-by-branch sharing, corrected-cache writeback, and isolated full-anchor predictor history. The one-prompt setting uses the current main-table pack. Table 12 reports the completed four-setting study. Increasing the number of calibration trajectories does not yield a clear quality benefit.

Random single-prompt calibration. To test sensitivity to prompt identity, we independently fit ten step-by-branch scalar packs by centered least squares, each on one randomly selected COCO

Table 12: Calibration-set-size sensitivity for FLUX.1-dev with 20 steps, $N = 5$, warmup-3, and step-by-branch sharing on DrawBench-200. Each calibration prompt supplies one trajectory pair.

Calib. prompts	PSNR↑	SSIM↑	LPIPS↓	CLIP↑	IR↑
1	19.1014	0.67348	0.42925	31.9461	0.89891
4	19.2467	0.68151	0.41684	31.8786	0.90214
16	19.1494	0.67215	0.43108	31.8759	0.88410
64	19.1270	0.66995	0.43479	31.8941	0.88120

Table 13: Random single-prompt calibration on FLUX.1-dev, DrawBench-200, $N = 5$, warmup-3. Each of ten independently fitted step-by-branch packs uses the same calibration noise seed; evaluation prompts and seeds are shared. Mean, sample standard deviation, and range are across the ten pack-level aggregates.

Metric	Mean ± std	Range
CLIP↑	31.9113 ± 0.0255	$[31.8755,\ 31.9534]$
IR↑	0.89312 ± 0.01033	$[0.87758,\ 0.90675]$
PSNR↑	19.1151 ± 0.0848	$[18.9346,\ 19.2567]$
SSIM↑	0.67070 ± 0.00673	$[0.65619,\ 0.68096]$
LPIPS↓	0.43368 ± 0.00956	$[0.41778,\ 0.45368]$

prompt disjoint from DrawBench-200. The selection seed is fixed in advance, and all ten fits use the same calibration noise seed and unified deployment policy. Every pack is evaluated on the same 200 DrawBench prompts and evaluation seeds. Table 13 reports the mean, sample standard deviation, and range of the resulting DrawBench-200 aggregate for each metric across the ten packs. The small dispersion across packs indicates limited sensitivity to the identity of the single calibration prompt in this setting.

A.9 Additional Comparison with L^2P

We additionally compare CalibCache with L^2P (Shen et al., 2026), a data-driven linear predictor that learns timestep-specific combinations of historical features from full-compute trajectories. This comparison is placed in the appendix because the publicly released L^2P implementation does not support experiments on all three models evaluated in the main text, so we developed our own implementation. However, we find that our adaptation of L^2P performs very poorly on DiT.

Adaptation and evaluation protocol. The released implementation does not provide DiT support, so the results below therefore use our adaptation of its output-level predictor to DiT-XL/2. We retain its causal lower-triangular weight matrix, identity initialization, masked mean-squared-error objective, and Adam optimization with learning rate 0.01 for 201 updates. The predictor is fitted on 64 full-compute trajectories, with 16 additional trajectories used only for validation monitoring. At each cache step, it predicts the complete sampler-facing model output from preceding full or predicted outputs and appends the prediction to its history. This preserves L^2P's output-level design: one prediction per cache step avoids maintaining and predicting attention/MLP features at every Transformer layer, keeping prediction and history-storage overhead low. We report the best-performing completed adaptation.

Both methods use ImageNet-256, DDIM-20, CFG 1.5, and the same $N = 5$/warmup-3 schedule, with logical full-compute steps $\{19, 18, 17, 12, 7, 2\}$. Quality is evaluated on 50K generated images. Latency and FLOPs follow the denoising-only accounting in Appendix A.3; latency uses one H800, batch size one, and 30 CUDA-event measurements after 10 execution warmup runs.

Table 14 confirms the latency advantage of output-level prediction, but also shows a large quality gap: L^2P obtains FID 39.9860 and sFID 28.7022, compared with 10.3691 and 6.4061 for CalibCache. CalibCache improves all five quality metrics while using one calibration trajectory pair and 8 deployed coefficients, compared with 64 trajectories and 151 coefficients for L^2P.

Table 14: Comparison with our L^{2}P adaptation on DiT-XL/2, ImageNet-256, using DDIM-20, CFG 1.5, $N = 5$, warmup-3, and 50K generated images. Traj. counts calibration conditioning/noise realizations: L^{2}P uses 64 full-compute trajectories, whereas CALIBCACHE uses one full-compute/naive-cache trajectory pair. Coeff. counts active deployed coefficients.

Method	Traj.	Coeff.	Latency (s)↓	FLOPs (T)↓	FID↓	sFID↓	IS↑	Precision↑	Recall↑
L^{2}P (adapted)	64	151	**0.0829**	**2.7461**	39.9860	28.7022	79.2690	0.47538	0.54490
CALIBCACHE	**1**	**8**	0.1354	2.7622	**10.3691**	**6.4061**	**151.7564**	**0.66828**	**0.57380**

Mismatch between fitting and deployment histories. L^{2}P is fitted using true historical outputs from full-compute trajectories, whereas its deployment history also contains previously predicted outputs. Let f_k denote the full-compute reference output at execution index k, let $\widetilde{f}_k$ denote the deployed output, and define $e_k = \widetilde{f}_k - f_k$. At a cache step, the prediction error satisfies

$$e_k = \underbrace{\left(\sum_{j<k} w_{k,j} f_j - f_k\right)}_{\text{true-history prediction residual}} + \underbrace{\sum_{j<k} w_{k,j} e_j}_{\text{propagated history errors}} . \tag{11}$$

Thus, a small prediction residual under true history does not guarantee a small deployed error. Prediction errors also alter the latent trajectory, so later full-compute outputs can differ from the reference even when those steps are evaluated exactly.

To test the effect of history mismatch, we keep the fitted weights fixed and continue to use predicted outputs for sampling at cache steps, but populate the predictor history with true model outputs evaluated on the current deployed trajectory. Across 32 paired samples from two random seeds, this diagnostic reduces final latent MSE relative to full computation from 0.08146 to 0.02995, a 63.2% reduction.

Our reproduction of L^{2}P performs poorly on DiT, a discrete diffusion model, for which no official implementation is publicly available. Moreover, L^{2}P primarily reports acceleration results under 50-step sampling, whereas our study focuses on shorter sampling schedules, particularly 20 steps. Given these limitations, we do not consider our reproduction sufficiently validated to serve as a reliable baseline in the main text.

A.10 ADDITIONAL COMPARISON WITH TC-PADÉ

We additionally reproduce TC-Padé (Cui et al., 2026) from the authors' public repository (commit `e2d6302`) on the same FLUX.1-dev DrawBench-200 evaluation used in the main text. The released repository snapshot contains executable inference code for FLUX.1-dev, but not corresponding implementations for Qwen-Image or DiT-XL/2. TC-Padé therefore cannot be evaluated under the main text's cross-model protocol, so we report the available FLUX-specific comparison in the appendix rather than adding it to the main benchmark tables. Our evaluation wrapper retains the released Transformer forward and residual predictor without modifying the repository source, while providing the current Diffusers interface, indexed prompt seeds, and per-prompt history reset. All rows in Table 15 use 20 denoising steps, 1024^2 resolution, guidance 3.5, and the same 200 prompts and seeds. For TC-Padé, $N_{\text{th}} \in \{0.7, 1.0, 1.3\}$ denotes the repository's curvature threshold, where a larger value permits more aggressive prediction; it is not the fixed cache interval N used by CALIBCACHE. The reproduced configurations use predictor order 3, history size 6, and interval 8; prediction starts at step 4 for $N_{\text{th}} = 0.7, 1.0$ and step 3 for $N_{\text{th}} = 1.3$. Latency uses one H800 at batch size one, with 30 CUDA-event measurements after 10 warmup runs; FLOPs combine profiler-visible operations with analytic fused-attention matrix multiplications. Both measurements cover the 20-step denoising process only and exclude text encoding, VAE decoding, device-to-host transfer, and I/O, matching the main-table accounting boundary. Because TC-Padé makes prompt-dependent prediction decisions, its FLOPs are measured on one warmed trajectory using the first DrawBench prompt and its fixed evaluation seed.

Across the displayed configurations, CALIBCACHE gives substantially higher full-compute fidelity than the reproduced TC-Padé configurations in PSNR, SSIM, and LPIPS, and also improves ImageReward in the matched DrawBench evaluation. CLIP is closer and configuration-dependent:

Table 15: Efficiency and quality comparison with our reproduction of the public TC-Padé implementation on FLUX.1-dev, DrawBench-200, with 20 denoising steps at 1024^2. TC-Padé's N_{th} is a curvature threshold, whereas CALIBCACHE's N is a fixed cache interval.

Method	Latency (s)↓	FLOPs (T)↓	CLIP↑	IR↑	PSNR↑	SSIM↑	LPIPS↓
TC-Padé ($N_{\text{th}} = 0.7$)	5.8379	1116.023	31.6134	0.89883	11.7447	0.52002	0.62702
TC-Padé ($N_{\text{th}} = 1.0$)	3.4730	669.747	31.5535	0.82747	11.8787	0.53759	0.65077
TC-Padé ($N_{\text{th}} = 1.3$)	3.0231	520.987	31.4045	0.69882	11.9939	0.54075	0.66214
CALIBCACHE ($N = 3$)	3.2094	621.914	31.5376	0.95549	18.6202	0.72185	0.34093
CALIBCACHE ($N = 4$)	2.8173	544.223	31.6116	**0.96672**	**19.8328**	**0.74463**	**0.28787**
CALIBCACHE ($N = 5$)	**2.4644**	**466.550**	**31.9461**	0.89891	19.1014	0.67348	0.42925

TC-Padé at $N_{\text{th}} = 0.7$ is slightly higher than CALIBCACHE at $N = 3, 4$, while CALIBCACHE at $N = 5$ obtains the highest CLIP score in this comparison. More aggressive TC-Padé thresholds reduce latency and FLOPs, while CALIBCACHE at $N = 5$ remains faster and uses fewer FLOPs than all three reproduced TC-Padé configurations.

B ADDITIONAL QUALITATIVE COMPARISONS

Figures 3, 4, and 5 show more appearance and detail comparisons using matched conditioning and initial noise; aggregate quality is reported in the quantitative tables. All CALIBCACHE examples use one-trajectory centered scalar fitting, corrected-cache writeback, and isolated full-anchor history, without image retouching or method-specific enhancement.

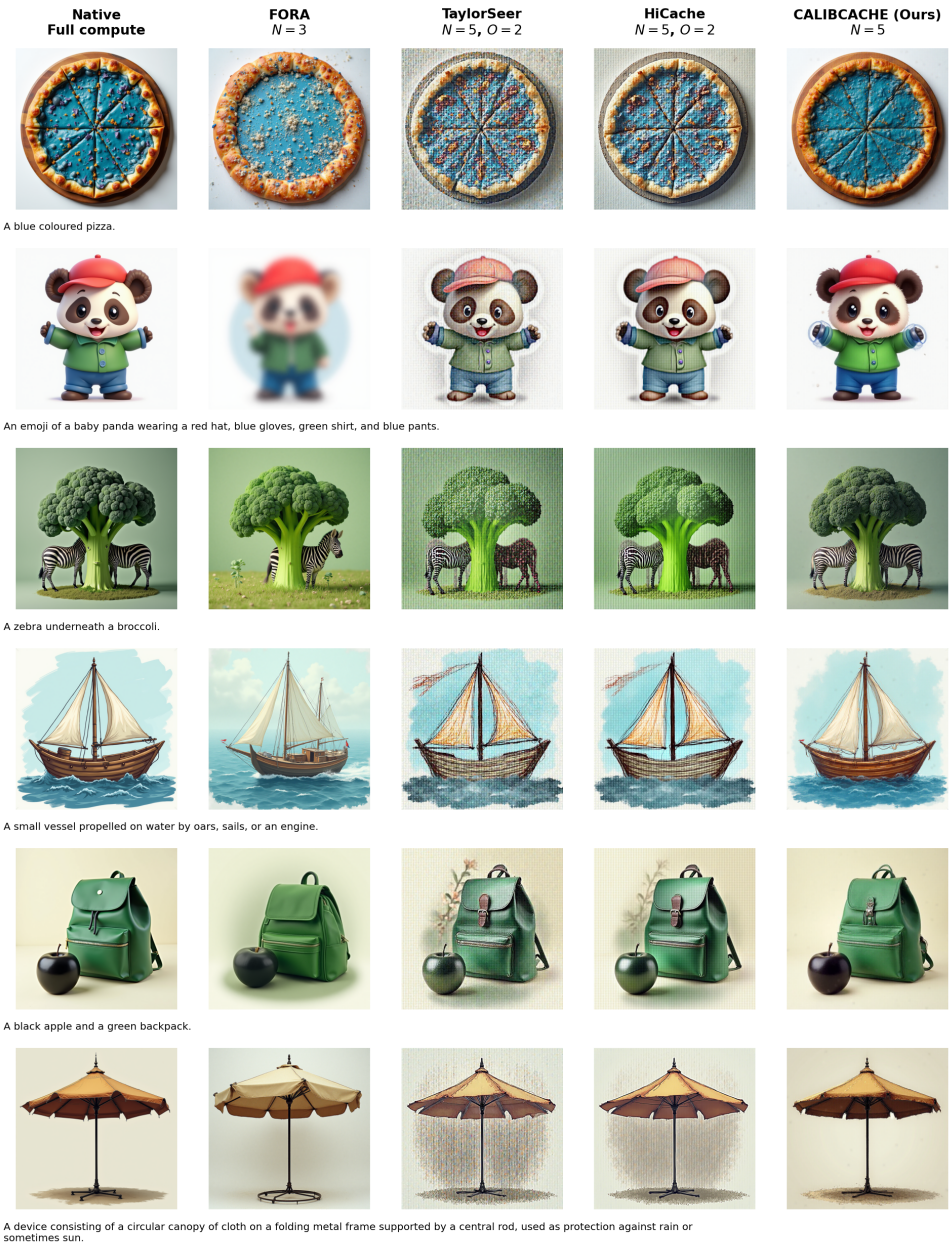


Figure 3: FLUX.1-dev comparisons on DrawBench at 1024^2 with 20 denoising steps. Six prompts are shown in rows, with the exact prompt below each row. As in the introductory figure, FORA uses $N = 3$ without extra warmup; TaylorSeer, HiCache, and CALIBCACHE use $N = 5$ and warmup-3. TaylorSeer and HiCache use $O = 2$, and HiCache uses scale 0.6. These examples highlight reduced texture artifacts and closer preservation of the full-compute appearance and layout.

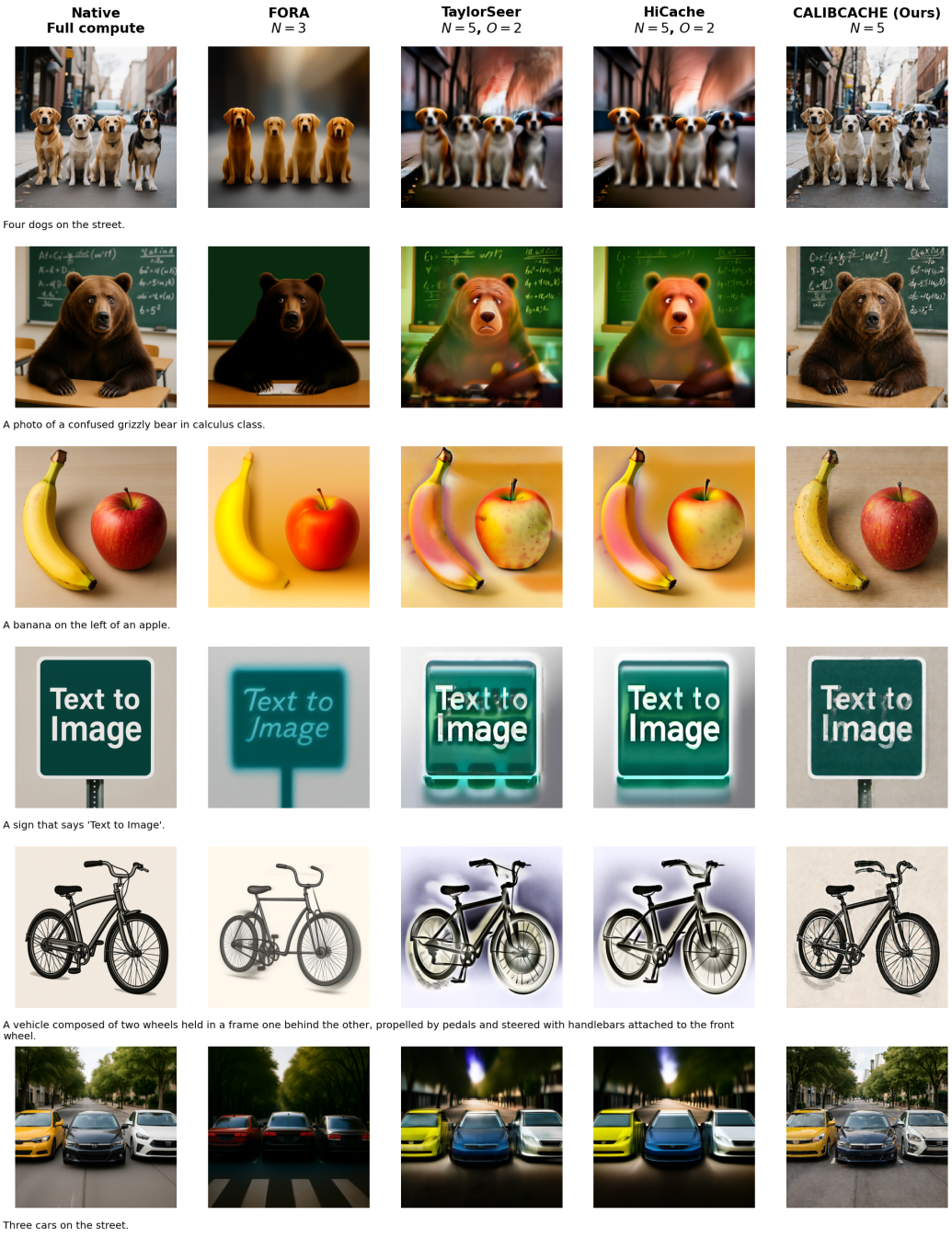


Figure 4: Qwen-Image comparisons on DrawBench at 1024^2 with 20 denoising steps. FORA uses $N = 3$ without extra warmup; TaylorSeer, HiCache, and CALIBCACHE use $N = 5$ and warmup-3. TaylorSeer and HiCache use $O = 2$, and HiCache uses scale 0.6. These examples illustrate clearer subjects and backgrounds and closer preservation of the full-compute scene and material appearance.

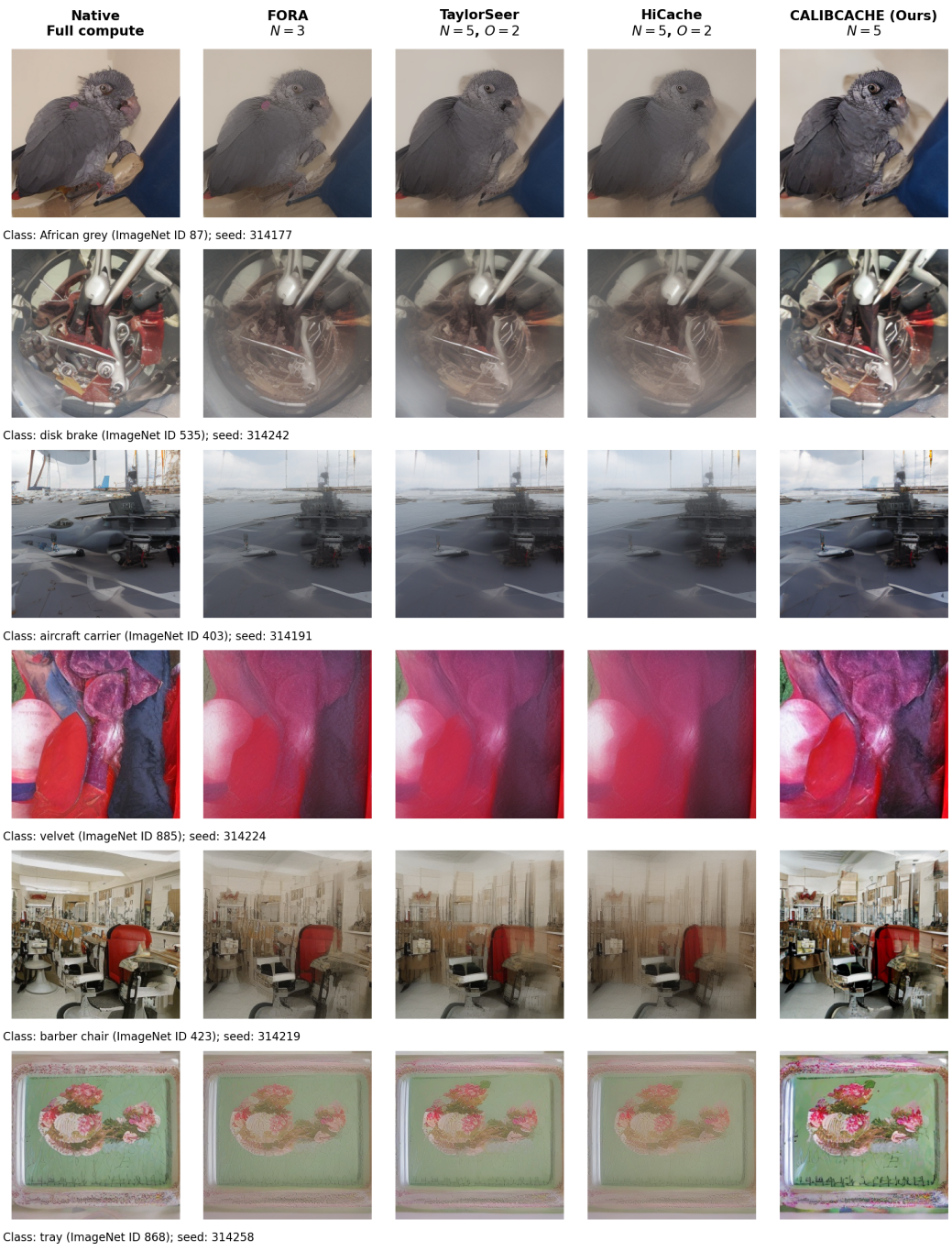


Figure 5: DiT-XL/2 comparisons on ImageNet-256 with DDIM-20 and CFG 1.5. Six examples are selected from 100 paired class–noise inputs, with the class and noise seed below each row. All methods start from the identical saved latent and class label for each example. FORA uses $N = 3$ without extra warmup; TaylorSeer, HiCache, and CALIBCACHE use $N = 5$ and warmup-3. TaylorSeer and HiCache use $O = 2$, and HiCache uses scale 0.6. These examples illustrate differences in texture and detail; aggregate quality is reported in Table 3.