StreamDiT: Real-Time Streaming Text-to-Video Generation

Supplementary Material

8. Inference Details

StreamDiT is specifically designed to achieve real-time responsiveness and interactivity, and its inference pipeline is structured accordingly. An overview of this pipeline is provided in Fig. 11. In the main thread (Thread 1), the system performs the denoising operation, refills the stream queue, and emits denoised video frames from the queue to forward them to a separate decoder thread (Thread 2). This decoder thread runs concurrently, decoding the latent video frames to actual video frames. These resulting frames are then rendered in real time, allowing users to observe the changes immediately.

Additionally, a prompt callback function operates continuously on another thread (Thread 3), listening for new user prompts in real time. When a user provides a new prompt, it is converted into text embedding by text encoders, and the embedding is sent to the DiT thread to update the existing embedding. Subsequent denoising steps then use this updated embedding through a cross-attention mechanism, changing the direction of text guidance dynamically (Fig. 12). This design enables users to interactively influence and modify video content in real time through prompt inputs.

In StreamDiT, since information from both preceding and succeeding video segments is always present in the context, it is essential to consider not only the explicit text guidance but also the implicit influence of video guidance. When the chunk size is small, the noise level difference between adjacent blocks in the Stream Queue decreases, amplifying the effect of video guidance. Therefore, to allow for larger content transformations, such as morphing, a larger chunk size is preferable. Conversely, if the goal is to maintain semantically continuous changes, such as variations in the walking direction of a character, a smaller chunk size is more suitable.

To perform stream denoising, the Stream Queue is filled with latent video frames that vary gradually in noise level. In the case where an initial video is provided in advance, such as in video-to-video scenarios, the video can first be encoded and then filled into the Stream Queue by adding Gaussian noise with appropriate stepwise noise levels. However, in the case of text-to-video generation, as shown in Fig. 13, the process starts with chunk generation using a standard T2V model. During this stage, the intermediate video latents at each denoising step must be cached. Afterward, from the cached intermediate latents, those corresponding to the appropriate noise levels and video frames are selected in accordance with the StreamDiT inference configuration, and used to populate the Stream Queue. Once the queue is prepared, infinite-length video generation becomes possible by autoregressively and continuously performing stream denoising. Furthermore, as illustrated in the bottom row of Fig. 13, when the number of stream chunks N is set to one and one or more reference frames are used ($K \geq 1$), conventional chunk-based video extension is also possible. Because of StreamDiT's highly flexible unified architecture, it is possible to choose the chunk size and block size mixed during training according to the intended use. This enables a single model to flexibly switch between various inference patterns during deployment.

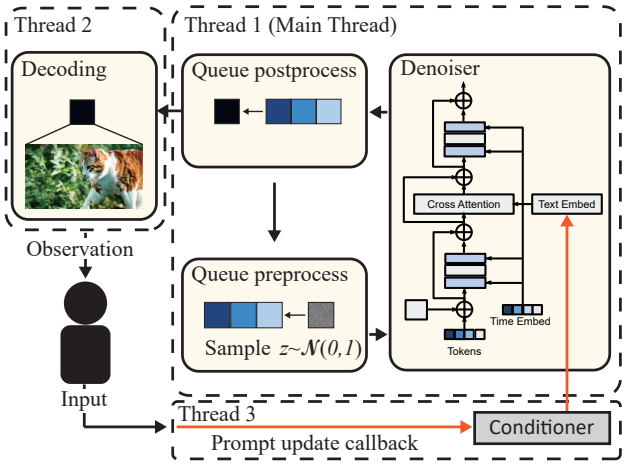

Figure 11. Interactive inference pipeline of StreamDiT: To decrease latency, generative models, decoder and text encoder are in separate process.

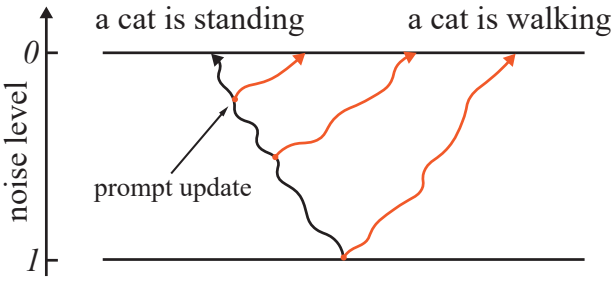

Figure 12. Denoising trajectory change with text guidance update: As denoising progresses toward the final stages, it becomes increasingly difficult to deviate from the outcome dictated by the original text guidance.

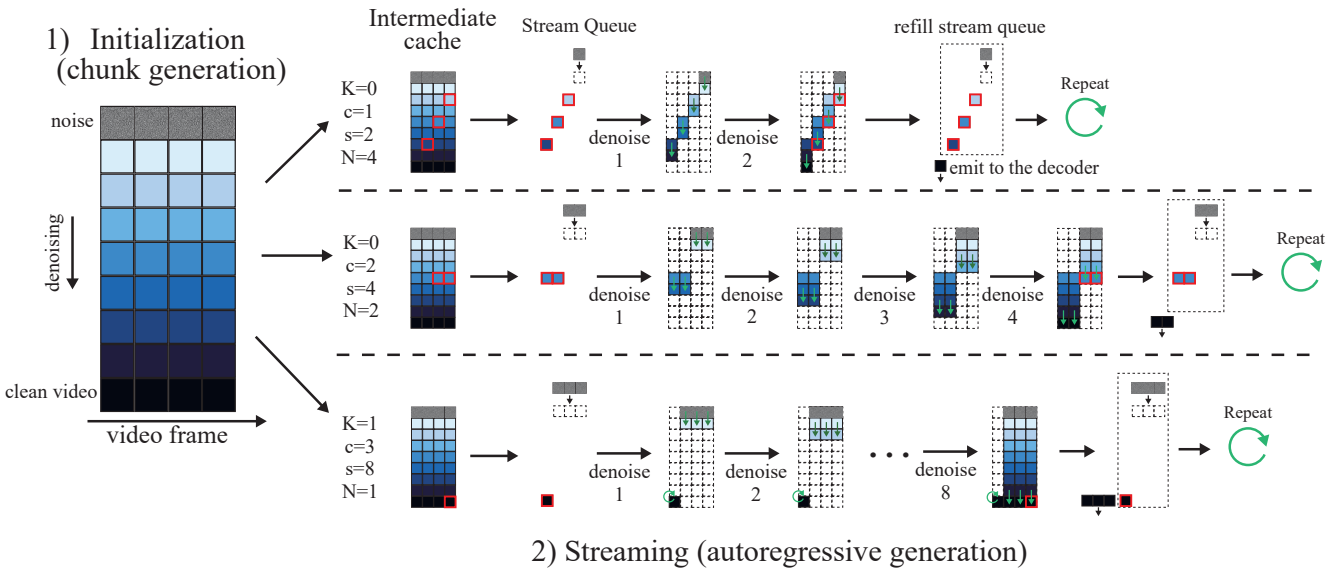

Figure 13. In the T2V scenario, StreamDiT first performs standard chunk generation to prepare intermediate latent cache. Once the intermediate cache is fully populated, the appropriate blocks are retrieved to construct the initial Stream Queue. Different inference configurations can be activated, provided that those configurations were included during mixed chunk training.

9. Design Choices and Training Details

Our design process begins by defining the global context length B of the base text-to-video (T2V) model. Once B is established, we determine a corresponding micro-step budget, enabling the informed selection of an appropriate chunk size c to meet latency constraints. The choice of chunk size critically balances responsiveness and visual fidelity; larger chunk sizes yield more stable and higher-quality outputs but incur increased latency. To ensure flexibility across various inference scenarios, we adopt mixed-chunk training, which minimally impacts overall performance. Notably, incorporating the full-context chunk size ($c = B$) into mixed-chunk training substantially improves output quality by preserving key characteristics of the original T2V model, as demonstrated in Tab. 3. For our distilled real-time model, we select a chunk size of $c = 2$ to achieve optimal performance at 16 FPS. Choosing a smaller chunk size ($c = 1$) would reduce performance to approximately 8 FPS in a step-distilled scenario. The micro-step size s is not tuned independently; rather, it is directly determined by the chosen chunk size and the total number of denoising steps ($T = s \times N$).

In this paper, we trained StreamDiT using a partitioning strategy based on a *linear noise schedule*. However, depending on the denoising scenario, it may be beneficial to adopt a *nonlinear noise schedule* during inference. For instance, stronger denoising may be preferred in earlier frames, while detailed reconstruction might be prioritized in later ones. To enhance the performance of stream denoising under such conditions, it is crucial that the noise distribution during training closely matches the intended inference-time

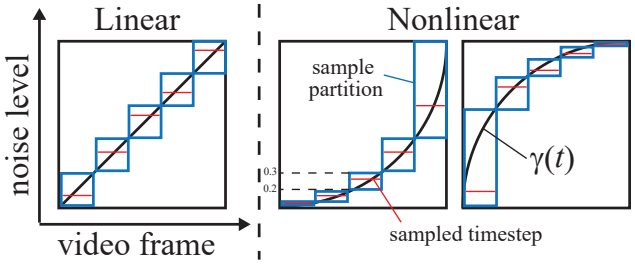

Figure 14. Partitioned Noise Training: The partitioning strategy can be defined by an arbitrary function—linear or nonlinear—that aligns with the chosen noise scheduling strategy.

noise schedule (Fig. 14). To achieve this, we define a general noise scheduling function $\gamma(t)$, mapping the normalized time domain $[0, 1]$ to the noise level range $[0, 1]$. Here, t represents normalized temporal positions, such as frame indices or timestamps. We partition the time domain into N equal segments, and within each segment $\left(\frac{i-1}{N}, \frac{i}{N}\right]$, we randomly select a time point t_i from a uniform distribution. The noise level at this sampled point, $\gamma(t_i)$, is then applied uniformly across the entire interval. The resulting stepwise noise function $\hat{\gamma}(t)$ is expressed as:

$$\hat{\gamma}(t) = \sum_{i=1}^{N} \gamma(t_i) \cdot \mathbf{1}\left[t \in \left(\frac{i-1}{N}, \frac{i}{N}\right]\right],$$
$$\text{where } t_i \sim \text{Uniform}\left(\left[\frac{i-1}{N}, \frac{i}{N}\right]\right). \quad (9)$$

This approach enables us to approximate various nonlinear

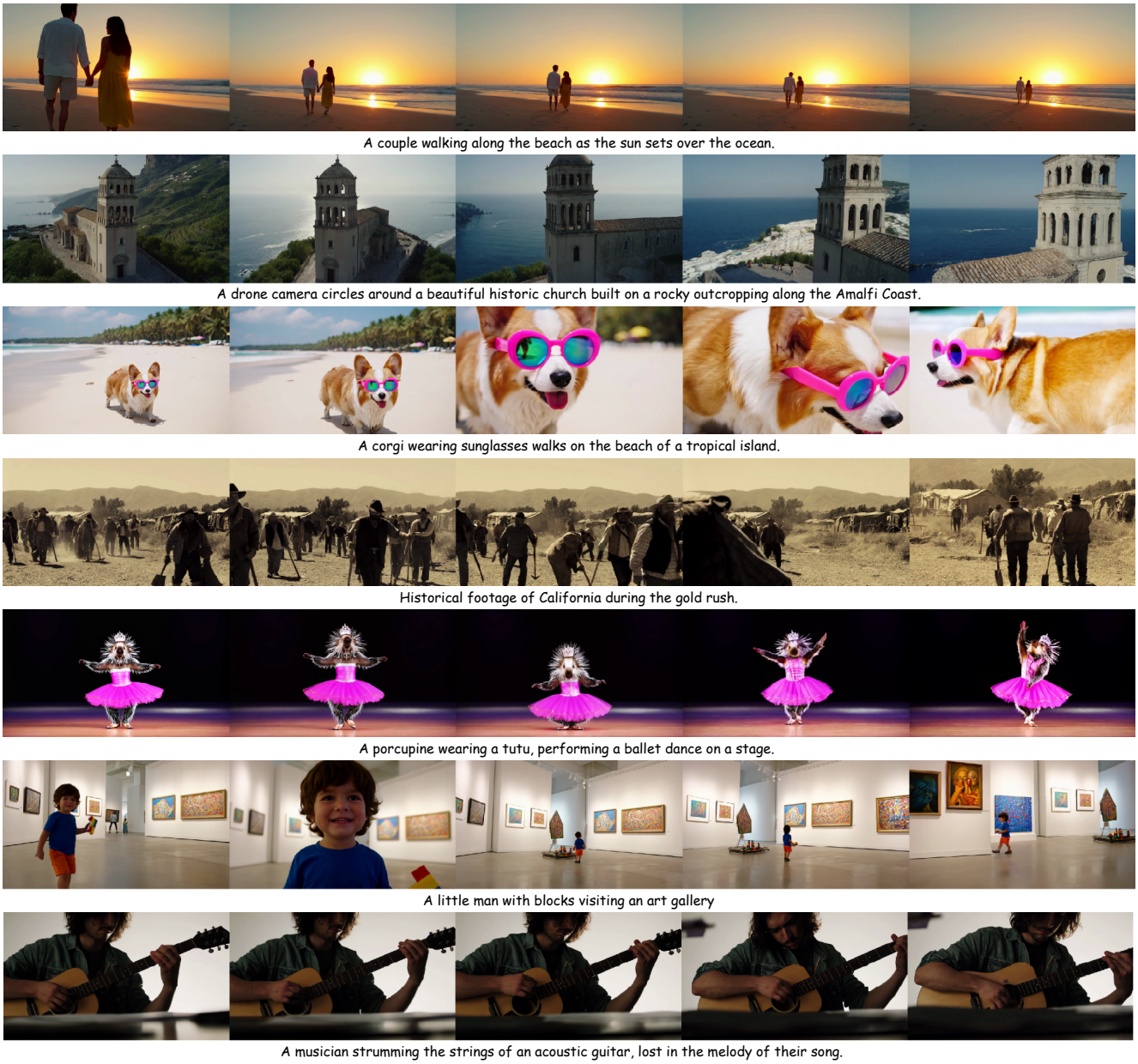

Figure 15. High-quality long videos generated by StremDiT-30B, demonstrating the scalability of our method.

noise schedules with an easily implementable step function. For example, a linear schedule corresponds to $\gamma(t) = t$, while exponential schedules can be modeled as $\gamma(t) = t^k$ with $k > 1$. Thus, our formulation provides flexibility to match different inference-time behaviors.

Furthermore, StreamDiT involves latent representations with different noise levels interacting within a shared context. Due to this design, the model is sensitive to the chosen noise scheduling strategy and the context length. Therefore, careful consideration is required when setting these parameters. For instance, if most frames are assigned high noise levels, these frames will convey significantly less information compared to frames with lower noise. This scenario effectively reduces the usable context size, restricts information flow across frames, and can increase the difficulty of training. Consequently, improper design of noise schedules can negatively impact the denoising quality and temporal consistency of the model's outputs. Hence, selecting an appropriate noise scheduling strategy is crucial for achieving optimal model performance.

10. More Results

To evaluate the capabilities in generating higher-quality videos, we further fine-tuned the 30B model from Movie

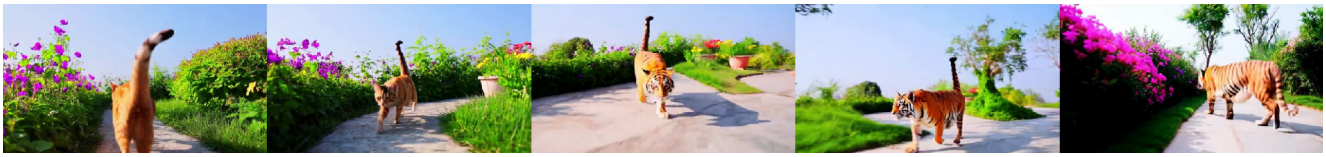

Figure 16. Sequential storytelling prompts can mitigate repetitive content and enable dynamic contents change.

Config		Quality	Subject	Background	Flickering	Motion	Dynamic	Aesthetic	Imaging
Reference frames	fixed 1	0.8175	0.9668	0.9769	0.9813	0.9910	0.4767	0.5500	0.6683
	fixed 2	0.8202	0.9643	0.9763	0.9810	0.9911	0.5533	0.5482	0.6536
	fixed 4	0.8151	0.9634	0.9748	0.9736	0.9886	0.5523	0.5292	0.6713
	fixed 8	0.8222	0.9664	0.9768	0.9809	0.9913	0.5320	0.5538	0.6680
	mixed 1	0.8228	0.9652	0.9765	0.9797	0.9907	0.5733	0.5421	0.6704
	mixed 2	0.8244	0.9657	0.9766	0.9787	0.9903	0.6067	0.5410	0.6685
	mixed 4	0.8230	0.9647	0.9769	0.9787	0.9902	0.5880	0.5413	0.6691
	mixed 8	0.8246	0.9654	0.9761	0.9784	0.9901	0.6120	0.5418	0.6688
T2V mix ratio	0.1	0.8187	0.9690	0.9796	0.9823	0.9909	0.4852	0.5437	0.6693
	0.3	0.8218	0.9611	0.9761	0.9817	0.9913	0.5600	0.5497	0.6605
	0.5	0.8186	0.9626	0.9752	0.9770	0.9893	0.5778	0.5396	0.6600

Table 4. Full VBench scores of our ablation studies on the 30B model.

Gen [30]. While the 30B model is not available for real-time generation at the present, it effectively supports the creation of exceptionally high-quality videos, as illustrated in Fig. 15. Our model consistently produces extended videos characterized by impressive visual quality, content coherence, and diverse scenes aligned accurately with provided text prompts.

Additionally, ablation studies on the 30B model using VBench are summarized in Tab. 4. The observed quality scores exhibit minimal variation, underscoring the stability and robustness of our method across various configurations.

Due to the inherent context-length limitations of base T2V models, even when augmented by autoregressive denoising improvements such as StreamDiT, the effective temporal context remains restricted. Consequently, repeatedly using the same prompt results in increasingly repetitive content, limiting the diversity of the generated videos. To address this limitation, we propose using a sequence of different story telling prompts during inference. This strategy maintains temporal coherence while allowing dynamic variation in the generated content (Sec. 8). As shown in Fig. 16, sequential storytelling prompts significantly mitigate repet-

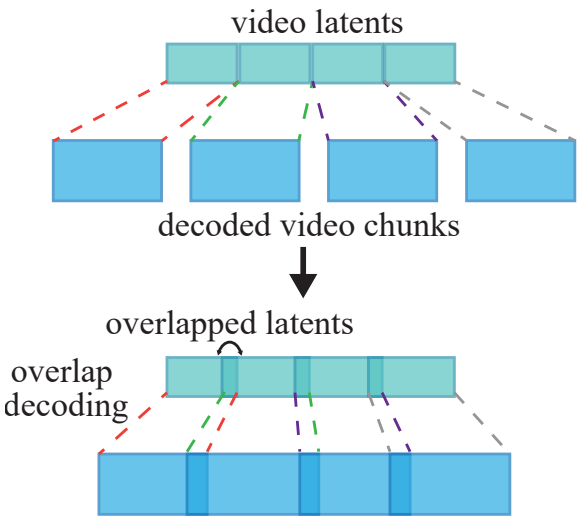

Figure 17. Overlap decoding

itive visual patterns and enhancing the feasibility in long-form video generation tasks. This approach facilitates the production of coherent videos with creative content, smooth object transformations, and seamless scene transitions.

11. Limitations

As described in Sec. 8, the effective context length of StreamDiT fundamentally depends on the base T2V model, and thus lacks long-term memory. When content falls outside the short-term memory window (*i.e.* context length) of StreamDiT, the associated information is likely to be lost. This can lead to issues such as identity mismatches in a person's face, or background inconsistencies when the camera makes a full rotation. However, since StreamDiT is orthogonal to additional long-term memory mechanisms, it is possible to address this issue in the future by combining it with long-term memory architectures such as State Space Models like Mamba [8].

Furthermore, with the current decoding strategy of StreamDiT, although the video frames are smoothly connected at the latent level, because video latent chunks are sequentially emitted from the Stream Queue and decoded separately, slight seams or flickering may be observed between decoded chunks in the final video. This occurs because a smooth connection at the latent level does not guarantee a seamless reconstruction in the decoded video. As a potential solution shown in Fig. 17, when decoding, one could concatenate the end of a cached previous video latent to the beginning of a newly emitted video latent from the Stream Queue to create an extended chunk, which is then decoded. This overlapping strategy helps to reduce the appearance of seams.

The improvement of StreamDiT with those additional approaches is left as future work.