Accelerating Diffusion Transformers with Gaussian Process Rectified Feature Cache

Zhirong Shen[1,3⋆], Rui Huang[1,3⋆], *ChangZou*[1,3], Shikang Zheng[1], Jiacheng Liu[1,4], Peiliang Cai[1], Zhengyi Shi[5], Yaosong Du[2], Liang Feng[6], Xiaobing Tu[2], Jinkui Ren[2], Xiantao Zhang[2], Linfeng Zhang[1⋆⋆]

[1] Shanghai Jiao Tong University
[2] Terminal Intelligent Computing Division, Alibaba Cloud
[3] University of Electronic Science and Technology of China
[4] Shandong University
[5] Xiamen University
[6] Fudan University

{2024080907011,huang_rui}@std.uestc.edu.cn zhanglinfeng@sjtu.edu.cn

Abstract. Diffusion Transformers have become the dominant paradigm in generative AI, but their high computational costs severely hinder real-time applications. Prediction-based feature caching is widely used to accelerate diffusion transformers; however, as the number of steps increases, the deviation between its predictions and the reference full-compute trajectory gradually grows. An intuitive idea is to use an online regression model to dynamically correct this deviation, but it faces the issue of label data being unavailable during the acceleration process. This paper presents a statistical observation that the residuals between the features of full computation steps using caching methods and reference full-compute trajectory locally exhibit a zero-mean Gaussian distribution. By treating the features of full computation steps as noisy observations of reference features, the data acquisition problem is resolved. Based on this observation, a plug-and-play GP-Refiner correction framework is proposed. This method utilizes Gaussian Process Regression for correction and, leveraging the properties of GPR, introduces an uncertainty-adaptive computation strategy that triggers necessary full-computation calibration by monitoring the posterior variance in real time. Experiments demonstrate significant improvements across different models when combined with various state-of-the-art methods. Integrating the proposed framework with **TaylorSeer** reduces the computational load by **19.3%** while improving PSNR by **0.9** dB and reducing LPIPS from **0.46** to **0.29**. Code is available in https://github.com/Aredstone/GP-Refiner.

Keywords: Diffusion Transformers · Feature Caching · Gaussian Process Regression

⋆ Equal contribution.
⋆⋆ Corresponding author.

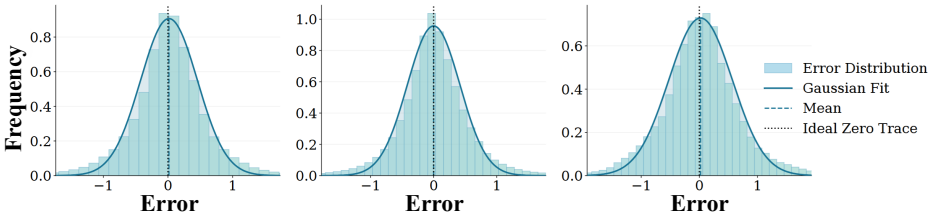

Fig. 1: Histogram of the error caused by the feature caching. Across different images, the residuals are centered near zero and closely match a Gaussian fit, consistent with an approximately zero-mean Gaussian noise model.

1 Introduction

Diffusion models [8, 22, 34] have become the dominant paradigm for visual generation, yet the adoption of Transformer-based architectures has drastically increased inference cost [1, 3, 4]. Existing *training-free* feature caching techniques exploit temporal coherence along the diffusion trajectory by directly reusing intermediate features [20, 26]; however, under high acceleration ratios, the growing feature mismatch across long skip intervals leads to accumulated errors and noticeable quality degradation [14]. To overcome this bottleneck, *forecasting-based* cache acceleration has recently emerged [7, 38]. Instead of pure reuse, it explicitly models the temporal evolution of intermediate representations to *predict* future features from historical states, mitigating accuracy decay at large skip intervals and substantially reducing the number of expensive denoising-network forward evaluations while preserving generation fidelity.

Forecasting-based caching can substantially reduce the computation of diffusion sampling, but its prediction error typically accumulates with the skip interval, causing the generation trajectory to progressively drift away from the reference full-compute trajectory [2, 15]. We aim to develop an adaptive rectification mechanism that requires no offline training and can be seamlessly plugged into existing forecasting-based caching accelerators, mapping the predicted features back to the reference full-compute trajectory under full-step sampling, denoted as f_t. However, during accelerated inference, once earlier steps have been skipped, even if we perform a full forward computation at the current timestep, we can only obtain $\hat{f}_t$, which is computed on an already drifted state and thus differs from the unbiased feature needed as supervision for training the rectifier.

Our residual analyses of f_t and $\hat{f}_t$ at both local and global levels show that, as illustrated in Fig. 1 and Fig. 2, the residual between the full-compute-step features obtained under caching and the corresponding reference features exhibits a pronounced zero-mean Gaussian characteristic. Based on this observation, we can treat the full-compute-step feature $\hat{f}_t$ under caching as a Gaussian-noisy observation of the reference feature, thereby resolving the label acquisition challenge described above.

Motivated by the fact that Gaussian process regression (GPR) naturally accommodates observations corrupted bThe complete pipeline ofy zero-mean

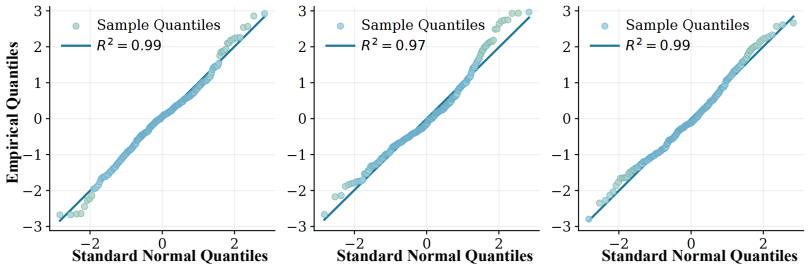

Fig. 2: Q–Q plots of randomly projected feature residuals. The empirical quantiles closely follow a standard normal line ($R^2 \approx 0.97$–0.99), supporting an approximately zero-mean Gaussian residual model.

Gaussian noise, and that it matches the requirements of DiT cache acceleration, *few-shot*, *non-parametric*, and capable of fitting highly nonlinear mappings, we propose **GP-Refiner**, which leverages GPR to compensate for the bias in the baseline cache predictor. Moreover, we reformulate the problem as a joint task of one-step evolution and rectification: starting from a biased baseline prediction, we infer the unbiased reference value at the next moment. During inference, GP-Refiner takes the baseline predicted feature as input and performs GPR-based correction, driving the accelerated trajectory closer to the reference features while incurring almost no additional computation overhead.

Furthermore, we exploit another key property of GPR to enable an adaptive computation mechanism [5, 29]. By monitoring the posterior variance online, our algorithm can quantify the confidence of the current correction; when the uncertainty exceeds a predefined threshold, it forcibly triggers a full forward computation for state recalibration. This mechanism effectively establishes a dynamic balance between inference speed and generation quality.

The main contributions are divided into three aspects:

1. **Noisy Observation Modeling from Statistical Evidence:** We reveal that the residuals between cached features and the corresponding reference (full-compute) features follow an approximately zero-mean Gaussian distribution. This allows us to treat full-compute-step features obtained under caching as Gaussian-noisy observations of the unbiased reference features, thereby resolving the supervision/label unavailability issue in accelerated inference and laying the foundation for regression-based bias correction.
2. **GP-Refiner for Plug-and-Play Bias Rectification:** We propose **GP-Refiner**, which leverages Gaussian Process Regression to dynamically rectify the predictions of baseline cache forecasters. With nearly no additional compute overhead, GP-Refiner substantially improves generation fidelity under high acceleration.
3. **Adaptive Re-computation via Posterior Uncertainty:** We introduce an adaptive computation strategy that quantifies prediction uncertainty via the Gaussian process posterior variance. When uncertainty exceeds a predefined threshold, we trigger a full forward evaluation to recalibrate the state,

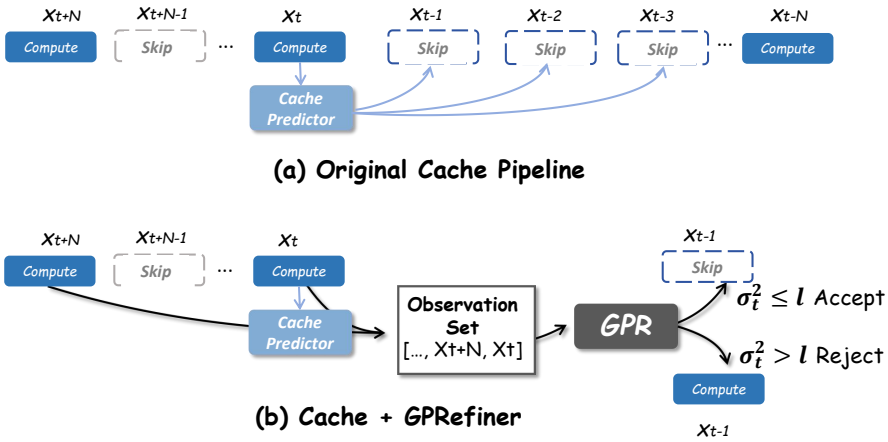

Fig. 3: Overall workflow of GP-Refiner. (a) Original cache pipeline: features are computed at selected timesteps and cached, while skipped steps are filled by a cache predictor. **(b)** Cache + GP-Refiner: we collect an online observation set from full-compute steps, use GPR to rectify the predictor output, and trigger recomputation when the posterior variance exceeds a threshold ($\sigma_t^2 > l$).

achieving a dynamic trade-off between inference speed and generation quality.

2 Related Work

The computational burden of Diffusion Transformers (DiT) [9, 21, 39] mainly arises from two factors: (i) the heavy Transformer computation when operating on high-resolution inputs [10, 13], and (ii) the dozens of sequential denoising iterations required to preserve generation quality [25, 30, 37]. Accordingly, most existing acceleration efforts target these two bottlenecks.

2.1 Acceleration via Model Modification and Retraining

A first line of research reduces inference cost by modifying the denoising model or compressing the sampling trajectory [6, 35], but requires additional retraining or fine-tuning [12, 41]. Representative examples include structural pruning and token reduction to lower per-step Transformer compute [23, 36], as well as trajectory compression via consistency models and flow matching to enable few-step generation [16, 28, 33]. Despite their efficiency, these approaches are less plug-and-play for large, already-trained DiT models, which motivates our focus on training-free inference-time acceleration.

2.2 Sampling Schedulers and Numerical Solvers

A representative training-free acceleration direction is to improve the sampling scheduler by adopting more accurate numerical solvers, so that larger discrete step sizes can be used without sacrificing generation fidelity [19]. Typical examples include DDIM [27] and the DPM-Solver family [17, 18], which leverage higher-order ODE-based solvers to stabilize coarse discretization. While effective in reducing the total number of steps, these methods still require a full forward pass of the denoising network at each step, leaving the per-step DiT computation largely unchanged and motivating complementary cache-based acceleration.

2.3 Cache-Based Inference Acceleration

Cache-based acceleration is a training-free strategy that skips denoising-network computation at selected timesteps by exploiting the temporal coherence of intermediate representations along the diffusion trajectory. Existing methods can be broadly organized into three representative paradigms.

Cache-then-Reuse. Reuse-based methods [20, 26] cache features at timestep t and directly reuse them for subsequent skipped steps. They are simple and introduce minimal overhead, making them effective at modest speedups [31, 42]. However, feature similarity drops rapidly as the reuse interval increases, which causes pronounced drift and error accumulation under high acceleration ratios.

Cache-then-Forecast. To support larger skip intervals, forecasting-based methods explicitly model the temporal evolution of features and predict future states from historical trajectories. TaylorSeer [14] pioneers this direction by treating feature evolution as a continuous-time process and extrapolating future features via Taylor-style high-order prediction. Subsequent works further improve long-horizon forecasting from different perspectives, including more stable polynomial or basis-function estimation and ODE-inspired formulations (e.g., HiCache [7], FoCa [38]). While substantially more robust than direct reuse, forecasting errors are still inevitable and can accumulate over long horizons, motivating the need for additional mechanisms to control drift.

Forecast-and-Correct. To mitigate drift under aggressive skipping, recent work augments forecasting with feedback-driven correction mechanisms that introduce additional signals (e.g., verification or probing) to decide when to recompute and how to better track the denoising trajectory. SpeCa [15] proposes a speculative caching framework that couples a lightweight predictor with low-cost verification, and rolls back or recomputes once the predicted trajectory exceeds a tolerance, improving robustness at high speedups. DiCache [2] explores online alignment by leveraging probing signals to adaptively schedule caching and better match the underlying denoising dynamics. Overall, these designs go beyond

pure forecasting by explicitly suppressing long-horizon error accumulation while retaining most of the computational gains from cache-based acceleration.

Overall, as the skip interval grows, both reuse-based and forecasting-based caching can accumulate errors and drift away from the true denoising trajectory. Existing corrective designs mitigate this drift using verification or probing signals to trigger recomputation, but they introduce extra overhead and remain limited for highly nonlinear feature errors. This motivates a *training-free*, *plug-and-play* module that performs *nonlinear* correction while providing trustworthy *uncertainty* or *confidence* estimates for adaptive recomputation.

3 Method

3.1 Preliminary

Diffusion Transformer (DiT). DiT is a diffusion-model backbone built entirely from Transformer layers and augmented with adaptive normalization. Given an image at time t, the model operates on a sequence of patch tokens $\mathbf{x}_t = \{x_i\}_{i=1}^{H\times W}$, where each x_i encodes one image patch. The network is a composition of L Transformer blocks, $\mathcal{G} = g_L \circ \cdots \circ g_2 \circ g_1$, and each block combines self-attention, cross-attention, and a feed-forward subnetwork:

$$g_\ell(\cdot) = f_{\mathrm{MLP}}^{(\ell)}\left(f_{\mathrm{CA}}^{(\ell)}\left(f_{\mathrm{SA}}^{(\ell)}(\cdot)\right)\right), \quad \ell = 1, \ldots, L. \tag{1}$$

Here, $f_{\mathrm{SA}}^{(\ell)}$ captures intra-token dependencies within the current latent sequence, $f_{\mathrm{CA}}^{(\ell)}$ conditions on external signals (e.g., text or timestep embeddings), and $f_{\mathrm{MLP}}^{(\ell)}$ provides non-linear mixing in token space. Adaptive normalization layers modulate these components with timestep- and/or conditioning-dependent parameters, enabling the model to tailor its computations across the diffusion trajectory.

Feature Caching for Diffusion Transformer. Temporal feature caching in DiT can be organized into two complementary paradigms: *reuse-based* caching and *forecasting-based* caching. In the reuse-based paradigm, a periodic schedule with interval $\mathcal{N}$ selects an anchor step t at which all layer features are computed and stored, $\mathcal{C}(x_t^l) := \mathcal{F}(x_t^l)$ for $l \in \{0, \ldots, L-1\}$. For the following $\mathcal{N}-1$ steps within the period, computation is skipped and features are reused,

$$\mathcal{F}(x_{t-k}^l) := \mathcal{C}(x_t^l), \quad k \in \{1, \ldots, \mathcal{N}-1\}, \tag{2}$$

yielding an approximate FLOPs reduction of $(\mathcal{N}-1)/\mathcal{N}$ at the cost of an accumulated temporal mismatch as $\mathcal{N}$ grows. In the *forecasting-based* paradigm, instead of directly reusing the anchor features, the skipped-step features are *predicted* from cached temporal statistics that summarize local feature dynamics. Among forecasting-based methods, a representative approach is TaylorSeer [14], which maintains the anchor feature together with its temporal finite differences up to order m,

$$\mathcal{C}(x_t^l) := \left\{\mathcal{F}(x_t^l), \Delta\mathcal{F}(x_t^l), \ldots, \Delta^m\mathcal{F}(x_t^l)\right\}, \tag{3}$$

and estimates feature at step $t-k$ via a truncated Taylor expansion around t,

$$\mathcal{F}_{\text{pred},m}(x_{t-k}^l) = \mathcal{F}(x_t^l) + \sum_{i=1}^{m} \frac{\Delta^i \mathcal{F}(x_t^l)}{i! \mathcal{N}^i} (-k)^i, \tag{4}$$

where $\Delta^i \mathcal{F}(x_t^l)$ denotes the i-th temporal difference and the factor $\mathcal{N}^{-i}$ normalizes by the period length. This Taylor-series modeling explicitly captures short-horizon evolution of layer features and helps mitigate drift at larger skip intervals.

Gaussian Process Regression. In the context of probabilistic modeling, a Gaussian Process (GP) is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution. Given a training set of input vectors $X = \{\mathbf{x}_i\}_{i=1}^n$ and a symmetric positive definite kernel function $k(\cdot, \cdot)$, we first define the training covariance matrix:

$$\mathbf{K} = [k(\mathbf{x}_i, \mathbf{x}_j)] \in \mathbb{R}^{n \times n}. \tag{5}$$

For a given test point $\mathbf{x}_*$, the covariance vector between the training points and the test point is defined as:

$$\mathbf{k}_* = [k(\mathbf{x}_1, \mathbf{x}_*), \ldots, k(\mathbf{x}_n, \mathbf{x}_*)]^\top \in \mathbb{R}^n. \tag{6}$$

The prior self-covariance of the test point is expressed as the scalar:

$$k_{**} = k(\mathbf{x}_*, \mathbf{x}_*). \tag{7}$$

We assume that the observed outputs $\mathbf{y}$ are related to the latent function values through an additive i.i.d. Gaussian noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_n^2 \mathbf{I})$. Consequently, the joint distribution of the observed training data $\mathbf{y}$ and the latent test value g_* follows a multivariate Gaussian distribution:

$$\begin{bmatrix} \mathbf{y} \\ g_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{k}_* \\ \mathbf{k}_*^\top & k_{**} \end{bmatrix}\right). \tag{8}$$

By conditioning the joint distribution on the observed data $\mathbf{y}$ and applying the properties of Schur complements, we derive the predictive distribution for the test point. The predictive mean, which represents the point estimate at $\mathbf{x}_*$, is given by:

$$\mu_* = \mathbf{k}_*^\top \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{y}. \tag{9}$$

Furthermore, the uncertainty associated with this prediction is quantified by the predictive variance:

$$\sigma_*^2 = k_{**} - \mathbf{k}_*^\top \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{k}_*. \tag{10}$$

These foundational expressions allow the model to provide both a regression value μ_* and a formal measure of confidence σ_*^2 for any unseen input.

3.2 Observation

In the accelerated generation process of diffusion models, an intuitive approach to correct the cumulative errors caused by approximation methods is to train a regression model for feature compensation. However, this faces a fundamental dilemma in practical applications: the problem of acquiring sample data. Due to the error accumulation from preceding timesteps, the features obtained during the acceleration process always deviate from the reference features on the standard denoising trajectory. This implies that we cannot collect accurate, real features online to serve as unbiased training samples.

To overcome this limitation, we conducted an in-depth analysis of the statistical properties of the feature deviation. Let $\hat{f}_t$ denote the feature obtained at a full computation step when using the caching method, and f_t denote the true unbiased feature at the corresponding timestep. We define the residual between them as $\epsilon_t = f_t - \hat{f}_t$. Through extensive statistical testing, we observed a crucial phenomenon: the residual between the full computation step feature using the caching method and the reference feature exhibits significant characteristics of a zero-mean Gaussian distribution. We rigorously verified this phenomenon through experiments at both the micro and macro levels.

First, at the micro level, we extracted an arbitrary single residual vector ϵ_t and conducted a visual analysis of all its dimensional elements. As shown in Figure 1, the results indicate that the empirical probability density distribution of the elements within a single residual vector closely fits a standard Gaussian bell curve. This intuitively demonstrates that the elements of a single vector approximately follow a one-dimensional Gaussian distribution.

Meanwhile, we conducted multivariate Gaussian tests on multiple residual vectors at the macro level. We leveraged a core property from multivariate statistical theory: if a high-dimensional random vector follows a multivariate Gaussian distribution, its linear projection in any arbitrary direction must necessarily follow a one-dimensional Gaussian distribution. As shown in Figure 2, we linearly projected multiple residual vectors onto several randomly generated unit directions and plotted Q-Q plots for the projected one-dimensional data. Experimental calculations indicate that the scatter points of the projected data are highly concentrated along the theoretical diagonal of the Q-Q plot. Across 100 random projections, the average correlation coefficient exceeded 0.98, with the lowest correlation coefficient reaching 0.9424. This rigorous quantitative test strongly confirms that the residual vector ϵ_t overall approximately follows a multivariate Gaussian distribution.

3.3 Feature Correction Modeling via Gaussian Processes

To rectify the cumulative errors during the accelerated generation of diffusion models, we reformulate the tasks of feature evolution and correction into a unified framework. The complete pipeline of GP-Refiner is illustrated in Figure 3, this section elaborates on how to infer unbiased ground-truth values for the next step by combining biased baseline predictions with historical observations.

Formalization of Error Observation Based on the statistical analysis of feature deviations in Section 3.2, we observed a critical phenomenon: the residuals between the features generated by the baseline prediction method and the reference features exhibit a distinct Gaussian distribution. Based on this, we can mathematically reformulate the feature evolution process. Specifically, we treat the feature $\hat{f}_t$ obtained from a full-computation step as a "noisy observation" centered around the corresponding feature f_t of the standard denoising process, perturbed by Gaussian noise. This relationship is formally expressed as:

$$\hat{f}_t = f_t + \epsilon \tag{11}$$

where the residual ϵ is modeled as Gaussian noise with zero mean, denoted as:

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2 \mathbf{I}) \tag{12}$$

This shift in modeling perspective is crucial. By integrating with Gaussian Process Regression (GPR), it elegantly bypasses the dilemma of being unable to obtain large-scale unbiased training samples in acceleration scenarios.

Dynamic Construction of Baseline Methods and Observation Sets Guided by the aforementioned observation, the model dynamically maintains an observation set $\mathcal{D}_t$ during the inference stage to collect the data for GPR. The specific acquisition and construction mechanism of this set is as follows:

When the model triggers a full network forward pass at a historical time step k, we first obtain the accurate full-computation feature $\hat{f}_k$. Subsequently, this value is used to update the state of the baseline prediction method. Utilizing the updated state and the extrapolative capability of the baseline method, we "predict" the feature of the previous step. We define the predicted feature output by the baseline method as:

$$\hat{p}_{k-1} = \text{BaseMethod}(k-1) \tag{13}$$

Following this, we pair the baseline prediction for the previous step $\hat{p}_{k-1}$ with the current full-computation feature $\hat{f}_k$ to form a data pair $(\hat{p}_{k-1}, \hat{f}_k)$, which is then formally added to the observation set. For consistency in subsequent mathematical expressions, we denote the i-th data pair added to the observation set as $(\hat{p}_{\tau_i-1}, \hat{f}_{\tau_i})$, where $\hat{p}_{\tau_i-1}$ represents the baseline predicted feature and $\hat{f}_{\tau_i}$ represents the noisy observation of the local ground-truth feature.

Before reaching the current time step t, if the system has collected n historical observation pairs, the observation set can be formally defined as:

$$\mathcal{D}_t = \{(\hat{p}_{\tau_i-1}, \hat{f}_{\tau_i})\}_{i=1}^{n} \tag{14}$$

Implementing this pairing mechanism between adjacent steps is of significant theoretical importance. It strictly confines the focus of error correction to the local residuals of single-step evolution. This local pairing strategy effectively reduces the fitting pressure on the model, allowing it to concentrate on short-term evolutionary deviations rather than attempting to correct highly non-linear, long-span error accumulations.

Introduction and Derivation of Gaussian Process Regression With the construction of the observation set clearly defined, we formally introduce the GPR framework. Distinct from traditional time-series fitting, we treat the baseline predicted features $\hat{p}$ as the input space and the full-computation features $\hat{f}$ as noisy observations within that space. This approach allows the model to learn the mapping residuals from baseline predictions to unbiased reference features, thereby calibrating the feature trajectory in real-time during inference.

To perform numerical inference, we first extract information from the observation set $\mathcal{D}_t$ to construct the core mathematical entities. First is the observation vector $\mathbf{Y}$, which aggregates the ground-truth feature values captured in all historical full-computation steps:

$$\mathbf{Y} = [\hat{f}_{\tau_1}, \hat{f}_{\tau_2}, \ldots, \hat{f}_{\tau_n}]^T \tag{15}$$

Correspondingly, we construct the prediction input matrix $\mathbf{X}$, which contains the associated baseline predicted features:

$$\mathbf{X} = [\hat{p}_{\tau_1-1}, \hat{p}_{\tau_2-1}, \ldots, \hat{p}_{\tau_n-1}]^T \tag{16}$$

In this context, $\mathbf{X}$ serves as the feature input for the regression task, while $\mathbf{Y}$ represents the target observations we aim to approximate.

Next, we construct the training covariance matrix (Gram matrix) $\mathbf{K}$. Each element K_{ij} is computed via a kernel function $k(\cdot, \cdot)$:

$$K_{ij} = k(\hat{p}_{\tau_i-1}, \hat{p}_{\tau_j-1}) \tag{17}$$

Since we defined the full-computation feature $\hat{f}$ as a noisy observation with variance σ_n^2, we utilize the modified covariance matrix $(\mathbf{K} + \sigma_n^2\mathbf{I})$ for the matrix inversion during inference.

Simultaneously, to characterize the correlation between the current predicted feature $\hat{p}_t$ to be corrected and the historical observations, we construct the test covariance vector $\mathbf{k}_*$:

$$\mathbf{k}_* = [k(\hat{p}_t, \hat{p}_{\tau_1-1}), k(\hat{p}_t, \hat{p}_{\tau_2-1}), \ldots, k(\hat{p}_t, \hat{p}_{\tau_n-1})]^T \tag{18}$$

Based on the conditional distribution theory of Gaussian Processes, we can derive the corrected unbiased feature prediction mean μ_t for the current step t:

$$\mu_t = \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{Y} \tag{19}$$

The framework simultaneously provides the prediction variance σ_t^2 as follows:

$$\sigma_t^2 = k(\hat{p}_t, \hat{p}_t) - \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_* \tag{20}$$

3.4 Adaptive Computation via Uncertainty

The GP-Refiner framework not only rectifies feature deviations but also provides a real-time quantification of inference trajectory uncertainty through the predicted variance σ_t^2. To overcome the limitations of fixed computation intervals,

we introduce a preset variance threshold l, which allows the model to dynamically select the computation mode based on whether the current uncertainty exceeds a tolerable range.

At each time step t, the system first derives the predicted variance σ_t^2 of the current corrected feature via the Gaussian Process. This variance intuitively reflects the confidence level of the current baseline prediction $\hat{p}_t$ relative to the known feature manifold (i.e., the observation set $\mathcal{D}_t$). By setting an activation threshold l, we define the feature acquisition logic as the following piecewise decision process:

$$f_t^* = \begin{cases} \text{FullCompute}(t), & \text{if } \sigma_t^2 > l \\ \mu_t, & \text{if } \sigma_t^2 \leq l \end{cases} \tag{21}$$

where f_t^* represents the feature used for the current inference step. The selection of the threshold will be detailed in the appendix. It exhibits strong robustness and does not significantly impact the results when kept within a certain range. If the predicted variance σ_t^2 exceeds the preset threshold, indicating that the untrustworthiness of the baseline prediction has reached its limit, the system immediately triggers a full computation to obtain the accurate feature; otherwise, it employs the corrected mean μ_t from the Gaussian Process for rapid inference.

This threshold-based activation mechanism ensures that full-computation steps are naturally clustered during stages of the denoising trajectory where non-linearity is strongest and error sensitivity is highest, effectively finding an optimal balance between inference speed and image fidelity.

4 Experiments

4.1 Experiment Settings

Model Configurations. To validate the effectiveness of our method and its generality as a cache-rectification plugin, we conduct experiments on commonly used text-to-image generators: **Qwen-Image** [32] and **FLUX.1-dev** [11]. Both base models operate at 50 inference steps. To further demonstrate robustness under extreme low-step regimes, we also include the Qwen-Image-Lightning variant running at 8 steps. As acceleration backbones, we integrate our rectifier with three representative caching methods: **TaylorSeer** [14], **ClusCa** [40], and **HiCache** [7].

For comprehensive comparison, we benchmark against standard step reduction and recent caching baselines including **FORA**, **ToCa**, **DuCa**, and **TeaCache**. We instantiate the Gaussian-process rectifier with an *angular RBF* kernel, and for regression, we use a fixed FIFO window of h=10 most recent features.
Evaluation and Metrics. For text-to-image generation, we employ 200 DrawBench [24] prompts as inference inputs. To comprehensively assess the performance, we measure both generation fidelity and computational efficiency. We evaluate the visual quality of generated images using SSIM, PSNR, and LPIPS against the unaccelerated reference images, reporting the mean over all prompts. For efficiency evaluation, we report the actual inference latency in seconds and

Table 1: Quantitative comparison in text-to-image generation for FLUX.

Method	Latency(s) ↓	Speed ↑	FLOPs(T) ↓	Speed ↑	PSNR↑	SSIM↑	LPIPS↓
[dev]: 50 steps	11.55	1.00×	3719.50	1.00×	-	-	-
60% **steps**	7.12	1.62×	2231.70	1.67×	30.310	0.7819	0.2461
50% **steps**	5.93	1.95×	1859.75	2.00×	29.576	0.7337	0.3106
40% **steps**	4.82	2.40×	1487.80	2.50×	29.122	0.6971	0.3619
34% **steps**	4.15	2.78×	1264.63	2.94×	28.881	0.6776	0.3913
FORA ($\mathcal{N} = 5$)	3.31	3.49×	893.54	4.16×	28.432	0.6029	0.4918
FORA ($\mathcal{N} = 7$)	2.90	3.98×	670.44	5.55×	28.315	0.5870	0.5409
`ToCa` ($\mathcal{N} = 6$)	7.28	1.59×	854.42	4.35×	29.135	0.6532	0.4081
`ToCa` ($\mathcal{N} = 9$)	6.45	1.79×	784.54	4.74×	28.889	0.6407	0.4525
`DuCa` ($\mathcal{N} = 8$)	3.46	3.34×	676.79	5.50×	29.133	0.6150	0.4534
`DuCa` ($\mathcal{N} = 10$)	3.22	3.59×	606.91	6.13×	28.953	0.5957	0.4935
TeaCache ($l = 0.6$)	3.76	3.07×	1115.44	3.33×	29.029	0.6801	0.4026
TeaCache ($l = 1.0$)	2.77	4.17×	743.63	5.00×	28.606	0.6360	0.4773
TaylorSeer ($O = 2, \mathcal{N} = 6$)	3.81	3.03×	744.80	4.99×	28.943	0.6558	0.4020
TaylorSeer ($O = 2, \mathcal{N} = 7$)	3.60	3.21×	670.44	5.55×	28.671	0.6237	0.4542
TaylorSeer + ours ($O = 2$)	**3.42**	**3.38**×	**643.13**	**5.78**×	**29.550**	**0.6972**	**0.3497**
HiCache ($\mathcal{N} = 6$)	3.77	3.06×	744.80	4.99×	29.104	0.6685	0.3742
HiCache ($\mathcal{N} = 7$)	3.57	3.23×	670.44	5.55×	28.937	0.6572	0.3982
HiCache + ours	**3.45**	**3.35**×	**631.33**	**5.89**×	**29.694**	**0.6912**	**0.3881**
ClusCa ($O = 1, \mathcal{N} = 6$)	4.09	2.82×	748.48	4.97×	28.818	0.6557	0.4093
ClusCa ($O = 1, \mathcal{N} = 7$)	3.88	2.98×	674.12	5.52×	28.596	0.6276	0.4567
ClusCa + ours ($O = 1$)	**3.69**	**3.13**×	**635.46**	**5.85**×	**29.823**	**0.7100**	**0.3334**

the theoretical computation cost in FLOPs, alongside their respective speedup ratios compared to the full-step baselines.

4.2 Main Results

FLUX.1-dev The quantitative results in Table 1 demonstrate that our method achieves a superior balance between acceleration and generation quality on the FLUX architecture. Fig. 4 illustrates the visual comparison of various baseline models combined with our method running on FLUX. By integrating the Gaussian Process-based uncertainty quantification module into existing frameworks such as **TaylorSeer**, **HiCache**, and **ClusCa**, we significantly enhance image fidelity while maintaining approximately 3× latency speedup and 5.8× FLOPs reduction. Specifically, **ClusCa + ours** achieves the best PSNR of **29.823** and the lowest LPIPS of **0.3334** among all compared acceleration schemes. Compared to traditional step-reduction or baseline caching methods, the experimental data proves that our proposed refinement mechanism effectively mitigates error accumulation during the caching process.

Qwen-Image We further evaluate the proposed method on the larger Qwen-Image model to verify its scalability. As shown in Table 2, while the baseline (50 steps) requires a substantial latency of 36.68s, our GP-based refinement consistently achieves superior efficiency. Notably, the integration with **HiCache** and **TaylorSeer** enables a remarkable FLOPs speedup of over **6.2**×, outperforming most caching-based baselines in both speed and fidelity. For instance, **TaylorSeer + ours** improves the PSNR from 28.20 to **29.48** (+1.28 dB) and significantly reduces the LPIPS from 0.65 to **0.29** compared to the $\mathcal{N} = 7$ baseline. These results highlight that our method is particularly effective for large-

Table 2: Quantitative comparison in text-to-image gen. for Qwen-Image.

Method	Latency(s) ↓	Speed ↑	FLOPs(T) ↓	Speed ↑	PSNR↑	SSIM↑	LPIPS↓
50 steps	36.68	1.00×	12917.56	1.00×	-	-	-
20% **steps**	7.82	4.69×	2583.51	5.00×	28.59	0.61	0.52
FORA ($\mathcal{N} = 4$)	10.76	3.41×	3359.99	3.84×	28.66	0.59	0.51
FORA ($\mathcal{N} = 6$)	8.49	4.32×	2326.74	5.55×	28.48	0.55	0.59
ToCa ($\mathcal{N} = 8$)	16.82	2.18×	2991.34	4.32×	28.93	0.63	0.44
ToCa ($\mathcal{N} = 12$)	14.56	2.52×	2406.20	5.37×	28.69	0.57	0.53
DuCa ($\mathcal{N} = 9$)	10.27	3.57×	2958.13	4.37×	28.45	0.58	0.55
DuCa ($\mathcal{N} = 12$)	8.41	4.36×	2171.56	5.95×	28.38	0.57	0.60
TaylorSeer ($\mathcal{N} = 6$)	9.71	3.78×	2583.97	5.00×	28.58	0.62	0.46
TaylorSeer ($\mathcal{N} = 7$)	8.99	4.08×	2323.30	5.56×	28.20	0.48	0.65
TaylorSeer + ours	**8.37**	**4.38**×	**2085.26**	**6.19**×	**29.48**	**0.75**	**0.29**
HiCache ($\mathcal{N} = 6$)	9.88	3.71×	2583.97	5.00×	28.89	0.66	0.40
HiCache ($\mathcal{N} = 7$)	9.11	4.03×	2323.30	5.56×	28.64	0.63	0.45
HiCache + ours	**8.54**	**4.30**×	**2067.17**	**6.25**×	**29.39**	**0.74**	**0.32**
Qwen-Image-Lightning-8steps	7.10	1.00×	2123.17	1.00×	∞	1.00	0.00
TaylorSeer ($\mathcal{N} = 3$)	**4.56**	**1.56**×	**1326.98**	**1.60**×	30.064	0.6709	0.2962
TaylorSeer + ours ($\mathcal{N} = 3$)	4.69	1.51×	1326.99	1.60×	**31.204**	**0.7665**	**0.1922**

scale models, where standard caching tends to accumulate errors more rapidly. By leveraging Gaussian Process priors, we recover critical visual features that are otherwise lost under extreme acceleration.

4.3 Ablation Study

Table 3: Ablation on GP Rectification and Dynamic Interval Scheduling.

Method	FLOPs(T) ↓	Speed ↑	PSNR↑	SSIM↑	LPIPS↓
[dev]: 50 steps	3719.50	1.00×	-	-	-
TaylorSeer ($O = 2, \mathcal{N} = 8$)	596.07	6.24×	28.513(+0.00)	0.5986(+0.00)	0.4854(+0.00)
TaylorSeer + ours ($O = 2, \mathcal{N} = 8$)	596.08	6.24×	29.129(+0.62)	0.6426(+0.04)	0.4469(-0.04)
TaylorSeer ($O = 2, \mathcal{N} = 7$)	670.44	5.55×	28.671(+0.00)	0.6237(+0.00)	0.4542(+0.00)
TaylorSeer + ours ($O = 2, \mathcal{N} = 7$)	670.45	5.55×	29.266(+0.60)	0.6622(+0.04)	0.4151(-0.04)
TaylorSeer ($O = 2, \mathcal{N} = 6$)	744.80	4.99×	28.943(+0.00)	0.6558(+0.00)	0.4020(+0.00)
TaylorSeer + ours ($O = 2, \mathcal{N} = 6$)	744.81	4.99×	29.432(+0.49)	0.6746(+0.02)	0.3895(-0.01)
TaylorSeer + Only Dynamic ($O = 2$)	650.21	5.72×	28.930(+0.00)	0.6604(+0.00)	0.3819(+0.00)
TaylorSeer + ours ($O = 2$)	**643.13**	**5.78**×	**29.550**(+0.62)	**0.6972**(+0.04)	**0.3497**(-0.03)

Ablation on GP Rectification and Dynamic Scheduling. We conduct a two-fold ablation study to verify the individual effectiveness of our proposed GP rectifier and the uncertainty-gated dynamic scheduling mechanism.

First, to evaluate the impact of **GP rectification**, we compare the baseline **TaylorSeer** with our integrated version (**TaylorSeer + ours**) under various fixed skip intervals $\mathcal{N} \in \{6, 7, 8\}$. As shown in Table 3, the addition of our rectifier consistently yields substantial fidelity gains across all configurations; for instance, at $\mathcal{N} = 7$, the PSNR improves from 28.671 to 29.266, while the LPIPS drops from 0.4542 to 0.4151. This demonstrates that the GP rectifier

Fig. 4: Visual comparison of generated images before and after combining different baseline cache methods with GP-Refiner.

can effectively learn and compensate for the numerical residuals in the Taylor expansion features, regardless of the caching frequency.

Second, we investigate the necessity of **dynamic scheduling**. We compare **TaylorSeer + Only Dynamic**, which utilizes Taylor features for interval adjustment without GP rectification, against our full model. While the Taylor-based dynamic approach achieves a 5.72× speedup, its PSNR (28.930) is lower than even our fixed-interval version at $\mathcal{N} = 6$.

In contrast, our full configuration, **TaylorSeer + ours** ($O = 2$), which leverages GP-derived uncertainty for both scheduling and rectification, achieves the best performance with a PSNR of **29.550** and a significant LPIPS reduction to **0.3497**.

These results confirm that while both modules are independently effective, their synergy is crucial: the dynamic scheduling optimizes resource allocation based on model uncertainty, while the GP rectifier ensures that the approximated steps maintain high-frequency structural integrity.

Distribution of Dynamic Sampling Steps. To further investigate the stability and predictability of our dynamic interval scheduling strategy, we evaluated the distribution of the required Number of Function Evaluations (NFEs) on a diverse test set of **200 generated images** using the **TaylorSeer [14] + ours** configuration on the **FLUX [11]** model. Experimental results demonstrate that **GP-Refiner** consistently converges within a **highly concentrated range** of steps. Specifically, among the 200 evaluated samples, **77** images required **8 NFEs**, **119** images required **8 NFEs**, and only **4** images required **9 NFEs**.

This highly concentrated NFE distribution proves that our stopping criterion possesses **exceptional stability**. It not only **avoid sdrastic fluctuations** in

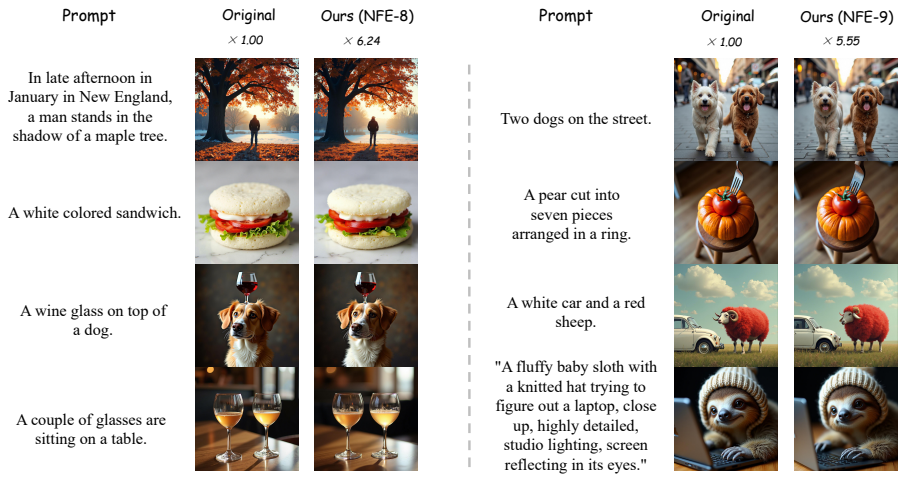

Fig. 5: Qualitative results of GP-Refiner (TaylorSeer + ours) on FLUX at NFE-8 and NFE-9.

sampling steps during inference but also ensures that the overall computational overhead remains **strictly controllable** throughout the inference process.

Furthermore, as illustrated in Figure 5, the difference in visual quality between images generated with 9 NFEs and 10 NFEs is **marginal**. Our dynamic allocation mechanism successfully **prunes redundant computations** while perfectly preserving the **high fidelity** and aesthetic quality of the images.

5 Conclusion

Addressing the challenge of cumulative feature cache errors faced by diffusion transformers during accelerated inference, a plug-and-play feature rectification framework, GP-Refiner, is proposed. Driven by the statistical observation that the baseline prediction residuals exhibit a local zero-mean Gaussian distribution, the framework transforms the acceleration process into a joint task of baseline estimation and rectification. Furthermore, utilizing the posterior variance derived from the regression, an uncertainty-aware adaptive computation strategy is designed to dynamically trigger full-computation calibration when the inference confidence falls below a threshold. Extensive experiments on large-scale generative models demonstrate that this mechanism significantly enhances image fidelity while substantially reducing computational complexity and accelerating inference speed, thereby establishing a new benchmark for the trade-off between computational efficiency and generation quality.

6 Acknowledgements

This paper was partially sponsored by WUYING - Alibaba Cloud.

Bibliography

[1] Blattmann, A., Dockhorn, T., Kulal, S., Mendelevitch, D., Kilian, M., Lorenz, D., Levi, Y., English, Z., Voleti, V., Letts, A., et al.: Stable video diffusion: Scaling latent video diffusion models to large datasets. arXiv preprint arXiv:2311.15127 (2023)

[2] Bu, J., Ling, P., Zhou, Y., Wang, Y., Zang, Y., Lin, D., Wang, J.: Dicache: Let diffusion model determine its own cache. In: The Fourteenth International Conference on Learning Representations (2026), https://openreview.net/forum?id=kflYZjGumW

[3] Chen, C., Hu, S., Zhu, J., Wu, M., Chen, J., Li, Y., Huang, N., Fang, C., Wu, J., Chu, X., et al.: Taming preference mode collapse via directional decoupling alignment in diffusion reinforcement learning. arXiv preprint arXiv:2512.24146 (2025)

[4] Chen, C., Zhu, J., Feng, X., et al.: S^2-guidance: Stochastic self guidance for training-free enhancement of diffusion models. arXiv preprint arXiv:2508.12880 (2025)

[5] Chen, Y., Zeng, L., ZHou, Y., Li, H., Zhai, J.: Jano: Adaptive diffusion generation with early-stage convergence awareness. arXiv preprint arXiv:2603.00519 (2026)

[6] Fang, G., Ma, X., Wang, X.: Structural pruning for diffusion models. arXiv preprint arXiv:2305.10924 (2023)

[7] Feng, L., Zheng, S., Liu, J., Lin, Y., Zhou, Q., Cai, P., Wang, X., Chen, J., Zou, C., Ma, Y., Zhang, L.: Hicache: Training-free acceleration of diffusion models via hermite polynomial-based feature caching (2025). https://doi.org/10.48550/arXiv.2508.16984

[8] Ho, J., Jain, A., Abbeel, P.: Denoising Diffusion Probabilistic Models (Dec 2020). https://doi.org/10.48550/arXiv.2006.11239, http://arxiv.org/abs/2006.11239, arXiv:2006.11239 [cs]

[9] Huang, R., Shao, S., Zhou, Z., Zhao, P., Guo, H., Ye, T., Bai, L., Yang, S., Xie, Z.: Accelerating diffusion model training under minimal budgets: A condensation-based perspective. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2026)

[10] Kong, W., Tian, Q., Zhang, Z., Min, R., Dai, Z., Zhou, J., Xiong, J., Li, X., Wu, B., Zhang, J., et al.: Hunyuanvideo: A systematic framework for large video generative models. arXiv preprint arXiv:2412.03603 (2024)

[11] Labs, B.F.: Flux. https://github.com/black-forest-labs/flux (2024)

[12] Li, S., Hu, T., Khan, F.S., Li, L., Yang, S., Wang, Y., Cheng, M.M., Yang, J.: Faster diffusion: Rethinking the role of unet encoder in diffusion models. arXiv preprint arXiv:2312.09608 (2023)

[13] Li, Z., Zhang, J., Lin, a.o.: Hunyuan-DiT: A powerful multi-resolution diffusion transformer with fine-grained chinese understanding. https://doi.org/10.48550/arXiv.2405.08748, http://arxiv.org/abs/2405.08748

[14] Liu, J., Zou, C., Lyu, Y., Chen, J., Zhang, L.: From reusing to forecasting: Accelerating diffusion models with taylorseers (2025). https://doi.org/10.48550/arXiv.2503.06923

[15] Liu, J., Zou, C., Lyu, Y., Li, K., Wang, S., Zhang, L.: Speca: Accelerating diffusion transformers with speculative feature caching. In: Proceedings of the 33rd ACM International Conference on Multimedia (MM '25). p. to appear. ACM, Dublin, Ireland (October 2025)

[16] Liu, X., Gong, C., et al.: Flow straight and fast: Learning to generate and transfer data with rectified flow. In: The Eleventh International Conference on Learning Representations (2023)

[17] Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., Zhu, J.: Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. Advances in Neural Information Processing Systems **35**, 5775–5787 (2022)

[18] Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., Zhu, J.: Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. arXiv preprint arXiv:2211.01095 (2022)

[19] Ma, X., Fang, G., Mi, M.B., Wang, X.: Learning-to-cache: Accelerating diffusion transformer via layer caching. arXiv preprint arXiv:2406.01733 (2024)

[20] Ma, X., Fang, G., Wang, X.: Deepcache: Accelerating diffusion models for free. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15762–15772 (2024)

[21] Peebles, W., Xie, S.: Scalable diffusion models with transformers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4195–4205 (2023)

[22] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-Resolution Image Synthesis with Latent Diffusion Models (Apr 2022). https://doi.org/10.48550/arXiv.2112.10752, http://arxiv.org/abs/2112.10752, arXiv:2112.10752 [cs]

[23] Saghatchian, O., Moghadam, A.G., Nickabadi, A.: Cached adaptive token merging: Dynamic token reduction and redundant computation elimination in diffusion model (2025)

[24] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S.K.S., Ayan, B.K., Mahdavi, S.S., Lopes, R.G., Salimans, T., Ho, J., Fleet, D.J., Norouzi, M.: Photorealistic text-to-image diffusion models with deep language understanding. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 24219–24238 (2022)

[25] Salimans, T., Ho, J.: Progressive distillation for fast sampling of diffusion models. arXiv preprint arXiv:2202.00512 (2022)

[26] Selvaraju, P., Ding, T., Chen, T., Zharkov, I., Liang, L.: Fora: Fast-forward caching in diffusion transformer acceleration. arXiv preprint arXiv:2407.01425 (2024)

[27] Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. In: International Conference on Learning Representations (2021)

[28] Song, Y., Dhariwal, P., Chen, M., Sutskever, I.: Consistency models. In: International Conference on Machine Learning. pp. 32211–32252. PMLR (2023)

[29] Tang, S., Wang, Y., Ding, C., Liang, Y., Li, Y., Xu, D.: Adadiff: Accelerating diffusion models through step-wise adaptive computation. In: European Conference on Computer Vision. pp. 73–90. Springer (2024)

[30] Wan, T., Wang, A., Ai, B., Wen, B., Mao, C., Xie, C.W., Chen, D., Yu, F., Zhao, H., Yang, J., Zeng, J., Wang, J., Zhang, J., Zhou, J., Wang, J., Chen, J., Zhu, K., Zhao, K., Yan, K., Huang, L., Feng, M., Zhang, N., Li, P., Wu, P., Chu, R., Feng, R., Zhang, S., Sun, S., Fang, T., Wang, T., Gui, T., Weng, T., Shen, T., Lin, W., Wang, W., Wang, W., Zhou, W., Wang, W., Shen, W., Yu, W., Shi, X., Huang, X., Xu, X., Kou, Y., Lv, Y., Li, Y., Liu, Y., Wang, Y., Zhang, Y., Huang, Y., Li, Y., Wu, Y., Liu, Y., Pan, Y., Zheng, Y., Hong, Y., Shi, Y., Feng, Y., Jiang, Z., Han, Z., Wu, Z.F., Liu, Z.: Wan: Open and advanced large-scale video generative models. https://doi.org/10.48550/arXiv.2503.20314, http://arxiv.org/abs/2503.20314

[31] Wimbauer, F., Wu, B., Schoenfeld, E., Dai, X., Hou, J., He, Z., Sanakoyeu, A., Zhang, P., Tsai, S., Kohler, J., et al.: Cache me if you can: Accelerating diffusion models through block caching. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6211–6220 (2024)

[32] Wu, C., Li, J., Zhou, J., Lin, J., Gao, K., Yan, K., ming Yin, S., Bai, S., Xu, X., Chen, Y., Chen, Y., Tang, Z., Zhang, Z., Wang, Z., Yang, A., Yu, B., Cheng, C., Liu, D.W., mei Li, D., Zhang, H., Meng, H., Wei, H., Ni, J.L., Chen, K., Cao, K., Peng, L., Qu, L., Wu, M., Wang, P., Yu, S., Wen, T., Feng, W., Xu, X.X., Wang, Y., Zhang, Y., Zhu, Y.A., Wu, Y., Cai, Y.J., Liu, Z.Y.: Qwen-image technical report. ArXiv **abs/2508.02324** (2025), https://api.semanticscholar.org/CorpusID:280422608

[33] Yan, H., Liu, X., Pan, J., Liew, J.H., Liu, Q., Feng, J.: Perflow: Piecewise rectified flow as universal plug-and-play accelerator. Advances in Neural Information Processing Systems **37**, 78630–78652 (2024)

[34] Yang, Z., Teng, J., Zheng, W., Ding, M., Huang, S., Xu, J., Yang, Y., Hong, W., Zhang, X., Feng, G., Yin, D., Gu, X., Zhang, Y., Wang, W., Cheng, Y., Xu, B., Dong, Y., Tang, J.: Cogvideox: Text-to-video diffusion models with an expert transformer. In: The Thirteenth International Conference on Learning Representations (2025), https://openreview.net/forum?id=LQzN6TRFg9

[35] Yuan, Z., Zhang, H., Pu, L., Ning, X., Zhang, L., Zhao, T., Yan, S., Dai, G., Wang, Y.: DiTFastattn: Attention compression for diffusion transformer models. In: The Thirty-eighth Annual Conference on Neural Information Processing Systems (2024), https://openreview.net/forum?id=51HQpkQy3t

[36] Zhang, E., Xiao, B., Tang, J., Ma, Q., Zou, C., Ning, X., Hu, X., Zhang, L.: Token pruning for caching better: 9 times acceleration on stable diffusion for free (2024), https://arxiv.org/abs/2501.00375

[37] Zhao, X., Jin, X., Wang, K., You, Y.: Real-time video generation with pyramid attention broadcast. arXiv preprint arXiv:2408.12588 (2024)
[38] Zheng, S., Feng, L., Wang, X., Zhou, Q., Cai, P., Zou, C., Liu, J., Lin, Y., Chen, J., Ma, Y., Zhang, L.: Forecast then calibrate: Feature caching as ode for efficient diffusion transformers (2025). https://doi.org/10.48550/arXiv.2508.16211
[39] Zheng, Z., Peng, X., Yang, T., Shen, C., Li, S., Liu, H., Zhou, Y., Li, T., You, Y.: Open-sora: Democratizing efficient video production for all (March 2024), https://github.com/hpcaitech/Open-Sora
[40] Zheng, Z., Wang, X., Zou, C., Wang, S., Zhang, L.: Compute only 16 tokens in one timestep: Accelerating Diffusion Transformers with Cluster-Driven Feature Caching. In: Proceedings of the 33rd ACM International Conference on Multimedia (MM '25). p. to appear. ACM, Dublin, Ireland (October 2025)
[41] Zhu, H., Tang, D., Liu, J., Lu, M., Zheng, J., Peng, J., Li, D., Wang, Y., Jiang, F., Tian, L., Tiwari, S., Sirasao, A., Yong, J.H., Wang, B., Barsoum, E.: Dip-go: A diffusion pruner via few-step gradient optimization (2024)
[42] Zou, C., Zhang, E., Guo, R., Xu, H., He, C., Hu, X., Zhang, L.: Accelerating diffusion transformers with dual feature caching (2024), https://arxiv.org/abs/2412.18911