TRAJECTORY-CONSISTENT CALIBRATION FOR CACHE-ACCELERATED DIFFUSION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Transformers require repeated denoiser evaluations during iterative sampling, making inference computationally expensive. Cache-based acceleration reduces this cost by reusing intermediate representations across denoising steps, but the resulting cache-side features can deviate from those produced by fresh computation and degrade generation quality. We identify two coupled calibration challenges. First, a reused or approximately computed feature can differ from its full-computation counterpart at the current calibration site. Second, applying an earlier correction changes the subsequent cache-side trajectory, so a later calibration operator may encounter a feature distribution different from the one used to fit it. We refer to these challenges as *direct cache mismatch* and *trajectory-induced calibration mismatch*, respectively. To address them, we propose *Trajectory-Consistent Calibration* (TCC), which fits closed-form, site-specific calibration operators from paired cache-side and full-computation features without modifying the denoiser parameters or the base cache strategy. Rather than fitting all operators on a single uncorrected cache trajectory, TCC processes selected denoising timesteps sequentially: it probes cache-side features, fits the corresponding operators, and replays and commits the calibrated step before estimating any later operators. Experiments on PixArt-α and DiT-XL/2 report lower FID values across the evaluated cache-based acceleration settings. In a representative PixArt-α setting based on FORA, TCC reduces FID from 29.83 to 27.35, slightly surpassing the corresponding full-computation baseline in terms of FID.

1 INTRODUCTION

Diffusion Transformers (DiTs) (Peebles & Xie, 2023) have substantially advanced visual generation by bringing scalable transformer backbones to diffusion models. However, their iterative denoising procedure requires repeated forward passes of a large backbone, making inference computationally expensive. Since generating a single sample involves multiple denoising steps (Ho et al., 2020; Song et al., 2021a), reducing repeated computation has become critical for efficient diffusion sampling.

Cache-based acceleration methods address this challenge by caching and reusing intermediate representations across nearby denoising steps in DiT. By exploiting temporal redundancy along the diffusion trajectory, they reduce repeated computation and provide a practical route to faster inference (Ma et al., 2024b; Wimbauer et al., 2024). However, cache reuse is inherently approximate rather than lossless. Because a reused representation is computed from a previous or neighboring denoising state (Ma et al., 2024b), it can deviate from the representation that full computation would produce at the current state. As sampling proceeds, this approximation error can further propagate through later denoising steps, causing the cache-accelerated trajectory to drift from the full-computation trajectory and ultimately degrade generation quality.

A key observation of this work is that calibrating cache reuse involves more than correcting an isolated feature error. At a cached site, the reused or approximately computed representation can directly differ from the feature produced by fresh computation; we call this discrepancy *direct cache mismatch*. Such local mismatches can propagate through subsequent denoising steps and contribute to drift of the cache-accelerated trajectory. A second calibration challenge arises once corrections are introduced. Applying an earlier correction changes the denoiser output and hence the latent and cache state entering later timesteps. Consequently, a later operator fitted on the original un-

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: Qualitative comparison on PixArt-α under cache-accelerated sampling, with prompts shown verbatim. These representative cases illustrate how TCC restores prompt-relevant objects weakened by cache reuse, such as the parking sign and green fire hydrants, and improves local details such as facial features while retaining the overall composition. All comparisons use matched prompts, seeds, and sampling settings.

corrected cache trajectory may be deployed on a different cache-side feature distribution. We call this estimation–deployment discrepancy *trajectory-induced calibration mismatch*. It is not an independent error introduced by cache reuse, but a failure mode of fitting later corrections without accounting for the history shaped by preceding ones.

Existing cache-acceleration research either selects what to reuse (Selvaraju et al., 2024; Ma et al., 2024a; Zou et al., 2024) or corrects errors on top of an existing policy. Offline plug-in methods precompute reusable signals: EOC uses adjacent-timestep trends, while increment-calibrated caching uses low-rank operators (Qiu et al., 2025b; Chen et al., 2025). TCC targets this offline plug-in setting and a specific estimation issue: an operator fitted on an uncorrected cache trajectory can become mismatched after preceding corrections change its deployment trajectory.

We propose *Trajectory-Consistent Calibration* (TCC), a calibration method that leaves the denoiser parameters and base cache strategy unchanged. TCC maintains a full-computation reference trajectory and a separately calibrated cache trajectory. At each selected denoising timestep, it collects paired full-computation and cache-side features, fits closed-form site-specific operators, and then replays and commits the calibrated cache step. Statistics for later operators are therefore collected only after the effects of preceding calibrations have entered the cache trajectory. During inference, the precomputed operators are applied directly to selected cache-side features, while all remaining computation follows the base cache strategy.

We validate TCC on PixArt-α and DiT-XL/2 across representative cache baselines, including FORA, ToCa, and L2C. Representative reported results include an FID change from 29.83 to 27.35 on PixArt-α with FORA and from 6.82 to 5.70 on DiT-XL/2 with FORA. Qualitative examples are shown in Figure 1, with additional comparisons provided in the supplementary material. Our contributions are summarized as follows:

- We identify two coupled calibration challenges in cache-accelerated diffusion: direct cache mismatch between cache-side and full-computation representations, and trajectory-induced calibration mismatch caused when preceding corrections change the feature distribution encountered by later operators.

- We propose *Trajectory-Consistent Calibration* (TCC), which combines closed-form local feature alignment with sequential, timestep-wise prior estimation along the calibrated cache trajectory, without modifying the cache strategy or model parameters.
- We evaluate TCC on PixArt-α and DiT-XL/2 with FORA, ToCa, and L2C, and compare trajectory-consistent prior estimation with one-shot calibration through controlled ablations.

2 Problem Formulation

Diffusion sampling generates a sample through a sequence of denoising states $x_T, x_{T-1}, \ldots, x_0$. At each timestep t, a transformer-based denoiser performs a forward pass on x_t and produces intermediate features at multiple internal sites. We index an internal feature site by

$$u = (t, \ell, m),$$

where t is the denoising timestep, ℓ identifies the transformer block, and m specifies a module output such as attention or MLP. For PixArt-α, m may additionally identify a cross-attention output. TCC is applied only to a selected subset of sites affected by a base cache strategy π. Let a_u denote the feature at site u along a full-computation reference trajectory, and let b_u denote the corresponding cache-side feature produced by reuse or approximate computation. The two features correspond to the same representative sample, timestep, and internal site, but their incoming representations may differ because preceding cache decisions or committed calibrations have already altered the cache-side trajectory. We apply a site-specific operator $\mathcal{C}_u$ to selected cache-side features while leaving the model, sampler, and base cache strategy unchanged.

Direct cache mismatch. At a cache-affected site, the base strategy may reuse a stale feature or use an approximate computation instead of freshly evaluating the corresponding module. The resulting cache-side feature therefore generally differs from its full-computation reference:

$$b_u \neq a_u.$$

We refer to this representation discrepancy as direct cache mismatch and use a_u as the representation-level calibration reference. Here, *direct* identifies the site at which the discrepancy is observed rather than a same-input causal decomposition: because a_u and b_u are collected along separate trajectories, their difference may already reflect trajectory effects inherited from preceding timesteps. TCC locally aligns the cache-side feature distribution with the corresponding full-computation distribution at each selected site.

Trajectory-induced calibration mismatch. This second mismatch is not an independent error introduced by cache reuse. It arises because correcting an earlier direct mismatch changes the denoiser output and hence the latent and cache state entering later timesteps. If all operators are fitted in one shot on the uncalibrated cache trajectory, a later operator may therefore encounter a different cache-side feature distribution during calibrated inference.

To express this dependence without tracking every preceding activation, let $\mathcal{T}_t$ denote the runtime state entering timestep t, including the current diffusion latent and the cache state maintained by the base accelerator. We write $b_u(\mathcal{T}_t)$ for the cache-side feature produced at site u when the base cache strategy enters timestep t with this state. We distinguish $\mathcal{T}_t^{\mathrm{cache}}$, reached by the base cache strategy without calibration, from $\mathcal{T}_t^{\mathrm{cal}}$, reached after committing calibrations at preceding timesteps. For two successive calibration timesteps, applying the earlier operator changes the state entering the later one, and a later cache-side feature may consequently differ under the two trajectory states:

$$b_u\big(\mathcal{T}_t^{\mathrm{cache}}\big) \neq b_u\big(\mathcal{T}_t^{\mathrm{cal}}\big).$$

One-shot calibration estimates later operators using features from the first trajectory but deploys them after preceding corrections have moved the runtime state to the second. We refer to this discrepancy as trajectory-induced calibration mismatch. It can accumulate across denoising timesteps because each calibrated step changes the state from which later operators are applied. TCC addresses direct cache mismatch with local statistical alignment and reduces trajectory-induced calibration mismatch by estimating operators sequentially along the cache trajectory produced by preceding calibrated timesteps.

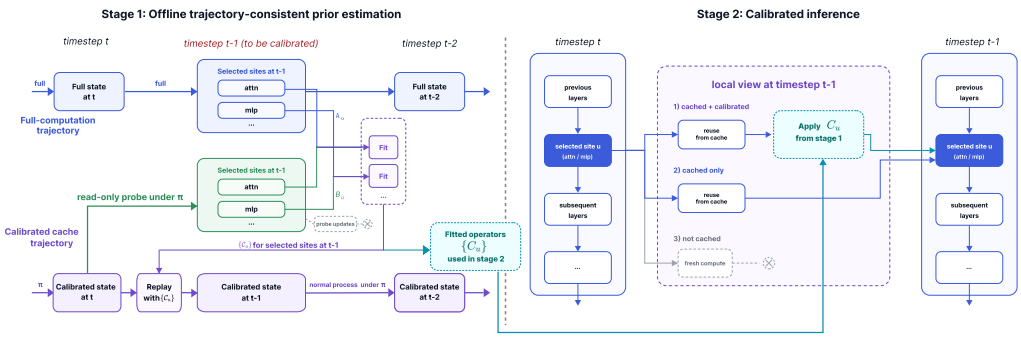


Figure 2: Overview of trajectory-consistent calibration (TCC). In Stage 1, TCC performs a full-computation pass and a read-only cache probe at each selected calibration timestep, fits one operator $\mathcal{C}_u$ for each selected site, and then replays the timestep with the fitted operators. This calibrated replay determines the runtime state used to estimate priors at subsequent timesteps. In Stage 2, the corresponding site-specific operator is applied inline to each selected reused feature, while other sites follow the base cache strategy. Operators within one timestep are fitted from the same probe pass; prior estimation is sequential across denoising timesteps. The calibrated replay is essential: it commits the updated state before statistics for later timesteps are estimated, thereby accounting for the trajectory shift induced by preceding corrections. The displayed $\{\mathcal{C}_u\}$ are the inference-time operators instantiated from the estimated calibration priors.

3 METHOD

This section presents *Trajectory-Consistent Calibration* (TCC), a closed-form calibration method for cache-accelerated diffusion models. As illustrated in Figure 2, TCC has two stages. During offline prior estimation, it collects paired full-computation and cache-side features, estimates site-specific calibration priors, and fits the corresponding operators. Here, a calibration prior refers specifically to the precomputed alignment statistics μ_A, μ_B, R, s at a fixed site; the operator $\mathcal{C}_u$ is instantiated from these statistics for the chosen calibration strength α. Crucially, TCC processes denoising timesteps sequentially, so the statistics used for a later operator are collected after the effects of preceding calibrated timesteps have entered the runtime trajectory state. During inference, the precomputed operator $\mathcal{C}_u$ is applied directly to the cache-side feature at its selected site. The local statistical transform determines how a cache-side feature is corrected at a fixed site, whereas trajectory-consistent prior estimation determines under which cache trajectory its statistics are estimated. The latter distinguishes TCC from one-shot calibration.

3.1 LOCAL STATISTICAL CALIBRATION

Consider a selected calibration site u. For clarity, we fix this site and omit its subscript from the paired statistics in the following derivation. Given K representative samples indexed by i, let a_i denote the full-computation feature and let b_i denote the corresponding cache-probe feature. Their paired collection from the full-computation reference and calibrated cache trajectories is described in Section 3.2. After the pooling used for fitting, each a_i and b_i is a D-dimensional feature vector. We stack them as rows of

$$A = [a_1; \ldots; a_K], \qquad B = [b_1; \ldots; b_K].$$

Thus, A and B each have shape $K \times D$, and row i corresponds to the same representative sample in both matrices. For class-conditioned DiT, we average token-pooled features from multiple sampled trajectories of the same class. For text-conditioned PixArt-α, each prompt provides one representative sample. Pooling is used only to estimate the shared transform; during inference, the fitted $D \times D$ transform is applied to each original, unpooled activation vector and shared across positions at the fixed site.

Our key modeling assumption is that, at a fixed calibration site, cache-side and full-computation features exhibit a systematic statistical shift that can be estimated from representative paired samples.

We approximate the dominant part of this shift with a global orthogonal transformation, scale, and translation. This simple similarity transform admits a stable closed-form fit and low-cost application, although it is not intended to represent arbitrary nonlinear or sample-specific errors.

Let μ_A and μ_B denote the feature-wise means across the K paired samples. The two means locate the centers of the respective feature distributions, and their difference captures the average translation between them. We remove the respective means by defining

$$A_c = A - \mu_A, \qquad B_c = B - \mu_B.$$

After centering, the cross-covariance between the paired features describes their directional correspondence. We estimate an orthogonal transformation R and a non-negative global scale s using a scaled Orthogonal Procrustes problem (Schönemann, 1966):

$$(R, s) = \underset{R^\top R = I,\ s \geq 0}{\arg\min} \|A_c - sB_cR\|_F^2 .$$

Let $B_c^\top A_c = U\Sigma V^\top$ be the singular value decomposition of the cross-covariance matrix. The closed-form solution is

$$R = UV^\top, \qquad s = \frac{\langle A_c, B_cR \rangle}{\|B_cR\|_F^2 + \epsilon},$$

where $\epsilon > 0$ is a small constant used for numerical stability.

At the fixed site, the precomputed statistics μ_A, μ_B, R, s constitute the calibration prior. The paired features A, B are the data used to estimate this prior, while $\mathcal{C}_u$ denotes the operator instantiated from it and applied during inference.

Here, R aligns the dominant feature directions, while s matches their overall magnitude. Together with the means, the resulting map first centers a cache-side feature, then orthogonally transforms and rescales it, and finally shifts it to the full-computation mean. For a cache-side feature b at the fixed site, this map is

$$\mathcal{P}(b) = \mu_A + s(b - \mu_B)R.$$

We then define the site-specific calibration operator as

$$\mathcal{C}_u = \mathrm{Fit}(A, B; \alpha), \qquad \mathcal{C}_u(b) = b + \alpha\big(\mathcal{P}(b) - b\big). \tag{1}$$

Here α controls the calibration strength. When $\alpha = 0$, $\mathcal{C}_u$ is the identity map and TCC reduces to the base cache strategy. When $\alpha = 1$, it applies the complete fitted Procrustes map. When $0 < \alpha < 1$, the operator interpolates between the raw cache-side feature and the fitted Procrustes output. Values $\alpha > 1$ extrapolate beyond the fitted map and provide a stronger correction, whose stability is evaluated empirically. The operator is obtained in closed form and requires no gradient-based training.

3.2 Timestep-Wise Trajectory-Consistent Prior Estimation

TCC estimates calibration priors sequentially across denoising timesteps. At the start of timestep t, the full-computation reference trajectory has runtime state $\mathcal{T}_t^{\mathrm{full}}$, while the calibrated cache trajectory has state $\mathcal{T}_t^{\mathrm{cal}}$. These states are maintained separately from the same representative samples. The calibrated state includes the effects of all calibration operators fitted at preceding denoising timesteps. For fixed A_u, B_u, α changes the instantiated operator, not the fitted prior statistics. However, because that operator is used in the committed replay, α changes the cache-side data from which priors at later timesteps are estimated. The resulting collection of site-specific priors is therefore α-specific and is re-estimated for each tested value. At a selected calibration timestep, TCC performs four operations.

1. Full reference. TCC advances the full-computation trajectory through timestep t and records $A_u = [a_{u,1}; \ldots; a_{u,K}]$ for every selected site u in this timestep.

2. Read-only cache probe. TCC saves the incoming calibrated state $\mathcal{T}_t^{\mathrm{cal}}$ before the probe. Starting from this saved state, it executes one cache-accelerated denoising step under the base cache strategy π, with operators for the current timestep disabled, and records $B_u = [b_{u,1}; \ldots; b_{u,K}]$. It then discards the latent and cache-state updates produced by the probe. The batches A_u and B_u are paired by representative sample, timestep, and site, although their local incoming representations need not

Algorithm 1 Offline timestep-wise trajectory-consistent prior estimation

Require: Base cache strategy π, selected calibration sites, representative samples, strength α
Initialize $\mathcal{T}^{\text{full}}$ and $\mathcal{T}^{\text{cal}}$ from the same samples
for each denoising timestep t **do**
  Run one full-computation step, record A_u for selected sites in timestep t, and update $\mathcal{T}^{\text{full}}$
  if timestep t contains selected calibration sites **then**
    Save $\mathcal{T}_t^{\text{cal}}$ as the common probe/replay start state
    Probe the cache step from the saved start state, record B_u for all selected sites, and discard the probed next state
    for each selected site u in timestep t **do**
      Estimate the calibration prior from A_u, B_u and instantiate $\mathcal{C}_u \leftarrow \text{Fit}(A_u, B_u; \alpha)$
    end for
    Replay the cache step from the saved start state, apply $\mathcal{C}_u$ inline, and commit the output to $\mathcal{T}^{\text{cal}}$
  else
    Advance $\mathcal{T}^{\text{cal}}$ using the base cache strategy π
  end if
end for
return the fitted site-specific operators $\{\mathcal{C}_u\}$

be identical because they are collected from separate trajectories. Operators fitted at preceding timesteps are already reflected in the saved state; disabling the current operators ensures that B_u is the uncorrected cache-side feature to which the new operator will be applied.

3. Prior estimation and operator fitting. For each selected site, TCC estimates the calibration prior from A_u, B_u and instantiates its operator using

$$\mathcal{C}_u = \text{Fit}(A_u, B_u; \alpha).$$

4. Calibrated replay. TCC restarts the cache step from the same saved incoming state, applies each newly fitted $\mathcal{C}_u$ inline at its selected site, and commits the resulting latent and cache state as the calibrated state for the next denoising timestep. Only this replay updates the calibrated trajectory; it is not used to refit the current operators. Because the replay is committed before any later operators are estimated, subsequent statistics are collected under a trajectory that already reflects preceding calibrations.

Timestep-wise rather than layerwise fitting. TCC uses one probe to collect all selected sites within a timestep. For full-branch reuse methods such as FORA, the reused attention and MLP outputs are stale tensors that do not depend on the current residual stream. Applying an earlier within-timestep calibration therefore does not change later raw cached outputs, and layerwise re-probing produces the same fitting data. For partially recomputed methods such as L2C and ToCa, an earlier calibration may affect later recomputed features through the residual stream. In these cases, one-probe timestep-wise fitting is an efficiency-oriented approximation. TCC uses this uniform procedure while enforcing sequential consistency across denoising timesteps by committing each calibrated timestep before estimating the next.

At full-computation timesteps, the base accelerator introduces no additional cache-reuse mismatch. Such a step may still propagate residual trajectory differences inherited from preceding cached timesteps; the calibrated and full-computation runtime states are therefore not assumed to become identical.

4 Experiments

4.1 Experiment Settings

Model configurations. We evaluate TCC on two representative generation tasks: text-to-image generation and class-conditional image generation, using NVIDIA H800 80GB GPUs. For text-to-image generation, we use PixArt-α (Chen et al., 2023) with DPM-Solver++ (Lu et al., 2025)

Table 1: Text-conditional image generation on MS-COCO 2017 with PixArt-α. Latency is measured end-to-end, including ToCa's dynamic token weighting and TCC calibration; hyperparameters are provided in the supplementary material.

Method	Latency (s) ↓	FLOPs ↓	Speed ↑	MS-COCO 2017 FID-30K ↓	MS-COCO 2017 CLIP ↑
PixArt-α	0.9832	11.1225	1.00×	28.15	31.03
50% steps	0.4997	5.5613	1.97×	37.47	30.50
FORA ($N = 3$)	0.4580	3.8902	2.15×	29.83	30.98
FORA ($N = 3$) + TCC	0.5388	4.3469	1.82×	**27.35**	30.75
ToCa ($N = 3$, $R = 60\%$)	0.9720	6.5357	1.01×	28.03	30.95
ToCa ($N = 3$, $R = 90\%$)	0.9210	4.6580	1.07×	29.63	30.95
ToCa ($N = 3$, $R = 90\%$) + TCC	1.0480	5.1146	0.94×	**27.36**	30.37

sampling at 256×256 resolution and 20 sampling steps. We compare TCC with representative cache-based acceleration baselines, including FORA (Selvaraju et al., 2024) and ToCa (Zou et al., 2024). We do not include L2C (Ma et al., 2024a) in this setting, since the original L2C work trains and evaluates its router on class-conditional DiT models.

For class-conditional generation, we adopt DiT-XL/2 (Peebles & Xie, 2023) with the DDIM (Song et al., 2021a) sampler and evaluate two 20-step settings: 256×256 and 512×512 generation. On DiT-XL/2, we compare with representative cache-based acceleration methods, including the learned-policy L2C and FORA. We further compare with EOC (Qiu et al., 2025b), our most directly matched baseline in the offline plug-in setting: both augment existing FORA or L2C policies with precomputed corrections, using adjacent-timestep trends for EOC and trajectory-consistent paired priors for TCC.

Prior estimation and evaluation. TCC estimates site-specific calibration priors offline and applies the corresponding precomputed operators during cached sampling. For PixArt-α, we construct a prompt-level prior-estimation set using 1K prompts sampled from the MS-COCO 2017 (Lin et al., 2015) training captions, without overlap with the 30K prompts used for FID (Heusel et al., 2018) evaluation. For DiT-XL/2, we generate 100 samples per ImageNet (Deng et al., 2009) class and average their token-pooled representations within each class to obtain class-level representative features for prior estimation. Following Section 3, each calibration prior consists of the precomputed site-specific alignment statistics μ_A, μ_B, R, s, from which the corresponding inference-time operator is instantiated. All prior statistics and operator parameters are computed in FP32. We repeat offline prior estimation for each model, base cache strategy, cache configuration, calibration configuration, and selected correction strength α. This offline cost is not included in per-sample inference latency; the reported latency includes the calibration transforms applied during cached sampling. We report both FLOPs and wall-clock latency because dynamic token selection and memory movement can make FLOPs an incomplete proxy for end-to-end speed. We provide a diagnostic analysis of the TCC application window in the supplementary material.

For text-to-image generation, we follow the MS-COCO-30K protocol used by ToCa and report FID-30K for distributional fidelity and CLIP Score for image-text alignment. The CLIP Score is computed using CLIP ViT-B/32 (Radford et al., 2021). For class-conditional generation, we follow the standard DiT evaluation protocol by uniformly sampling 50,000 images from the 1,000 ImageNet classes. We report FID-50K, sFID, and Inception Score (Salimans et al., 2016). For each base cache strategy and evaluation setting, we use a fixed calibration window, site set, and correction strength for all generated samples, rather than adapting these choices per sample. Further details on hyperparameter selection are provided in the supplementary material.

4.2 Main Results

Text-to-image generation. Table 1 shows that TCC lowers FID-30K from 29.83 to 27.35 for FORA and from 29.63 to 27.36 for the aggressive ToCa ($R = 90\%$) setting; FORA+TCC also yields lower FID than full-computation PixArt-α. For ToCa, increasing reuse from $R = 60\%$ to

Table 2: Results on ImageNet with DiT-XL/2 under 20-step DDIM sampling. TCC hyperparameters are summarized in the supplementary material.

Resolution	Method	Latency (s) ↓	FLOPs ↓	Speed ↑	FID ↓	sFID ↓	IS ↑
256×256	DiT	0.8164	9.5055	1.00×	3.51	4.93	223.54
	L2C	0.6704	7.4006	1.22×	3.50	4.66	**225.44**
	L2C + EOC	0.7249	7.4100	1.12×	3.45	4.68	223.96
	L2C + TCC	0.7127	7.6887	1.15×	**3.45**	**4.62**	224.54
	FORA ($N = 2$)	0.4727	4.7634	1.73×	6.82	8.65	190.01
	FORA ($N = 2$) + EOC	0.5533	4.7700	1.48×	5.82	**5.34**	**195.84**
	FORA ($N = 2$) + TCC	0.5208	5.0678	1.57×	**5.70**	5.57	194.57
512×512	DiT	4.6796	42.0977	1.00×	4.98	6.56	186.18
	L2C	3.7654	32.7711	1.24×	6.18	8.15	178.29
	L2C + TCC	3.9908	33.9235	1.17×	**6.03**	**6.94**	**178.56**
	FORA ($N = 2$)	2.5709	21.0647	1.82×	12.13	12.02	136.98
	FORA ($N = 2$) + TCC	2.6790	21.9778	1.75×	**10.99**	**11.06**	**139.27**

$R = 90\%$ lowers FLOPs but raises FID-30K from 28.03 to 29.63 before calibration. The gains are metric-dependent: CLIP Score remains close for FORA+TCC but decreases for ToCa+TCC. FORA+TCC retains clear wall-clock acceleration, whereas ToCa+TCC is slowed by dynamic token processing and calibration overhead; see the supplementary material.

Class-conditional image generation. Table 2 summarizes the class-conditional results on DiT-XL/2. On ImageNet 256×256, TCC improves both L2C and FORA, reducing FID from 3.50 to 3.45 for L2C and from 6.82 to 5.70 for FORA. Against EOC, TCC matches FID and improves sFID on L2C. On FORA, TCC reports lower FID and latency, whereas EOC reports lower FLOPs and sFID and higher IS; neither method dominates across all reported metrics. On ImageNet 512×512, TCC consistently improves both cache baselines: it reduces FID from 6.18 to 6.03 for L2C and from 12.13 to 10.99 for FORA. We omit EOC at 512×512 because the original paper does not report this setting. The improvements are most consistent on distributional image-quality metrics such as FID and sFID, while Inception Score remains broadly comparable across calibrated and uncalibrated cache variants.

4.3 Ablation Study

Trajectory-consistent prior estimation. We first study whether calibration priors should be estimated once from an uncorrected cache trajectory or estimated sequentially along the calibrated trajectory. This ablation is conducted on DiT-XL/2 at ImageNet 256×256 with 20-step DDIM sampling, using FORA($N = 2$) as the cache baseline. During offline prior construction, the one-shot variant estimates all calibration priors from the original uncorrected cache trajectory, whereas TCC estimates each later prior only after preceding calibrated timesteps have been replayed and committed. Once constructed, both variants use the resulting fixed operators during evaluation sampling. Thus, TCC collects calibration statistics under the trajectory distribution induced by earlier calibrations. Figure 3 shows the sensitivity to the correction strength α. The two variants start from the same baseline at $\alpha = 0$. As α increases, one-shot calibration rapidly over-corrects the cached representations, leading to severe degradation. This indicates that calibration transformations estimated only once from the uncorrected trajectory do not remain well matched after the sampling trajectory has been altered by preceding calibrations. In contrast, TCC remains stable and improves FID within the tested range. This supports the main design of TCC: calibration priors should be estimated sequentially along the calibrated trajectory, rather than being estimated once from a single uncorrected cache trajectory. The corresponding numerical results are provided in the supplementary material.

Effect of transformation components. We further ablate the transformation components used by TCC. Under the same FORA($N = 2$) setting, we compare three variants: shift-only calibration, which only aligns the representation mean; scale+shift calibration, which aligns the mean and a global scalar scale while removing the orthogonal component; and full TCC, which uses the complete similarity transformation. As shown in Table 3, shift-only calibration barely improves FID and

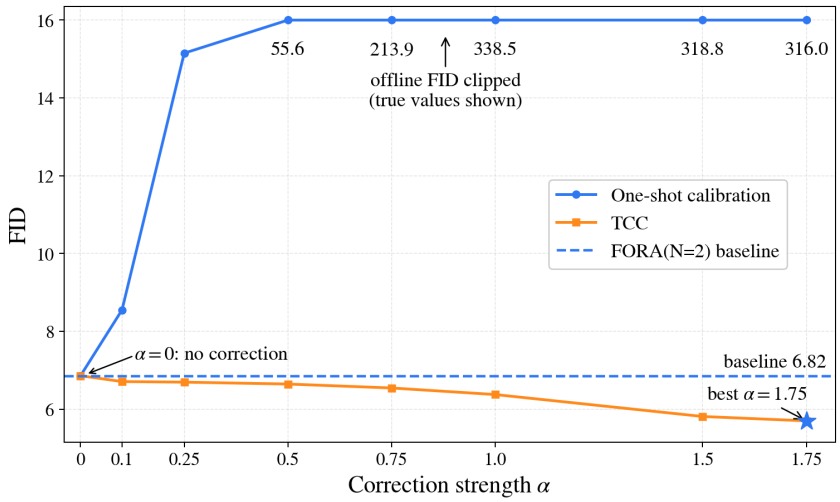


Figure 3: Sensitivity to the correction strength α on ImageNet 256×256 with 20-step DDIM sampling under FORA($N = 2$). When $\alpha = 0$, both variants reduce to the FORA($N = 2$) baseline. One-shot calibration becomes increasingly harmful as α grows, while TCC remains stable and improves FID within the tested range. For readability, one-shot points with very large FID are clipped and marked.

Table 3: Ablation on transformation components under FORA($N = 2$) on DiT-XL/2 (ImageNet 256×256, 20-step DDIM). For each calibration variant, we report the best result over the tested α values, with the selected α shown in parentheses.

Method	**FID** ↓	**sFID** ↓
FORA($N = 2$) baseline	6.82	8.65
Shift-only calibration ($\alpha = 0.50$)	6.81	8.87
Scale+shift calibration ($\alpha = 2.00$)	5.81	6.63
TCC ($\alpha = 1.75$)	**5.70**	**5.57**

slightly worsens sFID, indicating that mean alignment alone is insufficient. Scale+shift calibration recovers a large portion of the FID gain, but remains clearly worse than TCC in sFID. TCC with the complete similarity transformation achieves the best FID and sFID trade-off, suggesting that the complete transformation is important for preserving both image quality and structural fidelity. Full curves for FID and sFID are provided in the supplementary material.

5 Conclusion

We presented TCC, a closed-form calibration method for cache-accelerated diffusion inference. By estimating each site-specific prior after preceding calibrated timesteps have been committed, TCC reduces the estimation–deployment mismatch of one-shot calibration without changing the denoiser, sampler, or base cache strategy. Across the evaluated PixArt-α and DiT-XL/2 settings, TCC reports lower FID values for FORA, ToCa, and L2C. Its effectiveness and overhead remain configuration- and metric-dependent, as discussed in the supplementary material.

AI use statement

In this work, generative AI tools were used to improve the readability and linguistic presentation of the manuscript and to support the search for and identification of potentially relevant prior work. Generative AI tools were not used for any of the research tasks for which disclosure is required under the ICLR 2027 policy, such as developing the theoretical framework, formulating mathematical claims, designing the methodology or experiments, implementing the proposed method, analyzing

data, or interpreting results. All AI-assisted language edits and literature suggestions were reviewed by the authors, and suggested references were checked against their original sources before inclusion. The authors take responsibility for the final content of this work, including AI-assisted text edits and literature suggestions.

ETHICS STATEMENT

This work studies inference acceleration for existing diffusion models using public benchmark datasets and does not introduce a new dataset or involve human subjects. TCC changes the computational procedure used during inference rather than expanding the semantic capabilities of the underlying generative models. Nevertheless, it may inherit the biases, limitations, and potential misuse risks of those models, and reducing generation cost may also lower the cost of producing deceptive or harmful synthetic content. The limitations and broader impact of the method are discussed further in the appendix.

REPRODUCIBILITY STATEMENT

The main paper describes the models, samplers, datasets, evaluation metrics, cache baselines, calibration-data construction, and evaluation protocols used in the experiments. The method section and Algorithm 1 specify the calibration and inference procedures. The appendix reports the selected hyperparameters and calibration windows, additional ablations, multi-seed results, latency measurements, and compute resources. The supplementary material includes an anonymous code package containing the implementation, configurations, and evaluation scripts used in this work.

REFERENCES

Anirud Aggarwal, Abhinav Shrivastava, and Matthew Gwilliam. Evolutionary caching to accelerate your off-the-shelf diffusion model, 2026. URL https://arxiv.org/abs/2506.15682.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis, 2023.

Zhiyuan Chen, Keyi Li, Yifan Jia, Le Ye, and Yufei Ma. Accelerating diffusion transformer via increment-calibrated caching with channel-aware singular value decomposition, 2025. URL https://arxiv.org/abs/2505.05829.

Huanpeng Chu, Wei Wu, Guanyu Fen, and Yutao Zhang. Omnicache: A trajectory-oriented global perspective on training-free cache reuse for diffusion transformer models, 2025. URL https://arxiv.org/abs/2508.16212.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Yasaman Haghighi and Alexandre Alahi. Sencache: Accelerating diffusion model inference via sensitivity-aware caching. *arXiv preprint arXiv:2602.24208*, 2026.

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, 2018. URL https://arxiv.org/abs/1706.08500.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Kumara Kahatapitiya, Haozhe Liu, Sen He, Ding Liu, Menglin Jia, Chenyang Zhang, Michael S. Ryoo, and Tian Xie. Adaptive caching for faster video generation with diffusion transformers. *arXiv preprint arXiv:2411.02397*, 2024.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. URL https://arxiv.org/abs/1405.0312.

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. *arXiv preprint arXiv:2411.19108*, 2024.

Joseph Liu, Joshua Geddes, Ziyu Guo, Haomiao Jiang, and Mahesh Kumar Nandwana. Smoothcache: A universal inference acceleration technique for diffusion transformers. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR) Workshops*, pp. 3229–3238, June 2025a.

Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=PlKWVd2yBkY.

Ziming Liu, Yifan Yang, Chengruidong Zhang, Yiqi Zhang, Lili Qiu, Yang You, and Yuqing Yang. Region-adaptive sampling for diffusion transformers, 2025b. URL https://arxiv.org/abs/2502.10389.

Jinming Lou, Wenyang Luo, Yufan Liu, Bing Li, Xinmiao Ding, Weiming Hu, Yuming Li, and Chenguang Ma. Token caching for diffusion transformer acceleration, 2026. URL https://arxiv.org/abs/2409.18523.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *arXiv preprint arXiv:2206.00927*, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, 22: 730–751, 2025. doi: 10.1007/s11633-025-1562-4.

Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching, 2024a.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15762–15772, 2024b. doi: 10.1109/CVPR52733.2024.01492.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182, 2023. doi: 10.1109/ICCV51070.2023.00387.

Junxiang Qiu, Lin Liu, Shuo Wang, Jinda Lu, Kezhou Chen, and Yanbin Hao. Accelerating diffusion transformer via gradient-optimized cache. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17608–17617, 2025a.

Junxiang Qiu, Shuo Wang, Jinda Lu, Lin Liu, Houcheng Jiang, Xingyu Zhu, and Yanbin Hao. Accelerating diffusion transformer via error-optimized cache, 2025b. URL https://arxiv.org/abs/2501.19243.

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. URL https://arxiv.org/abs/2103.00020.

Amirmojtaba Sabour, Sanja Fidler, and Karsten Kreis. Align your steps: Optimizing sampling schedules in diffusion models, 2024.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016. URL https://arxiv.org/abs/1606.03498.

Peter H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966. doi: 10.1007/BF02289451.

Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. Fora: Fast-forward caching in diffusion transformer acceleration. *arXiv preprint arXiv:2407.01425*, 2024.

Tong Shao, Yusen Fu, Guoying Sun, Jingde Kong, Zhuotao Tian, and Jingyong Su. Plug-and-play fidelity optimization for diffusion transformer acceleration via cumulative error minimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=pt4iKnAm0M.

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

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=PxTIG12RRHS.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, Christian Rupprecht, Daniel Cremers, Peter Vajda, and Jialiang Wang. Cache me if you can: Accelerating diffusion models through block caching. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6211–6220, 2024. doi: 10.1109/CVPR52733.2024.00594.

Shuchen Xue, Zhaoqiang Liu, Fei Chen, Shifeng Zhang, Tianyang Hu, Enze Xie, and Zhenguo Li. Accelerating diffusion sampling with optimized time steps. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8292–8301, 2024. doi: 10.1109/CVPR52733.2024.00792.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *NeurIPS*, 2023.

Zhenyu Zhou, Defang Chen, Can Wang, and Chun Chen. Fast ode-based sampling for diffusion models in around 5 steps. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7777–7786, 2024. doi: 10.1109/CVPR52733.2024.00743.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. *arXiv preprint arXiv:2410.05317*, 2024.

A Related Work

Fast sampling for diffusion models. Fast diffusion sampling uses optimized timestep schedules, numerical solvers with fewer denoiser evaluations (Xue et al., 2024; Sabour et al., 2024; Song et al., 2021b; Liu et al., 2022; Lu et al., 2022; Zhao et al., 2023; Zhou et al., 2024), or learned few-step dynamics through distillation and consistency modeling (Salimans & Ho, 2022; Song et al., 2023).

Cache-based acceleration for diffusion models. Cache methods use fixed schedules (Selvaraju et al., 2024; Ma et al., 2024b; Wimbauer et al., 2024); adaptive criteria based on feature similarity, timestep information, sample complexity, or sensitivity (Liu et al., 2025a; 2024; Kahatapitiya et al., 2024; Haghighi & Alahi, 2026); learned layer- or token-level reuse policies (Ma et al., 2024a; Lou et al., 2026); fine-grained token or region reuse (Zou et al., 2024; Liu et al., 2025b); and optimized cache schedules (Chu et al., 2025; Aggarwal et al., 2026). These methods determine when and where to reuse computation.

Plug-in cache-error correction. After reuse decisions have been made, the reused representations may still deviate from their full-computation counterparts and propagate errors through later denoising computation. This motivates plug-in correction methods that operate on top of an existing reuse policy rather than redesigning when or where computation is cached. EOC provides the most directly matched baseline in our offline plug-in setting: it applies precomputed adjacent-timestep trends on top of FORA or L2C (Qiu et al., 2025b). Increment-calibrated caching uses low-rank, channel-aware SVD operators (Chen et al., 2025). TCC likewise precomputes corrections for existing policies, but fits paired-distribution Procrustes priors sequentially along the calibrated trajectory. GOC instead uses sampling-adaptive compensation, propagating recent cached-feature differences with inflection-aware gating (Qiu et al., 2025a); cumulative-error minimization optimizes cache schedules under acceleration budgets (Shao et al., 2026). These methods differ in their correction signal and when it is formed.

B Technical Diagnostics

B.1 Within-label dispersion during late denoising

For DiT, the main configuration forms one representative vector per ImageNet class by averaging token-pooled features from multiple sampled trajectories. This construction assumes that the class-level representative remains informative about individual samples at a fixed denoising step and layer. As a diagnostic of where this assumption may weaken, we measure the within-label dispersion of DiT activations during sampling.

We follow the reverse DDIM order, where denoising proceeds from step 19 to step 0. Thus, smaller step indices correspond to later denoising stages. For a fixed ImageNet class, we generate 100 samples and collect conditional activations from each DiT block at every denoising step. For each branch and layer, we compute the RMS of the within-label standard deviation across samples and compare it with the RMS of the corresponding class-mean activation.

As shown in Figure 4, within-label dispersion for the examined class increases as denoising proceeds. Panel A reports the absolute RMS statistics averaged over layers. In the early-to-middle stages, the class-mean activation remains comparable to, or larger than, the within-label standard deviation. However, the within-label standard deviation grows rapidly in later steps. Panel B summarizes this trend by measuring the fraction of layers whose within-label standard-deviation RMS exceeds the RMS of their corresponding class mean. After the trajectory passes step 12 toward step 0, more than half of the layers satisfy this condition. Panel C further shows the relative dispersion, defined as the ratio between within-label standard-deviation RMS and class-mean RMS. Values above 1 indicate that within-label variation exceeds the activation scale of the corresponding class mean. The ratio rises sharply after the middle steps and remains above 1 for most late denoising stages.

This single-class diagnostic is consistent with restricting TCC to an early-to-middle portion of the denoising trajectory rather than applying it uniformly to all steps. Under the DiT-256 20-step DDIM setting, the evaluated configuration uses steps 19–12. The selected window can vary with the sam-

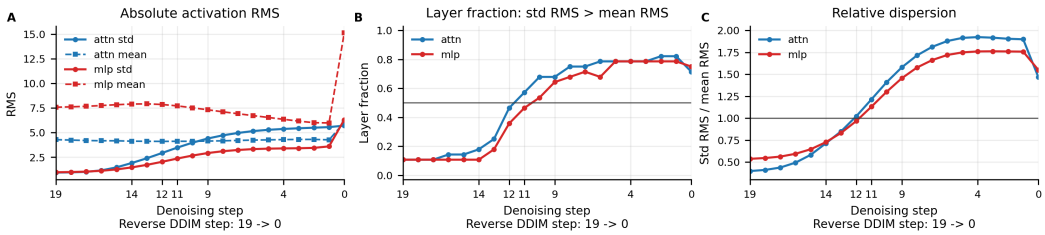


Figure 4: **Within-label activation dispersion increases along the reverse denoising trajectory.** We analyze 100 samples from the same ImageNet class and collect conditional activations from DiT blocks at each DDIM step. The reverse trajectory proceeds from step 19 to step 0. **A:** Absolute RMS statistics averaged over layers; solid lines denote within-label standard-deviation RMS, and dashed lines denote class-mean RMS. **B:** Fraction of layers whose within-label standard-deviation RMS exceeds their class-mean RMS. The horizontal line marks 50% of layers. **C:** Relative dispersion, measured as within-label standard-deviation RMS divided by class-mean RMS. Values above 1 indicate that within-label variation exceeds the class-mean activation scale.

pler, model, and cache schedule; this diagnostic alone does not establish the optimal window for every class or configuration.

B.2 Calibration window and hyperparameter selection

TCC introduces two main hyperparameters: the calibration strength α and the application window over denoising steps. We select them according to the scale and granularity of the errors induced by each base cache strategy, while keeping the denoiser, sampler, and cache schedule unchanged.

The calibration strength α controls how strongly the fitted statistical transform is applied. Because the evaluated cache policies reuse representations at different granularities, we tune α separately for each model and cache configuration. The selected values are larger for module-level FORA on DiT and smaller for the evaluated L2C, ToCa, and PixArt-α configurations. These choices summarize the reported pilot sweep rather than implying that one strength is optimal across models or cache policies.

The application window defines the early-to-middle range in which calibration sites are eligible. Within this range, operators are applied only at the selected sites affected by the base cache strategy; other computation follows that strategy. The window is chosen to cover the early-to-middle part of the reverse denoising trajectory. As discussed in Appendix B.1, late denoising steps contain stronger sample-specific variation, making statistics estimated from class- or prompt-level representatives less reliable. Therefore, we avoid applying TCC uniformly to all steps and restrict calibration to a fixed early-to-middle window. For 20-step DDIM sampling on DiT-XL/2, we typically apply TCC to steps 19–12; for the 512×512 FORA setting, we use a slightly shorter window 19–14. For 50-step sampling, we use the corresponding relative early-to-middle range, namely steps 49–30. For PixArt-α with 20-step DPM-Solver++ sampling, we apply TCC to steps 19–14.

Table 4 summarizes the hyperparameters used in the main experiments and additional ablations. The selected sites cover cache-affected attention and MLP outputs across transformer blocks. For PixArt-α, the attention sites include both self-attention and cross-attention outputs. All sites within a selected timestep are recorded in one read-only probe, after which the timestep is replayed with the fitted operators and committed before later priors are estimated. These settings are selected from a small pilot sweep and then fixed for the reported evaluation. All prior statistics and operator parameters are computed in FP32. Here, N denotes the cache interval, and R denotes the fraction of tokens reused by ToCa.

Table 4: Summary of TCC hyperparameters used in the reported experiments. The denoiser, sampler, and base cache schedule are kept unchanged in all settings.

Model / setting	Base cache	α	Window
PixArt-α 256×256, 20-step	FORA ($N = 3$)	0.50	19–14
PixArt-α 256×256, 20-step	ToCa ($N = 3, R = 90\%$)	0.25	19–14
DiT-XL/2 256×256, 20-step	FORA ($N = 2$)	1.75	19–12
DiT-XL/2 256×256, 20-step	L2C	0.50	19–12
DiT-XL/2 512×512, 20-step	FORA ($N = 2$)	1.25	19–14
DiT-XL/2 512×512, 20-step	L2C	0.50	19–12
DiT-XL/2 256×256, 50-step	FORA ($N = 2$)	1.75	49–30
DiT-XL/2 256×256, 50-step	L2C / ToCa	0.50	49–30

C Additional Experimental Details and Ablations

C.1 Multi-seed stability of the main results

Protocol and number of runs. We repeat the main 20-step experiments using experiment-level global random seeds $\{0, 2025, 2027\}$. Seed 0 is the run reported in the main paper, and seeds 2025 and 2027 are additional runs generated from scratch. Each method configuration is evaluated exactly once per seed: each seed–configuration pair contains one complete generation run followed by one metric-evaluation run, and no within-seed reruns are averaged. Thus, every aggregate in Table 5 is based on $n = 3$ configuration-level runs. Across the 19 configurations reported in this table, this corresponds to 57 seed–configuration runs, including 38 additional runs beyond the seed-0 results. The same seed identifiers are used for each cache baseline and its TCC variant, enabling paired seed-wise comparisons. For DiT, the seed is set through the `-global-seed` argument; for PixArt-α, it is set through the `-seed` argument.

For text-to-image experiments, each run generates $30{,}000$ images using the fixed MS-COCO 2017 prompt set. We report FID-30K and CLIP score using CLIP ViT-B/32, where CLIP score is 100 times the mean image–text cosine similarity. For class-conditional DiT experiments, each run generates $50{,}000$ ImageNet samples, from which we compute FID-50K, sFID, and Inception Score (IS). All methods within a setting use the same prompt or class protocol, model checkpoint, sampling schedule, and evaluation reference statistics.

Statistics and scope. For a metric x and the reported seed subset $\mathcal{S} = \{0, 2025, 2027\}$, we report

$$\bar{x} = \frac{1}{3}\sum_{s\in\mathcal{S}} x_s, \qquad \mathrm{Std}(x) = \sqrt{\frac{1}{3-1}\sum_{s\in\mathcal{S}}(x_s - \bar{x})^2}. \tag{2}$$

The table shows $\bar{x} \pm \mathrm{Std}(x)$ followed by the observed $[\min_s x_s, \max_s x_s]$ range. The bracketed range is not a confidence interval. We compute the aggregate statistics from the available per-seed metric values before rounding and report all summary statistics and observed ranges to two decimal places for consistency with the precision used in the main paper.

Observed stability of the TCC improvement. Across the six directly paired cache baselines—FORA and ToCa on PixArt-α, L2C and FORA on DiT-XL/2 at 256×256, and L2C and FORA on DiT-XL/2 at 512×512—the TCC variant has lower mean FID than its paired cache baseline in every setting reported in Table 5. The mean FID reductions (baseline minus TCC) are 2.46 for PixArt-FORA, 2.06 for PixArt-ToCa, 0.03 for DiT-256-L2C, 1.09 for DiT-256-FORA, 0.12 for DiT-512-L2C, and 1.11 for DiT-512-FORA. The four DiT TCC variants also have lower mean sFID than their paired cache baselines.

Comparison with EOC. For DiT at 256×256, TCC and EOC have similar aggregate means under L2C: their FID values are 3.43 and 3.40, their sFID values are 4.60 and 4.61, and their IS values are 225.74 and 225.71, respectively. Under FORA, TCC has lower mean FID and sFID than EOC: FID changes from 6.33 to 5.70, sFID from 8.27 to 5.54, and IS from 193.57 to 194.37. The larger variation of EOC under FORA, especially for sFID, also illustrates why the per-seed dispersion is reported rather than relying only on aggregate means.

Table 5: Multi-seed stability across the main 20-step settings. Results use $n = 3$ seeds $\{0, 2025, 2027\}$ and are reported as mean $\pm$ sample standard deviation [min, max]. Panel (a) reports FID-30K and CLIP score for PixArt-α; panels (b) and (c) report FID-50K, sFID, and IS for DiT-XL/2. Lower FID and sFID are better; higher CLIP and IS are better.

(a) PixArt-α at 256×256

Method	FID-30K ↓	CLIP ↑
DPM-Solver (20 steps)	28.11 ± 0.04 [28.07, 28.15]	31.04 ± 0.01 [31.03, 31.05]
DPM-Solver (10 steps; 50% steps)	37.42 ± 0.05 [37.38, 37.47]	30.52 ± 0.02 [30.50, 30.53]
FORA ($N = 3$)	29.79 ± 0.04 [29.76, 29.83]	30.99 ± 0.01 [30.98, 31.00]
FORA ($N = 3$) + TCC	27.34 ± 0.03 [27.30, 27.35]	30.74 ± 0.01 [30.73, 30.75]
ToCa ($N = 3, R = 60\%$)	27.97 ± 0.06 [27.93, 28.03]	30.95 ± 0.00 [30.95, 30.96]
ToCa ($N = 3, R = 90\%$)	29.57 ± 0.05 [29.53, 29.63]	30.95 ± 0.01 [30.94, 30.97]
ToCa ($N = 3, R = 90\%$) + TCC	27.52 ± 0.15 [27.37, 27.66]	30.35 ± 0.03 [30.32, 30.37]

(b) DiT-XL/2 at 256×256 with DDIM-20

Method	FID-50K ↓	sFID ↓	IS ↑
DDIM (20 steps)	3.50 ± 0.02 [3.47, 3.51]	4.91 ± 0.03 [4.88, 4.93]	223.31 ± 0.45 [222.79, 223.61]
L2C	3.46 ± 0.04 [3.42, 3.50]	4.65 ± 0.01 [4.64, 4.66]	226.41 ± 0.99 [225.44, 227.43]
L2C + EOC	3.40 ± 0.06 [3.34, 3.45]	4.61 ± 0.06 [4.56, 4.68]	225.71 ± 1.71 [223.96, 227.38]
L2C + TCC	3.43 ± 0.03 [3.39, 3.45]	4.60 ± 0.02 [4.58, 4.62]	225.74 ± 1.53 [224.54, 227.46]
FORA ($N = 2$)	6.79 ± 0.07 [6.71, 6.84]	8.69 ± 0.07 [8.64, 8.76]	191.54 ± 1.50 [190.01, 193.01]
FORA ($N = 2$) + EOC	6.33 ± 0.45 [5.82, 6.65]	8.27 ± 2.53 [5.34, 9.79]	193.57 ± 1.98 [192.21, 195.84]
FORA ($N = 2$) + TCC	5.70 ± 0.01 [5.69, 5.72]	5.54 ± 0.04 [5.49, 5.57]	194.37 ± 0.47 [193.83, 194.70]

(c) DiT-XL/2 at 512×512 with DDIM-20

Method	FID-50K ↓	sFID ↓	IS ↑
DDIM (20 steps)	5.02 ± 0.04 [4.98, 5.07]	6.01 ± 0.48 [5.73, 6.56]	184.51 ± 1.60 [182.99, 186.18]
L2C	6.14 ± 0.04 [6.12, 6.18]	7.51 ± 0.55 [7.19, 8.15]	177.86 ± 1.28 [176.42, 178.87]
L2C + TCC	6.02 ± 0.02 [5.99, 6.04]	7.01 ± 0.06 [6.94, 7.05]	178.13 ± 1.18 [176.80, 179.04]
FORA ($N = 2$)	12.09 ± 0.12 [11.96, 12.19]	12.02 ± 0.04 [11.97, 12.06]	137.28 ± 0.59 [136.91, 137.96]
FORA ($N = 2$) + TCC	10.98 ± 0.04 [10.94, 11.02]	10.99 ± 0.07 [10.92, 11.06]	138.66 ± 1.47 [136.99, 139.72]

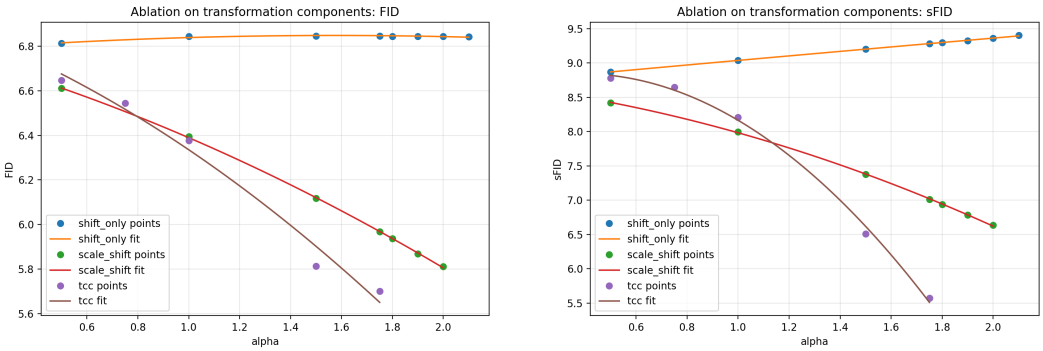
Metric-specific trade-offs. The FID improvements do not imply uniform improvement in every metric. In the PixArt-α experiments, the TCC variants have lower FID but also lower mean CLIP score than their paired cache baselines. We therefore report all requested metrics and restrict the consistency statement above to paired FID and DiT sFID, rather than claiming metric-wise dominance. To complement this metric-level comparison, Appendix C.6 presents additional matched-prompt qualitative examples, which show that TCC preserves the semantic content of the generated images and, in several cases, improves the correspondence between local visual details and the input text.

C.2 Numerical results for one-shot calibration and TCC

Table 6 reports representative points from the coarse α sweep used for the trajectory-consistent prior-estimation ablation in the main paper. When $\alpha = 0$, both variants reduce to the original FORA($N = 2$) baseline, since no correction is applied. Even with a weak correction strength, one-shot calibration already degrades generation quality. As α increases, the degradation becomes severe, indicating that one-shot priors estimated from the uncorrected trajectory yield operators that are not compatible with the trajectory shifted by previous calibrations. For each tested nonzero α, TCC repeats the sequential save–probe–fit–replay procedure because the committed replay changes the cache-side data used to estimate priors at later timesteps. In the reported sweep, the TCC points remain finite and yield lower FID than the FORA baseline over the displayed range.

Table 6: Comparison between one-shot calibration and TCC on ImageNet 256×256 with 20-step DDIM sampling under FORA($N = 2$). All variants apply calibration to all cached attention and MLP outputs.

Method	α	FID ↓	sFID ↓
FORA($N = 2$)	0	6.82	8.65
FORA($N = 2$) + One-shot calibration	0.1	8.55	14.69
FORA($N = 2$) + TCC	0.1	6.71	8.77
FORA($N = 2$) + One-shot calibration	0.5	55.61	198.80
FORA($N = 2$) + TCC	0.5	6.65	8.78
FORA($N = 2$) + One-shot calibration	1.5	318.75	225.75
FORA($N = 2$) + TCC	1.5	5.81	6.51



(a) FID versus α. **(b)** sFID versus α.

Figure 5: Transformation-component ablation under FORA($N = 2$). Markers denote measured results, and curves are quadratic fits for visualizing the trend. Shift-only calibration remains close to the cache baseline, while scale+shift calibration reports lower FID and approaches the full transformation. Compared with scale+shift, the full transformation reports a lower sFID in this setting.

C.3 Full curves for transformation-component ablation

This subsection provides the full results for the transformation-component ablation discussed in the main experimental section. In the main text, we report a compact comparison among shift-only calibration, scale+shift calibration, and the complete TCC transformation. Here, we further show the complete FID and sFID curves under different correction strengths α.

Since different calibration forms may reach their best FID and sFID at different correction strengths, a single α value may not fully reflect their respective behavior. Therefore, we evaluate a range of commonly used α values under the same experimental setting and plot the resulting curves in Figure 5. All variants are evaluated on DiT-XL/2 at ImageNet 256×256 with 20-step DDIM sampling, using FORA($N = 2$) as the cache baseline. TCC is applied to the cached attention and MLP outputs in the same application window and layer range as in the main ablation.

The results are consistent with the compact table in the main text. Shift-only calibration changes FID only marginally and yields a slightly higher sFID than the cache baseline in this setting. Scale+shift calibration recovers most of the reported FID reduction, but its sFID remains higher than that of the complete transformation. The full transformation reports the lowest FID and sFID among the evaluated variants. This ablation indicates that the orthogonal component contributes to the observed trade-off in this configuration; it does not by itself establish that the same component is necessary for every cache policy.

Table 7: Ablation on pooling strategy for TCC under different correction strengths. All results are evaluated on ImageNet 256×256 and use all transformer layers.

α	Pooling strategy	FID ↓	sFID ↓	IS ↑
1.25	Class-pool	15.05	33.93	132.78
1.25	Mixed	8.58	13.81	176.46
1.25	Token-pool	**6.13**	**7.20**	**194.32**
1.50	Class-pool	17.37	40.48	121.20
1.50	Mixed	8.00	10.94	179.21
1.50	Token-pool	**5.81**	**6.27**	**196.72**
1.75	Class-pool	20.43	49.43	107.20
1.75	Mixed	7.34	8.08	182.11
1.75	Token-pool	**5.70**	**5.57**	**194.57**

Table 8: Class-conditional image generation on ImageNet 256×256 with 50-step DDIM sampling. For ToCa, we use $R = 93\%$; TCC hyperparameters are summarized in Table 4.

Method	Latency (s) ↓	FLOPs ↓	Speed ↑	FID ↓	sFID ↓	IS ↑	Precision ↑	Recall ↑
DiT	2.0435	23.7637	1.00×	2.26	4.32	238.21	0.81	0.60
L2C	1.6494	18.1264	1.24×	2.25	**4.28**	**246.47**	0.81	0.58
L2C + TCC	1.7638	18.8494	1.16×	**2.22**	4.32	245.16	0.81	0.59
FORA($N = 2$)	1.1929	11.9085	1.71×	2.65	4.71	**238.04**	0.80	0.59
FORA($N = 2$) + TCC	1.3163	12.6695	1.55×	**2.49**	**4.69**	237.00	0.80	0.59
ToCa($N = 2$)	1.5386	13.5404	1.33×	2.60	**4.43**	**236.57**	0.80	0.58
ToCa($N = 2$) + TCC	1.6186	14.3775	1.26×	**2.55**	4.49	235.84	0.80	0.59

C.4 Pooling strategy for calibration statistics

We ablate how calibration features are aggregated before fitting the TCC calibration transforms. Table 7 compares three pooling strategies on ImageNet 256×256 under different correction strengths. At each calibration site, token-pool averages over calibration samples and token positions within each class, yielding one feature vector per class as a fitting point for a shared Procrustes transform. Class-pool first averages over calibration samples within each class while retaining token positions, and then averages these token-wise statistics across classes, yielding token-indexed prototypes. Mixed independently constructs both corrections and linearly blends their corrected outputs; the reported experiments assign equal weight to the two corrections.

Across all evaluated correction strengths, token-pool achieves the lowest FID and sFID and the highest IS, class-pool performs the worst, and mixed produces intermediate results. At $\alpha = 1.75$, the observed FID values are 20.43 for class-pool, 7.34 for mixed, and 5.70 for token-pool. The same ordering is observed at $\alpha = 1.50$, where token-pool also achieves the best sFID and IS. These results suggest that, in the evaluated DiT-XL/2 and FORA configuration, class-conditioned representatives obtained through token pooling provide a more effective basis for estimating the shared calibration transform than the evaluated token-indexed prototypes obtained through class pooling.

C.5 Additional ImageNet results with 50-step DDIM sampling

We further evaluate TCC on ImageNet class-conditional generation with DiT-XL/2 under the 256×256, 50-step DDIM setting. This setting allows us to compare with L2C, FORA, and ToCa under the same benchmark. In contrast, ToCa does not report a tuned token-caching ratio for the 20-step setting, and directly reusing the reported ratio leads to a poor baseline. Therefore, we use the 20-step setting in the main paper for L2C and FORA, and provide the 50-step comparison here as an additional result.

As shown in Table 8, the TCC variants report lower FID than all three cache baselines in this setting. For L2C, FID changes from 2.25 to 2.22, while keeping precision unchanged and slightly improving

recall from 0.58 to 0.59. For FORA($N = 2$), FID changes from 2.65 to 2.49 and sFID from 4.71 to 4.69. For ToCa($N = 2$), TCC reduces FID from 2.60 to 2.55, with a slight increase in sFID. These are single reported point estimates; repeated-run uncertainty is not evaluated in this table, particularly for the small L2C and ToCa differences.

C.6 Additional qualitative comparisons

We provide additional qualitative comparisons to further illustrate the visual effect of TCC under cache-accelerated sampling. These examples complement the quantitative results in the main paper by showing how TCC affects generated samples under matched prompts or class labels, random seeds, and sampling configurations.

For text-to-image generation, we show more PixArt-α comparisons under the same setting as the PixArt-α qualitative comparison in the main paper. Specifically, we compare the base cache-accelerated sampler with its TCC-calibrated counterpart using matched MS-COCO prompts, seeds, and 20-step DPM-Solver++ sampling at 256×256 resolution. In the displayed cases, the calibrated samples preserve the overall layout of the cached samples while changing several prompt-relevant local details.

For class-conditional generation, we additionally visualize DiT-XL/2 samples on ImageNet at 256×256 resolution with 20-step DDIM sampling and classifier-free guidance scale 4.0. We compare the corresponding cache baseline and TCC-calibrated results under matched class labels and random seeds. The displayed cases retain the semantic category while showing changes in artifacts, object boundaries, and texture details. Because the examples are qualitative and selected, they are not used as evidence of population-level improvement.

C.7 Number of prompts for PixArt representative selection

We further study how many text prompts are needed to estimate representative TCC statistics for text-to-image generation. In this experiment, PixArt-α is used as the base model, ToCa is used as the cache acceleration method, and TCC is applied as a calibration module. We fix the ToCa configuration to $N = 3$ and $R = 90\%$, and only vary the number of MS-COCO2017 captions used for representative selection. To isolate the effect of prompt count, all settings use the same correction strength $\alpha = 0.25$ and the same application window 19–14.

Table 9 compares the results obtained with 1K, 2K, and 5K representative prompts. The observed FID and CLIP values vary non-monotonically with prompt count in this sweep.

Table 9: Ablation on the number of representative prompts used to estimate TCC statistics for PixArt-α under the same ToCa-accelerated sampling pipeline.

Prompts	**MS-COCO2017**	
	FID-30K $\downarrow$	**CLIP** $\uparrow$
1K	**27.36**	**30.37**
2K	28.13	30.25
5K	27.79	30.33

Increasing the representative prompt set from 1K to 5K does not produce a monotonic improvement in either metric. The 1K run has the lowest observed FID and highest observed CLIP score among these three runs. This table reports point estimates and does not measure variation across prompt subsets or generation seeds.

C.8 Latency overhead of ToCa on PixArt-α

Latency and FLOPs measurement. Latency is measured with batch size 8 using CUDA events. The latency values reported in the tables correspond to the end-to-end wall-clock time for generating one batch of eight images. For each method, we first perform one warm-up sampling run and then report the average latency over five timed sampling runs under the same sampling configuration. The

speedup values are computed from these batch latency measurements relative to the corresponding full-computation baseline.

The FLOPs reported in the tables are normalized per generated image. They are computed separately from the CUDA-event timing: for DiT and PixArt-α cache/TCC variants, FLOPs are obtained from analytical estimators matched to the corresponding cache schedule and TCC application window; for the DiT-ToCa baseline, FLOPs are taken from the original ToCa FLOPs counter. Computing FLOPs separately avoids adding profiling overhead to the latency measurements and makes the FLOPs numbers reflect estimated dense model-computation cost rather than measured runtime. Since all methods within each table are compared under the same batch-size and FLOPs-normalization protocol, these numbers are intended to support relative speed–cost comparisons rather than to equate wall-clock latency and FLOPs units directly.

The FLOPs estimates include the denoiser forward computations and the additional TCC calibration operators, while excluding VAE decoding, text encoding, I/O, and metric computation. The additional FLOPs introduced by TCC come only from applying the fitted operators at selected cached sites. For cache baselines, reused cached representations are counted as zero additional dense-computation FLOPs, and only the freshly computed modules or tokens are counted according to the corresponding cache schedule.

Latency analysis of ToCa+TCC. The end-to-end latency numbers reported in the main paper are measured under the above protocol and are used for the main speed-quality comparison. To further explain the apparently unusual latency behavior of ToCa+TCC, especially the fact that it can be slower in wall-clock time despite lower estimated FLOPs, we conduct an additional fine-grained profiling experiment on a single sampling pass. This extra profiling run is not used to report the main latency numbers; it is used only to analyze the relative composition of ToCa's runtime overhead.

As shown by the results in the main paper, ToCa reduces the estimated model FLOPs on PixArt-α, but the corresponding wall-clock acceleration is limited. This discrepancy mainly comes from two sources. First, ToCa performs token-level reuse rather than whole-module reuse. Even under aggressive reuse, fresh tokens are still recomputed in the cross-attention and MLP modules, so a non-trivial portion of transformer computation remains. Second, ToCa introduces a dynamic token-management path, including token scoring, freshness reweighting, sorting, token gathering/scattering, and cache-index updates. These operations contribute modestly to model FLOPs, but can introduce noticeable wall-clock overhead due to indexing, memory movement, synchronization, and many small GPU kernels.

For this analysis, we further instrument ToCa-r90 on PixArt-α at 256×256 resolution with 20 sampling steps using CUDA events. This profiling is used only to analyze the relative latency composition. Because module-level profiling requires additional CUDA events and synchronization points, the profiled absolute latency is not intended to match the end-to-end latency reported in the main paper.

Figure 8 shows the measured latency composition. In this profiling run, ToCa-r90 takes 1474.4 ms per batch in total. The dynamic token-selection path accounts for 420.1 ms, including token selection before cross-attention cache computation and before MLP cache computation:

$$\texttt{cross_token_select} = 216.2 \text{ ms},$$
$$\texttt{mlp_token_select} = 203.8 \text{ ms}.$$

Together, these two selection stages contribute 28.5% of the total profiled latency. Inside this dynamic path, the main measured components include score evaluation (146.7 ms, 9.9%), spatial bonus computation (71.2 ms, 4.8%), sorting (41.6 ms, 2.8%), and cache-index maintenance (39.3 ms, 2.7%). These costs are not well captured by model-FLOPs counting, since they are dominated by memory access, indexing, synchronization, and small-kernel overheads rather than dense matrix multiplication.

The dynamic selection path, however, is not the only reason for the limited latency speedup. As shown in Figure 8, fresh-token recomputation in cached modules also takes a substantial fraction of runtime: 180.2 ms (12.2%) for cross-attention and 68.1 ms (4.6%) for MLP. Therefore, even if the measured selection overhead were removed, the ToCa-r90 latency would decrease from 1474.4 ms to about 1054.3 ms, which is still not close to the acceleration suggested by FLOPs reduction

alone. This is because ToCa still performs partial token computation in attention and MLP modules, whereas static cache methods such as FORA reuse whole module outputs on cache steps.

Overall, the latency gap of ToCa on PixArt-α comes from the combination of retained fresh-token computation and dynamic token-management overhead. This explains why ToCa has wall-clock latency close to the native PixArt-α baseline reported in the main paper, despite reducing the estimated FLOPs. TCC further applies a calibration operator to cached representations. Although this adds less than 10% extra FLOPs over the ToCa-r90 setting, the base ToCa latency is already close to the native baseline. Therefore, ToCa+TCC can become slower than native PixArt-α in wall-clock time. The ToCa+TCC result on PixArt-α should thus be interpreted as a quality-recovery setting under aggressive token-level reuse, with an explicit latency trade-off.

C.9 Compute resources

Run counts. For generation-quality metrics, one algorithm run denotes one complete sampling pass and its subsequent metric evaluation for a fixed method, setting, and random seed. Each entry in the multi-seed study represents one such run per seed, as detailed in Appendix C.1. Other quality tables report one run for each displayed method–setting point unless the text explicitly states a different aggregation. Latency uses a separate protocol: one warm-up run followed by five timed runs per method, as described in Appendix C.8.

The reported experiments were conducted on NVIDIA H800 GPUs with 80GB memory. We report approximate compute in H800 GPU-hours, computed as wall-clock time multiplied by the number of GPUs. For the 256×256 settings, DiT-XL/2 ImageNet 50K-sample 20-step TCC runs take about 1–2 GPU-hours per method setting, and the corresponding offline prior estimation takes about 2 GPU-hours. For PixArt-α, MS-COCO-30K sampling runs take about 1–2 GPU-hours per method setting, and prior-estimation runs, including prompt-count ablations up to 5K prompts, take about 1 GPU-hour per operator pack.

For storage overhead, in the DiT-XL/2 256×256 20-step FORA($N = 2$) setting, the stored calibration operators for one calibrated cache timestep occupy about 0.3GB. This size is determined mainly by the DiT backbone and the selected calibration sites, rather than by the particular base cache policy. For example, four calibrated cache timesteps within the 19–12 apply window require about 1.2GB of stored calibration operators in the DiT-XL/2 256×256 20-step FORA($N = 2$) setting. This operator pack is constructed with 5 H800 GPUs in parallel.

D Limitations and Broader Impact

TCC requires an offline prior-estimation stage and additional calibration operators during cached inference, which introduce extra FLOPs and can reduce wall-clock speedup when the underlying cache method already incurs dynamic token-selection or memory-movement overhead; a detailed overhead analysis is provided in Appendix C.8. Calibration strength and sites are chosen empirically per base cache strategy, reflecting strategy-dependent error magnitudes and granularities. For the main 20-step configurations, Appendix C.1 reports aggregate variability over three seeds across the evaluated settings. The additional 50-step study and several ablations remain single-run point estimates, so their small differences should be interpreted cautiously. Improvements also do not transfer uniformly to every metric; CLIP score or sFID can be less favorable in some settings. TCC reduces dense computation only in configurations that retain an end-to-end speedup; energy consumption was not measured. Lower-cost image generation may also facilitate misuse for deceptive or harmful synthetic content.



Figure 6: Additional qualitative comparisons on PixArt-α under cache-accelerated text-to-image generation. All comparisons use matched prompts, random seeds, and sampling settings. The examples illustrate local visual differences while preserving the overall composition.

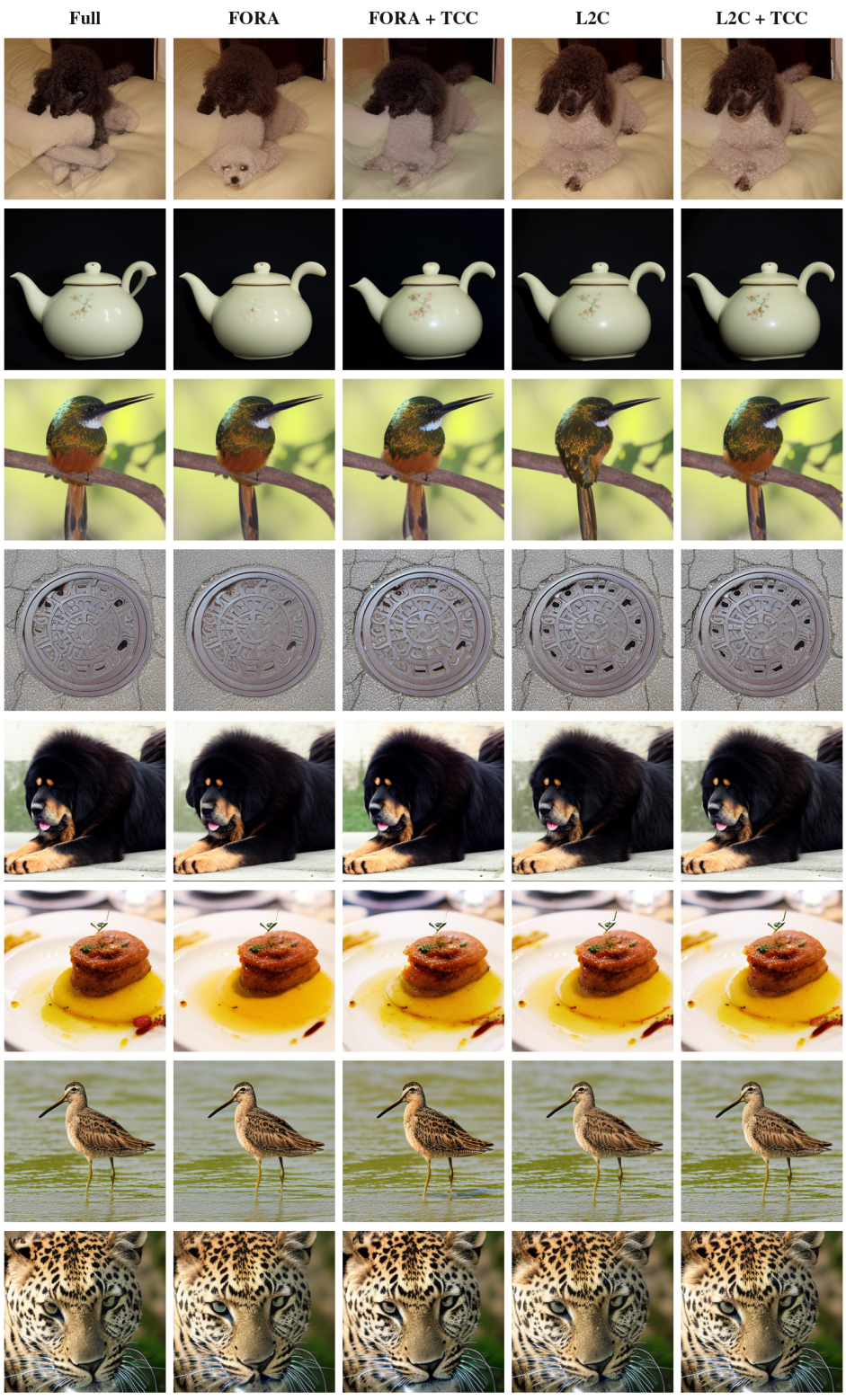


Figure 7: Additional qualitative comparisons on DiT-XL/2 for ImageNet class-conditional generation at 256×256 resolution with 20-step DDIM sampling and classifier-free guidance scale 4.0. All comparisons use matched class labels, random seeds, and sampling settings.

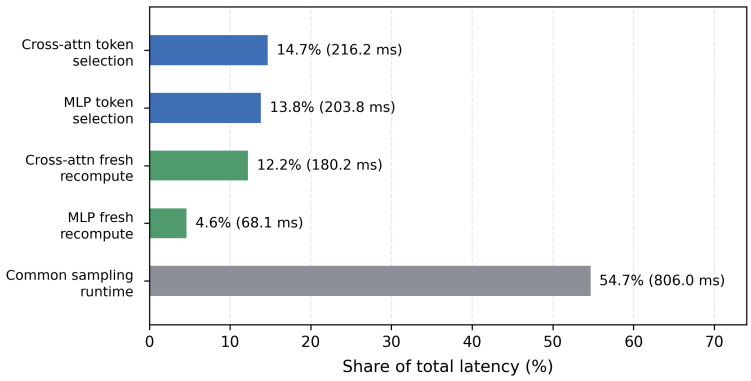


Figure 8: Latency breakdown of ToCa-r90 on PixArt-α at 256×256 resolution with 20 sampling steps. Blue bars denote dynamic token-selection overhead, green bars denote fresh-token recomputation in cached modules, and the gray bar denotes common sampling runtime, including regular forward computation, residual normalization, cache reuse operations, and other non-ToCa-specific sampling costs. Percentages are normalized by the total profiled latency.