SODA: Sensitivity-Oriented Dynamic Acceleration for Diffusion Transformer

Supplementary Material

Overview

This is the appendix for our paper titled *SODA: Sensitivity-Oriented Dynamic Acceleration for Diffusion Transformer*. This appendix supplements the main paper with the following content:

A. More Details of our SODA

To adapt to the complex internal sensitivity to acceleration, we propose SODA, a Sensitivity-Oriented Dynamic Acceleration method. As illustrated in Fig. **??**, SODA first conducts **(1) Offline Fine-grained Sensitivity Modeling (OFS)**: Defining error to measure the fine-grained sensitivity of timesteps, layers and modules before inference. Then, SODA adopts **(2) Dynamic Caching Scheduling Optimization (DCS)**: Employing dynamic programming to identify the optimal combination of cache intervals that yields the minimal sensitivity impact. Finally, when pruning and cache reuse, SODA proposes **(3) Unified Adaptive Strategy Formulation (UAS)**: Achieving adaptive scheduling for pruning timing and rate guided by sensitivity errors.

Due to space limitations in the main text, we provide additional analysis and implementation details of this module here.

A.1. Details of Offline Fine-grained Sensitivity Modeling (OFS)

Although the integration of caching and pruning balances the high acceleration of cache with the flexibility of pruning, existing fixed or heuristic methods struggle to accurately perceive the sensitivity introduced by such acceleration operations. As a result, they inevitably skip important computations, leading to degraded generation quality.

A.1.1. Sensitivity error and generation fidelity.

We emphasize the importance of the sensitivity error because it is the fundamental cause of the degradation in generation quality.

Once an acceleration strategy is determined, the diffusion inference results at each timestep and layer exhibit slight deviations due to acceleration operations such as cache reuse or pruning, thereby introducing computational errors. As the inference process unfolds, minor errors introduced by acceleration operations accumulate rapidly, often in an exponential manner. This accumulation results in substantial deviations from the ground-truth outputs, severely compromising generation quality. The essence of current acceleration optimization strategies lies in minimizing the acceleration-induced errors as much as possible, so that the accelerated diffusion process can closely approximate the original diffusion outputs.

However, the introduction of such errors is influenced by the model's intrinsic sensitivity to acceleration, which is highly complex and variable. The degree of sensitivity differs across models, timesteps, layers, and modules, making it difficult to capture through prior knowledge or fixed heuristic rules. As a result, existing static or heuristic acceleration strategies fail to fully account for these sensitivity-induced errors and inevitably lead to performance degradation.

To address this challenge, we propose a sensitivity quantification approach that models the fine-grained, dynamic distribution of sensitivity errors within different modules, enabling a more precise characterization of the model's response to acceleration.

A.1.2. Purpose of sensitivity error modeling.

The essential goal of our sensitivity error analysis is to:

- Through error analysis, we reveal that sensitivity varies significantly across different models, timesteps, layers, and modules.
- It highlights the limitations of current heuristic-based approaches and underscores the urgent need for adaptive methods that leverage the sensitivity error distribution to better balance acceleration and output quality.
- Based on the estimated sensitivity errors, we derive the optimal caching strategy through dynamic programming prior to inference by minimizing the cumulative sensitiv-

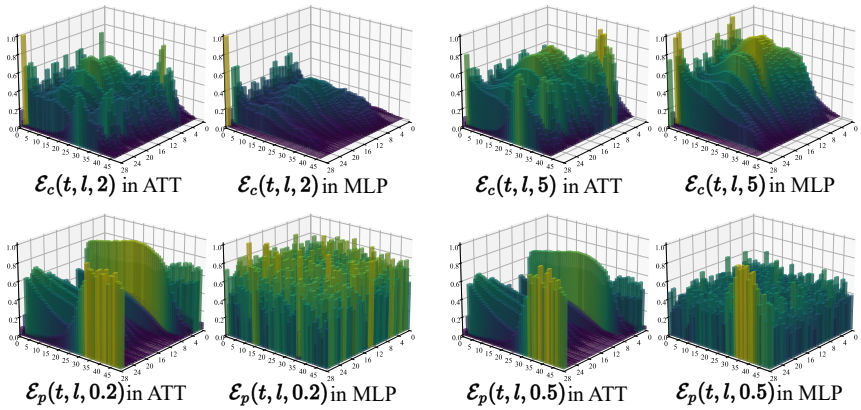

Figure 1. **Sensitivity error visualization of DiT-XL/2.** We visualize sensitivity errors of each module (attention is abbreviated as ATT) at every timestep and layer during the denoising process, under different cache intervals and pruning rates.

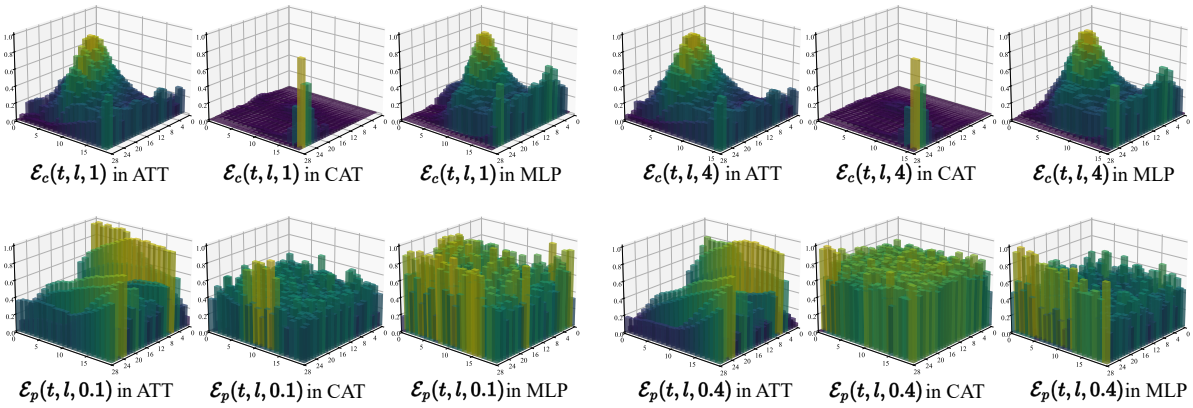

Figure 2. **Sensitivity error visualization of PixArt-α.** We visualize the sensitivity error of each module (attention is abbreviated as ATT, cross-attention as CAT) at every timestep and layer during the denoising process, under different cache intervals and pruning rates.

ity error. This not only reduces sensitivity errors but also avoids any overhead during the actual inference.

- By comparing the estimated pruning sensitivity error with the caching sensitivity error, we adaptively determine when to apply pruning operations. It allows us to retain a subset of tokens to mitigate sensitivity errors, thus achieving acceleration while simultaneously correcting for cache-induced quality degradation.

A.1.3. Details of sensitivity error.

As described in the main paper, we model the sensitivity errors caused by cache operations ($\mathcal{E}_c$) and pruning ($\mathcal{E}_p$) based on the Cosine distance between the accelerated outputs and the ground-truth outputs.

We compute both $\mathcal{E}_c$ and $\mathcal{E}_p$ separately across different models and modules. For $\mathcal{E}_c$, we evaluate the error under cache intervals ranging from 1 to 9. In contrast, since the pruning rate is a floating-point value rather than a discrete integer like the cache interval, we analyze the general trend of $\mathcal{E}_p$ by varying the pruning rate from 0.1 to 0.9 with a step size of 0.1.

During the detailed analysis, $\mathcal{E}_c$ is influenced by multiple variables: the models (DiT-XL/2, PixArt-α, OpenSora), modules (ATT, CAT, MLP), timesteps ($1-T$), layers ($1-L$), and cache intervals ($1-9$). $\mathcal{E}_p$ follows the same pattern. When analyzing a specific dimension, we average all other factors to observe the overall trend. For example, when comparing the error distribution across different models, we average the layers, modules, and operations, retaining only the model and timestep dimensions for comparison.

A.1.4. More sensitivity analysis.

We perform sensitivity error estimation analyses on DiT-XL/2, PixArt-α, and OpenSora to cover a wide range of tasks and model architectures.

In DiT, we visualize two different acceleration ratios under various acceleration operations, and compare the results across different modules in Fig. 1. It can be observed that even under the same acceleration operation, different mod-

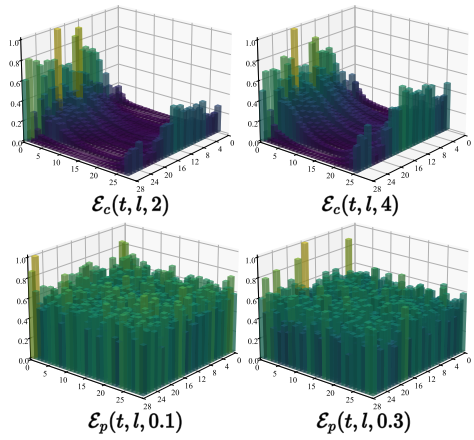

Figure 3. **Sensitivity error visualization of MLP in OpenSora.** We visualize sensitivity errors of each module at every timestep and layer during the denoising process, under different cache intervals and pruning rates.

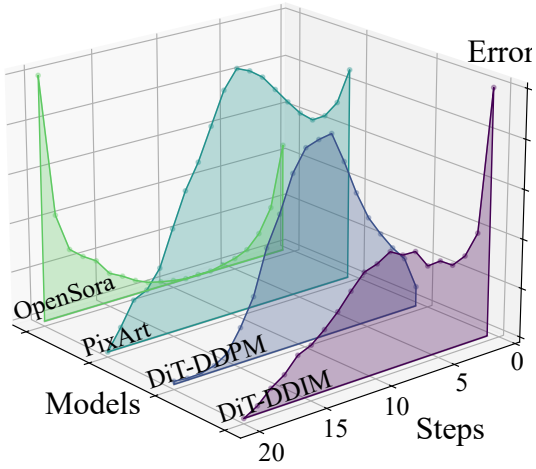

Figure 4. **Sensitivity error visualization of different models.** There exist distinct phases where the sensitivity error reduction is particularly pronounced in every model.

ules exhibit significantly different sensitivity error distributions. For example, attention modules are more sensitive to acceleration, while modules that do not emphasize contextual information like MLP tend to present error patterns that are nearly random under pruning strategy.

Furthermore, the discrepancy across different acceleration operations is significant. As a result, for our approach that combines caching and pruning, analyzing the sensitivity error characteristics of each acceleration strategy is critically important.

Similar analyses are performed on PixArt and OpenSora, where comparable discrepancies can be observed, as illustrated in Fig. 2 and Fig. 3. By comparing the analysis results across different models, we observe that while there are certain commonalities along specific dimensions (PixArt-α tends to exhibit higher sensitivity errors from caching-based acceleration in the middle stages of denoising, whereas OpenSora shows more concentrated errors at the beginning and end), these patterns are difficult to generalize at each module level. As a result, current heuristic approaches inevitably introduce relatively high sensitivity errors, which in turn lead to noticeable degradation in output quality.

Therefore, based on the sensitivity error estimation, our analysis leads to the following conclusions:

- Diffusion models exhibit highly diverse sensitivity error behaviors under different acceleration operations, with significant variations across models, timesteps, layers, and modules. Such complex sensitivity error distributions are difficult to accurately capture through manually crafted heuristics.
- The observed heterogeneity persists across diverse image and video generation tasks, regardless of content differences. This consistency supports its use as a prior for guiding inference-time decisions.

A.2. Details of Dynamic Caching Scheduling Optimization (DCS)

We propose a caching strategy formulated as a dynamic programming problem, where sensitivity errors are treated as step-wise costs. This allows us to adaptively determine the optimal caching intervals and anchor timestep sets that minimize the overall accumulated error.

The main idea and procedure of the algorithm have already been presented in the main paper. Here, we provide a supplementary explanation of the proposed algorithm, demonstrating how the dynamic programming procedure can be flexibly adapted to accommodate specific practical scenarios.

During the adaptive caching process, we observe that the DiT architecture typically exhibits a special stage in which the introduced error is abnormally reduced. As shown in Fig. 4, this phenomenon is consistently observed across all models in our experiments. This suggests that the DiT architecture possesses an inherent error-correction capability and demonstrates a degree of robustness to introduced sensitivity errors.

Since this special phase varies significantly across different models, we adopt an adaptive approach to incorporate this extension, in order to ensure the generality of our method.

We first compute the gradient $\nabla\mathcal{E}_{dp}$ of the step-wise cost $\mathcal{E}_{dp}$. After smoothing, this special stage can be identified by simply determining the sign of the gradient $\nabla\mathcal{E}_{dp}$. Within this identified stage, the candidate cache interval set $\mathcal{N}$ is restricted as $\mathcal{N}'$ by favoring smaller cache intervals, thus

Algorithm 1 Dynamic Caching Scheduling Optimization

Input: Total steps T, caching times N_s, cache interval candidate set $\mathcal{N}$, constrained cache interval candidate set $\mathcal{N}'$, sensitivity error here $\mathcal{E}_{dp}(t,n)$
Output: timestep set $\mathcal{A}$ and cache interval set $\mathcal{I}$

Initialize $dp[t][i] \leftarrow \infty$, $dp[T][1] \leftarrow 0$, $prev[t][i] \leftarrow$ None, $\forall t \in [T,1], \forall i \in [1,N_s]$
for $i = 1$ to N_s **do**
  for $t = T$ down to 1 **do**
    if $\Delta\mathcal{E}_{dp}(t,n) < 0$ **then**
      $\mathcal{N}^* \leftarrow \mathcal{N}'$
    else
      $\mathcal{N}^* \leftarrow \mathcal{N}$
    for all $n \in \mathcal{N}^*$ **do**
      if $t > 0$ **then**
        if $dp[t][i+1] > dp[t+n][i] + \mathcal{E}_{dp}(t,n)$ **then**
          $dp[t][i+1] \leftarrow dp[t+n][i] + \mathcal{E}_{dp}(t,n)$
          $prev[t][i+1] \leftarrow (t,n)$
Backtracking: $\mathcal{A} \leftarrow \{\}, \mathcal{I} \leftarrow \{\}, t \leftarrow 1, i \leftarrow N_s$
while $i > 0$ **do**
  $(t_{\text{next}}, n) \leftarrow \text{prev}[t][i]$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{t\}, \mathcal{I} \leftarrow \mathcal{I} \cup \{n\}$
  $t \leftarrow t_{\text{next}}, i \leftarrow i-1$
return $\mathcal{A}, \mathcal{I}$

preserving the inherent robustness of DiT to perturbations.

With this approach, we are able to explicitly retain more timesteps that fall within this special phase during the dynamic programming process, thereby preserving the inherent noise-suppression capability of the DiT architecture.

The extended algorithm, as shown in Alg. 1 (we highlight the extended parts in blue font), primarily narrows the search space during the optimal substructure search by introducing targeted constraints on specific timesteps. Specifically, suppose that the original candidate set $\mathcal{N} = \{n | n \in \mathbb{Z}, 2 \leq n \leq 6\}$. We can define a reduced set $\mathcal{N}' = \{n | n \in \mathbb{Z}, 2 \leq n \leq 3\}$, which contains only half of the elements of the original one. This allows the dynamic programming process to adopt a more conservative caching interval during the special phase.

By extending this idea, we demonstrate the flexible application of dynamic programming in caching strategies, contributing to a deeper integration of this mathematical framework with DiT-based acceleration. We plan to further investigate this integration in future work.

A.3. Details of Unified Adaptive Strategy Formulation (UAS)

In the main paper, we have already explained whether pruning is applied at each timestep and the corresponding pruning rate:

$$x' = \begin{cases} \mathcal{C}_{l,m}(\Omega), \ if\ \delta_{t+1,l,m} \geq \mathcal{E}_c(t,l,m,n), \\ \mathcal{P}_{\alpha_{t+1,l,m},\gamma}(x) \cup \mathcal{C}_{l,m}(\gamma), \ else. \end{cases} \tag{1}$$

if pruning is triggered, we compute the mean value of the feature activations at the current node as the importance metric. According to the corresponding pruning rate, the top-a tokens are then selected based on the Top-K operation. These important tokens continue to participate in subsequent computations, while the remaining tokens are replaced with the cached results from previous steps.

We use the feature mean to measure token importance rather than employing the modeled sensitivity error, mainly because the pruning positions are highly correlated with the generated content. As previously discussed, our modeled sensitivity error is primarily model-specific and largely independent of the generation content. We use the feature mean instead of the more common attention weights because obtaining attention weights requires explicitly constructing intermediate attention computations, which conflicts with the widely adopted FlashAttention acceleration mechanism.

During pruning, both the pruning timing and pruning rate are fully determined by the offline-modeled sensitivity error, thus introducing no additional computational overhead during inference. The cost of computing the pruning positions is already included in the reported inference speed of the main experiments.

A.4. Overall Analysis

Generalization and Limitation. Our sensitivity modeling is analogous to downloading model weights, only a one-time setup is required for permanent use. When the model is fine-tuned or changed with a new model, SODA requires sensitivity re-modeling. Note that it is fully automated, without manual heuristics, and generalizes across different image sizes, CFGs and seeds on the same model.

Scalability. SODA's modeling is conducted via random generations, so its time depends on the model inference. Larger models incur longer modeling time. As modeling is insensitive to sample count, we reduce it to mitigate overhead (e.g., 10 for video vs. 100 for image models).

B. More Experiments

In the experiments, we provide the model implementation details omitted in the main text, along with the parameters used for the baselines and the specifications of the benchmarks and evaluation metrics.

B.1. More Implementation Details

We apply the SODA model to DiT-XL/2, PixArt-α, and OpenSora, which represent category-to-image,

Model	Sampling	Latency(s)↓	FLOPs(G)↓	Speed↑	FID↓	sFID↓	IS↑	Precision↑	Recall↑
DiT	DDPM	2.508	118.68	1.00×	2.23	4.57	275.65	0.82	0.58
SODA (N_s=125)	DDPM	1.778	76.70	1.55×	2.21	4.72	272.01	0.82	0.58
SODA (N_s=72)	DDPM	1.070	43.42	2.73×	2.47	5.09	262.30	0.81	0.59
SODA (N_s=69)	DDPM	0.839	35.25	3.37×	2.88	6.11	252.03	0.80	0.59
SODA (N_s=63)	DDPM	0.704	29.66	4.00×	3.33	6.76	237.69	0.78	0.60
DiT	DDIM	0.533	23.74	1.00×	2.25	4.33	239.97	0.80	0.59
SODA (N_s=31)	DDIM	0.433	15.61	1.52×	2.37	4.51	242.22	0.82	0.59
SODA (N_s=25)	DDIM	0.314	12.21	1.94×	2.39	4.55	240.75	0.81	0.58
SODA (N_s=18)	DDIM	0.263	9.55	2.49×	2.75	4.56	235.65	0.80	0.58
SODA (N_s=12)	DDIM	0.162	6.25	3.80×	3.62	5.88	228.41	0.79	0.59

Table 1. **More results of DiT-XL/2 on class-conditional image generation.** The arrow denotes whether lower or higher values indicate superior performance. N_s represents the number of cache intervals and is used to adjust the acceleration budget. All speeds are measured on RTX 4090 GPU.

text-to-image, and text-to-video generation tasks, respectively.

B.1.1. DiT-XL/2

Experiments conducted on DiT-XL/2 [33] employ both DDPM [15] and DDIM [43] samplers. The DDPM and DDIM samplers that we adopt consist of 250 and 50 steps, respectively.

For DDPM, implementation of different methods is as follows: FORA [40] adopts a fixed caching interval of $\mathcal{N} = 3$. ToCa [55] employs an average caching ratio of $R = 93\%$ and an average caching interval of $\mathcal{N} = 4$, and DuCa [56] employs a caching ratio of $R = 95\%$ and an average caching interval of $\mathcal{N} = 3$. For our method SODA, we perform a pre-inference error analysis using 100 images, setting $\lambda = 0.3$ and $\beta = 0.4$. The overall acceleration efficiency is controlled via the number of caching intervals, denoted by N_s.

For DDIM, FORA adopts a fixed caching interval of $\mathcal{N} = 3$. ToCa employs a caching ratio of $R = 93\%$ and an average caching interval of $\mathcal{N} = 3$, and DuCa employs an average caching ratio of $R = 95\%$ and an average caching interval of $\mathcal{N} = 3$. The basic settings of our SODA are kept consistent with those in DDPM.

B.1.2. PixArt-α

For PixArt-α [3], we adopt the DPM++ [28] with 20 steps as sampler. The implementation of different methods is as follows: FORA adopts a fixed caching interval of $\mathcal{N} = 2$. ToCa employs an average caching ratio of $R = 70\%$ and an average caching interval of $\mathcal{N} = 3$, and DuCa employs an average caching ratio of $R = 25\%$ and an average caching interval of $\mathcal{N} = 3$. For our method SODA, we perform a pre-inference error analysis using 100 images, setting $\lambda = 0.4$ and $\beta = 0.8$. The overall acceleration efficiency is controlled via the number of caching intervals, denoted by N_s.

B.1.3. OpenSora

Experiments on OpenSora are conducted with 30 reflective flow steps. The implementation of different methods is as follows: FORA adopts a fixed caching interval of $\mathcal{N} = 2$. ToCa sets different average caching intervals for different modules, with $\mathcal{N} = 3$ for temporal attention, spatial attention, and MLP, and $\mathcal{N} = 6$ for cross-attention. The average caching ratio is set to 100% for all modules except the MLP, which has an average caching ratio of 85%. DuCa adopts the same scheme as ToCa. Δ-DiT [5] divides the complete denoising process into two stages. PAB [54] sets the caching interval to $\mathcal{N} = 2$ for temporal attention, $\mathcal{N} = 4$ for spatial attention, $\mathcal{N} = 6$ for cross-attention. The MLP module is fully computed. For our method SODA, we perform a pre-inference error analysis using 10 videos to reduce the time cost of error analysis, setting $\lambda = 0.1$ and $\beta = 0.0$ for higher acceleration. The overall acceleration efficiency is controlled via the number of caching intervals, denoted by N_s.

B.1.4. Evaluation and metrics

For class-conditional image generation, we use DiT-XL/2 to randomly generate 50k 256×256 images from ImageNet [9], use FID [14], sFID, IS [39], Precision, and Recall to measure the quality of the generated images. The Fréchet Inception Distance (FID) measures the distributional discrepancy between real and generated images in the Inception-V3 feature space, where a lower score indicates higher fidelity and realism. Its spatial variant, sFID, computes feature distribution differences at the spatial level rather than the global pooling layer, making it more sensitive to structural and texture coherence. The Inception Score (IS) evaluates image quality and diversity based on the confidence and entropy of predicted class probabilities from an Inception network. A higher score implies clearer and more diverse generations. Precision quantifies perceptual fidelity by measuring how many generated samples fall

within the manifold of real data, while Recall evaluates diversity by estimating the coverage of the real image distribution. Together, these metrics provide a comprehensive evaluation of the trade-off between fidelity and diversity in class-conditional image generation.

For text-to-image generation, we evaluate performance using FID-30K and the CLIP Score [13], which jointly assess generation quality and text-image alignment of images generated by PixArt-α. FID-30K is computed between 30,000 generated images and the corresponding real images from the MS-COCO2017 dataset[22], measuring the Fréchet distance between their feature distributions in the Inception-V3 embedding space. A lower value indicates that the generated images are closer to real ones, reflecting higher visual fidelity and realism. The CLIP Score evaluates semantic consistency between an image and its associated text prompt by computing the cosine similarity of their representations in the CLIP model's joint vision-language embedding space. A higher CLIP Score denotes stronger semantic alignment and better text-image correspondence. Together, these two metrics provide complementary perspectives on both the perceptual quality and the semantic faithfulness of text-to-image generation.

For text-to-video generation, we adopt VBench [17] to comprehensively assess the quality of generated videos. We use 950 text prompts and generate five videos per prompt, resulting in a total of 4,750 videos, each rendered at 480p resolution with a 9:16 aspect ratio and 2 seconds duration. VBench evaluates 16 aspects covering semantic alignment, visual quality, temporal consistency, and aesthetic appeal, providing a holistic measurement of video generation performance.

B.2. More Experiment Results

To supplement the experimental results presented in the main paper and further demonstrate the effectiveness of our method, this section includes additional acceleration results and ablation studies.

B.2.1. More results of Dit-XL/2

We provide additional results of the SODA method under various acceleration ratios in Tab. 1.

It can be observed that our method preserves most of the model performance under low acceleration ratios. Due to the minimal accumulated cache error obtained via dynamic programming and the adaptive pruning decision that further reduces the cache error, our method can even improve the performance of the original model in certain cases. For example, the results of DDPM with N_s=125 and DDIM with N_s=31.

Moreover, as the acceleration ratio continues to increase, the model inevitably begins to omit important computations after removing redundant ones, leading to a gradual degradation in performance.

Model	Latency(s)↓	FLOPs(G)↓	VBench(%)↑			
			Total	Imaging quality	Subject consistency	Scene
OpenSora	102.287	3283.20	79.13	**61.14**	95.09	52.46
SODA (N_s=21)	**86.684**	**2319.94**$_{1.42\times}$	**79.13**	60.63	**95.26**	**53.35**
Δ-DiT (ArXiv24)	-	3166.47	78.21	-	-	-
PAB (ICLR25)	-	2558.25	76.95	-	-	-
FORA (ArXiv24)	-	1751.32	76.91	-	-	-
ToCa (ICLR25)	54.698	1394.03	78.34	56.04	93.92	50.21
DuCa (ICLR25)	50.637	1315.62	78.39	56.54	94.02	50.41
SODA (N_s=12)	49.896	1314.42	**78.49**	**58.14**	**94.60**	**53.33**
SODA (N_s=11)	**48.022**	**1277.59**$_{2.57\times}$	78.41	57.35	94.14	52.00

Table 2. **More VBench results of OpenSora on text-to-video generation. All speeds are measured on 80G A100 GPU.**

This phenomenon reflects the natural trade-off between generation quality and acceleration. The advantage of our method lies in its ability to improve model performance under low acceleration ratios and significantly delay the trade-off point, thereby preserving more of the model's performance even as the acceleration increases.

B.2.2. More results of OpenSora

We have reported the overall performance on VBench for OpenSora in the main paper. However, since the VBench metric aggregates multiple evaluation criteria with varying importance, the differences between methods on this metric are often minor (typically less than 1), which hinders a clear analysis of our model's advantages.

Considering that VBench primarily focuses on videos depicting motion of specific objects, we argue that analyzing the corresponding aspects of quality, consistency, and overall perceptual experience is more important.

Therefore, we report the relevant detailed metrics within VBench in Tab. 2 to further highlight the effectiveness of our method. It can be observed that under a low acceleration ratio (1.42×), our method even outperforms the original model in terms of subject consistency and scene perception, indicating that it is capable of improving performance during acceleration in video generation models.

Moreover, even at the same or higher acceleration ratios compared to the baseline, our method consistently maintains superior performance, with particularly significant advantages in metrics such as image quality. This indicates that our method can effectively mitigate the error introduced by acceleration, achieving speed-up while preserving generation quality.

B.2.3. More generation models

We extend experiments to Qwen-Image [50] (image) and Wan2.1 [47] (video) to demonstrate our effectiveness. In Tab. 3, "Base" is a baseline caching steps with fixed intervals (ToCa/DuCa require manual heuristics, limiting their transferability). We achieve higher fidelity under the same acceleration.

Qwen-Img	Lat.(s)↓ /Spe.↑	CLIP↑	IR↑
Origin	$73.431_{1.00\times}$	35.69	1.0242
Base	$41.135_{1.79\times}$	35.14	0.8677
Ours	$\mathbf{40.749}_{1.80\times}$	**35.72**	**1.0352**

Wan2.1	Lat.(s)↓ /Spe.↑	Vbench (%)↑
Origin	$187.46_{1.00\times}$	78.97
Base	$118.82_{1.58\times}$	74.24
Ours	$\mathbf{119.46}_{1.57\times}$	**75.61**

Table 3. Results on more generation models.

DiT-XL/2	Lat.(s)↓	Spe.↑	IS↑	FID↓	sFID↓	P↑	R↑
DDPM/250	2.508	1.00×	275.65	2.23	4.57	0.82	0.58
Δ-DiT [5]	-	1.52×	271.03	2.31	-	-	-
Ours	1.778	1.41×	**272.01**	**2.21**	**4.72**	**0.82**	0.58
Ours	**1.651**	**1.52×**	271.77	2.23	4.76	**0.82**	**0.59**
DDIM/20	0.243	1.00×	223.23	3.51	4.92	0.79	0.57
GOC(FORA) [34]	0.167	1.46×	191.44	6.78	8.74	0.74	0.53
Ours	0.168	1.45×	**205.05**	**5.03**	**5.80**	**0.76**	**0.56**

Table 4. Comparison with more baselines.

Training caching	Lat.(s)↓ /Spe.↑	IS↑	FID↓	sFID↓
DiT-XL/2	$0.533_{1.00\times}$	239.97	2.25	4.33
L2C [29]	$\text{None}_{1.25\times}$	233.26	2.62	4.50
HarmoniCa [16]	$\text{None}_{1.30\times}$	238.74	2.36	4.24
Ours	$\mathbf{0.403}_{1.32\times}$	**241.71**	2.37	4.58

Distillation one-step	Spe.↑	Fidelity↑ ($\frac{\text{Method}}{\text{Origin}}$)(%)	Costs↓ GPU/time
DiT-XL/2	1×	100	-
ShortCut [10]	∼50×	∼40	TPUV3s/2days
π-flow [2]	∼50×	∼78	Not mentioned
Ours	∼4×	**∼98**	**Free**

Table 5. Comparison with training-based caching and few-step distillation.

Model	VBench(%)↑			
	Total	Imaging quality	Subject consistency	Scene
Vanilla	76.41	54.23	93.54	50.51
OFS w/ DCS	78.37	56.54	94.12	53.41
OFS w/ UAS	78.41	57.55	94.07	53.03
SODA (Ours)	**78.49**	**58.14**	**94.60**	**53.33**

Table 6. **Ablation study of OpenSora on text-to-video generation.**

B.2.4. More baselines

We include more training-free baselines and achieve superior fidelity in Tab. 4 (Δ-DiT is not open-sourced, so we adopt its setting).

B.2.5. Comparison with training methods.

In Tab. 5, compared to training caching, SODA achieves the highest IS and outperforms L2C in FID without training. Compared to distillation, SODA aims for a better balance between acceleration and fidelity, while distillation trades extremely training cost and generation degradation for maximal acceleration. Overall, SODA's advantage is training-free nature, enabling generalization without manual heuristics.

FID↓		β				
		0.4	0.5	0.6	0.7	0.8
λ	0.4	27.46	27.44	27.42	27.38	27.33
	0.5	27.44	27.42	27.37	27.33	27.32
	0.6	27.36	27.35	27.34	27.33	27.32

Table 7. **The impact of parameters λ and β on PixArt-α.**

DiT-XL/2	FlashAtt	Lat.(s)↓	Spe.↑	IS↑	FID↓	sFID↓	P↑	R↑
Mean	✓	0.201	2.65×	**233.22**	**2.80**	**4.61**	**0.80**	**0.58**
Variance	✓	0.211	2.53×	233.20	2.82	2.67	0.80	0.58
Norm	✓	**0.197**	**2.70×**	233.18	2.83	4.66	0.80	0.58
Attn weights	✗	0.300	1.78×	233.25	2.81	4.63	0.80	0.58

Table 8. Ablation study for pruning location.

B.3. More Ablation Study

B.3.1. More ablation study on OpenSora.

To demonstrate the adaptability and generalization ability of our method across different tasks, we not only report additional main experimental results, but also include ablation studies conducted on OpenSora.

As shown in Tab. 6, compared to the vanilla caching strategy, our proposed DCS module improves performance by 1.96% on average by minimizing cumulative caching error through dynamic programming based on estimated error. In addition, the adaptive pruning mechanism (UAS) corrects caching error by retaining important tokens, resulting in a 2% improvement. The combination of both modules leads to an average increase of 2.08 on the VBench, with nearly a 4% gain in imaging quality.

On the one hand, this demonstrates the effectiveness of our proposed modules; on the other hand, it highlights the general applicability of our SODA.

B.3.2. More parameter analysis on PixArt-α.

To further demonstrate the robustness of the hyperparameters introduced by the UAS module, in addition to the parameter space reported on DiT in the main text, we also report results on the parameter space explored on PixArt in Tab. 7.

It can be observed that the FID fluctuates within a narrow range of 0.14 across the entire parameter space, indicating a high degree of robustness and suggesting that our method is largely insensitive to the hyperparameters introduced in UAS.

Typically, λ is fixed. Given acceleration factor $r > 1$, we first obtain the number of cache interval $N_s = N/r - 1$ (leave room for pruning), then select β: $\sum_t^T \sum_l^L \{\lambda \mathcal{E}_c(t,l,n) + \beta\} = 1/r$ (omit transformation from t to n), it means the proportion of pruned tokens is the inverse of speed-up factor. Due to the adaptive pruning decisions, β may fluctuate slightly, but it has negligible impact on acceleration factor.

Figure 5. **Visualization results of different acceleration methods on DiT-XL/2.** Comparisons are highlighted with red dashed boxes.

B.3.3. More comparison on pruning methods

In Tab. 8, we use feature mean and top-K selection to determine pruning positions, to identify important tokens with a simple and efficient way, **avoiding** large time overhead. Attention weights are excluded due to incompatibility with FlashAttention, despite their effectiveness.

C. More Visualization

We provide additional visual comparisons of acceleration results, along with detailed example analyses.

C.1. DiT-XL/2

It can be observed that our method preserves most of the model performance under low acceleration ratios. Due to the minimal accumulated cache error obtained via dynamic programming and the adaptive pruning decision that further reduces the cache error, our method can even improve the performance of the original model in certain cases. For example, the results of DDPM with N_s=125 and DDIM with N_s=31.

Moreover, as the acceleration ratio continues to increase, the model inevitably begins to omit important computations after removing redundant ones, leading to a gradual degradation in performance.

This phenomenon reflects the natural trade-off between generation quality and acceleration. The advantage of our method lies in its ability to improve model performance under low acceleration ratios and significantly delay the trade-off point, thereby preserving more of the model's performance even as the acceleration increases.

Our method SODA obtains further acceleration, with the generation speed 180% faster than the original model, and 12% faster than DuCa. The visualization results are shown in Fig. 5. Compared with DuCa, SODA-generated images demonstrate finer detail clarity (e.g., the face of the dogs in the upper Col.C and the lower Col.F), while images from DuCa exhibits notable degree of distortion.

SODA-generated images possess better background texture and overall alignment with the images from the original model (e.g., the background of the woodlouse in upper Col.B and the overall structure of the bridge in lower

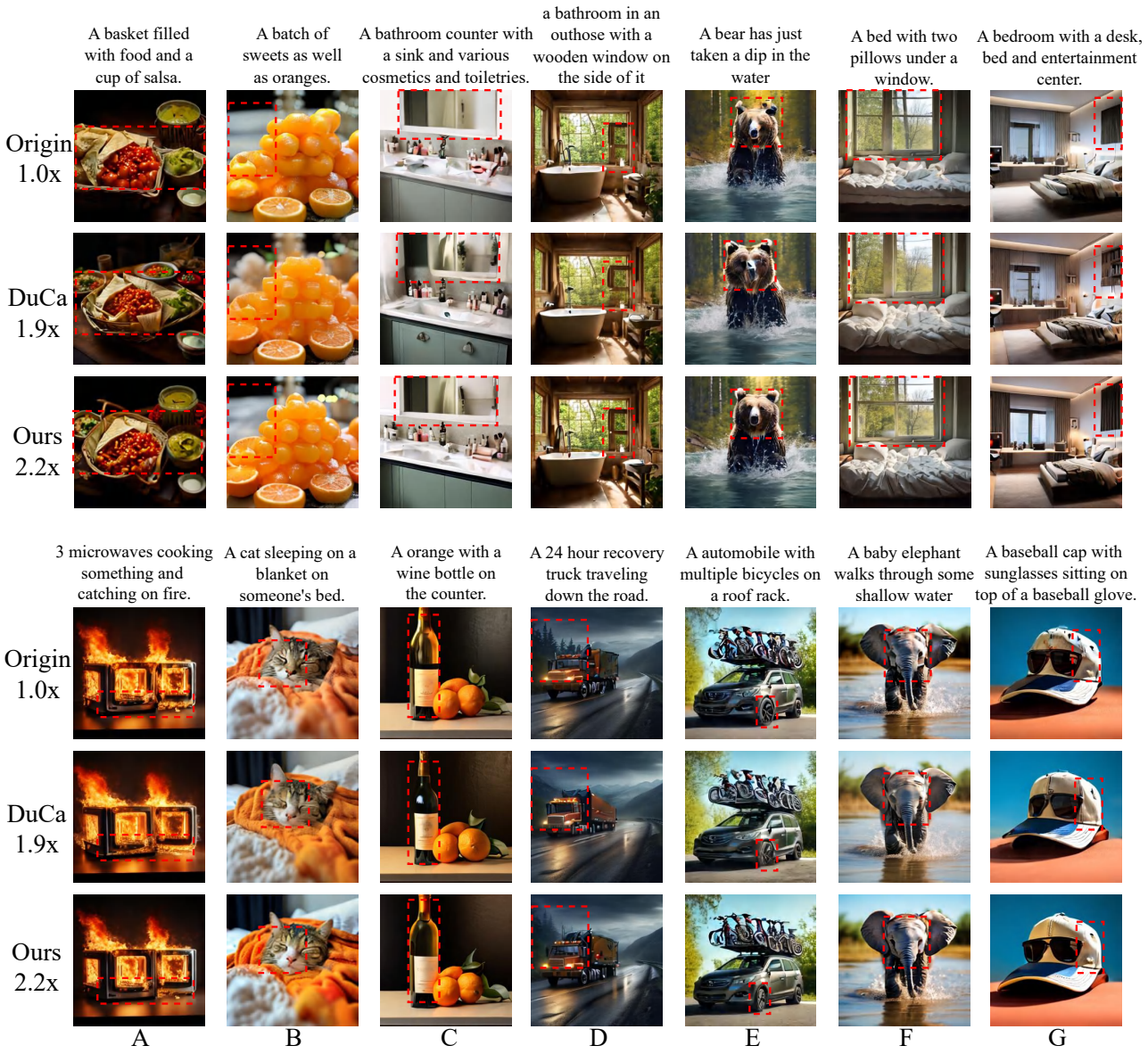

Figure 6. **Visualization results of different acceleration methods on PixArt-α.**

Col.A).

SODA also mitigates some of the defects in DuCa-generated images. For example, numeracy confusion (e.g., a third duck by DuCa in lower Col.E, and a second hot-air balloon by DuCa in upper Col.D) and redundant attributes(e.g., the hat on the dog head by DuCa in upper Col.H).

C.2. PixArt-α

Our method SODA achieves a higher acceleration ratio of $2.2\times$ compared with the $1.9\times$ of DuCa. The visualization results are shown in Fig. 6. SODA-generated images exhibit better detail and overall consistency with those generated by the original model (e.g., the basket of food in upper Col.A and the details of the recovery truck and the bear in lower Col.C and upper Col.D, respectively).

It should be noted that our method sometimes even demonstrate superior generation quality compared with the original model (e.g., the right ear of the bear in upper Col.E is more complete, and the light shade on the wine bottle in lower Col.C is more real). This aligns with our quantitative analysis in the Experiment Section.

C.3. OpenSora

Our method SODA achieves an extra $0.1\times$ acceleration based on DuCa, which is now 160% faster than the orig-

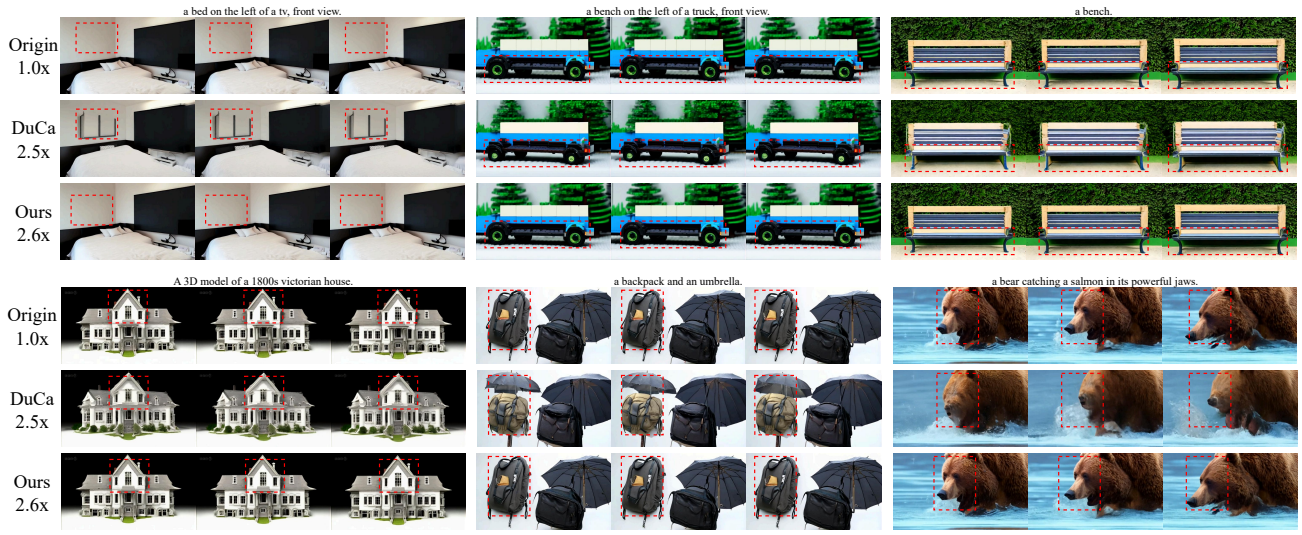

Figure 7. **Visualization results of different acceleration methods on OpenSora.**

inal model. The visualization results are shown in Fig. 7. Compared with DuCa, our method mitigates problems such as object redundancy (e.g., the redundant decoration above the bed by DuCa in the upper left frames) and detail distortion (e.g., the distorted face of the bear by DuCa in the lower right frames).

Our method can also better maintain the consistency with the original model (e.g., in the lower middle frames, DuCa-generated frames contain two umbrellas and the color of the backpack on the left is khaki, while there is only one umbrella and the color of the backpack on the left is gray in the frames generated by both the original model and our method).

Moreover, our method can better handle the relationship of foreground objects and background. For example, in the upper right frames, there exists a clear boundary between the foreground bench and the background bushes in the frames generated by our method. However, in the frames generated by DuCa, the edges of the bench already blend slightly with the bushes.

As illustrated by the visual examples, SODA effectively reduces the caching error while achieving acceleration.

D. Related Work

D.1. Diffusion Transformer

D.1.1. Diffusion Model.

Diffusion models [15, 35] synthesize visual content from pure noise by progressively denoising it through noise prediction. The training and inference process of diffusion models can be divided into two stages: noise diffusion stage and reverse denoising stage, respectively.

During the noise diffusion stage, noise of different magnitudes is added to the original image from the training set with respect to the timestep $t \in [1, T]$, where T is the total number of steps. The noise diffusion process at timestep t can be expressed as:

$$\boldsymbol{x_t} = \sqrt{\bar{a}_t}\boldsymbol{x_0} + \sqrt{1-\bar{a}_t}\boldsymbol{\epsilon}, \tag{2}$$

where $\boldsymbol{x_t}$ is the noisy image at timestep t, $\boldsymbol{x_0}$ is the original image, a_t is a constant related to t that controls the noise magnitude, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is the random noise sampled from the Gaussian distribution.

The model then learns to predict the noise added at a certain timestep by minimizing the following objective [15]:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_t[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon_\theta}(\sqrt{\bar{a}_t}\boldsymbol{x_0} + \sqrt{1-\bar{a}_t}\boldsymbol{\epsilon}, t)||^2], \tag{3}$$

where $\boldsymbol{x_0} \sim q(\boldsymbol{x}), \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, $q(\boldsymbol{x})$ is the distribution of the training data, and $\boldsymbol{\epsilon_\theta}$ is the estimated noise with the noisy image $\boldsymbol{x_t}$ and timestep t as input, parameterized by $\boldsymbol{\theta}$.

During the reverse denoising stage, the trained model predicts the noise added at a certain timestep and removes it to obtain a less noisy image. When reversing from timestep T to 1, random noise can be gradually denoised to a specific image. The specific denoising process varies among different solvers. Taking DDPM [15] as an example, the denoising process from $\boldsymbol{x_t}$ to $\boldsymbol{x_{t-1}}$ can be modeled as:

$$\boldsymbol{x_{t-1}} = \frac{1}{\sqrt{a_t}}(\boldsymbol{x_t} - \frac{1-a_t}{\sqrt{1-\bar{a}_t}}\boldsymbol{\epsilon_\theta}(\boldsymbol{x_t}, t)) + \sigma_t \boldsymbol{z}, \tag{4}$$

where $a_t, \bar{a}_t, \sigma_t$ are constants related to t, and $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

D.1.2. Diffusion Transformer.

Diffusion transformer [33] integrates the transformer [46] architecture into the diffusion process, enabling improved

controllability and higher generation quality compared with U-Net [36]. It has a hierarchical structure consisting of a total number of L DiT blocks. The architecture of DiT and within a specific DiT block l can be expressed as follows:

$$\begin{aligned} \mathcal{G} &= g_1 \circ g_2 \circ \cdots \circ g_L, \\ g_l &= F^l_{\mathrm{SA}} \circ F^l_{\mathrm{CA}} \circ F^l_{\mathrm{MLP}}, \end{aligned} \tag{5}$$

where $\mathcal{G}$ denotes the DiT model, g_l denotes the DiT block, SA denotes the self-attention layer, CA denotes the cross-attention layer, and MLP denotes the multilayer perceptron.

Like standard Transformers, the input image of size $H \times W$ is segmented into a number of spatial patches of size $p \times p$, resulting in a total of $\frac{H}{p} \times \frac{W}{p}$ patches. The input $\boldsymbol{x}$ to DiT is a sequence of tokens corresponding to the patches. When $\boldsymbol{x}$ enters a DiT block, it will go through a residual connection, expressed as follows:

$$f^l(\boldsymbol{x}) = \boldsymbol{x} + \mathrm{AdaLN} \circ F^l_{\star}(\boldsymbol{x}), \tag{6}$$

where l denotes the index of DiT block, AdaLN denotes the adapative layer normalization and $\star \in \{\mathrm{SA}, \mathrm{CA}, \mathrm{MLP}\}$.

Although DiT has achieved substantial progress, its slow inference process remains a major bottleneck, limiting its broader adoption and real-world applicability. The computational overhead largely comes from the attention layer, which has a time complexity of $\mathbf{O}(\boldsymbol{N^2})$, where $\boldsymbol{N}$ is the number of input tokens.

D.2. Training-free DiT acceleration

In this paper, we investigate training-free acceleration approaches that do not require additional training or distillation, thereby avoiding the significant computational overhead typically associated with post-training optimization. Existing training-free acceleration methods for diffusion transformers can generally be categorized into two paradigms: feature caching, which reuses intermediate activations for efficiency, and token pruning, a strategy adapted from token reduction techniques originally developed for large language models (LLMs).

D.2.1. Caching-based Acceleration

Recently, feature caching has emerged as a training-free method that reuses the results from previous timesteps or layers to avoid redundant computation. Caching-based methods can be categorized into two types according to their granularity: layer-wise caching and token-wise caching.

1) Layer-wise caching can refer to caching the layers in the UNet [36] or DiT [33] backbones or caching the specific attention or MLP modules. In terms of UNet-based diffusion models, DeepCache [30] and FasterDiffusion [20] cache intermediate feature output of certain blocks for later reuse, while TGATE [24] splits the denoising process into two phases and caches the self-attention and cross-attention output, respectively. For diffusion transformer, FORA [40] uses an intuitive caching schedule by computing and caching the output of attention layers and MLP layers in DiT every $\mathcal{N}$ steps, and reusing the results in the subsequent $\mathcal{N}-1$ steps. Δ-DiT [5] caches the deviation between adjacent feature maps to better preserve previous sampling information. TaylorSeer [26] utilizes the Taylor expansion to optimize approximations in long-range feature reuse scenarios. TeaCache [23] and AdaCache [18] adopt adaptive schedules to decide whether to cache or not at a specific timestep.

2) Token-wise caching caches features at a finer granularity. Instead of caching the entire output feature map, it selectively caches some of the tokens based on certain importance or redundancy criteria. ToCa [55] selects suitable tokens to cache under the guidance of four scores that comprehensively indicate the influence of caching a specific token. DuCa [56] applies layer-wise and token-wise caching alternately during the denoising process to better balance efficiency and quality.

It should be noted that most of the methods mentioned above adopt a fixed caching schedule (e.g., FORA [40]) or a manually crafted heuristic schedule (e.g., DuCa [56]). Although these methods are intuitive and easy to implement, they are at risk of neglecting the dynamic magnitudes of feature evolvement in different timesteps, layers and modules, and may result in quality degradation. Therefore, current research calls for adaptive methods that can dynamically shift their caching schedules as the denoising process progresses.

D.2.2. Pruning-based Acceleration

Token reduction is originally intended for the acceleration of large language models(LLM) [11, 25] and multimodal large language models(MLLM) [4, 42, 53]. However, it also excels in accelerating diffusion models. Token reduction methods can be roughly classified into two classes: token pruning and token merging.

1) Token pruning preserves only a fraction of the total tokens that are the most important and discards the rest before performing compute-intensive operations like attention. Tokens are selected to be preserved or discarded following different criteria. AT-EDM [48] proposes a graph-based algorithm that uses attention maps to measure the importance of tokens. CAT-Pruning [7] selects tokens based on noise variations across timesteps, selection frequency, and spatial structure. DaTo [52] first identifies tokens with minimal temporal noise difference as base tokens within each patch of the image, and then discards the tokens that have the highest similarity to the base tokens. After the preserved tokens have gone through the intensive computation, the pruned tokens need to be recovered for subsequent computations like convolution. This can be achieved

by similarity-based copying or token-wise caching, which is elaborated in the previous subsection.

2) Token merging [1, 37, 44] follows similar principles to token pruning. However, it additionally merges the pruned tokens with the preserved ones before omitting them, so as to maintain more information. ToFu[19] combines both token pruning and merging to achieve optimal performance.

Although token reduction methods demonstrate excellent performance because of their fine-grained operation, intricate pruning strategies are likely to add additional workload, making their efficiency still lag behind caching-based acceleration. In addition, some of these methods leverage the attention maps, which are incompatible with CUDA-based acceleration such as FlashAttention [8].

D.3. Other Acceleration

The high latency during the inference stage of diffusion models has drawn continuous attention from the research community, and researchers are working to improve the inference speed of diffusion models from different perspectives.

In general, classic acceleration methods can be categorized into the following domains: *Distillation* [31, 38, 51] often includes a lightweight student network that derives from the original teacher network through knowledge transfer [32]. *Quantization* [12, 21, 41] transforms the precision of model weights to smaller bit width(*e.g.* FP32 to INT8) via retraining, thus achieves model compression. Advanced *samplers*, such as DDPM [15], DDIM [43], DPM-Solver [27], aim to minimize the sampling steps required to generate an image. However, although these methods achieve an excellent acceleration ratio, most of them demand heavy retraining on high-end GPUs, which is formidable to common practices.

Meanwhile, the acceleration of diffusion models can also be achieved via *hardware-level* optimization [6, 8, 45, 49]. These methods focus on minimizing memory access frequency or optimizing the efficiency of computing kernels to fully leverage the computing capacity of modern GPUs. However, hardware-level optimization often suffers from poor portability, high engineering complexity, and limited adaptability to evolving model architectures.

References

[1] Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4599–4603, 2023. 12

[2] Hansheng Chen, Kai Zhang, Hao Tan, Leonidas Guibas, Gordon Wetzstein, and Sai Bi. pi-flow: Policy-based few-step generation via imitation distillation. *arXiv preprint arXiv:2510.14974*, 2025. 7

[3] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, et al. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. *arXiv preprint arXiv:2310.00426*, 2023. 5

[4] Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, pages 19–35. Springer, 2024. 11

[5] Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. Delta-dit: A training-free acceleration method tailored for diffusion transformers. *arXiv preprint arXiv:2406.01125*, 2024. 5, 7, 11

[6] Yu-Hui Chen, Raman Sarokin, Juhyun Lee, Jiuqiang Tang, Chuo-Ling Chang, Andrei Kulik, and Matthias Grundmann. Speed is all you need: On-device acceleration of large diffusion models via gpu-aware optimizations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4651–4655, 2023. 12

[7] Xinle Cheng, Zhuoming Chen, and Zhihao Jia. Cat pruning: Cluster-aware token pruning for text-to-image diffusion models. *arXiv preprint arXiv:2502.00433*, 2025. 11

[8] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022. 12

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[10] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. *arXiv preprint arXiv:2410.12557*, 2024. 7

[11] Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*, 2024. 11

[12] Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. *arXiv preprint arXiv:2305.10657*, 2023. 12

[13] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021. 6

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 5

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 5, 10, 12

[16] Yushi Huang, Zining Wang, Ruihao Gong, Jing Liu, Xinjie Zhang, Jinyang Guo, Xianglong Liu, and Jun Zhang. Harmonica: Harmonizing training and inference for better

feature caching in diffusion transformer acceleration. *arXiv preprint arXiv:2410.01723*, 2024. 7

[17] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, et al. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21807–21818, 2024. 6

[18] Kumara Kahatapitiya, Haozhe Liu, Sen He, Ding Liu, Menglin Jia, Chenyang Zhang, Michael S Ryoo, and Tian Xie. Adaptive caching for faster video generation with diffusion transformers. *arXiv preprint arXiv:2411.02397*, 2024. 11

[19] Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1383–1392, 2024. 12

[20] Senmao Li, Taihang Hu, Fahad Shahbaz Khan, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of unet encoder in diffusion models. *CoRR*, 2023. 11

[21] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17535–17545, 2023. 12

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 6

[23] 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 *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 7353–7363, 2025. 11

[24] Haozhe Liu, Wentian Zhang, Jinheng Xie, Francesco Faccio, Mengmeng Xu, Tao Xiang, Mike Zheng Shou, Juan-Manuel Perez-Rua, and Jürgen Schmidhuber. Faster diffusion via temporal attention decomposition. *arXiv preprint arXiv:2404.02747*, 2024. 11

[25] Junyi Liu, Liangzhi Li, Tong Xiang, Bowen Wang, and Yiming Qian. Tcra-llm: Token compression retrieval augmented large language model for inference cost reduction. *arXiv preprint arXiv:2310.15556*, 2023. 11

[26] Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. *arXiv preprint arXiv:2503.06923*, 2025. 11

[27] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in neural information processing systems*, 35:5775–5787, 2022. 12

[28] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, pages 1–22, 2025. 5

[29] Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. *Advances in Neural Information Processing Systems*, 37:133282–133304, 2024. 7

[30] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15762–15772, 2024. 11

[31] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14297–14306, 2023. 12

[32] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009. 12

[33] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023. 5, 10, 11

[34] Junxiang Qiu, Lin Liu, Shuo Wang, Jinda Lu, Kezhou Chen, and Yanbin Hao. Accelerating diffusion transformer via gradient-optimized cache. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17608–17617, 2025. 7

[35] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 10

[36] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 11

[37] Omid Saghatchian, Atiyeh Gh Moghadam, and Ahmad Nickabadi. Cached adaptive token merging: Dynamic token reduction and redundant computation elimination in diffusion model. *arXiv preprint arXiv:2501.00946*, 2025. 12

[38] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 12

[39] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016. 5

[40] 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. 5, 11

[41] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1972–1981, 2023. 12

[42] Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. Llava-prumerge: Adaptive token reduction for efficient

large multimodal models. *arXiv preprint arXiv:2403.15388*, 2024. 11

[43] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 5, 12

[44] Wenhao Sun, Rong-Cheng Tu, Jingyi Liao, Zhao Jin, and Dacheng Tao. Asymrnr: Video diffusion transformers acceleration with asymmetric reduction and restoration. *arXiv preprint arXiv:2412.11706*, 2024. 12

[45] Shidi Tang, Ruiqi Chen, Rui Liu, Yuxuan Lv, Pengwei Zheng, He Li, and Ming Ling. Diff-acc: An efficient fpga accelerator for unconditional diffusion models. *ACM Transactions on Embedded Computing Systems*, 2025. 12

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 10

[47] Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. 6

[48] Hongjie Wang, Difan Liu, Yan Kang, Yijun Li, Zhe Lin, Niraj K Jha, and Yuchen Liu. Attention-driven training-free efficiency enhancement of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16080–16089, 2024. 11

[49] Zhican Wang, Guanghui He, and Hongxiang Fan. Sdacc: Accelerating stable diffusion through phase-aware sampling and hardware co-optimizations. *arXiv preprint arXiv:2507.01309*, 2025. 12

[50] 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, 2025. 6

[51] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6613–6623, 2024. 12

[52] 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. 11

[53] Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, et al. Sparsevlm: Visual token sparsification for efficient vision-language model inference. *arXiv preprint arXiv:2410.04417*, 2024. 11

[54] Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast. *arXiv preprint arXiv:2408.12588*, 2024. 5

[55] Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. *arXiv preprint arXiv:2410.05317*, 2024. 5, 11

[56] Chang Zou, Evelyn Zhang, Runlin Guo, Haohang Xu, Conghui He, Xuming Hu, and Linfeng Zhang. Accelerating diffusion transformers with dual feature caching. *arXiv preprint arXiv:2412.18911*, 2024. 5, 11