Tuning-Free Multi-Event Long Video Generation via Synchronized Coupled Sampling

Supplementary Material

A. Foundational concepts and derivations

A.1. Further details on preliminaries

This section provides additional details on the preliminaries introduced in Section 2 to further guide the readers.

Diffusion models. Diffusion models are generative models that learn to gradually transform a noise sample from a tractable noise distribution towards a target data distribution. This transformation is consisting of two processes: a forward process and a reverse process. In the forward process, noise is incrementally added to the data sample over a sequence of timestep t, leading to a gradual loss of structure in the data. This forward process is defined as following:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \tag{4}$$

where $\bar{\alpha}_t$ is a pre-defined coefficient that regulates noise schedule over time, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ denotes a noise sampled from a standard normal distribution. The reverse diffusion process aims to invert the forward process, gradually removing the noise and recovering the original target data distribution. This reverse transformation is modeled using a neural network (referred to as the diffusion model, $\boldsymbol{\epsilon}_\Phi$), which is trained using a loss function based on denoising score matching [17, 44]. The loss function is defined as:

$$\mathcal{L}_{\texttt{diff}}(\Phi; \mathbf{x}) = \mathbb{E}_{t, \boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[w(t)\|\boldsymbol{\epsilon}_\Phi(\mathbf{x}_t; t) - \boldsymbol{\epsilon}\|_2^2\right], \tag{5}$$

where $w(t)$ is a weighting function applied to each timestep t, and $t \sim \mathcal{U}(0, 1)$ is drawn from a uniform distribution.

Score distillation sampling. Score distillation sampling (SDS) [35] introduces a new way to optimize any arbitrary differentiable parametric function g using diffusion models as a critic by posing generative sampling as an optimization problem. The flexibility of SDS in optimizing diverse differentiable operators has made it a versatile tool for visual tasks. Specifically, given $\mathbf{x} = g(\theta)$, the gradient of the diffusion loss function of Equation 5 with respect to the parameter θ is expressed as below:

$$\begin{aligned}&\nabla_\theta \mathcal{L}_{\texttt{SDS}}(\Phi; \mathbf{x} = g(\theta)) \\ &\quad\triangleq \mathbb{E}_{t,\boldsymbol{\epsilon}}\left[w(t)\left(\boldsymbol{\epsilon}_\Phi^\gamma(\mathbf{x}_t; \mathbf{c}, t) - \boldsymbol{\epsilon}\right)\frac{\partial \boldsymbol{\epsilon}_\Phi^\gamma(\mathbf{x}_t; \mathbf{c}, t)}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \theta}\right]\end{aligned} \tag{6}$$

In DreamFusion [35], it has been shown that by omitting the Jacobian term of the U-Net (the middle term) in Equation 6 yields an effective gradient for estimating an update direction that follows the score function of diffusion models, thereby moving towards a higher-density region. The simplified expression is given as Equation 2 in Section 2.

Collaborative score distillation sampling. The original SDS method considers only a single sample, $\mathbf{x}$, during the optimization process. Collaborative Score Distillation (CSD) [22] extends SDS to enable inter-sample consistency by synchronizing gradient updates across multiple samples, $\{\mathbf{x}^{(i)}\}_{i=1}^N$. Specifically, CSD generalizes SDS by using Stein Variational Gradient Descent (SVGD) [26] to align gradient directions across multiple samples as Equation 3 in Section 2. For a positive definite kernel, k, in Equation 3, CSD employs the standard Radial Basis Function (RBF) kernel. This approach ensures that each parameter update is influenced by other samples with scores adjusted via importance weights based on sample affinity, thereby effectively promoting inter-sample consistency during optimization.

A.2. SynCoS with different diffusion models

In this paper, we present our method using diffusion models with ϵ-prediction networks for clarity. However, our framework is compatible with any T2V denoising model. To illustrate this, we describe how applying our framework to different diffusion settings.

Flow-based models. Recently, rectified flow [25], a unified ODE-based framework for generative modeling, introduces a simplified denoising process that optimizes the trajectories in diffusion space to be as straight as possible. Given a data sample $\mathbf{x}$ from the data distribution and a noise sample $\boldsymbol{\epsilon}$ from a Gaussian distribution, rectified flow defines the forward process as:

$$\mathbf{x}_t = t\boldsymbol{\epsilon} + (1-t)\mathbf{x}_0, \quad t \in [0, 1] \tag{7}$$

Accordingly, the reverse process is governed by an ODE that maps $\boldsymbol{\epsilon}$ to $\mathbf{x}_0$:

$$d\mathbf{x}_t = \mathbf{v}(\mathbf{x}_t; t)dt, \quad t \in [0, 1] \tag{8}$$

where $\mathbf{v}$ is a velocity field estimated by a learnable neural network Φ, where the model is trained using the following objective:

$$\mathcal{L}_{\texttt{flow}}(\Phi; \mathbf{x}) = \mathbb{E}_{t,\boldsymbol{\epsilon}}\left[w(t)\|(\boldsymbol{\epsilon} - \mathbf{x}_0) - \mathbf{v}(\mathbf{x}_t; t)\|_2^2\right]. \tag{9}$$

Score distillation with flow-based models. While SDS and CSD have been implemented on denoising diffusion models with ϵ-prediction networks, the core concept of score distillation—using diffusion models as generative priors for

I would like you to act as a sentence separator, breaking down long sentences into a global prompt and a specified number of concise local prompts based on action verbs. The long sentence will be referred to as the "**Scenario**" and divided into a given number of short sentences, or "**Local Prompts**," based on the "**Number of Prompts**."

First, consider the context of the long sentence, replacing any pronouns with specific nouns mentioned in the sentence, using an indefinite article if necessary.

Next, break the modified long sentence into the specified number of local prompts, focusing on the main action verbs. Please follow these rules:

1. Each set should consist of one global prompt that is shared across all local prompts, along with the specified number of local prompts.
2. The global prompt should describe the shared properties across all the local prompts.
3. The global prompt must provide detailed, concise descriptions for the shared entity to avoid vagueness, enhancing clarity and context for the local prompts.
4. Each local prompt should include the global prompt.
5. Each local prompt should focus on only one action verb. The tense and form of these verbs must match the original long sentence.
6. Each local prompt should be self-contained, following the structure of subject, verb, and background.
7. Each local prompt should include all relevant background information linked to the main action verb.
8. No other verbs should be included apart from the specified main action verbs.
9. Avoid redundancy between the global prompt and local prompts; the global prompt should not contain the main actions described in the local prompts.
10. The tense of the local prompts (present, present progressive, or present participle) must align with the original long sentence.

Example:

Global prompt: An adorable kawaii cheeseball on the moon, smiling, 3D render, octane, soft lighting, dreamy bokeh, light sparkles floating in the background.

Local prompt 1: An adorable kawaii cheeseball on the moon, smiling, 3D render, octane, soft lighting, dreamy bokeh, light sparkles floating in the background. The cheeseball wiggles slightly, its rounded shape catching the soft light as it shimmers.

Local prompt 2: An adorable kawaii cheeseball on the moon, smiling, 3D render, octane, soft lighting, dreamy bokeh, light sparkles floating in the background. A soft sparkle lands on the cheeseball's surface, making it blush a gentle pink, adding to its cuteness.

Local prompt 3: An adorable kawaii cheeseball on the moon, smiling, 3D render, octane, soft lighting, dreamy bokeh, light sparkles floating in the background. The cheeseball's eyes blink slowly, a soft reflection of the stars above shimmering in them.

Local prompt 4: An adorable kawaii cheeseball on the moon, smiling, 3D render, octane, soft lighting, dreamy bokeh, light sparkles floating in the background. The cheeseball hops slightly on the moon's surface, leaving a tiny imprint in the soft, glowing moon dust.

From this point onward, I will provide prompts for you to rephrase according to these rules. The prompts may sometimes be simple, with minimal description for the entity or background. Feel free to enhance the details as needed to improve video quality.

Figure 8. **Instruction for generating structured prompt**. This instruction follows the guidelines to create individual local prompts and a shared global prompt based on a scenario and the number of prompts the user gives.

optimization—can also be extended to flow-based models. Below, we derive the equation for score distillation adapted to flow-based models. By computing the gradient of the training objective $\mathcal{L}_{\texttt{flow}}$ for flow-based models, Φ, with respect to θ, the score distillation sampling adapted to flow-based models can be expressed as:

$$\begin{aligned}&\nabla_{\theta_i}\mathcal{L}_{\texttt{Flow}-\texttt{SDS}}(\Phi, \mathbf{x}=g(\theta))\\ &\quad\triangleq \mathbb{E}_{\boldsymbol{\epsilon},t}\left[w(t)\left(\boldsymbol{v}_\Phi(\mathbf{x}_t,t)-(\boldsymbol{\epsilon}-\mathbf{x})\right)\frac{\partial \mathbf{x}}{\partial \theta}\right].\end{aligned} \tag{10}$$

Here, $w(t)$ is a time-dependent weighting function, and $\boldsymbol{v}_\Phi$ is estimated by the pre-trained flow-based denoising network Φ. In accordance with SDS conventions, the (transformer) Jacobian term is omitted to enhance computational efficiency, enabling the optimization of $\mathbf{x}$ using the rectified flow model. This loss is then could be reinterpreted within the Collaborative Score Distillation (CSD) framework [22], allowing for synchronous updates across multiple samples instead of treating each gradient independently.

B. Experimental details

Details of structured prompt. The structured prompt is designed to divide a cohesive story (*i.e.*, scenario) into multiple prompts, enabling the generation of long videos controlled by distinct prompts. It consists of a single global prompt that describes shared properties across all local prompts (e.g., object appearances or styles) and distinct local prompts that specify changes in objects, motions, or styles. To create a structured prompt, we leverage a Large Language Model (LLM), as in previous works [31]. The process involves instructing the LLM to segment a given scenario into multiple local prompts and a detailed global prompt, as illustrated in Figure 8. We use GPT-4o [32] to generate these structured prompts, ensuring consistency across all experiments. These structured prompts are applied in all experiments, including the main evaluations of our method and the baselines, as well as in ablation studies, ensuring a fair comparison.

To extensively verify our method under various challenging scenarios, we consider both previously established scenarios [21, 29, 31, 36, 47, 50] and our new, more challenging scenarios, including: background changes, object motion changes, camera movements, compositional generation, complex scene transitions, cinematic effects, physical transformation, and storytelling. We experiment with 48 structured prompt scenarios, including 11 with two local prompts, 12 with three, and 25 with four, extending video length by a factor of four or five. Full lists are provided on our project page: https://syncos2025.github.io/

Implementation details of the main experiments. DDIM sampling was performed with 50 steps, setting DDIM η to 0. The stride s is set to 4 for CogVideoX-2B and 6 for Open-

Table 3. **Quantitative ablations study of the three coupled stages in SynCoS**, omitting each stage during one-timestep denoising, demonstrates the importance of all three stages for coherent long video generation with multiple events.

Backbone	Stage 1	Stage 2	Stage 3	Temporal Quality: Subject Consistency ↑	Temporal Quality: Background Consistency ↑	Temporal Quality: Motion Smoothness ↑	Temporal Quality: Dynamic Degree ↑	Temporal Quality: Num Scenes ↓	Frame-wise Quality: Aesthetic Quality ↑	Frame-wise Quality: Imaging Quality ↑	Semantics: Glb Prompt Fidelity ↑	Semantics: Loc Prompt Fidelity ↑
M	✓	✗	✓	80.46%	91.14%	**98.55%**	97.92%	1.229	53.36%	58.42%	0.318	**0.348**
	✗	✓	✗	78.88%	91.63%	97.70%	14.58%	21.33	45.56%	42.27%	0.305	0.300
	✓	✓	✓	**82.70%**	**91.85%**	98.24%	**100.00%**	**1.042**	**54.56%**	**65.53%**	**0.325**	**0.348**

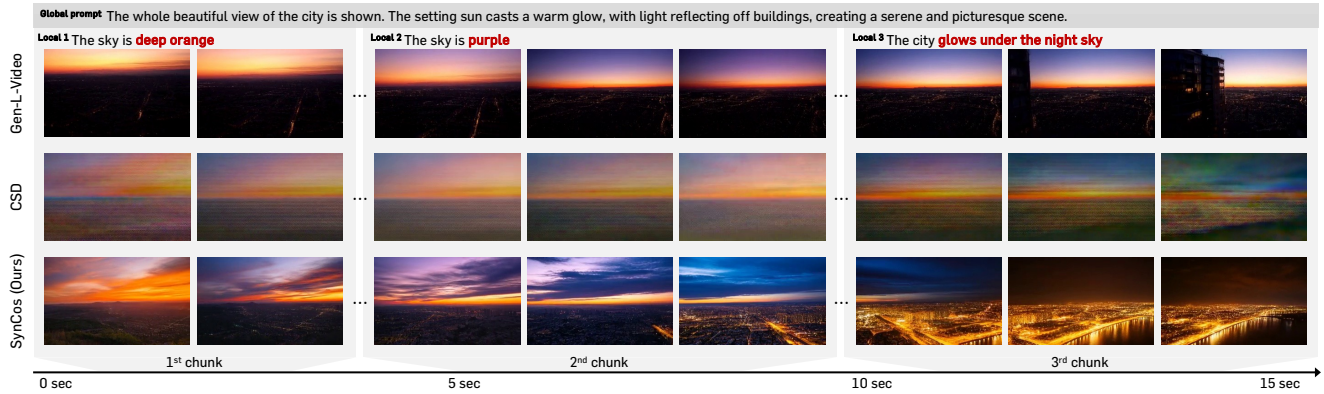


Figure 9. **Qualitative visualization of the ablation study on the three coupled stages of SynCoS**. All examples in the second box are 3 times longer in duration compared to the underlying base model.

Sora Plan (v1.3). In second stage, we set $t_{\text{min}} \in [800, 900]$, learning rate, $lr \in [0.5, 1]$ using AdamW Optimizer [28], and $iters \in [20, 50]$. The scale of the classifier-free guidance is set to 6 for CogVideoX-2B and 7.5 for Open-Sora Plan (v1.3). For Gen-L-Video, we use the same strides and guidance scale as in our experiments. For FIFO-Diffusion, we set $n = 4$ for the number of partitions in latent partitioning and lookahead denoising, following their best parameter configurations.

Measurements of computation time. We measure the computation time required to generate a $4\times$ longer video compared to the underlying base model of CogVideoX-2B, using a single H100 80GB GPU (Table 4), including both main baselines and our approach. Although our method takes $1.38\times$ longer than the baselines, this overhead remains comparable while delivering substantial gains in quality and consistency, leading to superior quantitative and qualitative performance. With a multi-GPU implementation, the runtime can be further reduced to **3.9 minutes**. In addition, the computation cost can be flexibly tuned based on video scenarios and synchronization requirements—for instance, reducing the number of iterations in the second-stage optimization lowers runtime.

C. Additional ablations

Effect of the three coupled stage. In addition to Section 3 and Figure 4, we provide further quantitative evaluations (Table 3) and qualitative visualizations (Figure 9) by skipping each stage in SynCoS to assess the efficacy of its three-stage process in generating high-quality multi-event long videos.

Table 4. Measurements and comparisons of computation time on CogVideoX-2B.

Gen-L-Video [50]	FIFO [21]	SynCoS (Ours)
21 min.	21 min.	29 min.

As discussed in Section 3.1, omitting the first stage in SynCoS, which corresponds to temporal co-denoising with DDIM (*i.e.*, Gen-L-Video), causes denoising paths across chunks to diverge, leading to overlapping artifacts, as shown in Figure 4. This results in reduced temporal consistency and frame-wise quality, as quantified in Table 3, due to abrupt changes. Additionally, this variant often struggles to faithfully follow prompts, as the simple fusion of denoising paths dilutes prompt guidance unique to each chunk. This issue is evident in the second example of a city transition (Figure 9), where the scene fails to properly reflect changes in glowing, particularly in the third chunk.

Conversely, relying solely on the second stage, corresponding to temporal co-denoising with CSD, degrades image quality significantly. As shown in Figure 4 and Figure 9, the video exhibits noise-like artifacts, leading to a severe loss of frame-wise quality and prompt fidelity, as quantified in Table 3. While this approach integrates information across local and distant video frames, it does not produce

high-quality long videos, as the resulting video suffers from low image quality, completely failing for high-quality long video generation.

In contrast, SynCoS effectively couples both stages, leveraging the output of the first stage as a refining source for the second stage, which enhances inter-sample long-range consistency. This integration enables high-quality, long video generation with multiple prompts, achieving smooth transitions, semantic consistency throughout the video, and strong prompt fidelity for each chunk.

Ablation study on stride. By default, we set the stride (step size between chunks) to 4 for CogVideoX-2B without extensive tuning for each scenario. For prompts with frequent content changes, reducing the stride improves long-term consistency by increasing information sharing across overlaps. Users can adjust the stride to balance vibrant changes (larger stride) and stronger synchronization (smaller stride), enhancing content consistency. Figure 10 provides qualitative evidence: while a stride of 4 introduces slight variations in the knight's appearance, a stride of 1 ensures greater consistency throughout.

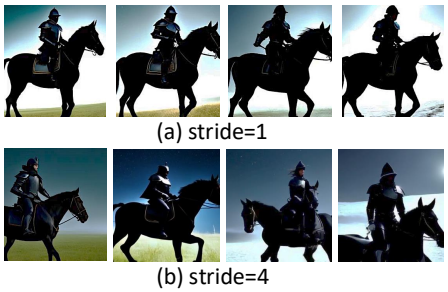
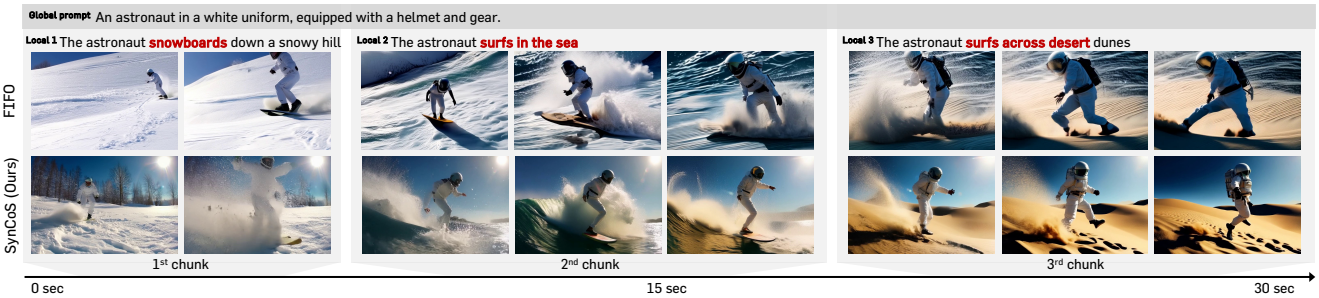
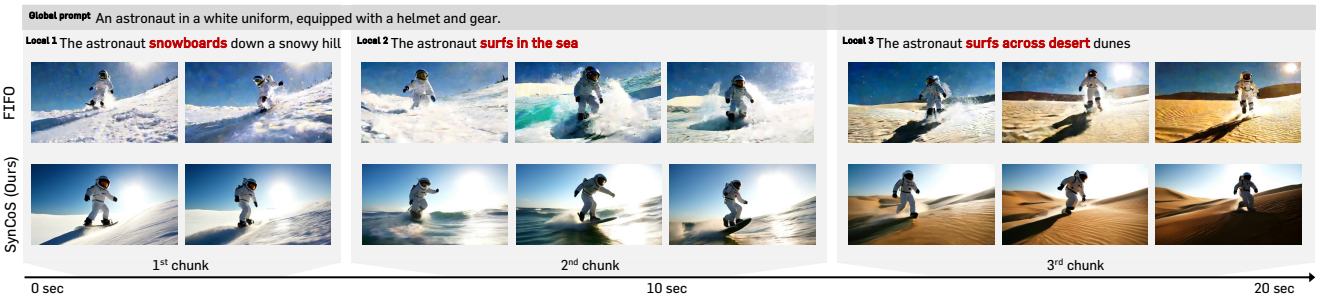


Figure 10. **Qualitative ablation study on stride.** Reducing the stride enhances content consistency, which is beneficial in scenarios like a knight running as the background transitions from grass to snow.

Although reducing the stride increases computation time slightly—from 55 minutes (Table 4) to 60 minutes—the impact is minimal. This is because stride reduction does not affect the second stage optimization time, where most of the computational overhead occurs. Instead of processing all chunks simultaneously in this stage, SynCoS uses a minibatch approach, randomly selecting B chunks from the total N chunks at each iteration. Since B remains the same for both stride 1 and stride 4, the overall optimization cost remains largely unaffected.



(a) Long video generation on CogVideoX-2B, where a single video chunk consists of 26 frames.



(b) Long video generation on Open-Sora Plan (v1.3), where a single video chunk consists of 49 frames.

Figure 11. **Long video generation results**. All generated videos are 4 times longer than the underlying base model. FIFO significantly suffers from noise-like artifacts on Open-Sora Plan (v1.3) due to inevitable training-inference discrepancy in their design.

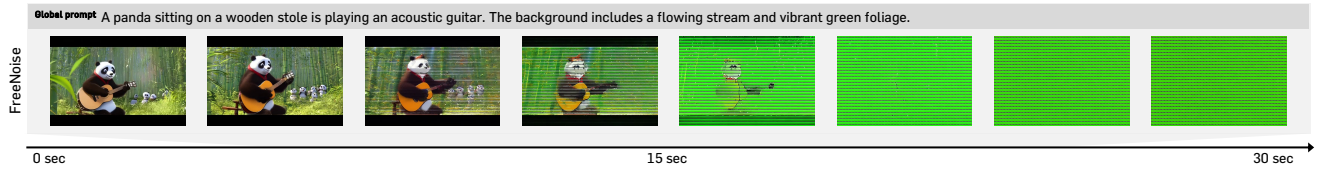


Figure 12. **Long video generation result of FreeNoise on CogVideoX-2B**. This generated video is 4 times longer than the underlying base model. Each frame is generated well in the early frames, where no fusion is applied. However, as soon as the fusion of attentional features is applied, the generated video shows stagnant results of repeated object motion without any scene changes, eventually leading to the entire failure of video generation.

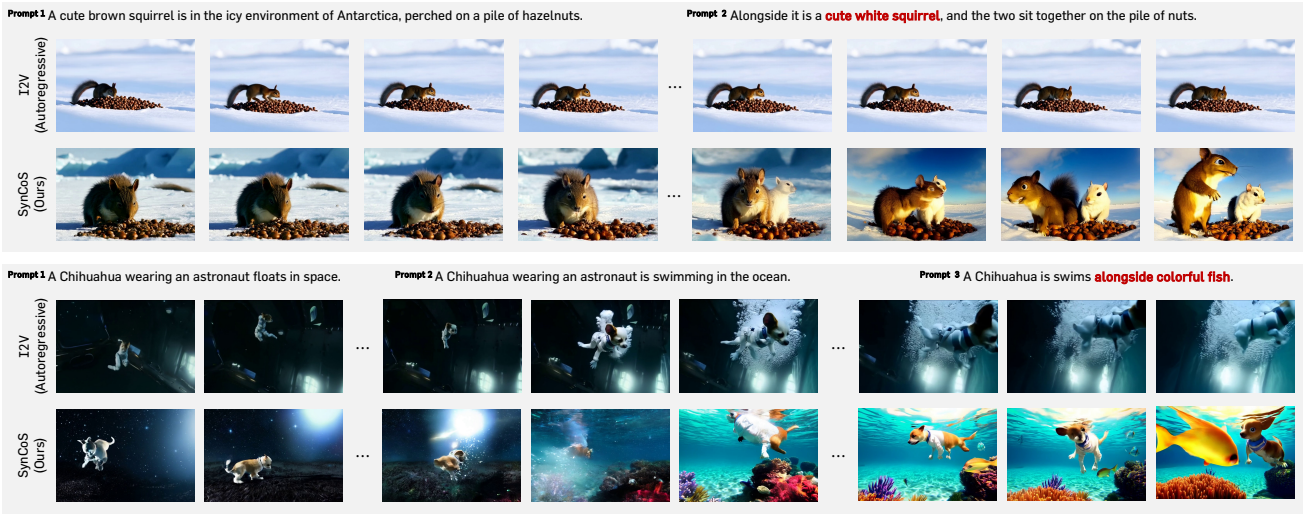


Figure 13. **Qualitative comparison** with autoregressive generation using image-to-video (I2V) model for long videos. While autoregressive generation with I2V models effectively handles scene transitions, it often struggles to introduce new components into the video.

D. Further discussions on previous work

D.1. Limitations of previous approaches

Following the recent success of T2V models, several works have explored extending video diffusion models for longer video generation without additional training. However, we observe that current tuning-free approaches often exhibit undesirable artifacts when applied to recent video diffusion models. In the following sections, we will detail the limitations and failures of these existing methods.

Discussion on FIFO-Diffusion. In video diffusion models, all frames within a single video chunk are processed at the same timestep during both training and inference. FIFO-diffusion [21] proposes a new sampling technique that uses a queue to store a series of consecutive frames, with the timestep increasing for each frame. While this approach enables the generation of infinitely long videos, it introduces avoidable discrepancies between the timesteps used during training and those used during inference. These discrepancies become more pronounced when applied to recent video diffusion models. This is because as the number of frames processed within a single chunk increases, the timestep gap between frames in the same chunk widens. For example, FIFO-diffusion is not susceptible to per-frame artifacts on CogVideoX-2B [55], which encodes 13 frames per chunk. However, in Open-Sora Plan (v1.3) [23], which encodes 24 frames per chunk, these artifacts become significantly more noticeable, as shown in Figure 12. We note that in contrast to existing tuning-free long video generation methods, SynCoS does not introduce training-inference discrepancy and can be seamlessly applied to any video diffusion model.

Discussion on FreeNoise. The video diffusion models as-is often lack the capability to maintain content consistency across different video chunks beyond a single video chunk. FreeNoise [36] addresses this limitation by fusing attention features from temporal layers to establish long-range temporal correlations between different chunks. While this method works effectively with earlier video diffusion models that separate spatial and temporal attention layers (*i.e.*, enabling the fusion of only temporal attention features), we observe that it is not applicable to newer video diffusion models, as illustrated in Figure 12. This limitation arises because recent DiT-based models, widely used in frontier T2V approaches [23, 55], lack dedicated temporal layers. Instead, these models tokenize an entire video chunk into patches and apply attention across all patches. As a result, fusing all attention features without additional considerations severely degrades performance, demonstrating that naive fusion techniques are unsuitable for DiT-based models in long video generation.

D.2. Additional comparisons with architecture-specific approaches

In our main experiment (Section 5), we primarily compare SynCoS with architecture-compatible, tuning-free methods for multi-event long video generation. These approaches remain compatible with newer diffusion models, leveraging backbone improvements for enhanced generation quality. However, several existing methods [5, 29, 36, 45] are restricted to specific diffusion backbones, making them incompatible with newer or alternative models and limiting their performance to existing architectures.

Nonetheless, to ensure a comprehensive and fair comparison, we implement SynCoS using the respective architectures of existing tuning-free methods and evaluate its performance in Table 5. We compare SynCoS with Video-Infinity [45] and FreeNoise [36] by applying SynCoS to VideoCrafter2 [52], a U-Net-based video diffusion model. Additionally, to compare against DitCtrl [5] (built on Multi-Modal Diffusion Transformer (MM-DiT)), we evaluate SynCoS under the same backbone of CogVideoX-2B [55].

Notably, SynCoS significantly outperforms all baselines in temporal consistency and video quality while achieving comparable prompt fidelity, demonstrating its robustness across various diffusion backbones.

Table 5. **Quantitative comparison with architecture-specific baselines**. *Abbreviations: subject consistency (SC), background consistency (BC), aesthetic quality (AQ), and prompt fidelity (PF).

Backbone	Method	Temporal		Frame	Semantics
		SC ↑	BC ↑	AQ ↑	PF ↑
VideoCrafter2 [52]	Video-Infinity [45]	0.879	0.943	0.645	**0.365**
	FreeNoise [36]	0.904	0.946	**0.650**	0.356
	SynCoS	**0.911**	**0.947**	0.648	**0.365**
CogVideoX-2B [55]	DitCtrl [5]	0.821	0.916	0.635	**0.394**
	SynCoS	**0.864**	**0.927**	**0.643**	0.381

These results are qualitatively supported by Figures 14 and 15, where SynCoS achieves both coherent astronaut appearance and the highest prompt fidelity.

Comparisons with autoregressive generation using I2V. Long videos can also be generated using image-to-video (I2V) models by generating a single video chunk with T2V and then conditioning the last frame of the previous chunk to the I2V model. For two reasons, we do not directly include autoregressive generation using the I2V model in the main paper. First, using an I2V model requires a different prompt structure. Unlike our structured prompts, which focus on describing changes in local chunks, I2V models require explicit descriptions of transitions between chunks. For example, to generate an artistic video of a butterfly on a flower across changing seasons, our structured prompts might include: (1) "In spring, a butterfly is on a flower," and (2) "In summer, a butterfly is on a flower." In contrast, I2V prompts would need to explicitly describe transitions, such as: (1) "In spring, a butterfly is on a flower," and (2) "The season changes from

spring to summer, and the butterfly is on a flower." Second, the base models used in our experiments do not support I2V generation.

Nonetheless, to comprehensively analyze long video generation, we compare SynCoS with an I2V model (CogVideo-5X-I2V) that uses a backbone supporting I2V generation and appropriately tuned prompts. As shown in Figure 13, I2V-based autoregressive generation enables smooth scene transitions (e.g., a Chihuahua floating in space transitioning to the Chihuahua swimming in the ocean). However, it often struggles to introduce new objects (e.g., a brown squirrel and a white squirrel, or a Chihuahua with fish), limiting its ability to generate long videos that naturally add or change components over time.

D.3. Longer video generation results and comparison with PA-VDM

SynCoS can generate 1-minute single-event and 2-minute multi-event videos without any quality degradation (Figure 16 and Figure 17). In Figure 16, the *training-based* PA-VDM [54], limited to single-event, shows temporal quality degradation and noticeable subject drift. In contrast, our *tuning-free* framework maintains consistent character appearance and visual quality throughout the entire sequence.

D.4. Single-event long video generation

We present 4× longer single-event video generation results in Table 6 using the same prompts as those used in the baselines. Our method consistently outperforms baselines. We will also include qualitative examples in the final manuscript to further support these results.

Table 6. **Quantitative comparisons of single-event long video generations on CogVideoX-2B.** *Abbreviations: subject consistency (SC), background consistency (BC), aesthetic quality (AQ), and prompt fidelity (PF)

	Temporal		Frame	Semantics
Method	SC ↑	BC ↑	AQ ↑	PF ↑
Gen-L-Video	0.897	0.941	0.623	**0.388**
FIFO	0.874	0.922	0.604	0.383
SynCoS	**0.927**	**0.951**	**0.633**	**0.388**

Figure 14. **Qualitative comparison with DiTCtrl on CogVideoX.**

Figure 15. **Qualitative comparisons on VideoCrafter2.**

Figure 16. **1-minute video comparison with PA-VDM.**

Figure 17. **2-minute (3072 frames) videos generated by SynCoS.**

E. Pseudo-code implementation of each stage.

We describe each stage with pseudo-code implementations. In the first stage of temporal co-denoising with DDIM, the video is divided into overlapping chunks, where each chunk undergoes a diffusion forward pass and a DDIM update to compute $\mathbf{x}_0^{(i)}$. The chunks are then fused to obtain $\mathbf{x}'_0$, as detailed in Figure 18. In the second stage, the fused $\mathbf{x}'_0$ is further refined by re-dividing it into overlapping chunks and applying CSD-based optimization. While we use CSD-based optimization to enforce global coherence, it differs from the original CSD, as the second stage is adjusted using a grounded timestep and a fixed baseline noise, ensuring synchronous coupling across all three stages. The iterative refinement process is illustrated in Figure 19. Lastly, the refined $\mathbf{x}'_0$ from the second stage is converted using baseline noise to prepare for subsequent steps in the diffusion pipeline.

F. Additional qualitative results

We consider long videos of multiple events with the following challenging scenarios: object motion control, cinematic effects, storytelling, camera control, background changes, physical transformations, complex scene transitions, and compositional generation. For generated videos handling object motion control and camera control scenes, please refer to Figure 1, and other video examples are visualized in Figure 20 and Figure 21. To view all generated videos, refer to our project page: https://syncos2025.github.io/.

```python
def first_stage(
    videos,                       # video tensor of shape [T, C, H, W]
    prompt_embeds_for_chunks,      # Text prompt embeddings for each chunk
    model,                         # The denoising model
    scheduler,                     # Scheduler to refine videos
    guidance_scale,                # Scale factor for classifier-free guidance
    timestep,                      # Current timestep in the diffusion process
    stride                         # Stride size for dividing videos
):

    # Initialize list to store video chunks and count tensor
    videos_for_chunks = []
    count = torch.zeros_like(videos)

    # Split videos into strides for diffusion forward
    for start in range(0, videos.shape[0], stride):
        # Determine the end of the current stride
        end = start + stride

        # Slice the videos for the current stride
        chunks = videos[start:end]

        # Increment count for the covered range
        count[start:end] += 1

        # Append the current stride to the list for processing
        videos_for_chunks.append(chunks)

    # Concatenate all processed video chunks into a single tensor for inference
    videos_for_chunks_input = torch.cat(videos_for_chunks, dim=0)

    # Diffusion forward
    noise_pred = predict_noise(model, videos_for_chunks_input, prompt_embeds_for_chunks, timestep)

    # Apply classifier-free guidance to refine noise predictions
    noise_pred = apply_classifier_free_guidance(noise_pred, guidance_scale)

    # Perform a scheduler step to update videos and extract the denoised sample
    res = scheduler_step(scheduler, noise_pred, timestep, videos_for_chunks)
    videos_x0 = res.pred_original_sample

    # Initialize the aggregated result tensor
    value_x0 = torch.zeros_like(videos)

    # Aggregate the results into `value_x0`
    idx = 0
    for start in range(0, videos.shape[0], stride):
        # Define the range for the current stride
        start + stride

        # Accumulate the processed values for the corresponding stride
        value_x0[start:end] += videos_x0[idx]
        idx += 1

    # Compute the final denoised prediction by fusing
    # Wherever `count > 0`, compute the averaged value, else fallback to `value_x0`
    pred_original_sample = torch.where(count > 0, value_x0 / count, value_x0)

    return pred_original_sample
```

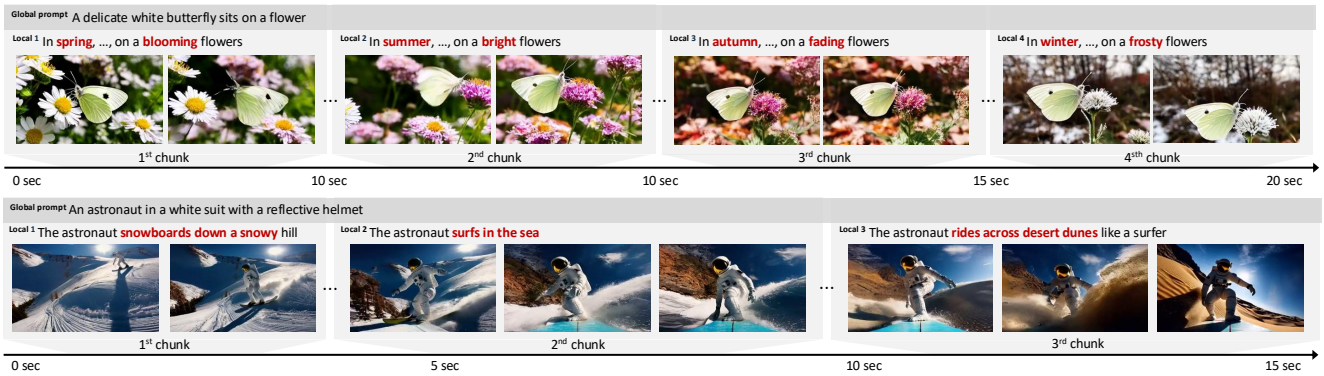
Figure 18. **Pseudo-code implementation of the first stage of SynCoS**.

```python
def second_stage(
        pred_original_sample,  # Initial video prediction
        prompt_embeds_for_chunks,      # Text prompt embeddings for each chunk
        timestep,              # A grounded timestep
        cfg_scale,             # Guidance scale for classifier-free guidance
        opt                    # Optimization hyperparameters and options
    ):


    # Initialize target predictions with gradients enabled
    tgt_pred_sample = pred_original_sample.clone().detach()
    tgt_pred_sample.requires_grad = True

    # Optimization hyperparameters
    lr = opt.lr
    wd = opt.wd
    decay_iter = opt.decay_iter
    decay_rate = opt.decay_rate
    num_steps = opt.num_steps
    batch_size = opt.batch_size
    stride = opt.stride

    # Initialize optimizer and learning rate scheduler
    optimizer = opt.optimizer([tgt_pred_sample], lr=lr, weight_decay=wd)
    scheduler = opt.scheduler(optimizer, step_size=decay_iter, gamma=decay_rate)

    # Fix baseline noise for the entire tensor
    base_noise_all = torch.randn_like(tgt_pred_sample)

    for step in range(num_steps):
        optimizer.zero_grad()

        # Initialize tensors for counting and score accumulation
        count = torch.zeros_like(tgt_pred_sample, dtype=tgt_pred_sample.dtype)
        score_value = torch.zeros_like(tgt_pred_sample, dtype=tgt_pred_sample.dtype)

        # Initialize lists to store videos and noise chunks
        videos_for_chunks = []
        basenoise_chunks = []
        prompt_embeds_chunks = []

        # Determine valid subset indices for this iteration
        num_chunks = (tgt_pred_sample.shape[0] + stride - 1) // stride
        subset_indices = torch.randperm(num_chunks)[:batch_size]

        # Process selected chunks
        for i, start in enumerate(range(0, tgt_pred_sample.shape[0], stride)):
            if i not in subset_indices:
                continue

            # Determine the end of the current stride
            end = start + stride

            # Slice the videos for the current stride and add noise
            chunks = tgt_pred_sample[start:end]
            noisy_chunks = add_noise(chunks, base_noise_all[start:end], timestep)

            basenoise_chunks.append(base_noise_all[start:end].unsqueeze(0))
            prompt_embeds_chunks.append(prompt_embeds_for_chunks[i])

            # Update count and append to chunks list
            count[start:end] += 1
            videos_for_chunks.append(noisy_chunks.unsqueeze(0))

        # Concatenate videos for model inference
        videos_for_chunks_input = torch.cat(videos_for_chunks, dim=0)
        basenoise_chunks = torch.cat(basenoise_chunks, dim=0)
        prompt_embeds_chunks = torch.cat(prompt_embeds_chunks, dim=0)

        # Predict noise using the model
        noise_pred = predict_noise(opt.model, videos_for_chunks_input, prompt_embeds_chunks, timestep)

        # Apply classifier-free guidance
        noise_pred = apply_classifier_free_guidance(noise_pred, cfg_scale)

        # Compute CSD scores
        scores = calculate_csd_loss(videos_for_chunks_input, noise_pred, basenoise_chunks)

        # Aggregate scores back to the appropriate positions
        for i, start in enumerate(range(0, tgt_pred_sample.shape[0], stride)):
            if i not in subset_indices:
                continue

            # Determine the end of the current stride
            end = start + stride

            # Accumulate scores for the current range
            score_value[start:end] += scores[start:end]

        # Compute gradients using scores normalized by count
        grad_all = torch.where(count > 0, score_value / count, score_value)
        tgt_pred_sample.backward(gradient=grad_all)

        # Perform optimization step
        optimizer.step()
        scheduler.step()

    # Detach and return the updated target prediction
    tgt_pred_sample = tgt_pred_sample.clone().detach().requires_grad_(False)

    return tgt_pred_sample, base_noise_all
```

Figure 19. **Pseudo-code implementation of the second stage of SynCoS**.

(a) Cinematic effects

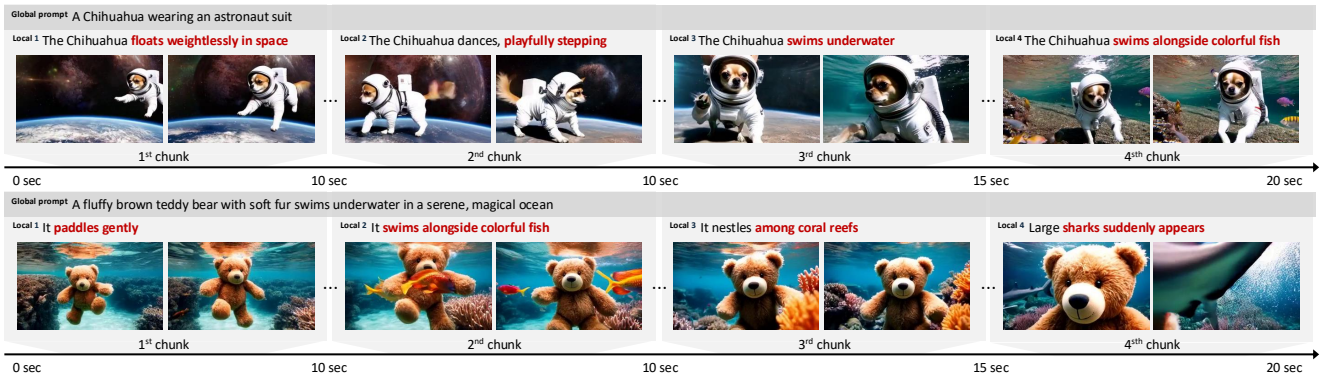
(b) Storytelling



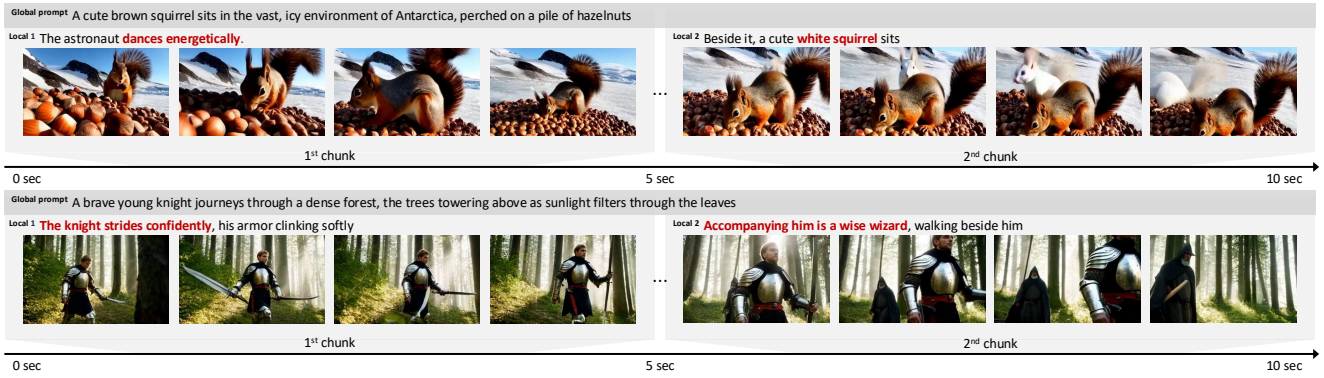
(c) Background changes

Figure 20. **Multi-event long video generation results** showcasing challenging scenarios, including cinematic effects, storytelling, and background changes. Each example is 2-4 times longer in duration compared to the underlying base model, resulting in 11 to 21-second videos at 24 fps, with a total of 256 to 512 frames.

(d) Physical transformation



(e) Complex scene transitions



(f) Compositional generation

Figure 21. **Multi-event long video generation results** showcasing challenging scenarios, including physical transformation, complex scene transitions, and compositional generation. Each example is 2-4 times longer in duration compared to the underlying base model, resulting in 11 to 21-second videos at 24 fps, with a total of 256 to 512 frames.