CACHEFLOW: UNIFYING STEP DISTILLATION AND FEATURE CACHING FOR FEW-STEP GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Step distillation and feature caching accelerate diffusion models by reducing sampling steps and per-step computation, respectively. Under extreme acceleration, distillation struggles to approximate teacher trajectories, sacrificing semantic fidelity and fine details; training-free feature caching degrades on the resulting sparse trajectories, amplifying error propagation when combined with distillation. We introduce *CacheFlow*, which addresses extreme few-step acceleration from a frequency-aware perspective. We identify asynchronous frequency evolution as the primary source of quality degradation: low-frequency semantics stabilize rapidly, while high-frequency details suffer trajectory collapse and cross-step distortion. We therefore propose frequency-aware policy distillation to provide adaptive multi-band supervision for faithful trajectory approximation and detail preservation. A lightweight, learnable cache policy captures rapid cross-step feature dynamics, overcoming rigid extrapolation in sparse regimes. Jointly optimizing the few-step student and cache policy suppresses caching errors. Experiments on FLUX.1-dev, Qwen-Image, Qwen-Image-Edit, and Qwen-Image-2.1 demonstrate competitive visual quality across text-to-image generation and image editing, with a measured speedup of up to 55.31×. Code will be released.

1 INTRODUCTION

Recent years have witnessed diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Rombach et al., 2022) emerging as the foundational paradigm for both visual generation and video synthesis. Along the path of scaling up Diffusion Transformers (DiTs) (Peebles & Xie, 2023; Black Forest Labs, 2024; Wu et al., 2025), the visual fidelity and prompt-following capabilities of DiTs have made remarkable progress. However, their iterative denoising nature incurs substantial computational overhead, posing a severe bottleneck for interactive and latency-sensitive deployments. To improve inference efficiency, existing acceleration efforts primarily branch into two orthogonal directions: the training-based step distillation (Song et al., 2023; Yin et al., 2024b; Chen et al., 2026) to minimize the number of function evaluations (NFEs), and the training-free feature caching (Ma et al., 2024; Liu et al., 2025a;b) to reduce the computational cost per evaluation. Both acceleration paradigms have achieved notable progress when deployed independently on current DiT models.

Despite their individual merits, pushing either paradigm toward extreme acceleration reveals fundamental boundaries. Existing step-distillation methods (Salimans & Ho, 2022; Meng et al., 2023; Song et al., 2023; Yin et al., 2024b;a) train a student to take long-stride shortcuts across trajectories. However, because these multi-step shortcuts lack direct ground-truth velocity from the teacher, traditional approaches rely on complex consistency (Song et al., 2023; Luo et al., 2023; Kim et al., 2024a), distribution-matching (Yin et al., 2024b;a), or adversarial (Cheng et al., 2026) objectives, which often suffer from gradient instability and mode collapse (Chen et al., 2026). While recent policy-based distillation methods (Chen et al., 2026; Yang et al., 2026) mitigate this by modeling dynamic policies over large time intervals, they typically enforce a monolithic temporal parameterization over the entire velocity field. Such a uniform formulation fails to accommodate the non-uniform dynamics across distinct denoising stages, leading to severe detail loss and fidelity degradation when compressed to extreme few-step (*e.g.*, 1–4 step) regimes.

Feature caching, on the other hand, skips redundant computation by reusing or forecasting representations across adjacent steps (Li et al., 2024; Ma et al., 2024; Selvaraju et al., 2024; Chen et al.,

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.

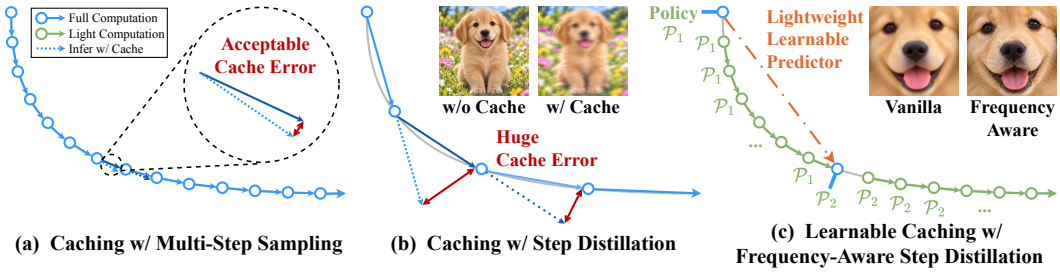

Figure 1: **Feature caching under multi-step and distilled sampling.** (a) Dense trajectories enable accurate prediction from nearby features. (b) Step distillation sparsifies the trajectory, producing substantial errors when conventional caching is directly applied. (c) CacheFlow jointly trains frequency-aware distillation and a lightweight cache predictor for reliable few-step generation.

2024). While early works directly copy historical features (Ma et al., 2024), recent schemes turn to predicting features through linear extrapolation (Liu et al., 2025a), Taylor expansions (Liu et al., 2025b), or higher-order polynomial forecasting (Zheng et al., 2026) to account for temporal drift. These methods fundamentally presuppose fine-grained temporal continuity across dense sampling trajectories. However, under aggressive caching ratios or large temporal intervals, fixed heuristic extrapolation rules break down against the complex, highly non-linear evolution of high-dimensional diffusion states (Zou et al., 2026a). As a result, the accumulated caching errors eventually manifest as severe visual artifacts and semantic degradation.

An intuitive approach to breaching the speed ceiling is to synthesize step distillation with feature caching in a learnable manner. Yet, directly compounding them leads to catastrophic failure. As illustrated in Fig. 1(a), dense trajectories preserve the temporal smoothness that validates feature extrapolation. In contrast, aggressive step distillation drastically sparsifies sampling time steps along the trajectory (Fig. 1(b)), which could cause sharp cross-step feature differences to surge exponentially. Under such severe inter-step disparity, heuristic extrapolation completely collapses. Although some post-hoc methods append learnable cache predictors to pre-distilled models (Zou et al., 2026a; Li et al., 2026), their decoupled training cannot be easily adapted to the student's intrinsic trajectory shifts, necessitating intricate adversarial losses and still yielding marginal generation quality.

To resolve these issues, we propose **CacheFlow**, a unified joint training framework that harmonizes step distillation and feature caching through a frequency-aware perspective (Fig. 1(c)). We fundamentally re-examine trajectory dynamics in the spectral domain and uncover a critical insight: **quality collapse under aggressive acceleration stems from asynchronous frequency evolution.** Specifically, *low-frequency global structures stabilize rapidly in early steps, presenting immense temporal redundancy; conversely, high-frequency textures and details exhibit dramatic, non-linear cross-step fluctuations that monolithic policies and heuristic extrapolation fail to capture.* Grounded in this insight, CacheFlow addresses both distillation and caching through three coordinated designs:

- **Frequency-Aware Policy Distillation:** Rather than enforcing a monolithic temporal parameterization, the student predicts a multi-band generative policy at each step. By decoupling the temporal evolution across distinct frequency bands, it explicitly preserves high-frequency details while stabilizing global low-frequency semantics, providing high-fidelity teacher-trajectory imitation even under sparse sampling.
- **Learnable Cross-Step Cache Predictor:** To replace rigid handcrafted extrapolation rules, we use a lightweight, data-driven MLP predictor. Operating directly within the distilled state space, it accurately captures the non-linear cross-step feature dynamics of skipped transformer blocks.
- **End-to-End Joint Optimization:** Crucially, we train the frequency-aware student policy and the cache predictor jointly under unified teacher-trajectory supervision. This collaborative optimization allows the student to actively adapt to cached feature approximations, reducing error accumulation associated with conventional two-stage or training-free pipelines.

We conduct comprehensive evaluations across text-to-image generation and image editing benchmarks, including FLUX.1-dev, Qwen-Image, Qwen-Image-Edit, and Qwen-Image-2.1. CacheFlow achieves measured inference speedups of 29.20×, 55.31×, 21.67×, and 18.54× on these four models, respectively, while preserving visual fidelity comparable to the original multi-step teacher models and establishing a superior quality–efficiency Pareto frontier over existing methods.

In summary, our key contributions are as follows:

- **A Unified Framework for Distillation and Caching.** We propose the first end-to-end framework that jointly optimizes policy-based step distillation and learnable feature caching along the same trajectory, reducing both the number of sampling steps and the per-step evaluation cost.
- **Frequency-Aware Policy Distillation.** We identify the asynchronous evolution across frequency bands under extreme step compression and propose a multi-band policy distillation strategy to imitate teacher trajectories and alleviate high-frequency collapse.
- **Learnable Feature Caching with Error Co-adaptation.** We replace heuristic extrapolation with a lightweight learnable cache predictor and jointly train it with the student, allowing the model to dynamically compensate for cache errors without relying on auxiliary adversarial objectives.
- **Advanced Acceleration Performance.** Across visual generation and editing tasks on multiple large-scale DiTs, CacheFlow achieves a measured speedup of up to 55.31× while maintaining visual quality comparable to the original multi-step teacher models.

2 Related Works

Sampling timestep reduction directly reduces the number of sampling steps while preserving generation quality. Salimans & Ho (2022) and Meng et al. (2023) progressively compress teacher trajectories, while Consistency Models (Song et al., 2023), LCM (Luo et al., 2023), and CTM (Kim et al., 2024a) learn consistent mappings between trajectory states. Recently, DMD (Yin et al., 2024b), SenseFlow (Ge et al., 2026), and TwinFlow (Cheng et al., 2026) further employ distribution-matching or adversarial objectives for training. In contrast, π-Flow (Chen et al., 2026) predicts a generative policy at each step initially to imitate the teacher velocity, while ArcFlow (Yang et al., 2026) adds a momentum term to improve prediction. Despite stable trajectory supervision, their monolithic parameterization struggles to capture non-uniform dynamics across denoising stages.

Model compression reduces computation through knowledge distillation (Li et al., 2023), structured pruning (Fang et al., 2023; Zhu et al., 2024), quantization (Shang et al., 2023; Kim et al., 2025), and token reduction (Bolya & Hoffman, 2023; Kim et al., 2024b; Zhang et al., 2024). However, these methods require additional training and can degrade severely under overly aggressive compression.

Feature caching exploits temporal redundancy between adjacent denoising steps to reuse intermediate features. FasterDiffusion (Li et al., 2024) and DeepCache (Ma et al., 2024) apply caching to U-Nets; FORA (Selvaraju et al., 2024) and Δ-DiT (Chen et al., 2024) extend it to DiTs. ToCa (Zou et al., 2025), DuCa (Zou et al., 2026b), and TeaCache (Liu et al., 2025a) further adopt adaptive caching strategies. To further improve accuracy, TaylorSeer (Liu et al., 2025b) extrapolates features through Taylor expansion, while FoCa (Zheng et al., 2026) models feature evolution as an ODE. Nevertheless, predefined rules and functions can accumulate errors as the caching interval increases.

3 CacheFlow

3.1 Preliminaries

Flow Matching. Given a data sample $\mathbf{x}_0 \sim p_{\text{data}}$, Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$, and condition $\mathbf{c}$, Flow Matching (Lipman et al., 2023) constructs a continuous transformation from the data distribution to the noise distribution through the probability path $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon}$ and learns a conditional velocity field satisfying $\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. The model parameters are optimized with $\mathcal{L}_{\text{FM}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon},t}\|\mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - (\boldsymbol{\epsilon} - \mathbf{x}_0)\|_2^2$. At inference time, $\mathbf{x}_0$ is obtained by solving the ODE backward from $t = 1$ to $t = 0$. However, high-accuracy ODE solving typically requires repeated evaluations of the velocity network at multiple timesteps, resulting in substantial inference overhead.

Step Distillation. Given timesteps $1 = t_0 > t_1 > \cdots > t_J = 0$, at the j-th step, the student learns to directly approximate the teacher transition $\mathbf{x}_{t_{j+1}} = \mathbf{x}_{t_j} + \int_{t_j}^{t_{j+1}} \mathbf{v}_T(\mathbf{x}_t, t)\,\mathrm{d}t$ from $\mathbf{x}_{t_j}$. In policy-based distillation (Chen et al., 2026), the student $G_{\boldsymbol{\theta}}$ performs one forward pass at the beginning of each step and produces a policy $\mathcal{P}_j$. The velocity function defined by this policy advances the trajectory through multiple low-cost substeps and directly imitates the teacher velocity, simplifying

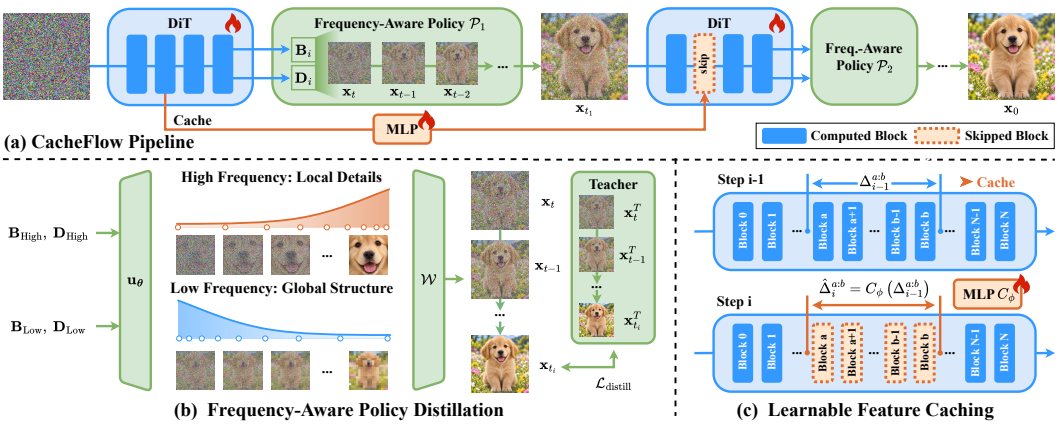

Figure 2: **Overview of CacheFlow. (a) CacheFlow Pipeline:** The DiT backbone predicts a frequency-aware policy to advance each denoising interval through low-cost substeps, while cached features allow selected blocks to be skipped. **(b) Frequency-Aware Policy Distillation:** Frequency bands are modeled independently to capture their temporal dynamics and accurately match the teacher's trajectory. **(c) Learnable Feature Caching:** A lightweight MLP predicts current-step feature changes from the preceding-step cache and is jointly optimized with the student.

the complex optimization objectives required by distribution-matching, consistency, or adversarial distillation and improving training stability. However, existing policies typically model the interval-wise velocity as a whole in the original latent space, making it difficult to precisely characterize the temporal variations of different components along the teacher trajectory.

Feature Caching. Let a DiT consist of L consecutive Transformer blocks $\{g_l\}_{l=1}^{L}$, and let $\mathbf{h}_i^l$ denote the input feature to the l-th block at the i-th denoising step. Naive caching (Ma et al., 2024; Selvaraju et al., 2024; Chen et al., 2024) directly reuses a feature computed at a previous step, i.e., $\widehat{\mathbf{h}}_i^{l+1} = \mathbf{h}_{i-1}^{l+1} = g_l(\mathbf{h}_{i-1}^l)$, thereby bypassing the evaluation of g_l. Prediction-based caching (Liu et al., 2025a;b; Zheng et al., 2026) instead estimates $g_l(\mathbf{h}_i^l)$ from historical features using prediction functions such as linear extrapolation, Taylor expansion, or higher-order polynomials. These methods exploit feature continuity between adjacent denoising steps to reduce the cost of each network evaluation, but their performance depends heavily on handcrafted prediction functions. At high acceleration ratios, errors accumulate rapidly and substantially degrade generation quality.

3.2 Frequency-Aware Policy Distillation

Images exhibit a natural frequency hierarchy: low-frequency components primarily encode global layout, contours, and color distribution, whereas high-frequency components describe edges, textures, and local details (Ren et al., 2026; Pavasovic et al., 2026). However, how this frequency structure manifests in the denoising dynamics of Flow Matching remains underexplored. We therefore decompose the denoising trajectories of Flow Matching models and analyze how each frequency-band component evolves over denoising time. As shown in Fig. 3, low-frequency components change rapidly during the early denoising stage to establish global structure, whereas high-frequency components change more rapidly during the later stage to refine local textures and details, revealing substantially different temporal dynamics across frequency bands.

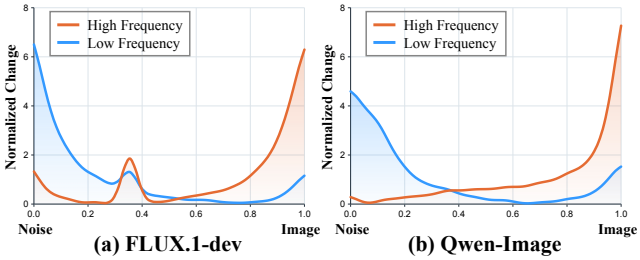

Figure 3: **Frequency dynamics of denoising trajectories.** Visualized using Haar DWT.

Building on this observation, we introduce a frequency-aware velocity modeling strategy. For an arbitrary denoising step from t_s to t_e, the student performs a single network evaluation at the starting

state $\mathbf{x}_{t_s}$ and produces a frequency policy $\mathcal{P} = G_{\boldsymbol{\theta}}(\mathbf{x}_{t_s}, t_s)$, which dynamically represents the complete velocity evolution over the corresponding interval. The policy uses $\mathbf{u}_{\boldsymbol{\theta},q}(t)$ to denote the q-th frequency-band component of the predicted velocity, models the temporal dynamics of each band separately, and reconstructs the complete velocity through the inverse frequency transform $\mathcal{W}$:

$$\mathbf{v}_{\boldsymbol{\theta}}\left(t \mid \mathbf{x}_{t_s}, t_s\right) = \mathcal{W}\left\{\mathbf{u}_{\boldsymbol{\theta},q}(t)\right\}_{q=1}^{Q}. \tag{1}$$

We implement $\mathcal{W}$ using the Haar discrete wavelet transform (DWT) (Pavasovic et al., 2026), which decomposes the velocity into four complementary frequency-band components $\mathcal{Q} = \{LL, LH, HL, HH\}$. The orthogonality and invertibility of the Haar DWT preserve all information in the spatial velocity and allow it to be reconstructed losslessly from the frequency-band components. For each band, we model its temporal variation as $\mathbf{u}_{\boldsymbol{\theta},q}(t) = \mathbf{B}_q + (t - t_s)\mathbf{D}_q$, where $\mathbf{B}_q$ and $\mathbf{D}_q$ denote the baseline spectral response and evolution direction of the corresponding frequency-band component, respectively. Consequently, each frequency-band component can independently adapt to its distinct dynamics throughout denoising.

At each denoising step, the frequency-aware policy predicts the dynamic parameters of individual frequency bands and rolls out the complete trajectory over the corresponding interval through low-cost ODE integration substeps, without reevaluating the student network. We therefore directly match the student-predicted velocity to the frozen teacher velocity, enabling the student to accurately approximate the teacher denoising trajectory. Specifically, let $t_s = t_0 > \cdots > t_{M+1} = t_e$. For each subinterval $[t_{i+1}, t_i]$, we define $t_{\mathrm{m}} = \lambda t_i + (1 - \lambda) t_{i+1}$ and obtain the corresponding state by sequentially integrating the teacher and student velocities:

$$\mathbf{x}_{t_{i+1}} = \mathbf{x}_{t_i} + \int_{t_i}^{t_{\mathrm{m}}} \mathbf{v}_T(\mathbf{x}_t, t)\,\mathrm{d}t + \int_{t_{\mathrm{m}}}^{t_{i+1}} \mathbf{v}_{\boldsymbol{\theta}}\left(t \mid \mathbf{x}_{t_s}, t_s\right)\,\mathrm{d}t. \tag{2}$$

Here, λ gradually increases from 0 to 1 during training. Accordingly, intermediate states are constructed primarily along the teacher trajectory at the beginning of training and progressively transition to the student's own trajectory, allowing the student to correct state deviations encountered during inference. Given an intermediate state $\mathbf{x}_t$, we query the instantaneous velocity of the frozen teacher and directly minimize the discrepancy between the student policy and the teacher velocity:

$$\mathcal{L}_{\text{distill}} = \mathbb{E}_{t,\mathbf{x}_t}\left[\left\|\mathbf{v}_{\boldsymbol{\theta}}\left(t \mid \mathbf{x}_{t_s}, t_s\right) - \mathbf{v}_T(\mathbf{x}_t, t)\right\|_2^2\right]. \tag{3}$$

This objective continuously supervises the student at multiple intermediate states along its own trajectory, allowing it to learn the velocity evolution over the entire denoising interval.

3.3 Learnable Feature Caching

Existing distillation methods typically have a finite compression limit, and further reducing the number of sampling steps often leads to severe quality degradation. An intuitive idea is to further reduce the per-step computation through feature caching. However, step distillation substantially enlarges the sampling interval, causing the handcrafted prediction functions used by existing caching methods to produce huge cross-step errors. To address it, we introduce a lightweight, learnable cache predictor that learns the actual cross-step feature changes after distillation in a data-driven manner.

Specifically, let $g_{a:b} = g_b \circ \cdots \circ g_a$ denote a selected segment of consecutive Transformer blocks, which maps the input feature $\mathbf{h}_i^a$ at the i-th step to the output feature $\mathbf{h}_i^{b+1} = g_{a:b}(\mathbf{h}_i^a)$. We partition the sampling steps into non-overlapping adjacent pairs. For any pair $(i-1, i)$, the segment is fully evaluated at the preceding step and its feature change $\Delta_{i-1}^{a:b}$ is cached. At the subsequent step, a lightweight predictor $C_{\boldsymbol{\phi}}$ estimates the current incremental change $\widehat{\Delta}_i^{a:b}$, allowing the segment output to be recovered while bypassing the corresponding blocks:

$$\Delta_{i-1}^{a:b} = \mathbf{h}_{i-1}^{b+1} - \mathbf{h}_{i-1}^{a}, \qquad \widehat{\Delta}_i^{a:b} = C_{\boldsymbol{\phi}}\big(\Delta_{i-1}^{a:b}\big), \qquad \widehat{\mathbf{h}}_i^{b+1} = \mathbf{h}_i^{a} + \widehat{\Delta}_i^{a:b}. \tag{4}$$

Compared with directly reusing historical features or employing handcrafted prediction functions, the data-driven cache predictor better captures cross-step feature changes and recovers the current-step output accurately. We implement $C_{\boldsymbol{\phi}}$ as a lightweight residual MLP. Empirically, the cache predictor introduces less than 0.4% of the parameters of the original diffusion model and less than 0.1% additional inference latency, while effectively recovering the features of the skipped blocks.

3.4 Joint Training of Distillation and Caching

Within each adjacent pair, the preceding step fully evaluates the network and caches its features, while the subsequent step uses the cache predictor to bypass the selected blocks. Each denoising step uses the frequency policy predicted at its starting point to extrapolate the complete velocity over the corresponding interval, while the velocity-matching objective jointly optimizes the student model and cache predictor. Because the predicted cached features directly contribute to the student velocity, teacher supervision continuously corrects caching errors. Meanwhile, the student model progressively learns intermediate representations that are more amenable to cross-step prediction and reuse. This co-adaptation continually reduces trajectory-matching deviations and caching errors without introducing additional complex reconstruction or adversarial objectives, delivering stable speedups while preserving generation quality.

4 Experiments

4.1 Text-to-Image Generation

Setups. We conduct experiments on FLUX.1-dev (Black Forest Labs, 2024), Qwen-Image (Wu et al., 2025), and Qwen-Image-2.1 (Qwen Team, 2026). To clearly compare performance, we define the effective NFE as the ratio of the number of Transformer blocks actually evaluated during student sampling to the total number of blocks in the original model. For example, FLUX.1-dev contains 57 Transformer blocks, and skipping blocks 6–19 or 6–29 at the second step yields 1.75 or 1.58 NFE, respectively. We also report variants without caching for direct comparison with existing methods. Following prior work, we evaluate with GenEval (Ghosh et al., 2023), DPG-Bench (Hu et al., 2024), and OneIG-Bench (Chang et al., 2025). See the appendix for more details.

Quantitative results. As shown in Tab. 1 and 2, CacheFlow clearly outperforms existing acceleration methods. Among step-distillation methods, SenseFlow and TwinFlow incur drops of 9.1%/5.1% and 5.7%/1.5% on GenEval/DPG-Bench, while ArcFlow drops by 3.5% on OneIG-Bench. Meanwhile, all feature-caching methods at 6 NFE already incur drops of 22.2%/12.4%/29.2%. In contrast, 2-NFE CacheFlow achieves 0.66/84.72, 0.86/88.58, and 0.51/83.76 on the three backbones at speedups of 22.89×/45.35×/14.47×, respectively. With learnable caching, CacheFlow-Medium reduces the NFE to 1.75 with speedups of 26.34×/51.61× and less than 0.5% quality degradation on average across GenEval/DPG-Bench, while the more aggressive CacheFlow-Fast limits the degradation to below 1%, relative to the no-cache variants. These results show that the frequency-aware policy more accurately approximates the teacher trajectory, while the jointly trained cache predictor effectively controls cross-step errors.

Qualitative results. As shown in Fig. 4, feature-caching methods exhibit evident blurring and artifacts, while few-step distillation methods suffer from distorted text (1st row), errors in complex relationships (2nd row), malformed objects (3rd row), or style shifts (4th row). Notably, even the original multi-step model sometimes fails to accurately follow the prompt (the extra knife in row 2, column 1). In contrast, CacheFlow preserves clear structures and accurate interactions even at a higher acceleration ratio. Fig. 5 further shows generated details. Compared with the competitive ArcFlow, CacheFlow presents clearer contours (1st row) and finer textures (2nd row), further demonstrating that frequency-aware policy distillation more effectively recovers high-frequency image details.

4.2 Image Editing

Setups. We conduct experiments on Qwen-Image-Edit (Qwen Team, 2025) and Qwen-Image-2.1 (Qwen Team, 2026) to validate our broad applicability. Following prior work, we evaluate with GEdit (Liu et al., 2025c) and ImgEdit-Bench (Ye et al., 2025). See the appendix for more details.

Quantitative results. As shown in Tab. 3, CacheFlow is competitive with existing step-distillation methods, while 4-NFE TwinFlow and Qwen-Image-Edit-Lightning incur drops of 4.1%/3.8% and 0.9%/1.2% on GEdit/ImgEdit-Bench. And for feature-caching methods, even with 8×2 NFE, the generation quality of FORA, TaylorSeer, and FoCa decreases by 67.6%/42.7%. In contrast, our 4-NFE no-cache variant and 3.27-NFE CacheFlow-Fast achieve 6.77/4.20 and 6.76/4.18 at speedups of 17.69×/21.67× on Qwen-Image-Edit, while maintaining competitive editing capability on Qwen-Image-2.1 at speedups of 10.94×/13.16×.

Table 1: **Quantitative comparisons for FLUX.1-dev** on GenEval, DPG-Bench, and OneIG-Bench. *Latency(s)* and *Speed* measure DiT-path latency and speedup.

Model	Type	Acceleration			GenEval↑	DPG-Bench↑	OneIG-Bench				
		NFE↓	Latency(s)↓	Speed↑			Alignment↑	Text↑	Diversity↑	Style↑	Reasoning↑
FLUX.1-dev (Black Forest Labs, 2024)		50	9.815	1.00×	0.66	84.16	0.790	0.556	0.238	0.307	0.257
FORA (Selvaraju et al., 2024)	Feature Caching	6	1.930	5.09×	0.56	77.97	0.727	0.112	0.147	0.249	0.188
TeaCache (Liu et al., 2025a)	Feature Caching	6	1.310	7.49×	0.58	78.58	0.724	0.158	0.143	0.222	0.219
TaylorSeer (Liu et al., 2025b)	Feature Caching	6	2.252	4.36×	0.42	78.73	0.675	0.188	0.208	0.194	0.205
FoCa (Zheng et al., 2026)	Feature Caching	6	1.931	5.08×	0.60	81.56	0.752	0.175	0.141	0.250	0.215
SenseFlow (Ge et al., 2026)	Step Distillation	2	0.435	22.56×	0.60	79.86	0.743	0.230	0.139	0.341	0.212
π-Flow (Chen et al., 2026)	Step Distillation	2	0.424	23.15×	0.58	82.36	0.764	0.141	0.216	0.332	0.212
ArcFlow (Yang et al., 2026)	Step Distillation	2	0.427	22.99×	0.65	84.29	0.798	0.368	0.210	0.350	0.224
CacheFlow (w/o Cache)		2	0.429	22.89×	**0.66**	**84.72**	**0.816**	**0.395**	**0.241**	**0.372**	**0.241**
CacheFlow-Medium		1.75	0.373	26.34×	**0.66**	84.36	0.803	0.357	0.238	0.362	0.230
CacheFlow-Fast		1.58	**0.336**	**29.20**×	0.65	84.08	0.792	0.344	0.229	0.356	0.217

Table 2: **Quantitative comparisons for Qwen-Image and Qwen-Image-2.1** on GenEval, DPG-Bench, and OneIG-Bench. *Latency(s)* and *Speed* measure DiT-path latency and speedup.

Model	Type	Acceleration			GenEval↑	DPG-Bench↑	OneIG-Bench				
		NFE↓	Latency(s)↓	Speed↑			Alignment↑	Text↑	Diversity↑	Style↑	Reasoning↑
Qwen-Image (Wu et al., 2025)		50×2	18.437	1.00×	0.87	88.32	0.880	0.888	0.194	0.427	0.306
FORA (Selvaraju et al., 2024)	Feature Caching	6×2	4.141	4.45×	0.45	50.96	0.478	0.427	0.165	0.326	0.198
TeaCache (Liu et al., 2025a)	Feature Caching	6×2	2.328	7.92×	0.80	79.01	0.738	0.548	0.178	0.343	0.251
TaylorSeer (Liu et al., 2025b)	Feature Caching	6×2	4.750	3.88×	0.63	77.38	0.700	0.382	0.182	0.301	0.204
FoCa (Zheng et al., 2026)	Feature Caching	6×2	4.208	4.38×	0.69	79.17	0.726	0.520	0.177	0.331	0.231
Lightning (Fan et al., 2025)	Step Distillation	2	0.453	40.70×	0.85	88.42	0.875	0.879	0.098	0.415	0.292
π-Flow (Chen et al., 2026)	Step Distillation	2	0.397	46.44×	0.83	86.45	0.837	0.634	0.176	0.382	0.259
TwinFlow (Cheng et al., 2026)	Step Distillation	2	0.482	38.25×	0.82	87.01	0.862	0.825	0.130	0.364	0.267
ArcFlow (Yang et al., 2026)	Step Distillation	2	0.401	45.98×	0.85	88.46	0.877	0.853	0.182	0.421	0.289
CacheFlow (w/o Cache)		2	0.407	45.35×	**0.86**	**88.58**	**0.879**	**0.881**	**0.189**	**0.426**	**0.293**
CacheFlow-Medium		1.75	0.357	51.61×	**0.86**	88.47	0.877	0.870	0.183	0.425	0.289
CacheFlow-Fast		1.63	**0.333**	**55.31**×	0.85	88.40	0.867	0.862	0.177	0.423	0.278
Qwen-Image-2.1 (Qwen Team, 2026)		40	6.069	1.00×	0.51	84.20	0.840	0.963	0.250	0.372	0.247
CacheFlow (w/o Cache)		2	0.419	14.47×	0.51	83.76	0.825	0.912	0.298	0.347	0.234
CacheFlow-Fast		1.69	0.327	18.54×	0.50	83.61	0.802	0.884	0.291	0.332	0.219



Figure 4: **Visualization of acceleration methods on Qwen-Image.** Compared with other methods, CacheFlow better preserves visual quality and prompt fidelity.

Qualitative results. Fig. 6 shows that existing acceleration methods exhibit instruction-following failure (1st row), structural inconsistency (2nd row), unedited-region corruption (3rd row), and fine-detail degradation (4th row). In contrast, CacheFlow accurately performs the target edits while better preserving unedited regions, subject structure, and local details.

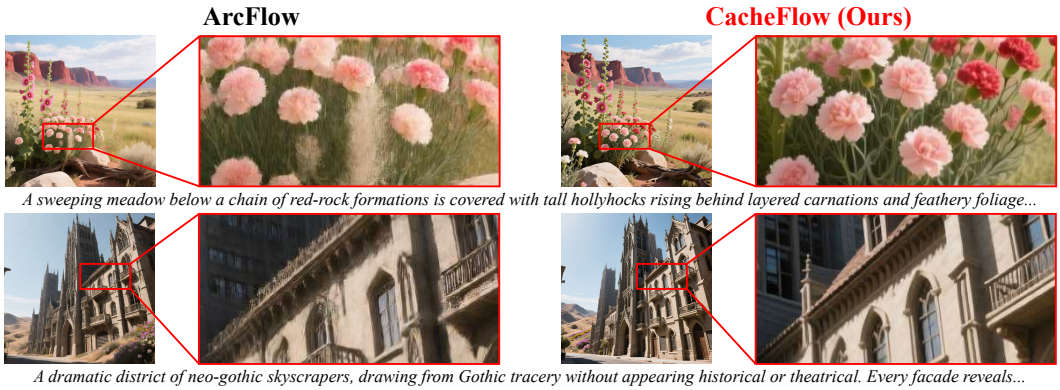

Figure 5: **Visual comparison with ArcFlow.** CacheFlow produces visibly finer details.

Table 3: **Quantitative comparisons for Qwen-Image-Edit and Qwen-Image-2.1** on GEdit and ImgEdit-Bench. *Latency(s)* and *Speed* measure DiT-path latency and speedup.

Model	Type	Acceleration			GEdit-CN (Full)			GEdit-EN (Full)			ImgEdit↑
		NFE↓	Latency(s)↓	Speed↑	G_SC↑	G_PQ↑	G_O↑	G_SC↑	G_PQ↑	G_O↑	
Qwen-Image-Edit (Qwen Team, 2025)		40×2	34.859	1.00×	7.44	7.03	6.87	7.39	6.86	6.71	4.22
FORA (Selvaraju et al., 2024)	Feature Caching	8×2	10.332	3.37×	2.34	4.85	2.38	2.28	4.91	2.35	2.41
TaylorSeer (Liu et al., 2025b)	Feature Caching	8×2	11.407	3.06×	1.72	4.67	1.85	1.90	4.34	1.90	1.98
FoCa (Zheng et al., 2026)	Feature Caching	8×2	10.416	3.35×	2.16	5.33	2.29	2.45	5.06	2.45	2.86
Lightning (Fan et al., 2025)	Step Distill.	4	1.899	18.36×	7.35	6.74	6.69	7.37	6.72	6.77	4.17
TwinFlow (Cheng et al., 2026)	Step Distill.	4	2.014	17.31×	6.37	6.72	6.51	6.33	6.73	6.51	4.06
CacheFlow (w/o Cache)		4	1.971	17.69×	**7.43**	**6.75**	**6.73**	**7.41**	**6.75**	**6.81**	**4.20**
CacheFlow-Fast		3.27	**1.609**	**21.67**×	7.41	6.71	6.72	7.40	6.70	6.79	4.18
Qwen-Image-2.1 (Qwen Team, 2026)		40	7.258	1.00×	7.97	7.85	7.31	8.80	8.16	8.21	4.18
CacheFlow (w/o Cache)		2	0.663	10.94×	7.90	7.53	7.23	8.72	7.96	8.08	4.14
CacheFlow-Fast		1.69	0.551	13.16×	7.82	7.41	7.16	8.65	7.84	7.93	4.12



Figure 6: **Visualization of acceleration methods on Qwen-Image-Edit.** CacheFlow better follows editing instructions while preserving reference content and local details.

4.3 Ablation Studies

Framework components. Tab. 4 validates each core design of CacheFlow, measuring feature prediction fidelity via Normalized Mean Squared Error (NMSE). Specifically: (1) Replacing the frequency-aware policy with a spatial-domain linear policy forces a monolithic temporal parameterization, dropping GenEval and DPG by 3.5% and 2.0%, respectively. (2) Replacing the learnable predictor with direct cached-residual reuse, while continuing student training, spikes cache error by 55.7% with noticeable visual degradation, proving that training-free caching cannot accommodate

Table 4: Component ablation of our proposed joint training framework.

Var.	Frequency Aware	Learnable Predictor	Joint Training	GenEval↑	DPG↑	Cache Error↓
(1)	✘	✔	✔	0.83	86.72	0.96
(2)	✔	✘	–	0.80	85.07	1.37
(3)	✔	✔	✘	0.85	88.04	0.93
Ours	✔	✔	✔	**0.86**	**88.47**	**0.88**

Table 5: Frequency transform ablation.

Transform ($\mathcal{W}$)	DPG↑
DCT	88.43
FFT	87.96
Laplacian	88.39
Haar DWT (Ours)	**88.47**

Table 6: Predictor architecture ablation.

Predictor (C_ϕ)	DPG↑
Blockwise MLP	87.68
Deep ResMLP	88.21
DiT Block	88.45
Simple MLP (Ours)	**88.47**

enlarged cross-step shifts. (3) Decoupling the pipeline into stagewise training hinders co-adaptation between distillation and caching, triggering a 5.7% surge in cache error and consistent metric drops (1.2% GenEval, 0.5% DPG). In contrast, our complete framework achieves the best generation fidelity and minimal cache error across all settings.

Frequency transforms. As Tab. 5 indicates, CacheFlow remains robust across diverse spectral decompositions. Haar DWT achieves the best performance owing to its orthogonality, invertibility, and compact spatial support, which together guarantee lossless reconstruction while cleanly disentangling structural semantics from high-frequency textures.

Predictor designs. As shown in Tab. 6, increasing network depth or adopting heavy DiT blocks yields diminishing returns. In contrast, a lightweight MLP achieves superior generation quality while introducing negligible overhead (<0.4% base parameters and 0.1% additional latency), striking an ideal balance between capacity and efficiency.

4.4 Performance across NFE Budgets

Fig. 7 compares CacheFlow with ArcFlow at different effective NFEs. On FLUX.1-dev, reducing NFE from 2 to 1.58 lowers our score by only 0.64 points (84.72 to 84.08); on Qwen-Image, reducing from 2 to 1.63 costs only 0.18 points (88.58 to 88.40). ArcFlow exhibits a substantially steeper decline on both models, demonstrating CacheFlow's superior quality–computation trade-off across the evaluated NFE budgets.

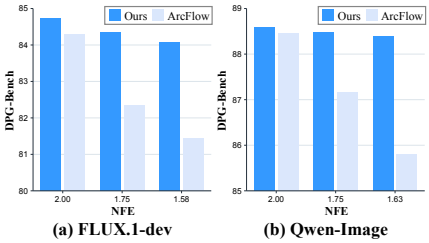

Figure 7: **NFE-budgeted Evaluation.**

4.5 Frequency-Band Analysis

To substantiate our frequency-aware policy, Fig. 8 visualizes the multi-band decomposition via Haar DWT. While isolated high-frequency sub-bands (LH, HL, HH) appear primarily residual, the low-frequency component (LL) governs the global semantic layout. Progressively reintegrating high frequencies systematically recovers object boundaries, text structures, and fine textures. This clear functional dichotomy confirms that different bands capture distinct levels of image synthesis, underscoring the need to decouple their disparate temporal dynamics during policy distillation.

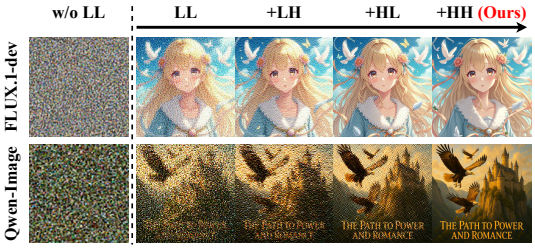

Figure 8: **Frequency-band analysis.** *Left*: high-frequency only. *Right*: starting with LL (low frequency), different bands are progressively added.

5 Conclusion

We present **CacheFlow**, a unified framework for extreme few-step acceleration. Motivated by asynchronous frequency evolution, it combines frequency-aware policy distillation with a lightweight cache predictor of non-linear cross-step feature dynamics. Joint teacher-trajectory supervision enables end-to-end co-adaptation, reducing distillation and caching errors. Evaluations on FLUX.1-dev, Qwen-Image, Qwen-Image-Edit, and Qwen-Image-2.1 demonstrate substantial acceleration while preserving competitive visual fidelity.

Reproducibility Statement

We provide the complete method formulation, training objectives, cache configurations, and evaluation protocol in the main paper and supplementary material. All implementation details and hyperparameters required to reproduce our experiments are documented, and the code and configurations will be released upon acceptance.

AI Use Statement

Generative AI tools were used only to improve the wording and readability of this paper. All research ideas, method design, implementation, experiments, analyses, and conclusions were developed and verified by the authors. The authors reviewed all AI-assisted edits and take full responsibility for the final content.

References

Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2024.

Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *IEEE CVPR Workshops*, 2023.

Jingjing Chang, Yixiao Fang, Peng Xing, Shuhan Wu, Wei Cheng, Rui Wang, Xianfang Zeng, Gang Yu, and Hai-Bao Chen. OneIG-Bench: Omni-dimensional nuanced evaluation for image generation. In *NeurIPS*, 2025.

Hansheng Chen, Kai Zhang, Hao Tan, Leonidas Guibas, Gordon Wetzstein, and Sai Bi. pi-Flow: Policy-based few-step generation via imitation distillation. In *ICLR*, 2026.

Junying Chen, Zhenyang Cai, Pengcheng Chen, Shunian Chen, Ke Ji, Xidong Wang, Yunjin Yang, and Benyou Wang. Sharegpt-4o-image: Aligning multimodal models with gpt-4o-level image generation, 2025. URL https://arxiv.org/abs/2506.18095.

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. *arXiv preprint arXiv:2406.01125*, 2024.

Zhenglin Cheng, Peng Sun, Jianguo Li, and Tao Lin. TwinFlow: Realizing one-step generation on large models with self-adversarial flows. In *ICLR*, 2026.

Xiangyu Fan, Zesong Qiu, Fanzhou Wang, Peng Gao, and Lei Yang. Qwen-Image-Lightning. https://github.com/ModelTC/LightX2V-Qwen-Image-Lightning, 2025.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. In *NeurIPS*, 2023.

Xingtong Ge, Xin Zhang, Tongda Xu, Yi Zhang, Xinjie Zhang, Yan Wang, and Jun Zhang. SenseFlow: Scaling distribution matching for flow-based text-to-image distillation. In *ICLR*, 2026.

Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. GenEval: An object-focused framework for evaluating text-to-image alignment. In *NeurIPS*, 2023.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. ELLA: Equip diffusion models with LLM for enhanced semantic alignment. *arXiv preprint arXiv:2403.05135*, 2024.

Dongjun Kim, Chieh-Hsin Lai, Wei-Hsiang Liao, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yutong He, Yuki Mitsufuji, and Stefano Ermon. Consistency trajectory models: Learning probability flow ODE trajectory of diffusion. In *ICLR*, 2024a.

Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *IEEE WACV*, 2024b.

Sungbin Kim, Hyunwuk Lee, Wonho Cho, Mincheol Park, and Won Woo Ro. Ditto: Accelerating diffusion model via temporal value similarity. In *IEEE HPCA*, 2025.

Haoyu Li, Tingyan Wen, Lin Qi, Zhe Wu, Yihuang Chen, Xing Zhou, Lifei Zhu, Xueqian Wang, and Kai Zhang. 1.x-Distill: Breaking the diversity, quality, and efficiency barrier in distribution matching distillation. *arXiv preprint arXiv:2604.04018*, 2026.

Senmao Li, Taihang Hu, Joost van de Weijer, Fahad Shahbaz Khan, Tao Liu, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of the encoder for diffusion model inference. In *NeurIPS*, 2024.

Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. SnapFusion: Text-to-image diffusion model on mobile devices within two seconds. In *NeurIPS*, 2023.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *ICLR*, 2023.

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. In *IEEE CVPR*, 2025a.

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

Shiyu Liu, Yucheng Han, Peng Xing, Fukun Yin, Rui Wang, Wei Cheng, Jiaqi Liao, Yingming Wang, Honghao Fu, Chunrui Han, et al. Step1X-Edit: A practical framework for general image editing. *arXiv preprint arXiv:2504.17761*, 2025c.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *IEEE CVPR*, 2024.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik P. Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *IEEE CVPR*, 2023.

Krunoslav Lehman Pavasovic, Théophane Vallaeys, Stéphane Mallat, Giulio Biroli, Luke Zettlemoyer, Brian Karrer, and Jakob Verbeek. WaiT for the signal: Simple frequency-aware flow-matching. *arXiv preprint arXiv:2607.28760*, 2026.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE ICCV*, 2023.

Qwen Team. Qwen-Image-Edit. https://github.com/QwenLM/Qwen-Image, 2025.

Qwen Team. Qwen-Image-2.1. https://github.com/QwenLM/Qwen-Image-2.1, 2026.

Sucheng Ren, Qihang Yu, Ju He, Xiaohui Shen, and Liang-Chieh Chen. Frequency-aware flow matching for high-quality image generation. In *IEEE CVPR*, 2026.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE CVPR*, 2022.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2022.

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.

Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *IEEE CVPR*, 2023.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *ICML*, 2023.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-Image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Zihan Yang, Shuyuan Tu, Licheng Zhang, Qi Dai, Yu-Gang Jiang, and Zuxuan Wu. ArcFlow: Unleashing 2-step text-to-image generation via high-precision non-linear flow distillation. *arXiv preprint arXiv:2602.09014*, 2026.

Yang Ye, Xianyi He, Zongjian Li, Bin Lin, Shenghai Yuan, Zhiyuan Yan, Bohan Hou, and Li Yuan. ImgEdit: A unified image editing dataset and benchmark. In *NeurIPS*, 2025.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T. Freeman. Improved distribution matching distillation for fast image synthesis. In *NeurIPS*, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Frédo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *IEEE CVPR*, 2024b.

Evelyn Zhang, Bang Xiao, Jiayi Tang, Qianli Ma, Chang Zou, Xuefei Ning, Xuming Hu, and Linfeng Zhang. Token pruning for caching better: 9 times acceleration on stable diffusion for free. *arXiv preprint arXiv:2501.00375*, 2024.

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. In *AAAI*, 2026.

Haowei Zhu, Dehua Tang, Ji Liu, Mingjie Lu, Jintu Zheng, Jinzhang Peng, Dong Li, Yu Wang, Fan Jiang, Lu Tian, Spandan Tiwari, Ashish Sirasao, Jun-Hai Yong, Bin Wang, and Emad Barsoum. DiP-GO: A diffusion pruner via few-step gradient optimization. In *NeurIPS*, 2024.

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

Chang Zou, Changlin Li, Songtao Liu, Zhao Zhong, Kailin Huang, and Linfeng Zhang. DisCa: Accelerating video diffusion transformers with distillation-compatible learnable feature caching. In *IEEE CVPR*, 2026a.

Chang Zou, Shikang Zheng, Evelyn Zhang, Runlin Guo, Haohang Xu, Zhengyi Shi, Conghui He, Xuming Hu, and Linfeng Zhang. Rethinking token-wise feature caching: Accelerating diffusion transformers with dual feature caching. *IEEE TIP*, 2026b.

A Tasks and Evaluation Protocols

A.1 Text-to-Image Generation

We conduct text-to-image experiments on FLUX.1-dev (Black Forest Labs, 2024), Qwen-Image (Wu et al., 2025), and Qwen-Image-2.1 (Qwen Team, 2026), generating images at a resolution of 1024×1024. For evaluation, GenEval (Ghosh et al., 2023) contains 553 compositional prompts spanning six tasks: single object, two objects, counting, colors, spatial relations, and color attribution. DPG-Bench (Hu et al., 2024) contains 1,065 long prompts and evaluates consistency in global scenes, entities, attributes, and relations. OneIG-Bench (Chang et al., 2025) uses 695, 200, 1,120, 114, and 225 prompts for five dimensions, respectively. Alignment evaluates prompt following for entities, attributes, and relations. Text combines edit distance, completeness, and word-level accuracy to assess text rendering. Diversity measures variations among repeated generations from the same prompt. Style measures similarity to reference-image styles, while Reasoning evaluates responses to knowledge- and reasoning-intensive prompts. All inference latencies are measured on an NVIDIA H200 GPU.

A.2 Image Editing

We conduct image-editing experiments on Qwen-Image-Edit-2509 (Qwen Team, 2025) and Qwen-Image-2.1 (Qwen Team, 2026). For evaluation, the Chinese and English subsets of GEdit (Liu et al., 2025c) each contain 606 prompts spanning 11 categories, including object editing, background and style transformation, appearance and pose modification, text editing, local-detail editing, and compositional editing. Semantic consistency G_{SC} measures instruction following, perceptual quality G_{PQ} evaluates clarity, realism, and artifacts, and the overall editing score G_{O} combines the two. ImgEdit-Bench (Ye et al., 2025) contains 737 prompts covering addition, removal, adjustment, replacement, style transfer, background and action modification, compositional editing, and subject extraction. Both benchmarks are evaluated using GPT-4.1. All inference latencies are measured on an NVIDIA H200 GPU.

B Experimental Settings

CacheFlow is initialized from the corresponding pretrained models and jointly optimizes the frequency-aware policy, LoRA parameters, output heads, and cache predictor under frozen-teacher trajectory supervision. Text-to-image training uses 2.3 million text prompts from π-Flow (Chen et al., 2026), while image-editing training uses 28,255 editing samples from ShareGPT-4o-Image (Chen et al., 2025). Detailed training configurations are provided in Tab. 7.

The cache predictor consists of LayerNorm, a $3072 \rightarrow 6144$ linear layer, GELU, a $6144 \rightarrow 3072$ linear layer, and a residual connection, with the final layer initialized to zero. The two-step FLUX.1-dev, Qwen-Image, and Qwen-Image-2.1 models use one predictor between the step pair $(1, 2)$, while the four-step Qwen-Image-Edit model uses separate predictors for the step pairs $(1, 2)$ and $(3, 4)$.

Table 7: **Detailed training configurations for CacheFlow.**

Setting	FLUX.1-dev	Qwen-Image	Qwen-Image-Edit	Qwen-Image-2.1
Student steps	2	2	4	2
Intermediate supervision points	4	4	4	4
Training steps	4,000	4,000	4,000	4,000
Global batch size	384	384	384	384
Optimizer	AdamW	AdamW	AdamW	AdamW
(β_1, β_2)	$(0.9, 0.95)$	$(0.9, 0.95)$	$(0.9, 0.95)$	$(0.9, 0.95)$
Weight decay	0	0	0	0
Main learning rate	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Policy-head learning rate	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Cache-MLP learning rate	$10^{-3} \to 10^{-4}$	$10^{-3} \to 10^{-4}$	$10^{-3} \to 10^{-4}$	$10^{-3} \to 10^{-4}$
LoRA rank / dropout	256 / 0.05	256 / 0.05	256 / 0.05	256 / 0.05
Teacher-ratio decay steps	2,000	1,000	1,000	1,000
Guidance scale	3.5	4.0	4.0	1.0
Noise shift	3.2	3.2	3.2	Dynamic

C Training and Inference Algorithms

Algorithms 1 and 2 summarize the joint training and inference procedures of CacheFlow, respectively.

Algorithm 1 Joint Training for CacheFlow

Require: Frozen teacher $\mathbf{v}_T$, student $G_{\boldsymbol{\theta}}$, cache predictor $C_{\boldsymbol{\phi}}$, schedule $\{t_j\}_{j=0}^{J}$, cached interval $[a, b]$, mixing ratio λ

Sample $\mathbf{x}_{t_0} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and condition $\mathbf{c}$; $\mathcal{L}_{\text{distill}} \leftarrow 0$
for $k = 0, \ldots, J/2 - 1$ **do**
  $j \leftarrow 2k$
  $(\mathcal{P}_j, \Delta_j^{a:b}) \leftarrow G_{\boldsymbol{\theta}}(\mathbf{x}_{t_j}, t_j, \mathbf{c})$
  Sample $\mathcal{S}_j \subset [t_{j+1}, t_j]$ and construct $\mathbf{x}_s$ by MixedIntegration
  $\mathcal{L}_{\text{distill}} \leftarrow \mathcal{L}_{\text{distill}} + \sum_{s \in \mathcal{S}_j} \|\mathbf{v}_{\boldsymbol{\theta}}(s \mid \mathbf{x}_{t_j}, t_j) - \mathbf{v}_T(\mathbf{x}_s, s)\|_2^2$
  $\mathbf{x}_{t_{j+1}} \leftarrow \text{stopgrad}\Big(\mathbf{x}_{t_j} + \int_{t_j}^{t_{j+1}} \mathbf{v}_{\boldsymbol{\theta}}(t \mid \mathbf{x}_{t_j}, t_j)\, \mathrm{d}t\Big)$
  $\widehat{\Delta}_{j+1}^{a:b} \leftarrow C_{\boldsymbol{\phi}}(\Delta_j^{a:b})$
  $\mathcal{P}_{j+1} \leftarrow G_{\boldsymbol{\theta}}(\mathbf{x}_{t_{j+1}}, t_{j+1}, \mathbf{c}; \widehat{\Delta}_{j+1}^{a:b})$
  Sample $\mathcal{S}_{j+1} \subset [t_{j+2}, t_{j+1}]$ and construct $\mathbf{x}_s$ by MixedIntegration
  $\mathcal{L}_{\text{distill}} \leftarrow \mathcal{L}_{\text{distill}} + \sum_{s \in \mathcal{S}_{j+1}} \|\mathbf{v}_{\boldsymbol{\theta}}(s \mid \mathbf{x}_{t_{j+1}}, t_{j+1}) - \mathbf{v}_T(\mathbf{x}_s, s)\|_2^2$
  $\mathbf{x}_{t_{j+2}} \leftarrow \text{stopgrad}\Big(\mathbf{x}_{t_{j+1}} + \int_{t_{j+1}}^{t_{j+2}} \mathbf{v}_{\boldsymbol{\theta}}(t \mid \mathbf{x}_{t_{j+1}}, t_{j+1})\, \mathrm{d}t\Big)$
end for
Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using $\nabla_{\boldsymbol{\theta}, \boldsymbol{\phi}} \mathcal{L}_{\text{distill}}$

Algorithm 2 Inference with CacheFlow

Require: Student $G_{\boldsymbol{\theta}}$, cache predictor $C_{\boldsymbol{\phi}}$, condition $\mathbf{c}$, schedule $\{t_j\}_{j=0}^{J}$, cached interval $[a, b]$

Sample $\mathbf{x}_{t_0} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $j = 0, \ldots, J-1$ **do**
  if j is a full-computation step **then**
    $(\mathcal{P}_j, \Delta_j^{a:b}) \leftarrow G_{\boldsymbol{\theta}}(\mathbf{x}_{t_j}, t_j, \mathbf{c})$
  else
    $\widehat{\Delta}_j^{a:b} \leftarrow C_{\boldsymbol{\phi}}(\Delta_{j-1}^{a:b})$
    $\mathcal{P}_j \leftarrow G_{\boldsymbol{\theta}}(\mathbf{x}_{t_j}, t_j, \mathbf{c}; \widehat{\Delta}_j^{a:b})$
  end if
  $\mathbf{x}_{t_{j+1}} \leftarrow \mathbf{x}_{t_j} + \int_{t_j}^{t_{j+1}} \mathbf{v}_{\boldsymbol{\theta}}(t \mid \mathbf{x}_{t_j}, t_j)\, \mathrm{d}t$
end for
return $\mathrm{VAE.Dec}(\mathbf{x}_0)$

D Frequency-Aware Policy Distillation

Let the spatial-domain velocity be $\mathbf{v}(t) \in \mathbb{R}^{C\times H\times W}$. The two-dimensional Haar DWT is applied independently to each channel. Its one-dimensional low-pass and high-pass filters are $\boldsymbol{\ell} = \frac{1}{\sqrt{2}}[1, 1]$ and $\boldsymbol{h} = \frac{1}{\sqrt{2}}[1, -1]$, respectively. By the separability of the Haar transform, the four two-dimensional analysis kernels are

$$\mathbf{K}_{LL} = \boldsymbol{\ell}^{\mathsf{T}}\boldsymbol{\ell}, \quad \mathbf{K}_{LH} = \boldsymbol{\ell}^{\mathsf{T}}\boldsymbol{h}, \quad \mathbf{K}_{HL} = \boldsymbol{h}^{\mathsf{T}}\boldsymbol{\ell}, \quad \mathbf{K}_{HH} = \boldsymbol{h}^{\mathsf{T}}\boldsymbol{h}. \tag{5}$$

For $q \in \mathcal{Q} = \{LL, LH, HL, HH\}$, the frequency-band coefficients are obtained by a stride-2 convolution:

$$\mathbf{u}_q(t) = \downarrow_2 (\mathbf{K}_q * \mathbf{v}(t)), \qquad \mathbf{u}_q(t) \in \mathbb{R}^{C\times H/2\times W/2}. \tag{6}$$

The LL band retains low-frequency responses along both spatial directions and represents global layout and smooth structure. The LH and HL bands retain directional high-frequency variations and describe directional edges, while the HH band responds to high-frequency textures along both directions.

Let $\mathcal{W}_f(\mathbf{v}) = \{\mathbf{u}_q\}_{q\in\mathcal{Q}}$ denote the forward transform and $\mathcal{W}$ its inverse. The orthogonality of the Haar basis gives the Parseval identity and perfect-reconstruction property:

$$\|\mathbf{v}\|_2^2 = \sum_{q\in\mathcal{Q}} \|\mathbf{u}_q\|_2^2, \qquad \mathcal{W}(\mathcal{W}_f(\mathbf{v})) = \mathbf{v}. \tag{7}$$

Each frequency band is therefore modeled in an orthogonal subspace and reconstructed into the spatial-domain velocity. Let the temporal dynamics of band q be $\mathbf{u}_{\boldsymbol{\theta},q}(t) = \mathbf{B}_q + (t - t_s)\mathbf{D}_q$. Since the inverse Haar transform is linear, the integral over any subinterval $[t_a, t_b] \subseteq [t_s, t_e]$ is

$$\int_{t_a}^{t_b} \mathbf{v}_{\boldsymbol{\theta}}(t)\, \mathrm{d}t = (t_b - t_a)\, \mathcal{W}\left\{ \mathbf{B}_q + \left(\frac{t_a + t_b}{2} - t_s \right) \mathbf{D}_q \right\}_{q\in\mathcal{Q}}. \tag{8}$$

Each internal substep is then updated by $\mathbf{x}_{t_b} = \mathbf{x}_{t_a} + \int_{t_a}^{t_b} \mathbf{v}_{\boldsymbol{\theta}}(t)\, \mathrm{d}t$ and matched to the teacher trajectory.

E Additional Experiments

E.1 Cached Block Intervals

We further investigate the effect of different cached block intervals on generation quality. All experiments use the two-step FLUX.1-dev and Qwen-Image models with fixed cache predictors, training configurations, numbers of cached blocks, and an NFE of 1.75. For each model, we compare five cache settings: three contiguous intervals located in the early, middle, and late blocks, and two mixed intervals with the same total number of cached blocks.

Table 8: **Ablation of cached block intervals** on FLUX.1-dev and Qwen-Image.

Cached Blocks	GenEval↑	DPG-Bench↑
FLUX.1-dev		
6–19	0.66	84.36
23–36	0.64	82.17
40–53	0.61	80.85
6–12 + 23–29	0.64	83.32
23–29 + 40–46	0.62	80.92
Qwen-Image		
9–23	0.86	88.47
24–38	0.83	87.01
39–53	0.82	86.42
9–15 + 24–31	0.84	87.21
24–30 + 39–46	0.82	86.65

As shown in Tab. 8, caching the early-block interval yields the best generation quality, while the middle- and late-block intervals cause larger performance drops. The mixed intervals exhibit the same trend. These results indicate that early blocks are more suitable for cross-step reuse. We therefore select blocks 6–19 for FLUX.1-dev and blocks 9–23 for Qwen-Image in the 1.75-NFE configuration. Other NFE configurations extend the cached range following the same principle.

F More Visualization Results

In this section, we provide additional visualization examples for text-to-image generation and image editing to further demonstrate that CacheFlow achieves higher generation and editing quality than previous acceleration methods, as shown in Fig. 9–13.



Figure 9: **Additional visualization results for text-to-image generation.**

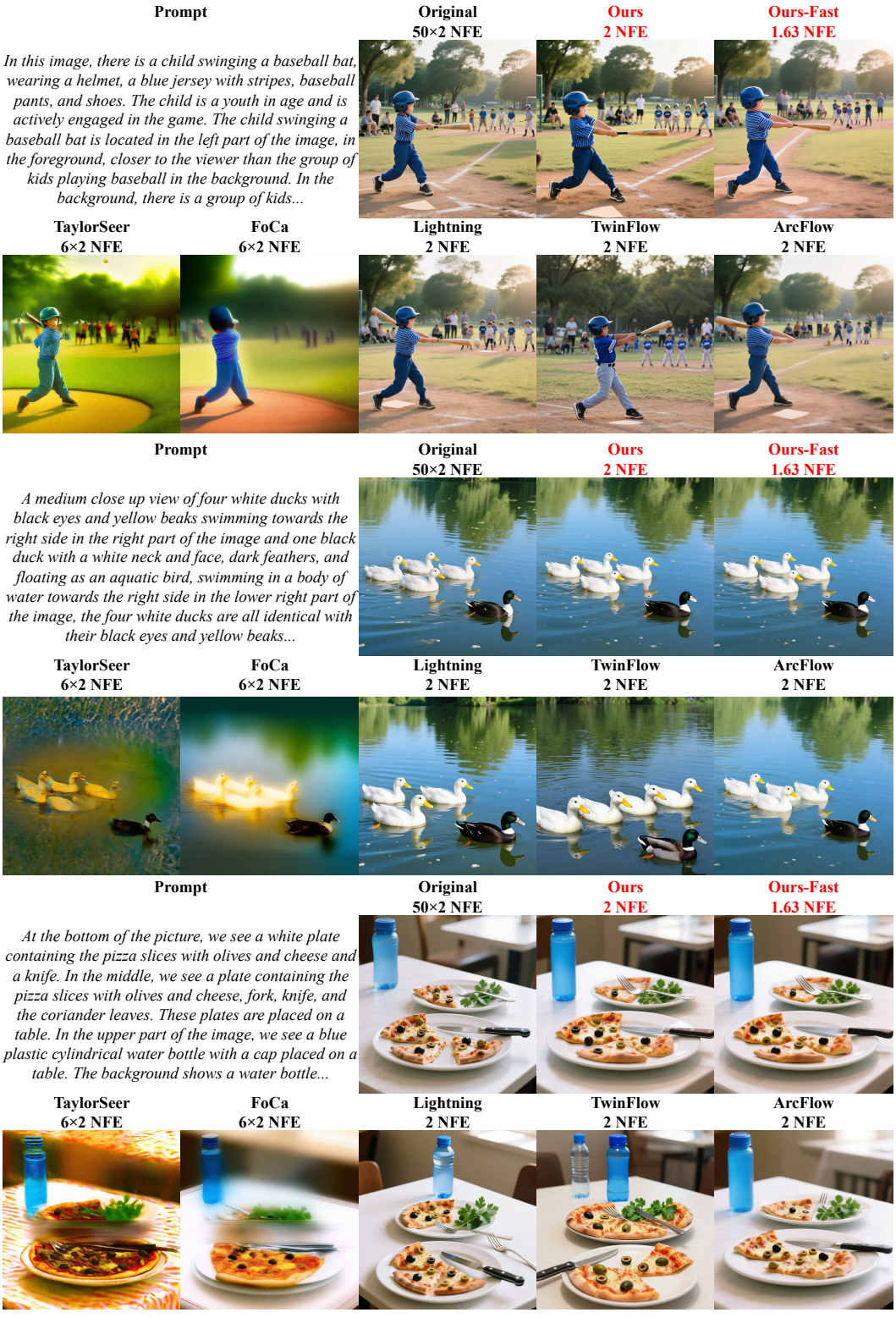

Figure 10: **Additional visualization results for text-to-image generation.**

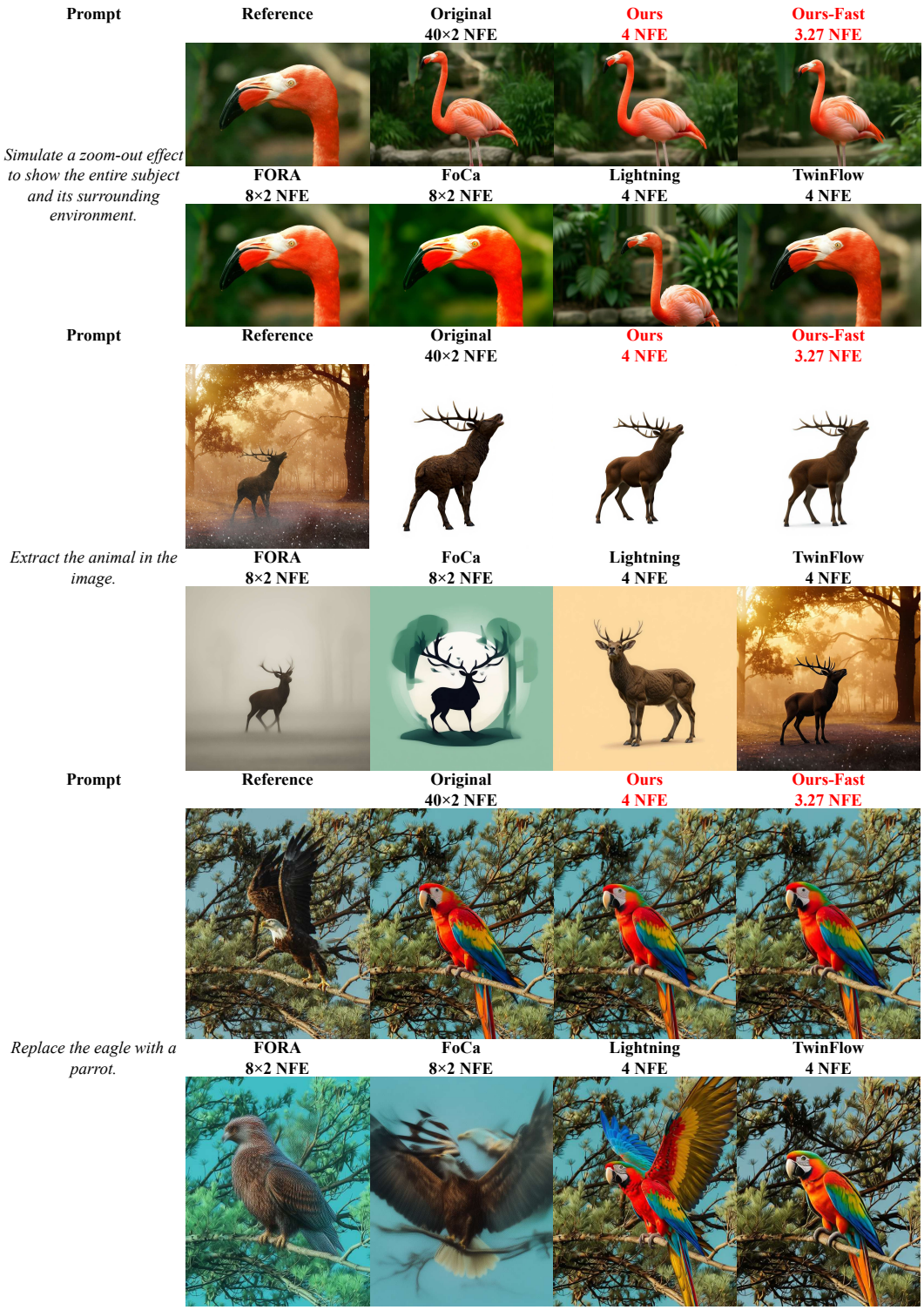

Figure 11: **Additional visualization results for image editing.**

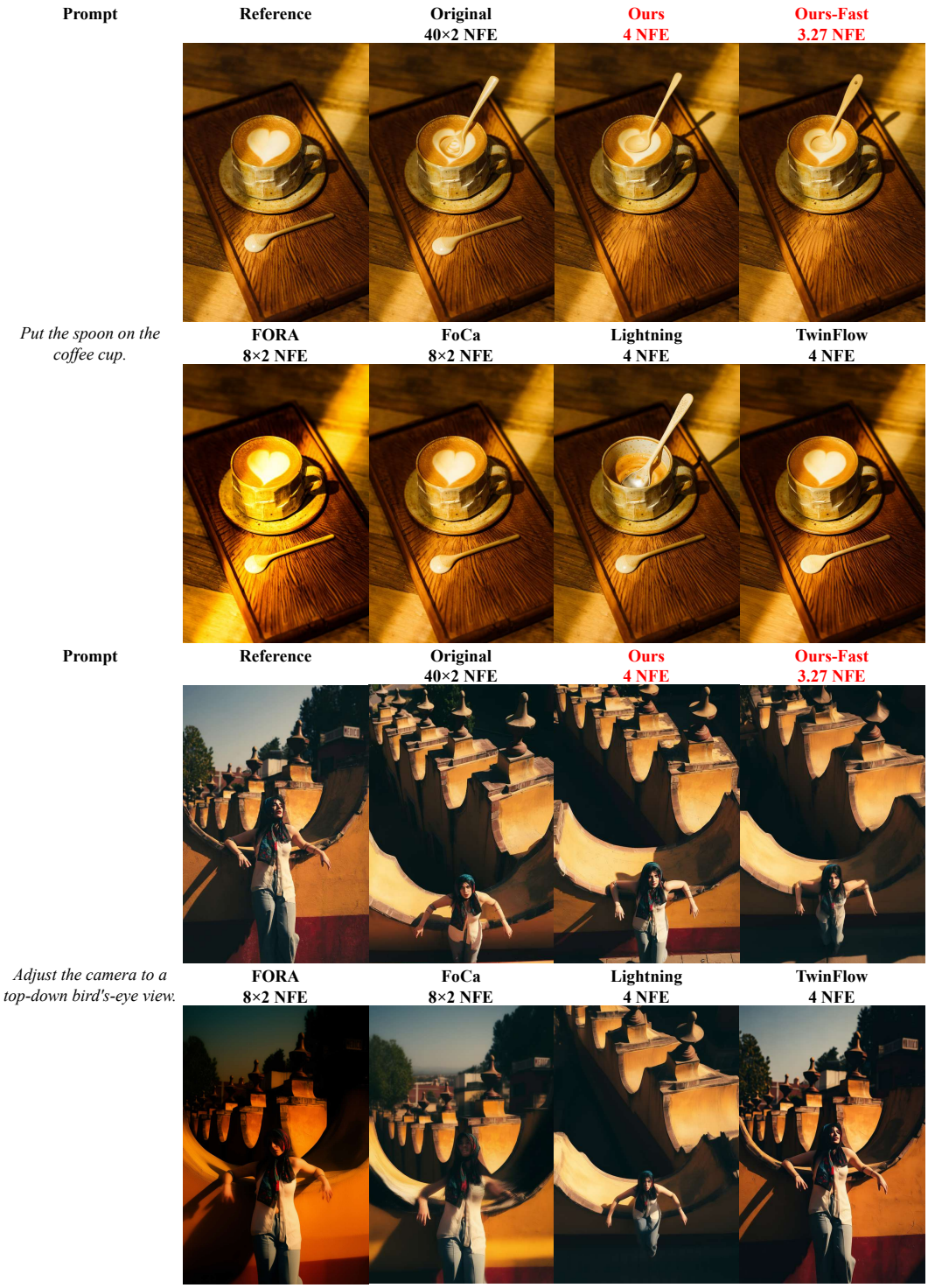

Figure 12: **Additional visualization results for image editing.**

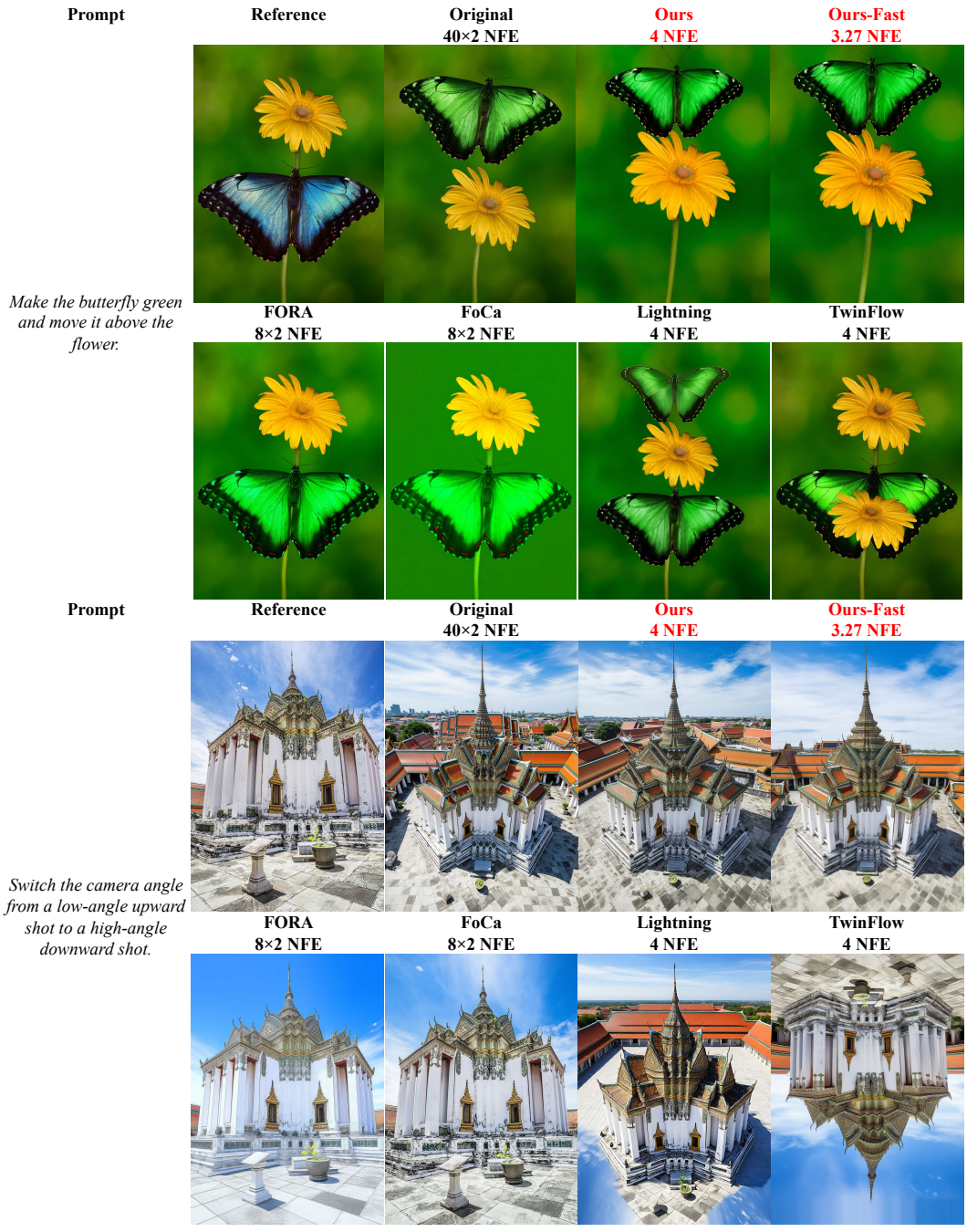

Figure 13: **Additional visualization results for image editing.**