A. Qualitative Results

The alignment of generated images to prompt conditioning using Residual Classifier-Free Guidance (R-CFG) is depicted in Fig. 7. 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. 7. This approach allows the generated images to blend more naturally with the original image.

Besides, Fig. 8 in the 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.

B. More method details

B.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. 9. 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.

B.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:

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

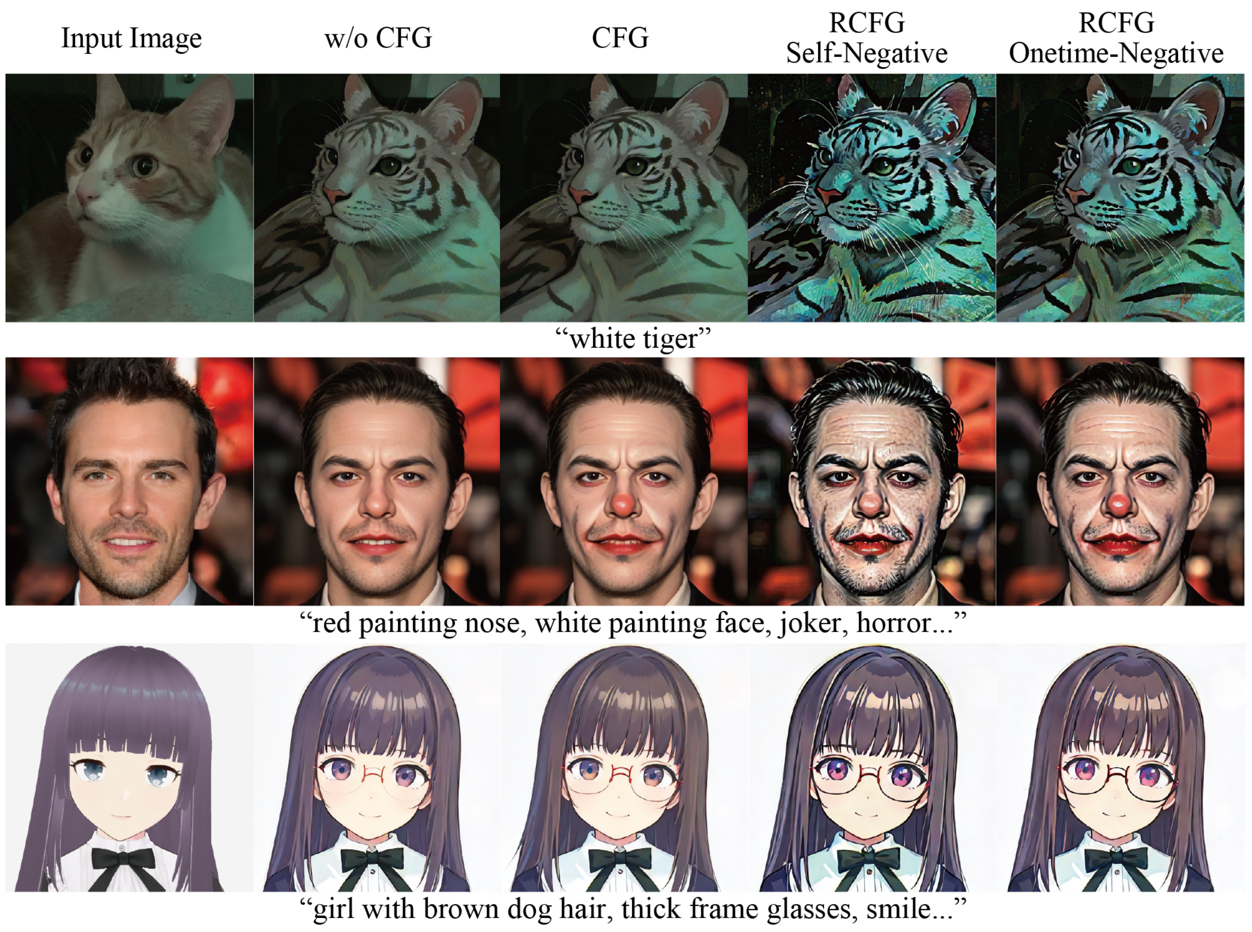
Input Image
w/o CFG
CFG
RCFG Self-Negative
RCFG Onetime-Negative
“white tiger”
“red painting nose, white painting face, joker, horror...”
“girl with brown dog hair, thick frame glasses, smile...”

Figure 7. 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 $\delta = 0.5$.

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.

B.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.

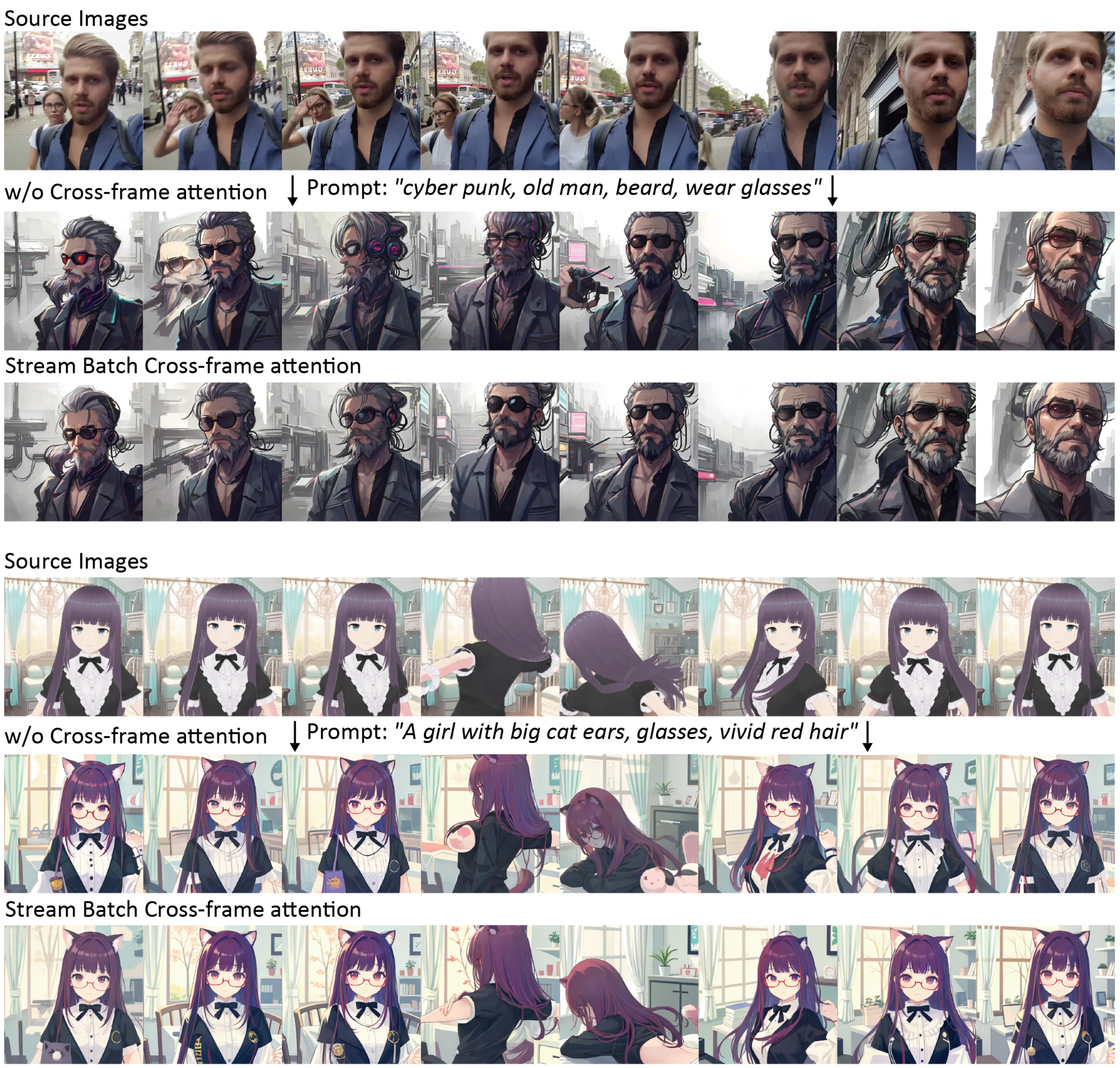
Source Images
w/o Cross-frame attention
Prompt: "cyber punk, old man, beard, wear glasses"
Stream Batch Cross-frame attention
Source Images
w/o Cross-frame attention
Prompt: "A girl with big cat ears, glasses, vivid red hair"
Stream Batch Cross-frame attention

Figure 8. 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.

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.

C. Text-to-Image Quality

The quality of standard text-to-image generation results is demonstrated in Fig. 10. Using the sd-turbo model, high-quality images like those shown in Fig. 10 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

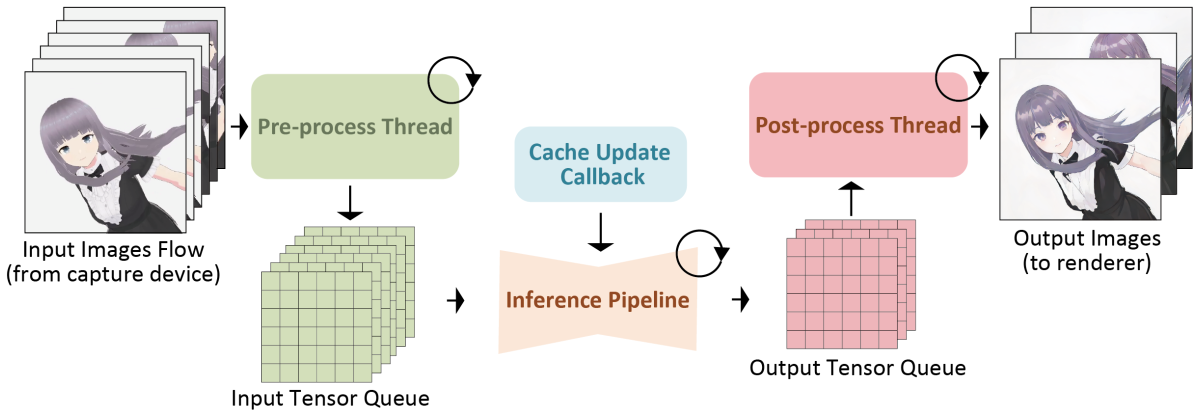

Figure 9. Input-Output Queue: The process of converting input images into a tensor data format manageable by the pipeline, and conversely, converting decoded tensors back into output images requires a non-negligible amount of additional processing time. To avoid adding these image processing times to 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.

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. 10 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.

D. GPU Usage Under Dynamic Scene

We also evaluate the GPU usage under dynamic scenes on one RTX 4090 GPU, as shown in the Figure. 12. The analysis of the GPU usage is shown in Section 4.2 of the main text.

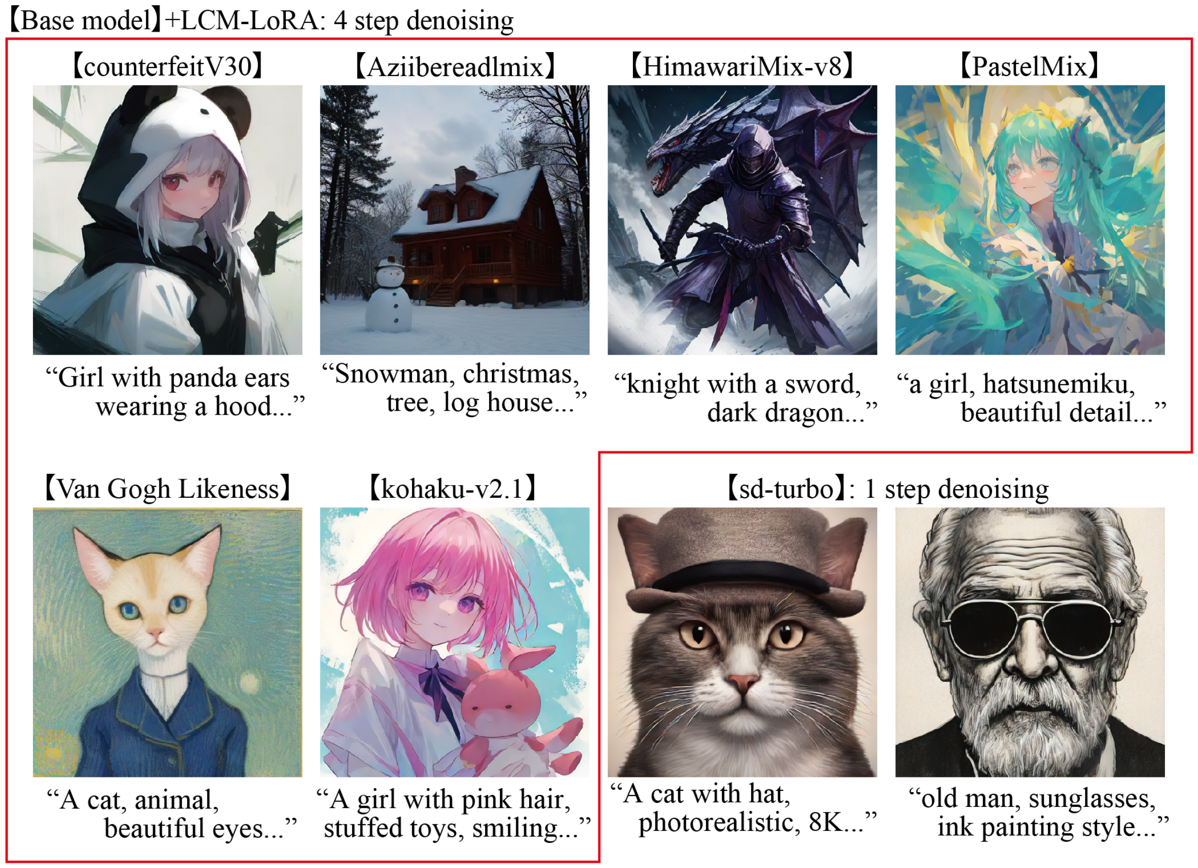
【Base model】+LCM-LoRA: 4 step denoising
【counterfeitV30】
【Aziibereadlmix】
【HimawariMix-v8】
【PastelMix】
"Girl with panda ears wearing a hood..."
"Snowman, christmas, tree, log house..."
"knight with a sword, dark dragon..."
"a girl, hatsunemiku, beautiful detail..."
【Van Gogh Likeness】
【kohaku-v2.1】
【sd-turbo】: 1 step denoising
"A cat, animal, beautiful eyes..."
"A girl with pink hair, stuffed toys, smiling..."
"A cat with hat, photorealistic, 8K..."
"old man, sunglasses, ink painting style..."

Figure 10. 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.

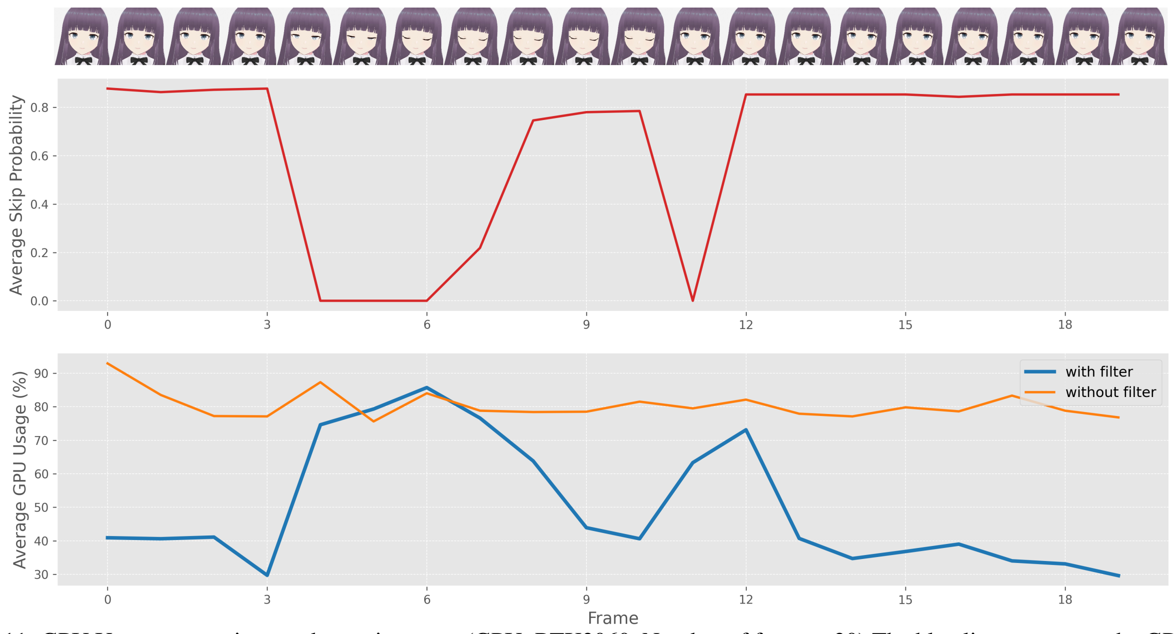

Figure 11. GPU Usage comparison under static scene. (GPU: RTX3060, Number of frames: 20) The blue line represents the GPU usage with SSF, the orange line indicates GPU usage without SSF, and the red line denotes the Skip probability calculated based on the cosine similarity between input frames. Additionally, 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.

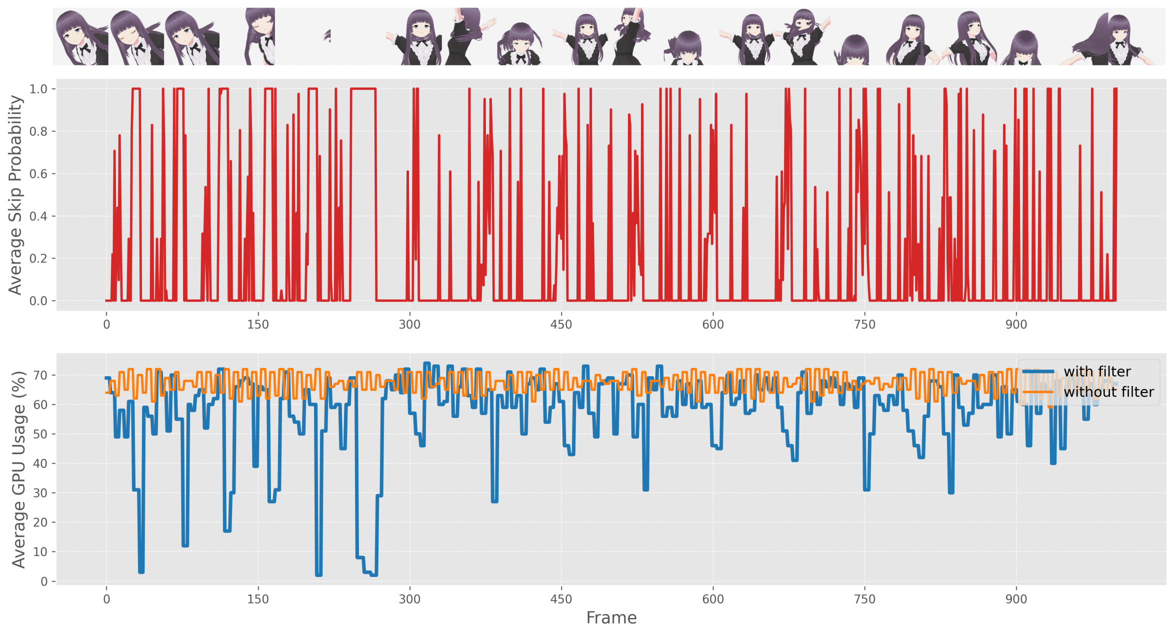

Figure 12. GPU Usage comparison under dynamic scene. (GPU: RTX4090, Number of frames: 1000) The blue line represents the GPU usage with SSF, the orange line indicates GPU usage without SSF, and the red line denotes the Skip probability calculated based on the cosine similarity between input frames. Additionally, 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.