StreamDiffusion: A Pipeline-level Solution for Real-time Interactive Generation

Akio Kodaira[1*] Chenfeng Xu[1,*] Toshiki Hazama[1,*] Takanori Yoshimoto[2] Kohei Ohno[3]
Shogo Mitsuhori[4] Soichi Sugano[5] Hanying Cho[6] Zhijian Kiu[7] Masayoshi Tomizuka[1] Kurt Keutzer[1]

[1]UC Berkeley [2]University of Tsukuba [3]International Christian University
[4]Toyo University [5]Tokyo Institute of Technology [6]Tohoku University [7]MIT

https://github.com/cumulo-autumn/StreamDiffusion

{akio.kodaira, xuchenfeng}@berkeley.edu

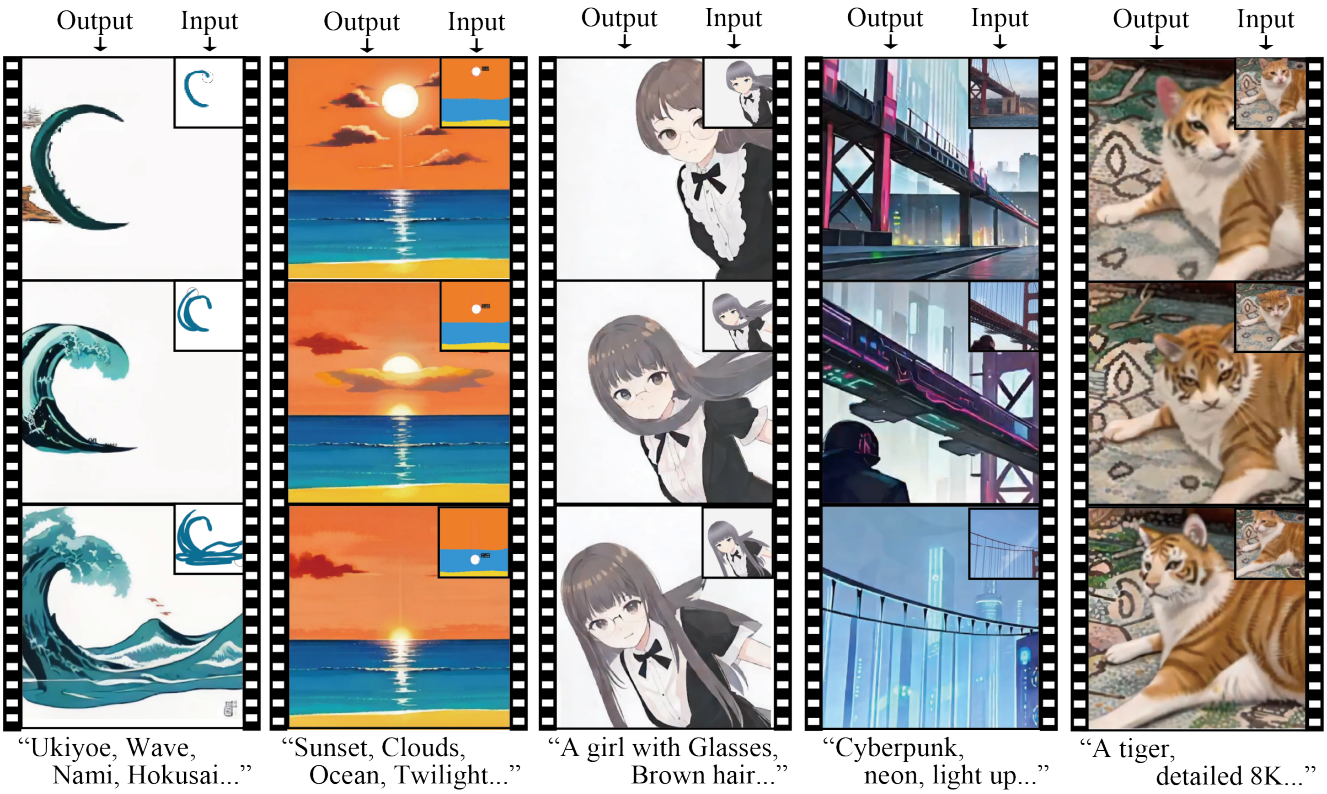

Figure 1. StreamDiT-4B: video generation can be streaming and real-time.

Abstract

We introduce StreamDiffusion, a real-time diffusion pipeline designed for streaming image generation. Existing diffusion models are adept at creating images from text or image prompts, yet they often fall short in real-time interaction. This limitation becomes particularly evident in scenarios involving continuous input, such as augmented/virtual reality, video game graphics rendering, live video streaming, and broadcasting, where high throughput is imperative. StreamDiffusion tackles this challenge through a novel pipeline-level system design. It employs unique strategies like batching the denoising process (Stream Batch), residual classifier-free guidance (R-CFG), and stochastic similarity filtering (SSF). Additionally, it seamlessly integrates advanced acceleration technologies for maximum efficiency. Specifically, Stream Batch reformulates the denoising process by eliminating the traditional wait-and-execute approach and utilizing

* denotes equal contribution
This work was done when Toshiki was a remote intern at UC Berkeley

a batching denoising approach, facilitating fluid and high-throughput streams. This results in 1.5x higher throughput compared to the conventional sequential denoising approach. R-CFG significantly addresses inefficiencies caused by repetitive computations during denoising. It optimizes the process to require minimal or no additional computations, leading to speed improvements of up to 2.05x compared to previous classifier-free methods. Besides, our stochastic similarity filtering dramatically lowers GPU activation frequency by halting computations for static image flows, achieving a remarkable reduction in computational consumption—2.39 times on an RTX 3060 GPU and 1.99 times on an RTX 4090 GPU, respectively. The synergy of our proposed strategies with established acceleration technologies enables image generation to reach speeds of up to 91.07 fps on a single RTX 4090 GPU, outperforming the throughput of AutoPipeline, developed by Diffusers, by more than 59.56x.

1. Introduction

Recently, there has been a growing trend in the commercialization of diffusion models [3, 28, 32, 34] for applications within the entertainment industry such as Metaverse, online video streaming, broadcasting, and even the robotic field [7]. A pertinent example is the use of diffusion models to create virtual YouTubers. These digital personas should be capable of reacting in a fluid and responsive manner to user input. These areas require diffusion pipelines that offer high throughput and low latency to ensure the efficient interactive streaming generation.

To advance the efficiency, current efforts primarily focus on reducing the number of denoising steps, such as decreasing from 50 denoise steps to just a few [24, 25] or even one [21, 42]. The strategy includes distilling the multi-step diffusion models into a few steps [36, 40] or re-framing the diffusion process with neural Ordinary Differential Equations (ODE) [22, 23]. Quantization has also been applied to diffusion models [14, 18] to improve efficiency. These methods share the common goal of approximating either the diffusion process itself or the original model weights to achieve efficiency gains. In this paper, we aim at an orthogonal direction and introduce StreamDiffusion, a pipeline-level solution that enables streaming image generation with high throughput. We highlight that existing model design efforts can still be integrated with our pipeline. Our approach enables the use of N-step denoising diffusion models while keeping high throughput and offers users more flexibility in choosing their preferred models.

Specifically, StreamDiffusion seamlessly incorporates a suite of novel strategies. Among these, we propose a simple yet novel approach termed **Stream Batch**. This method differs from the traditional sequential denoising mode, instead of batching the denoising steps. This subtle modification enhances efficiency without sacrificing the quality of image generation. We highlight Stream Batch enables a new capability of generating images conditioned on future frames in a streaming mode, which is something impossible in previous works. Via injecting the future frames, Stream Batch significantly improves the temporal consistency with few additional overhead. Furthermore, the key novelty of Stream Batch lies not only in its GPU parallelization; rather, it serves as a practical realization of a broader Stream Denoising framework. By denoising inputs at diagonally-offset timesteps within a streaming queue structure, diffusion models naturally achieve continuous, autoregressive generation—emitting one output frame per each newly sampled frame—while benefiting from parallel computation. Crucially, this design generalizes directly to sequential tasks such as video, audio, or robotic action-sequence generation, enabling interactive, unbounded-length synthesis.

Besides, we point out that it is time-consuming for existing diffusion pipelines to use classifier-free guidance for emphasizing the prompts during generation, due to the repetitive and redundant computations for negative conditions. To address this issue, we introduce an innovative approach termed as **residual classifier-free guidance (R-CFG)**. R-CFG approximates the negative condition with a virtual residual noise, which allows us to calculate the negative condition noise only during the initial step of the process. We also indicate that using the original input image latent as the residual term effectively generates results that diverge from the original input image according to the magnitude of the guidance scale, which is a special case of our R-CFG and does not require any computations for the negative condition term.

Furthermore, in real applications such as virtual youtuber and AR/VR cases, maintaining the diffusion models always in an active mode is energy-consuming as it keeps hitting GPU. To reduce the energy, we further apply a **stochastic similarity filtering (SSF)** strategy. In the pipeline, we compute the similarities between continuous inputs and determine whether the diffusion model should process the images based on the probability of an activated similarity. This enables both energy efficiency and visual fluency. In order to further improve the efficiency to cater to the real applications, we apply simple yet effective engineering implementations such as Input-Output Queue (IO-Queue), pre-computing caching, and TensorRT. The overview of the StreamDiffusion pipeline is shown in Fig. 2.

Experiments demonstrate that our proposed StreamDiffusion can achieve up to 91.07fps for image generation on one RTX4090 GPU, surpassing the diffusion Autopipeline from Diffusers [15] team by up to 59.6x. Besides, our stochastic similarity filtering strategy significantly reduces the GPU power usage by 2.39x on one RTX 3090GPU and by 1.99x on one RTX 4090GPU. Our proposed StreamDiffusion is

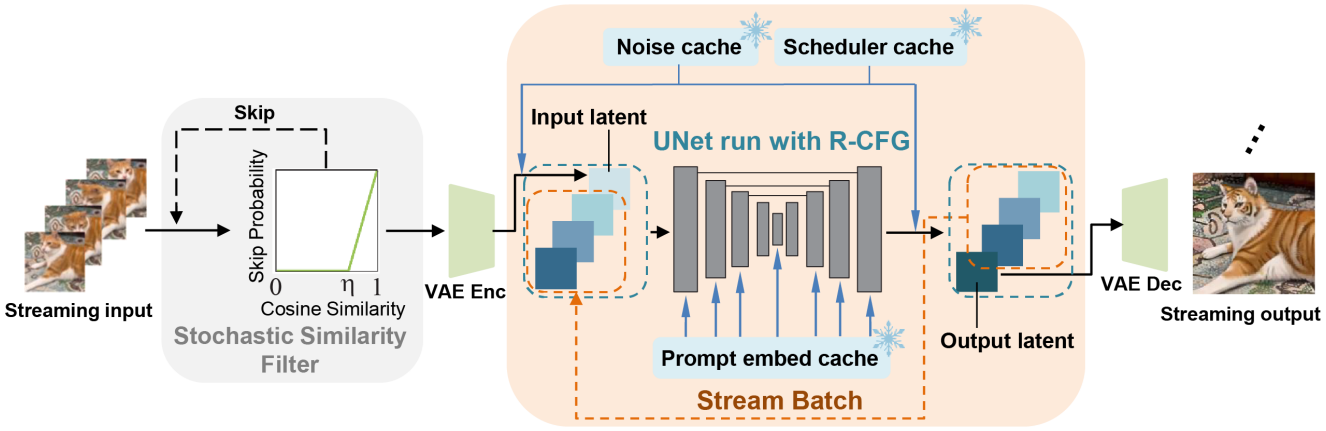

Figure 2. The overview of StreamDiffusion. StreamDiffusion combines several key components: (1) Stream Batch efficiently processes the denoising steps in batches. (2) Residual Classifier-Free Guidance approximates the negative condition term to reduces the unnecessary repetitive calculations in the UNet. (3) Stochastic Similarity Filter controls the pass of the image stream by calculating the similarity between frames to eliminate the redundant hit onto GPUs. Furthermore, StreamDiffusion leverages techniques like input-output queues for smooth data flow, cache management for faster processing through pre-calculated embedding, and a tiny VAE model to further contribute to overall efficiency. This synergistic combination allows StreamDiffusion to generate high-quality images at high throughputs while consuming minimal energy.

a new diffusion pipeline that is not only efficient but also energy-saving.

2. Related work

Efficient Diffusion Models Diffusion models [13, 29, 32, 39] have sparked considerable interest in the commercial sector due to their high-quality image/video generation capabilities. These models have been progressively adapted for various applications, including text-to-image generation [2, 30, 31], image editing [1, 33], video generation [4, 5] and even perception [16, 19, 41]. However, diffusion models are currently limited by their slow speed in generating outputs. In response to this challenge, a variety of strategies have been proposed. One of the mainstreams is to approximate the SDE-based diffusion process [38, 39] through an ordinary differentiable equation (ODE) framework. For example, DPM and DPM++ [22, 23] introduce ODE-based samplers, which significantly reduce the hundreds of denoising steps to between 15 and 20. Building upon the ODE formulation, InstaFlow [21] advances the reduction of denoising steps to a single instance through the novel strategy of rectified flow [20], while achieving performance close to that of Stable Diffusion [32]. Additionally, distillation from pre-trained diffusion models has also been explored as a method to facilitate few-step denoising. For instance, the consistency model [40] leverages the principle of self-consistency between noise at different denoising steps and uses pre-trained diffusion models [32] to guide the learning of a few-step denoising model, thereby enabling the generation of images within a minimal number of steps. In a notable extension of this concept, LCM [25, 26] applies the idea to the latent space rather than the pixel space. The use of distillation methods [35, 36, 42] to enhance the efficiency of the original Stable Diffusion model presents promising results. Besides improving efficiency through the lens of reducing denoising steps, quantization methods [14, 18] are proposed to make the model run in the regime of low float points, albeit with the potential trade-off of fidelity and efficiency. Moreover, parallel sampling [37] tries to utilize the approximate-parallel denoising strategy to improve the latency of the diffusion model. We emphasize that our work focuses on enhancing throughput and our work is orthogonal to [37]. Notably, our method is optimized for single-GPU, which are common among users. In contrast, the parallel sampling method reduces throughput on a single GPU.

Our proposed StreamDiffusion is significantly different from the approaches mentioned previously. While earlier methods primarily focus on the low latency of their individual model designs, our approach takes a different route. We introduce a *pipeline-level* solution specifically tailored for high throughput. Our pipeline can seamlessly integrate the low-latency diffusion models discussed above. Our proposed Stream Batch, residual classifier-free guidance, and the integration of other efficiency-enhancement methods focus on improving the efficiency of the whole pipeline instead of a single diffusion model.

Classifier-free Guidance Classifier-free guidance [12] is widely used for conditional generation due to the simplicity, efficiency, and stability compared to classifier-guidance [9].

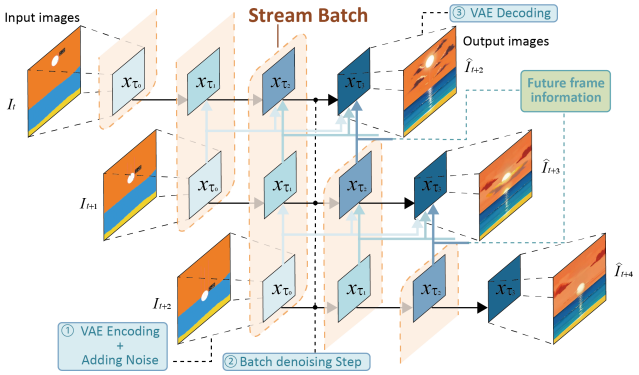

Figure 3. The concept of Stream Batch. In our approach, instead of waiting for a single image to be fully denoised before processing the next input image, we accept the next input image after each denoising step. This creates a denoising batch where the denoising steps are staggered for each image. By concatenating these staggered denoising steps into a batch, we can efficiently process continuous inputs using a U-Net for batch processing. The input image encoded at timestep t is generated and decoded at timestep $t+n$, where n is the number of the denoising steps.

It leverages negative prompts [8, 11, 32] and essentially operates the vector arithmetic shift in latent space, *i.e.,* we take a step of size (usually set by the guidance scale) away from the unconditional vector or negatively conditioned vector in the direction toward the conditioning manifold [12]. In the practical implementation, the classifier-free guidance is conducted by sharing the same UNet for both the conditional term and unconditional (or negative) term and subtracting the effect of the unconditional term from the conditioned one. We point out that the way of multiple denoising processes leads to unnecessary computations for the unconditional (or negative) term. To get rid of these redundant computations, we propose a novel residual-classifier-free guidance, termed as **R-CFG**, which approximates the conditional noise prediction with only requiring one or even zero-time computation for the negatively conditioned noise prediction for the UNet. We note that *R-CFG* is especially designed for the SDEdit method [27], as we mainly focus on the applications of translating streaming image flow. Our proposed R-CFG significantly improves the latency of the conditional image-to-image generation.

3. StreamDiffusion

StreamDiffusion is a new diffusion pipeline aiming for high throughput. It comprises three key components: Stream Batch strategy, Residual Classifier-Free Guidance (R-CFG), and Stochastic Similarity Filter. Besides, we also incorporate other acceleration methods like a novel input-output queue designed by us, the pre-computation procedure, the tiny-autoencoder, and model acceleration tools such as TensorRT. We elaborate on the details below.

3.1. Stream Batch: Batching the Denoise Step

In diffusion models, denoising steps are performed sequentially, resulting in a proportional increase in the processing time of U-Net relative to the number of steps. However, to generate high-fidelity images, it is necessary to increase the number of steps. To resolve this problem in interactive diffusion, we propose a method called Stream Batch.

The Stream Batch technique restructures sequential denoising operations into batched processes, wherein each batch corresponds to a predetermined number of denoising steps, as depicted in Fig. 3. The size of each batch is determined by the number of these denoising steps. This approach allows for each batch element to advance one step further in the denoising sequence via a single pass through U-Net. By iteratively applying this method, it is possible to effectively transform input images encoded at timestep t into their corresponding image-to-image results at timestep $t+n$, thereby streamlining the denoising procedure.

Stream Batch significantly reduces the need for multiple U-Net inferences. The processing time does not escalate linearly with the number of steps. This technique effectively shifts the trade-off from balancing processing time and generation quality to balancing VRAM capacity and generation quality. With adequate VRAM scaling, this method enables the production of high-quality images within the span of a single U-Net processing cycle, effectively overcoming the constraints imposed by increasing denoising steps.

Waiting and Batching can also increase the throughput of the diffusion pipeline. However, with naive Waiting and Batching (WB), denoising cannot begin immediately on the first input frame, leading to higher latency compared to Stream Batch. We mainly aim for smooth streaming applications. Yet achieving a smooth frame rate with WB requires additional engineering, such as precise inference speed estimation and input-output frame synchronization, and minor timing errors must be carefully managed. In contrast, Stream Batch automatically ensures a consistent interval between input and output frames, providing the advantage of lower latency while dynamically reaching the optimal throughput.

3.2. Improve Time Consistency by Stream Batch

Maintaining temporal consistency in video generation is challenging. Many approaches ensure frame coherence by referencing past frames, often through cross-frame attention. However, our Stream Batch method uniquely enables temporal consistency using information from future frames. As shown in Fig. 3, Stream Batch allows simultaneous denoising of multiple frames, passing information from future frames to the current frame. This supports real-time image translation that adapts to sudden changes in input while pre-

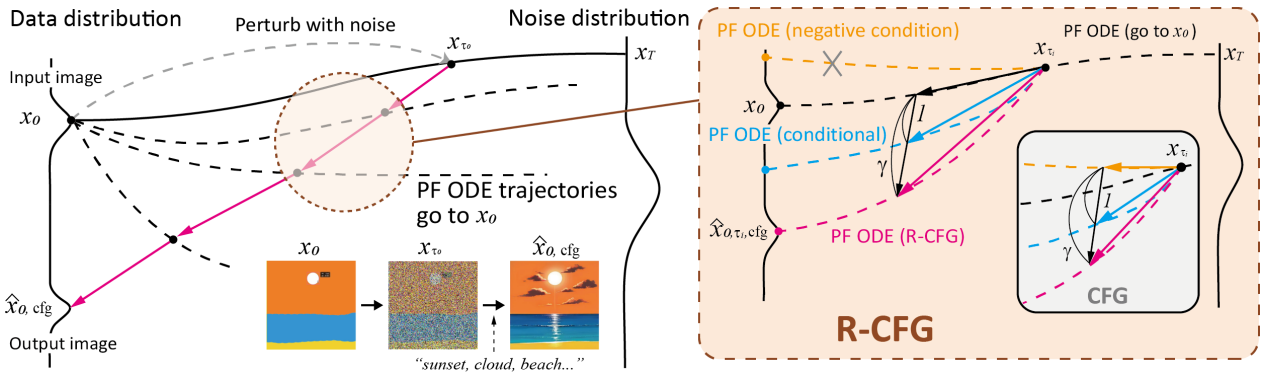

Figure 4. Virtual residual noise vectors: The orange vectors depict the virtual residual noise that starts from PF ODE trajectory and points to the original input latent x_0

serving consistency. In Stream Batch with n denoising steps, keys and values for each frame at each time step form the following batches:

$$K_{\text{Batch}} = \left[K_{t+i,0}, \ldots, K_{t,i}, \ldots, K_{t-(n-1-i),n-1}\right]$$

$$V_{\text{Batch}} = \left[V_{t+i,0}, \ldots, V_{t,i}, \ldots, V_{t-(n-1-i),n-1}\right]$$

These key and value batches incorporate information across different time frames and denoising steps. For example, if the frame at time step t has reached the i-th denoising step, the batch includes i future denoising steps for different frames and $n-1-i$ past frames. In Stream Batch Cross-frame Attention, rather than using the typical $K_{t,i}$ and $V_{t,i}$, we employ K_{Batch} and V_{Batch}, which integrate past and future frame information for the attention computation:

$$\text{Attn}(Q_{t,i}, K_{\text{Batch}}, V_{\text{Batch}}) = \text{Softmax}\left(\frac{Q_{t,i} \cdot K_{\text{Batch}}^T}{\sqrt{d}}\right) V_{\text{Batch}}$$

This approach enables the generation process to account for information from both past and future frames, as well as across different denoising stages, thus enhancing temporal consistency.

3.3. Residual Classifier-Free Guidance

Firstly, SDEdit based method [27] adds perturbation to the input image x_0 and transfers it to the noise distribution x_{τ_0} as follows,

$$x_{\tau_0} = \sqrt{\alpha_{\tau_0}} x_0 + \sqrt{\beta_{\tau_0}} \epsilon_0, \tag{1}$$

where α_{τ_0} and β_{τ_0} are values determined by a noise scheduler and ϵ_0 is a sampled noise from a Gaussian $\mathcal{N}(0, I)$. When using consistency models for conditional image editing, x_{τ_0} can be considered as a point on the PF ODE trajectory, which leads to the conditioning manifold. To intensify the conditioning by Classifier-Free Guidance (CFG)[10], it is imperative to compute a noise for a negative condition PF ODE trajectory, which is used in vector arithmetic shift for the guidance (Eq. 2).

$$\epsilon_{\tau_i,\text{cfg}} = \epsilon_{\tau_i,\bar{c}} + \gamma(\epsilon_{\tau_i,c} - \epsilon_{\tau_i,\bar{c}}), \tag{2}$$

This requirement introduces additional computational overhead at each denoising step. To reduce this computational overhead, R-CFG utilizes the fact that the original input image x_0 is referable at any stage of denoising steps.

For any latent x_{τ_i} at the denoising step τ_i, we can assume the existence of the virtual negative condition $\bar{c}'_{\tau_i}$, that satisfies the self-consistency described as Eq. 3 This implies that x_{τ_i} is on the PF ODE trajectory going back to the input image x_0.

$$x_0 \approx \hat{x}_{0,\tau_i,\bar{c}'_{\tau_i}} = f_\theta(x_{\tau_i}, \tau_i, \bar{c}'_{\tau_i}) \tag{3}$$

Following the LCM model parameterization [25] and our approximation for the inference time skip connections ($c_{\text{skip}}(\tau) = 0$, $c_{\text{out}}(\tau) = 1$ at $\tau \neq 0$), the self-consistency equation (Eq. 3) can be expressed as follows,

$$x_0 \approx \frac{x_{\tau_i} - \sqrt{\beta_{\tau_i}} \epsilon_{\tau_i,\bar{c}'_{\tau_i}}}{\sqrt{\alpha_{\tau_i}}} \tag{4}$$

Given the initial value x_0, and the subsequent values of x_{τ_i} obtained sequentially through the iterative denoising, the virtual noise vector $\epsilon_{\tau_i,\bar{c}'}$ in the direction toward the input image can be analytically determined by employing these values with the Eq. 4:

$$\epsilon_{\tau_i,\bar{c}'} = \frac{x_{\tau_i} - \sqrt{\alpha_{\tau_i}} x_0}{\sqrt{\beta_{\tau_i}}} \tag{5}$$

With the virtual noise $\epsilon_{\tau_i,\bar{c}'}$ obtained from Eq. 5, we formulate R-CFG by:

$$\epsilon_{\tau_i,\text{cfg}} = \delta\epsilon_{\tau_i,\bar{c}'} + \gamma(\epsilon_{\tau_i,c} - \delta\epsilon_{\tau_i,\bar{c}'}) \tag{6}$$

where δ is a magnitude moderation coefficient for the virtual residual noise that softens the effect and the approximation error of the virtual residual noise.

R-CFG that uses the original input image latent x_0 as the residual term can effectively generate results that diverge from the original input image according to the magnitude of the guidance scale γ, thereby enhancing the effect of conditioning without the need for additional U-Net computations. We call this method Self-Negative R-CFG.

Not only to deviate from the original input image x_0, but also to diverge from any negative condition, we can find the desired reference point $\hat{x}_{0,\tau_0,\bar{c}}$ that reflects the negative condition $\bar{c}$ using the same self-consistency formulation:

$$\hat{x}_{0,\tau_0,\bar{c}} = \frac{x_{\tau_0} - \sqrt{\beta_{\tau_0}}\epsilon_{\tau_0,\bar{c}}}{\sqrt{\alpha_{\tau_0}}} \tag{7}$$

We can obtain $\hat{x}_{0,\tau_0,\bar{c}}$ by computing the actual negative conditioned noise $\epsilon_{\tau_0,\bar{c}}$ using U-Net only one time for the first denoising step.

In Eq. 5, instead of x_0, using $\hat{x}_{0,\tau_0,\bar{c}}$, we can obtain the virtual negative conditioned noise $\epsilon_{\tau_{i+1},\bar{c}'}$ that can effectively diverge the generation results from the controllable negative conditioning $\bar{c}$. We name this method Onetime-Negative R-CFG. In contrast to the conventional CFG, which requires $2n$ computations of U-Net, the Self-Negative RCFG and Onetime-Negative RCFG necessitate only n and $n+1$ computations of U-Net, respectively, where n is the number of the denoising steps.

3.4. Stochastic Similarity Filter

When images remain unchanged or show minimal changes, particularly in scenarios without active user interaction or static environment, nearly identical input images are often repeatedly fed into the VAE and U-Net. This leads to the generation of identical or nearly identical images and unnecessary consumption of GPU resources. In contexts involving continuous inputs, such instances of unmodified input images can occasionally occur. To tackle this issue and minimize unnecessary computational load, we propose a strategy termed *stochastic similarity filter (SSF)*, as shown in Fig. 2.

We calculate the cosine similarity between the current input image I_t and the past reference frame image I_{ref}.

$$S_C(I_t, I_{\text{ref}}) = \frac{I_t \cdot I_{\text{ref}}}{\|I_t\|\|I_{\text{ref}}\|} \tag{8}$$

Based on this cosine similarity, we calculate the probability of skipping the subsequent VAE and U-Net processes. It is given by

$$\mathbf{P}(\text{skip}|I_t, I_{\text{ref}}) = \mathbf{max}\left\{0,\ \frac{S_C(I_t, I_{\text{ref}}) - \eta}{1-\eta}\right\}, \tag{9}$$

where η is the similarity threshold. This probability decides whether subsequent processes like VAE Encoding, U-Net, and VAE Decoding should be skipped or not. If not skipped, the input image at that time is saved and updated as the reference image I_{ref} for future use. This probabilistic skipping mechanism allows the network to operate fully in dynamic scenes with low inter-frame similarity, while in static scenes with high inter-frame similarity, the network's operational rate decreases, conserving computational resources. The GPU usage is modulated seamlessly based on the similarity of the input images, enabling smooth adaptation to scenes with varying dynamics.

Note: We emphasize that compared to determining whether we skip the compute via a hard threshold, the proposed probability-sampling-based similarity filtering strategy leads to a smoother video generation. Because the hard threshold is prone to making the video stuck, which hurts the impression of watching video streaming, while the sampling-based method significantly improves the smoothness. For the other efficiency improvement methods, we illustrate them in the supplementary material.

4. Experiments

We implement StreamDiffusion pipeline upon LCM, LCM-LoRA [25, 26] and SD-turbo [36]. As a model accelerator, we use TensorRT and for the lightweight efficient VAE, we use TAESD [17]. Our pipeline is compatible with the customer-level GPU. We test our pipeline on NVIDIA RTX4090 GPU, Intel Core i9-13900K CPU, Ubuntu 22.04.3 LTS, and NVIDIA RTX3060 GPU, Intel Core i7-12700K, Windows 11 for image generation. We note that we evaluate the throughput mainly via the average inference time per image through processing 100 images.

4.1. Quantitative Evaluation

We compare our method with the AutoPipelineForImage2Image, which is a pipeline developed by Huggingface diffusers [1]. The average inference time comparison is presented in Table. 1. Our pipeline demonstrates a substantial speed increase. When we use TensorRT, StreamDiffusion achieves a minimum speed-up of 13.0 times when running the 10 denoising steps, and reaching up to 59.6 times in scenarios involving a single denoising step. Even though without TensorRT, StreamDiffusion achieves a 29.7 times speed up compared to AutoPipeline when using one step denoising, and an 8.3 times speedup at 10 step denoising.

Efficiency comparison regarding Stream Batch. The efficiency comparison between Stream Batch and the original sequential U-Net loop is shown in Fig. 5. When implementing a denoising batch strategy, we observe a significant

[1] https://github.com/huggingface/diffusers

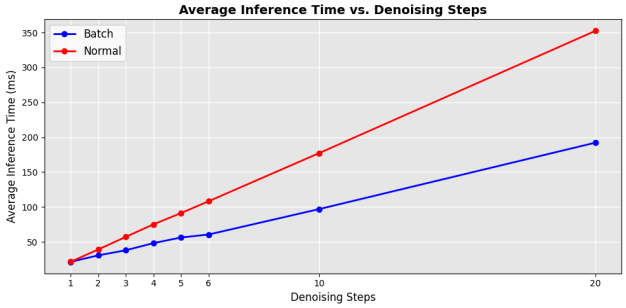

Figure 5. Average inference time comparison between Stream Batch and normal sequential denoising without TensorRT.

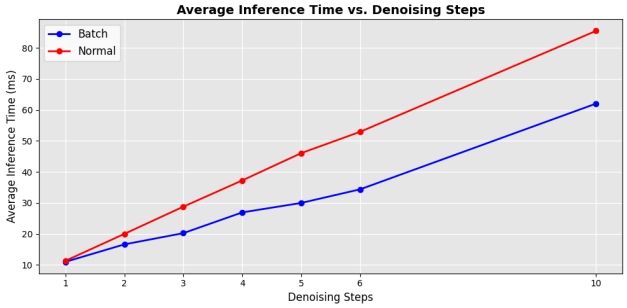

Figure 6. Average inference time comparison between Stream Batch and normal sequential denoising with using TensorRT.

Table 1. Comparison of average inference time (ms) at different denoising steps with speedup factors. The first column denotes the denoising steps and the AutoPipeline is from Diffusers [15].

Step	StreamDiffusion	StreamDiffusion w/o TRT	AutoPipeline Img2Img
1	10.65 (59.6x)	21.34 (29.7x)	634.40 (1x)
2	16.74 (39.3x)	30.61 (21.3x)	652.66 (1x)
4	26.93 (25.8x)	48.15 (14.4x)	695.20 (1x)
10	62.00 (13.0x)	96.94 (8.3x)	803.23 (1x)

improvement in processing time. It achieves a reduction by half when compared to a conventional U-Net loop at sequential denoising steps. Even though applying TensorRT, the accelerator tool for neural modules, our proposed Stream Batch still boosts the efficiency of the original sequential diffusion pipeline by a large margin at different denoising steps.

Efficiency comparison regarding R-CFG. Table. 2 presents a comparison of the inference times for StreamDiffusion pipelines with R-CFG and conventional CFG. The additional computations required to apply Self-Negative R-CFG are merely lightweight vector operations, resulting in negligible changes in inference time compared to when Self-Negative is not used. When employing Onetime-Negative R-CFG, additional UNet computations are necessary for the first step of the denoising process. Therefore, One-time-negative R-CFG and conventional CFG have almost identical inference times for a single denoising step case. However, as the number of denoising steps increases, the difference in inference time from conventional CFG to both Self-Negative and Onetime-Negative R-CFG becomes more pronounced. At denoising step 5, a speed improvement of 2.05x is observed with Self-Negative R-CFG and 1.79x with Onetime-Negative R-CFG, compared to conventional CFG.

4.2. Energy Consumption

We then conduct a comprehensive evaluation of the energy consumption associated with our proposed stochastic similarity filter (SSF), as depicted in Figure. 12 and Figure. 13. These figures provide the GPU utilization patterns when SSF (Threshold η set at 0.98) is applied to input videos containing scenes with periodic static characteristics. The comparative analysis reveals that the incorporation of SSF significantly mitigates GPU usage in instances where the input images are predominantly static and demonstrate a high degree of similarity.

Figure. 12 delineates the results derived from a meticulously executed two-denoise-step img2img experiment. This experiment was conducted on a 20-frame video sequence, employing NVIDIA RTX3060 graphics processing units with or without the integration of SSF. The experiment results indicate a substantial decrease in average power consumption from **85.96w** to **35.91w** on one RTX3060 GPU. Using the same static scene input video with one NVIDIA RTX4090GPU, the power consumption was reduced from **238.68w** to **119.77w**.

Furthermore, Figure. 13 expounds on the findings from a similar two-denoise-step img2img experiment using one RTX4090GPU. This time the evaluation of energy consumption is performed on a 1000-frame video featuring dynamic scenes. Remarkably, even under drastically dynamic conditions, the SSF efficiently extracted several frames exhibiting similarity from the dynamic sequence. This process results in a noteworthy reduction in average power consumption, from **236.13w** to **199.38w**. These findings underscore the efficacy of the Stochastic Similarity Filter in enhancing energy efficiency, particularly in scenarios involving static or minimally varying visual content.

Table 2. Comparison of average inference time (ms) at different denoising steps among different CFG methods

Step	Self-Negative R-CFG	Onetime-Negative R-CFG	CFG
1	11.04 (1.52x)	16.55 (1.01x)	16.74 (1x)
2	16.61 (1.64x)	20.64 (1.32x)	27.18 (1x)
3	20.64 (1.74x)	27.25 (1.32x)	35.91 (1x)
4	26.19 (1.90x)	31.65 (1.57x)	49.71 (1x)
5	31.47 (2.05x)	36.04 (1.79x)	64.64 (1x)

4.3. Ablation study

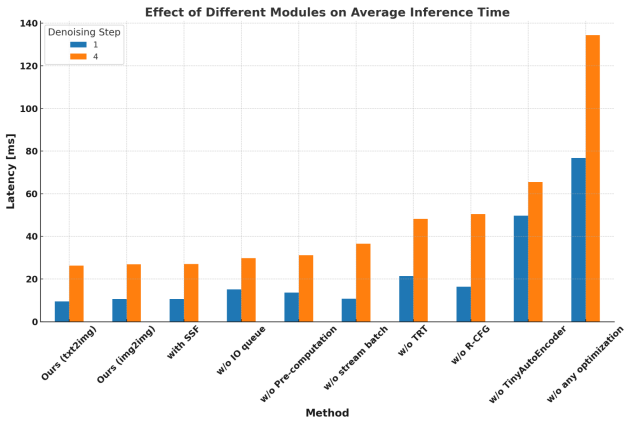

Figure 7. Ablation study on different components.

In our ablation study, as summarized in Fig. 7, we evaluate the average inference time of our proposed method under various configurations to understand the contribution of each component. Our proposed StreamDiffusion achieves an average inference time of 10.98/9.42 ms and 26.93/26.30 ms for denoising steps 1 and 4 on image-to-image/text-to-image generation, respectively. When the R-CFG is not used, we observe this results in the second largest efficiency drop. This demonstrates that R-CFG is one of the most critical components in our pipeline. When the Stream Batch processing is removed ('w/o stream batch'), we observe a large time consumption increase, especially at 4 denoising steps. We also evaluate the impact on the inference time of our pipeline regarding SSF. We observe that SSF plays a significant role in enabling energy saving and does not introduce extra time cost.

Besides, the absence of TensorRT ('w/o TRT') leads to a large increase in time cost. The removal of pre-computation also results in increased time cost but not much. We attribute the reason to the limited number of key-value computations in Stable Diffusion. Besides, the exclusion of input-output queue ('w/o IO queue') also demonstrates an impact on average inference time, which mainly aims to optimize the parallelization issue resulting from pre- and post-processing. In the AutoPipelineImage2Image's adding noise function, the precision of tensors is converted from fp32 to fp16 for each request, leading to a decrease in speed. In contrast, the StreamDiffusion pipeline standardizes the precision of variables and computational devices beforehand. It does not perform tensor precision conversion or computation device transfers during inference. Consequently, even without any optimization ('w/o any optimization'), our pipeline significantly outperforms the AutoPipelineImage2Image in terms of speed.

4.4. Quantitative Evaluation for the Image Quality.

We conduct a quantitative evaluation on the image quality. Specifically, we first evaluate the FID and CLIP scores on the text-to-image generation. We use the same dataset as [6] for the evaluation. We use LCM as our main baseline for the comparison. Note that our method is never trained; our method still improves LCM by a large margin in terms of the FID (29.69 vs. 26.79) and maintains a similar CLIP score (24.95 vs. 24.99). This demonstrates the effectiveness of our proposed method.

User study. We also conduct a user study to validate the visual quality of different components of our StreamDiffusion. The results are shown in the Table. 3. Our Stream Batch with future-frame attention significantly enhances time consistency compared to its absence. Additionally, our SSF method addresses the crucial yet rarely explored issue of energy efficiency. While SSF is simple, it is far from trivial: a common approach might apply a hard threshold on similarity to regulate streaming flow. However, as noted in the main text, we introduce a novel approach—using probability sampling to achieve superior streaming quality. As Table 3 illustrates, this approach is preferred by more users over the hard-threshold method for visual quality. Moreover, compared to the vanilla CFG method, both our self-Negative R-CFG and one-time R-CFG are more preferred by the users, demonstrating our method not only improves efficiency but also visual quality.

This pipeline enables image generation with very low throughput from input images received in real-time from cameras or screen capture devices. At the same time, it is capable of producing high-quality images that effectively align to the specified prompt conditions. These capabilities

# Users	Without Future-Frame (%)	With Future-Frame (%)
486	9.67	90.33
# Users	**Without SSF (%)**	**With SSF (%)**
144	48.61	51.39
# Users	**Hard Threshold (%)**	**SSF (%)**
45	24.44	75.56
# Users	**With / Without CFG (%)**	**Self-Neg R-CFG / One-Time R-CFG (%)**
257	22 / 20	35 / 23

Table 3. User study results: Future-frame attention consistency (486 users), SSF quality imperceptibility (144 users), streaming quality between hard threshold filter and SSF (45 users), comparison of with/without CFG and self-negative/one-time R-CFG (257 users).

demonstrate the applicability of our pipeline in various real-time applications, such as real-time game graphic rendering, generative camera effect filters, real-time face conversion, and AI-assisted drawing.

The alignment of generated images to prompt conditioning using Residual Classifier-Free Guidance (R-CFG) is depicted in Fig. 8. The generated images, without using any form of CFG, exhibit weak alignment to the prompt, particularly in aspects like color changes or the addition of non-existent elements, which are not effectively implemented. In contrast, the use of CFG or R-CFG enhances the ability to modify original images, such as changing hair color, adding body patterns, and even incorporating objects like glasses. Notably, the use of R-CFG results in a stronger influence of the prompt compared to standard CFG. R-CFG, although limited to image-to-image applications, can compute the vector for negative conditioning while continuously referencing the latent value of the input image and the initially sampled noise. This approach yields more consistent directions for the negative conditioning vector compared to the standard CFG, which uses UNet at every denoising step to calculate the negative conditioning vector. Consequently, this leads to more pronounced changes from the original image. However, there is a trade-off in terms of the stability of the generated results. While Self-Negative R-CFG enhances the prompt's effectiveness, it also has the drawback of increasing the contrast of the generated images. To address this, adjusting the $delta$ in Eq. 6 can modulate the magnitude of the virtual residual noise vector, thereby mitigating the rise in contrast. Additionally, using Onetime-Negative R-CFG with appropriately chosen negative prompts can mitigate contrast increases while improving prompt adherence, as observed in Fig. 8. This approach allows the generated images to blend more naturally with the original image.

Besides, Fig. 9 in appendix shows the image-to-image generation results using StreamBatch Cross-frame attention, with 4 denoising steps. As evident from the figure, compared to the results of StreamDiffusion without Cross-frame attention, the method incorporating information from future and past frames exhibits increased temporal consistency.

5. Conclusion

We propose StreamDiffusion, a pipeline-level solution for interactive diffusion generation. StreamDiffusion consists of several optimization strategies for both throughput and GPU usage, including Stream Batch with cross-attention, residual classifier-free guidance (R-CFG), IO-queue for parallelization, stochastic similarity filter, pre-computation, Tiny AutoEncoder, and the use of the model acceleration tool. The synergistic combination of these elements results in a marked improvement in efficiency. Specifically, StreamDiffusion achieves up to 91.07 frames per second (fps) on a standard consumer-grade GPU for image generation tasks. This performance level is particularly beneficial for a variety of applications, including but not limited to the Metaverse, online video streaming, and broadcasting sectors. Furthermore, StreamDiffusion demonstrates a significant reduction in GPU power consumption, achieving at least a 1.99x decrease. This notable efficiency gain underscores StreamDiffusion's potential for commercial application, offering a compelling solution for energy-conscious, high-performance computing environments.

6. Acknowledgments

We sincerely thank Taku Fujimoto and Huggingface team for their invaluable feedback, courteous support, and insightful discussions.

References

[1] Omri Avrahami, Dani Lischinski, and Ohad Fried. Blended diffusion for text-driven editing of natural images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 3

[2] Omri Avrahami, Ohad Fried, and Dani Lischinski. Blended latent diffusion. *ACM Trans. Graph.*, 2023. 3

[3] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, Wesam Manassra, Prafulla Dhariwal, Casey Chu, Yunxin Jiao, and Aditya Ramesh. Improving image generation with better captions. 2

[4] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023. 3

[5] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22563–22575, 2023. 3

[6] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. *arXiv preprint arXiv:2211.09800*, 2022. 8

[7] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023. 2

[8] Katherine Crowson, Stella Biderman, Daniel Kornis, Dashiell Stander, Eric Hallahan, Louis Castricato, and Edward Raff. Vqgan-clip: Open domain image generation and editing with natural language guidance. In *European Conference on Computer Vision*, pages 88–105. Springer, 2022. 4

[9] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021. 3

[10] Yilun Du, Shuang Li, and Igor Mordatch. Compositional visual generation with energy based models. In *Advances in Neural Information Processing Systems*, pages 6637–6647. Curran Associates, Inc., 2020. 5

[11] Yilun Du, Shuang Li, and Igor Mordatch. Compositional visual generation with energy based models. *Advances in Neural Information Processing Systems*, 33:6637–6647, 2020. 4

[12] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. 3, 4

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 3

[14] Yushi Huang, Ruihao Gong, Jing Liu, Tianlong Chen, and Xianglong Liu. Tfmq-dm: Temporal feature maintenance quantization for diffusion models. *arXiv preprint arXiv:2311.16503*, 2023. 2, 3

[15] Huggingface. https://huggingface.co/docs/diffusers/, 2024. 2, 7

[16] Aliasghar Khani, Saeid Asgari, Aditya Sanghi, Ali Mahdavi Amiri, and Ghassan Hamarneh. Slime: Segment like me. In *The Twelfth International Conference on Learning Representations*, 2024. 3

[17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. 6, 13

[18] 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 (ICCV)*, pages 17535–17545, 2023. 2, 3

[19] Ziyi Li, Qinye Zhou, Xiaoyun Zhang, Ya Zhang, Yanfeng Wang, and Weidi Xie. Open-vocabulary object segmentation with diffusion models. 2023. 3

[20] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022. 3

[21] Xingchao Liu, Xiwen Zhang, Jianzhu Ma, Jian Peng, and Qiang Liu. Instaflow: One step is enough for high-quality diffusion-based text-to-image generation. *arXiv preprint arXiv:2309.06380*, 2023. 2, 3

[22] 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. 2, 3

[23] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2022. 2, 3

[24] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference, 2023. 2

[25] Simian Luo, Yiqin Tan, Suraj Patil, Daniel Gu, Patrick von Platen, Apolinário Passos, Longbo Huang, Jian Li, and Hang Zhao. Lcm-lora: A universal stable-diffusion acceleration module. *arXiv preprint arXiv:2311.05556*, 2023. 2, 3, 5, 6, 12

[26] Simian Luo, Yiqin Tan, Suraj Patil, Daniel Gu, Patrick von Platen, Apolinário Passos, Longbo Huang, Jian Li, and Hang Zhao. Lcm-lora: A universal stable-diffusion acceleration module, 2023. 3, 6

[27] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2022. 4, 5

[28] OpenAI. https://openai.com/sora, 2024. 2

[29] William Peebles and Saining Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022. 3

[30] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2): 3, 2022. 3

[31] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 3, 13

[32] 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. 2, 3, 4

[33] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. 2022. 3

[34] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho†, David Fleet, and Mohammad Norouzi. Imagen: unprecedented photorealism × deep level of language understanding. 2022. 2

[35] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 3

[36] Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. *arXiv preprint arXiv:2311.17042*, 2023. 2, 3, 6

[37] Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[38] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. 3

[39] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020. 3

[40] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023. 2, 3

[41] Chenfeng Xu, Huan Ling, Sanja Fidler, and Or Litany. 3difftection: 3d object detection with geometry-aware diffusion features, 2023. 3

[42] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Frédo Durand, William T. Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. *arXiv*, 2023. 2, 3

.1. Qualitative Results

A. More Architecture details

A.1. Input-Output Queue

The current bottleneck in high-speed image generation systems lies in the neural network modules, including VAE and U-Net. To maximize the overall system speed, processes such as pre-processing and post-processing of images, which do not require handling by the neural network modules, are moved outside of the pipeline and processed in parallel.

In the context of input image handling, specific operations, including resizing of input images, conversion to tensor format, and normalization, are meticulously executed. To address the disparity in processing frequencies between the human inputs and the model throughput, we design an input-output queuing system to enable efficient parallelization, as shown in Fig. 10. This system operates as follows: processed input tensors are methodically queued for Diffusion Models. During each frame, Diffusion Model retrieves the most recent tensor from the input queue and forwards it to the VAE Encoder, thereby triggering the image generation sequence. Correspondingly, tensor outputs from the VAE Decoder are fed into an output queue. In the subsequent output image handling phase, these tensors are subject to a series of post-processing steps and conversion into the appropriate output format. Finally, the fully processed image data is transmitted from the output handling system to the rendering client.

A.2. Pre-computation

The U-Net architecture requires both input latent variables and conditioning embeddings. Typically, the conditioning embedding is derived from a text prompt, which remains constant across different frames. To optimize this, we precompute the prompt embedding and store it in a cache. In interactive or streaming mode, this pre-computed prompt embedding cache is recalled. Within U-Net, the Key and Value are computed based on this pre-computed prompt embedding for each frame. We have modified the U-Net to store these Key and Value pairs, allowing them to be reused. Whenever the input prompt is updated, we recompute and update these Key and Value pairs inside U-Net.

For consistent input frames across different timesteps and to improve computational efficiency, we pre-sample Gaussian noise for each denoising step and store it in the cache. This approach is particularly relevant for image-to-image tasks.

We also precompute α_τ and β_τ, the noise strength coefficients for each denoising step τ, defined as:

$$x_t = \sqrt{\alpha_\tau} x_0 + \sqrt{\beta_\tau}\epsilon \tag{10}$$

This is a minor point in low throughput scenarios, but at frame rates higher than 60 FPS, the overhead of recomputing these static values becomes noticeable.

We note that we have a specific design for the inference parameterization for latent consistency models (LCM). As per the original paper, we need to compute $c_{\text{skip}}(\tau)$ and $c_{\text{out}}(\tau)$ to satisfy the following equation:

$$f_\theta(x,\tau) = c_{\text{skip}}(\tau)x + c_{\text{out}}(\tau)F_\theta(x,\tau). \tag{11}$$

The functions $c_{\text{skip}}(\tau)$ and $c_{\text{out}}(\tau)$ in original LCM [25] is constructed as follows:

$$c_{\text{skip}}(\tau) = \frac{\sigma_{\text{data}}^2}{(s\tau)^2 + \sigma_{\text{data}}^2}, \quad c_{\text{out}}(\tau) = \frac{\sigma_{\text{data}} s\tau}{\sqrt{\sigma_{\text{data}}^2 + (s\tau)^2}}, \tag{12}$$

where $\sigma_{\text{data}} = 0.5$, and the timestep scaling factor $s = 10$. We note that with $s = 10$, $c_{\text{skip}}(\tau)$ and $c_{\text{out}}(\tau)$ approximate delta functions that enforce the boundary condition to the consistency models. (i.e., at denoising step $\tau = 0$, $c_{\text{skip}}(0) = 1$, $c_{\text{out}}(0) = 0$; and at $\tau \neq 0$, $c_{\text{skip}}(\tau) = 0$, $c_{\text{out}}(\tau) = 1$). At inference time, there's no need to recompute these functions repeatedly. We can either pre-compute $c_{\text{skip}}(\tau)$ and $c_{\text{out}}(\tau)$ for all denoising steps τ in advance or simply use constant values $c_{\text{skip}} = 0$, $c_{\text{out}} = 1$ for any arbitrary denoising step τ.

A.3. Model Acceleration and Tiny AutoEncoder

We employ TensorRT to construct the U-Net and VAE engines, further accelerating the inference speed. TensorRT is an optimization toolkit from NVIDIA that facilitates high-performance deep learning inference. It achieves this by performing several optimizations on neural networks, including layer fusion, precision calibration, kernel auto-tuning, dynamic tensor memory, and more. These optimizations are designed to increase throughput and efficiency for deep learning applications.

To optimize speed, we configured the system to use static batch sizes and fixed input dimensions (height and width). This approach ensures that the computational graph and memory allocation are optimized for a specific input size, leading to faster processing times. However, this means that if there is a requirement to process images with different shapes (i.e., varying heights and widths) or to use different batch sizes (including those for denoising steps), a new engine tailored to these specific dimensions must be built. This is because the optimizations and configurations applied in TensorRT are specific to the initially defined dimensions and batch size, and changing these parameters would necessitate a reconfiguration and re-optimization of the network within TensorRT.

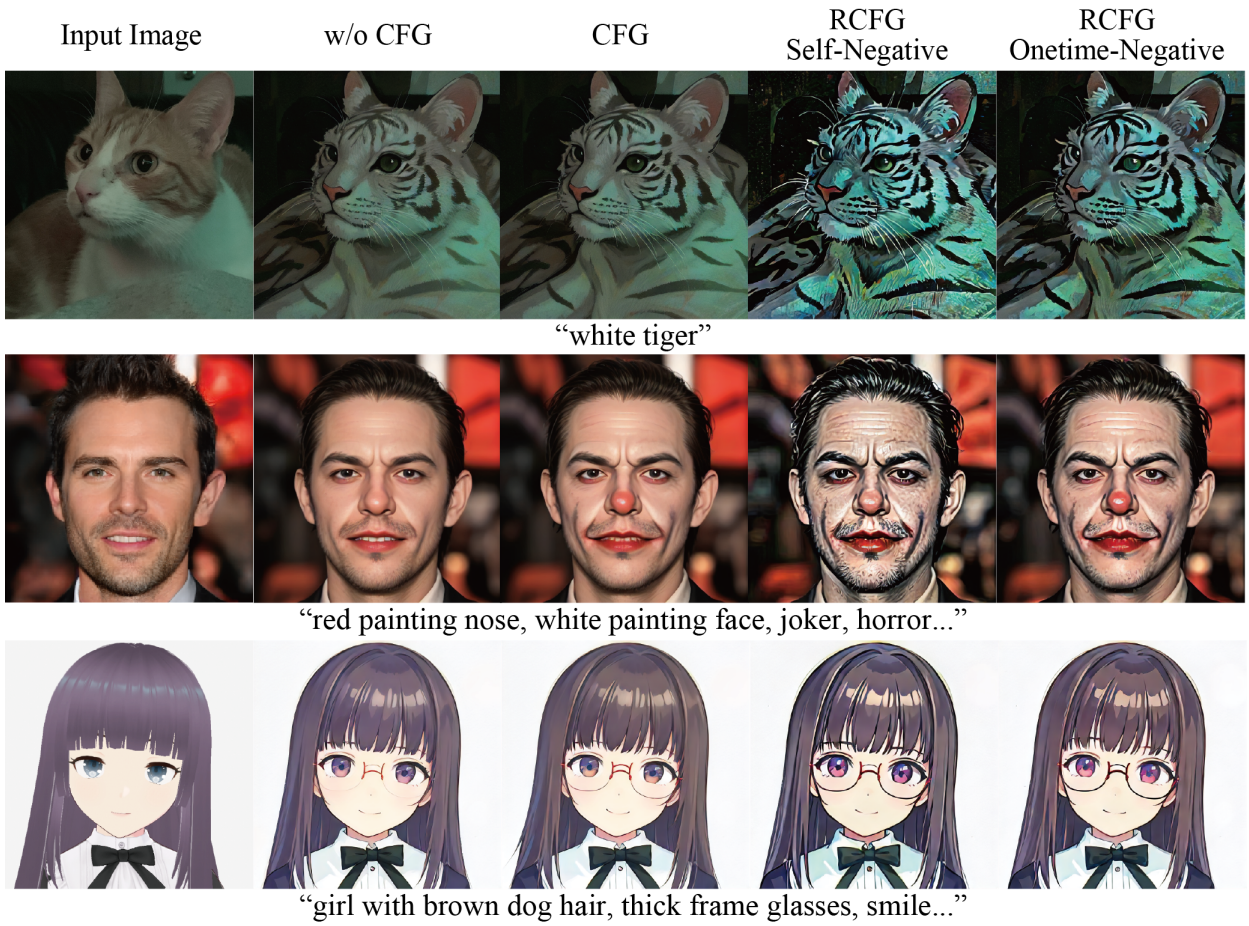

Figure 8. Results using no CFG, standard CFG, and R-CFG with Self-Negative and Onetime-Negative approaches. When compared to cases where CFG is not utilized, the cases with CFG utilized can intensify the impact of prompts. In the proposed method R-CFG, a more pronounced influence of prompts was observed. Both CFG and R-CFG use guidance scale $\gamma = 1.4$. For R-CFG, the first two rows use magnitude modelation coefficient $\delta = 1.0$, and the third row uses $\delta = 0.5$.

Besides, we employ a tiny AutoEncoder, which has been engineered as a streamlined and efficient counterpart to the traditional Stable Diffusion AutoEncoder [17, 31]. TAESD excels in rapidly converting latents into full-size images and accomplishing decoding processes with significantly reduced computational demands.

B. Text-to-Image Quality

The quality of standard text-to-image generation results is demonstrated in Fig. 11. Using the sd-turbo model, high-quality images like those shown in Fig. 11 can be generated in just one step. When images are produced using our proposed StreamDiffusion pipeline and SD-turbo model in an environment with GPU: RTX 4090, CPU: Core i9-13900K, and OS: Ubuntu 22.04.3 LTS, it's feasible to generate such high-quality images at a rate exceeding 100fps. Furthermore, by increasing the batch size of images generated at once to 12, our pipeline can continuously produce approximately 150 images per second. The images enclosed in red frames shown in Fig. 11 are generated in four steps using community models merged with LCM-LoRA. While these LCM models require more than 1 step for high quality image generation, resulting in a reduction of speed to around 40fps, these LCM-LoRA based models offer the flexibility of utilizing any base model, enabling the generation of images with diverse expressions.

C. GPU Usage Under Dynamic Scene

We also evaluate the GPU usage under dynamic scenes on one RTX 4090 GPU, as shown in the Figure. 13. The analysis of the GPU usage is shown in Section 4.2 of the main text.

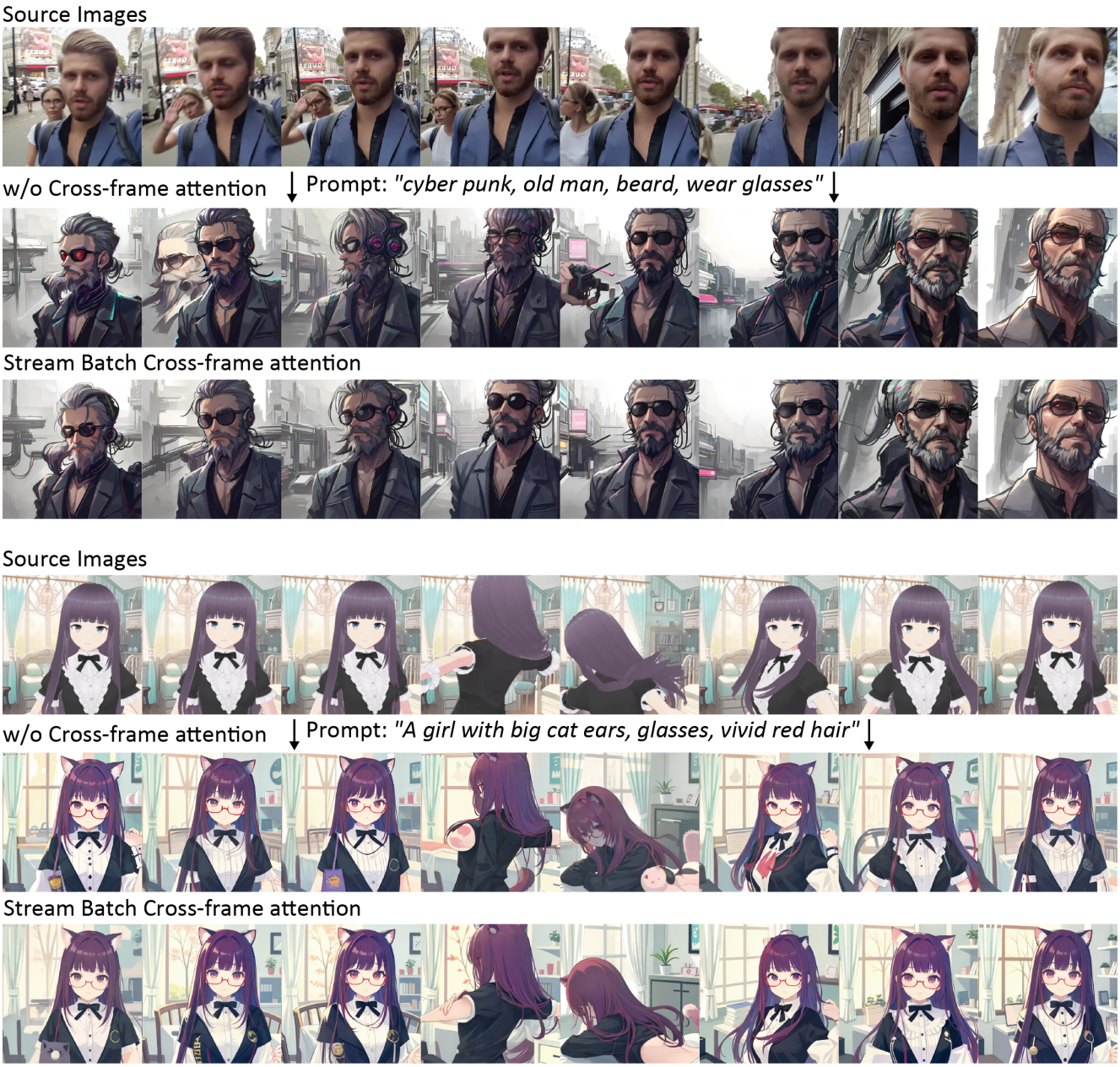

Figure 9. Time consistency qualitative evaluation: In cases where the subject's face moves significantly in intermediate frames, it can be observed that using StreamBatch Cross-frame attention produces more appropriate and temporally consistent generation results by leveraging the context from preceding and succeeding frames.

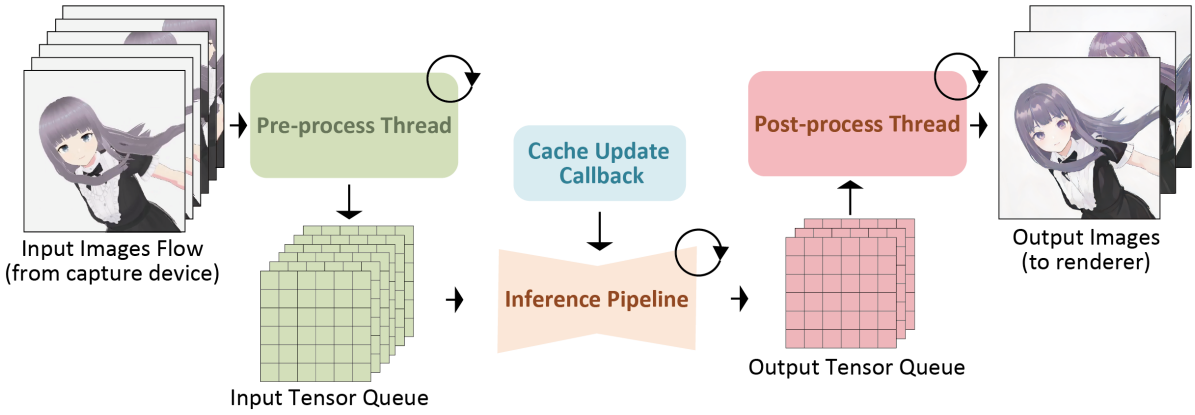

Figure 10. Input-Output Queue: The process of converting input images into a tensor data format manageable by the pipeline, and conversely, converting decoded tensors back into output images requires a non-negligible amount of additional processing time. To avoid adding these image processing times to the bottleneck process, the neural network inference process, we have segregated image pre-processing and post-processing into separate threads, allowing for parallel processing. Moreover, by utilizing an Input Tensor Queue, we can accommodate temporary lapses in input images due to device malfunctions or communication errors, enabling smooth streaming.

Figure 11. Text-to-Image generation results. We use four step denoising for LCM-LoRA, and one step denoising for sd-turbo. Our StreamDiffusion enables the real-time generation of images with quality comparable to those produced using Diffusers AutoPipeline Text2Image.

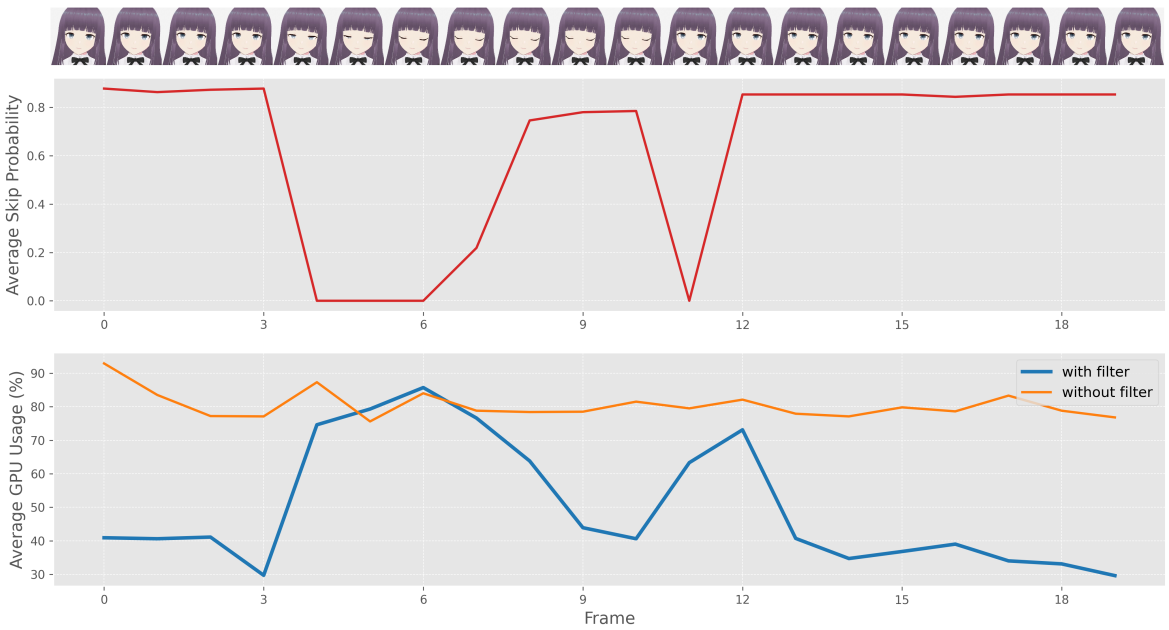

Figure 12. GPU Usage comparison under static scene. (GPU: RTX3060, Number of frames: 20) The blue line represents the GPU usage with SSF, the orange line indicates GPU usage without SSF, and the red line denotes the Skip probability calculated based on the cosine similarity between input frames. Additionally, the top of the plot displays input images corresponding to the same timestamps. In this case, the character in the input images is only blinking.

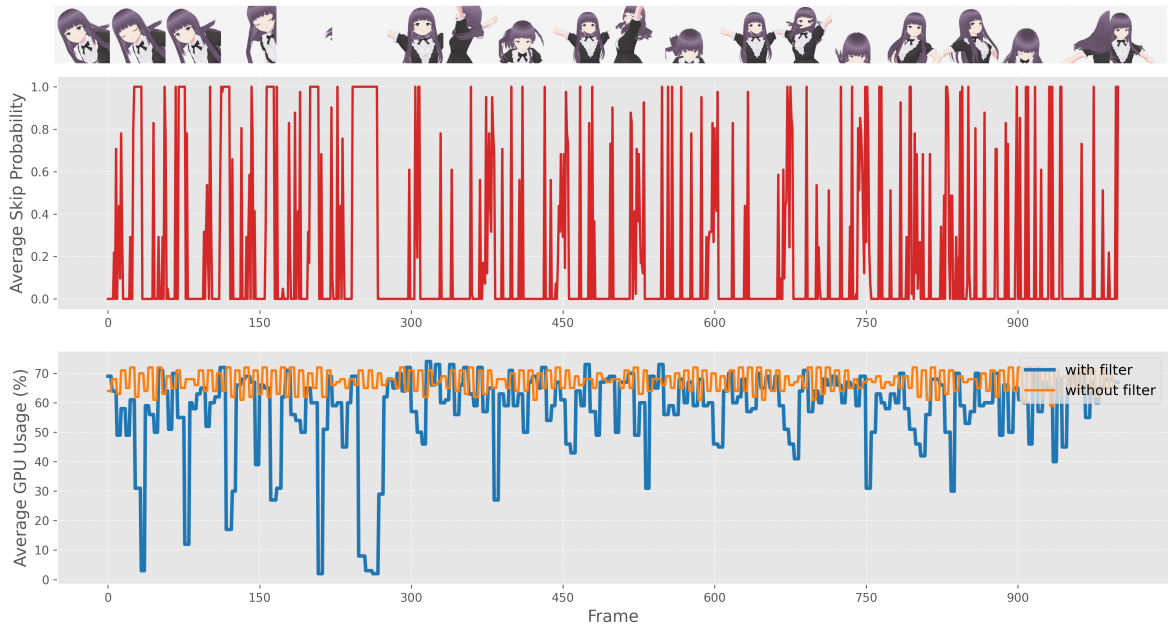

Figure 13. GPU Usage comparison under dynamic scene. (GPU: RTX4090, Number of frames: 1000) The blue line represents the GPU usage with SSF, the orange line indicates GPU usage without SSF, and the red line denotes the Skip probability calculated based on the cosine similarity between input frames. Additionally, the top of the plot displays input images corresponding to the same timestamps. In this case, the character in the input images keeps moving dynamically. Thus, this analysis compares GPU usage in a dynamic scenario.