One-Minute Video Generation with Test-Time Training

Karan Dalal*[4] Daniel Koceja*[2] Gashon Hussein*[2] Jiarui Xu*[1,3] Yue Zhao†[5] Youjin Song†[2]
Shihao Han[1] Ka Chun Cheung[1] Jan Kautz[1] Carlos Guestrin[2] Tatsunori Hashimoto[2] Sanmi Koyejo[2]
Yejin Choi[1] Yu Sun[1,2] Xiaolong Wang[1,3]
[1]NVIDIA [2]Stanford University [3]UCSD [4]UC Berkeley [5]UT Austin

Figure 1. TTT layers enable a pre-trained Diffusion Transformer to generate one-minute videos from text storyboards. We use *Tom and Jerry* cartoons as a proof of concept. The videos tell complex stories with coherent scenes composed of dynamic motion. Every video is produced directly by the model in a single shot, without editing, stitching, or post-processing. Every story is newly created.

Abstract

Transformers today still struggle to generate one-minute videos because self-attention layers are inefficient for long context. Alternatives such as Mamba layers struggle with complex multi-scene stories because their hidden states are less expressive. We experiment with Test-Time Training (TTT) layers, whose hidden states themselves can be neural networks, therefore more expressive. Adding TTT layers into a pre-trained Transformer enables it to generate one-minute videos from text storyboards. For proof of concept, we curate a dataset based on Tom and Jerry cartoons. Compared to baselines such as Mamba 2, Gated DeltaNet, and sliding-window attention layers, TTT layers generate much more coherent videos that tell complex stories, leading by 34 Elo points in a human evaluation of 100 videos per method. Although promising, results still contain artifacts, likely due to the limited capability of the pre-trained 5B model. The efficiency of our implementation can also be improved. We have only experimented with one-minute videos due to resource constraints, but the approach can be extended to longer videos and more complex stories.

Sample videos, code and annotations are available at: https://test-time-training.github.io/video-dit

1. Introduction

Despite the remarkable progress in visual and physical realism, state-of-the-art video Transformers are still generating mostly short clips of single scenes without complex stories. At the time of writing (March 2025), the maximum length of public APIs for video generation is 20 seconds for Sora (OpenAI), 16 seconds for MovieGen (Meta), 10 for Ray 2

*Joint first authors. † Joint second authors.

Accepted to The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2025

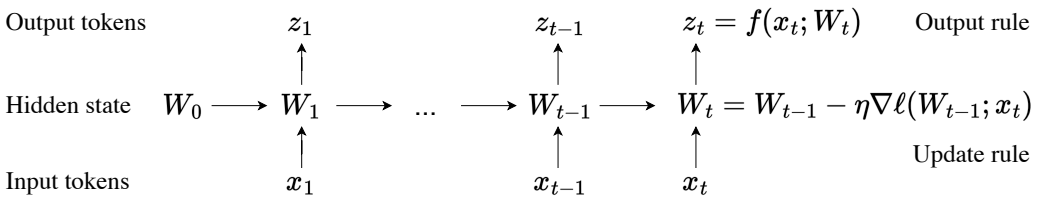


Figure 2. All RNN layers can be expressed as a hidden state that transitions according to an update rule. The key idea in [43] is to make the hidden state itself a model f with weights W, and the update rule a gradient step on the self-supervised loss ℓ. Therefore, updating the hidden state on a test sequence is equivalent to training the model f at test time. This process, known as Test-Time Training (TTT), is programmed into TTT layers. Figure and caption taken from [43].

(Luma), and 8 for Veo 2 (Google). None of these APIs can autonomously generate complex multi-scene stories.

A fundamental challenge behind these technical limitations is long context, because the cost of self-attention layers in Transformers increases quadratically with context length. This challenge is especially acute for video generation with dynamic motion, whose context cannot be easily compressed by a tokenizer. Using a standard tokenizer, each of our one-minute videos requires over 300k tokens in context. With self-attention, generating a one-minute video would have taken $11\times$ longer than generating 20 videos of 3 seconds each, and training would have taken $12\times$ longer.

To address this challenge, recent work on video generation has investigated RNN layers as an efficient alternative to self-attention, because their cost increases linearly with context length [47]. Modern RNN layers, especially variants of linear attention [23, 37] such as Mamba [8, 12] and DeltaNet [35, 53], have shown impressive results for natural language tasks. However, we have yet to see long videos with complex stories or dynamic motion generated by RNNs. Videos (link) in [47] are high resolution and one-minute long, but contain only single scenes and slow motion, let alone complex stories.

We believe that these RNN layers generate less complex videos because their hidden states are less expressive. RNN layers can only store past tokens into a hidden state of fixed size, which is only a matrix for linear attention variants such as Mamba and DeltaNet. It is inherently challenging to compress hundreds of thousands of vectors into a matrix with only thousands in rank. As a consequence, these RNN layers struggle to remember the deep relationships between distant tokens.

We experiment with an alternative class of RNN layers whose hidden states themselves can be neural networks. Specifically, we use two-layer MLPs with $2\times$ more hidden cells and richer nonlinearities than the linear (matrix) hidden states in linear attention variants. Since the neural network hidden states are updated by training even on test sequences, these new layers are called Test-Time Training (TTT) layers [43].

We start from a pre-trained Diffusion Transformer (CogVideo-X 5B [19]) that could only generate 3-second short clips at 16 fps (or 6 seconds at 8 fps). Then, we add TTT layers initialized from scratch and fine-tune this model to generate one-minute videos from text storyboards. We limit the self-attention layers to 3-second segments so their cost stays manageable. With only preliminary systems optimization, our training run takes the equivalent of 50 hours on 256 H100s.

We curate a text-to-video dataset based on ≈ 7 hours of *Tom and Jerry* cartoons with human-annotated storyboards. We intentionally limit our scope to this specific domain for fast research iteration. As a proof-of-concept, our dataset emphasizes complex, multi-scene, and long-range stories with dynamic motion, where progress is still needed; it has less emphasis on visual and physical realism, where remarkable progress has already been made. We believe that improvements in long-context capabilities for this specific domain will transfer to general-purpose video generation.

Compared to strong baselines such as Mamba 2 [8], Gated DeltaNet [53], and sliding-window attention layers, TTT layers generate much more coherent videos that tell complex stories with dynamic motion, leading by 34 Elo points in a human evaluation of 100 videos per method. For context, GPT-4o scores 29 Elo points over GPT-4 Turbo in LMSys Chatbot Arena [6].

Sample videos, code and annotations are available at: https://test-time-training.github.io/video-dit

2. Test-Time Training Layers

Following standard practice [44, 54], each video is preprocessed into a sequence of T tokens, where T is determined by its duration and resolution. This section reviews Test-Time Training (TTT) layers for general sequence modeling, using some of the exposition in Section 2 of [43]. We first discuss how to process general input sequences in a causal manner (chronological order). Section 3 discusses how to use RNN layers in a non-causal backbone by invoking them in opposite directions.

2.1. TTT as Updating a Hidden State

All RNN layers compress historical context in a hidden state of fixed size. This compression has two consequences. On one hand, mapping an input token x_t to output token z_t is efficient, because both the update rule and output rule take constant time per token. On the other hand, an RNN layer's ability to remember long context is limited by the amount of information its hidden state can store. The goal of [43] is to design RNN layers with expressive hidden states that can compress massive context. As an inspiration, they observe that self-supervised learning can compress a massive training set into the weights of a machine learning model.

The key idea in [43] is to use self-supervised learning to compress the historical context $x_1, \ldots, x_t$ into a hidden state W_t, by making the context an unlabeled dataset and the hidden state the weights of a machine learning model f. The update rule, illustrated in Figure 2, is a step of gradient descent on some self-supervised loss ℓ:

$$W_t = W_{t-1} - \eta \nabla \ell(W_{t-1}; x_t), \tag{1}$$

with learning rate η. Intuitively, the output token is just the prediction on x_t, made by f with the updated weights W_t:

$$z_t = f(x_t; W_t). \tag{2}$$

One choice of ℓ is reconstructing x_t itself. To make the learning problem nontrivial, one can first process x_t into a corrupted input $\tilde{x}_t$ (see Subsection 2.2), then optimize:

$$\ell(W; x_t) = \|f(\tilde{x}_t; W) - x_t\|^2. \tag{3}$$

Similar to denoising autoencoders [46], f needs to discover the correlations between dimensions of x_t in order to reconstruct it from partial information $\tilde{x}_t$.

As with other RNN layers and self-attention, this algorithm that maps an input sequence $x_1, \ldots, x_T$ to output sequence $z_1, \ldots, z_T$ can be programmed into the forward pass of a sequence modeling layer. Even at test time, the layer still trains a different sequence of weights $W_1, \ldots, W_T$ for every input sequence. Therefore, it is called *Test-Time Training (TTT) layer*.

Conceptually, calling backward on $\nabla \ell$ means taking gradients of gradients – a well-explored technique in meta-learning. TTT layers have the same interface as RNN layers and self-attention, therefore can be replaced in any larger network architecture. [43] refers to training the larger network as the *outer loop*, and training W within each TTT layer as the *inner loop*.

2.2. Learning a Self-Supervised Task for TTT

Arguably, the most important part of TTT is the self-supervised task specified by ℓ. Instead of handcrafting a self-supervised task from human priors, [43] takes a more end-to-end approach, learning it as part of the outer loop. Starting from the naive reconstruction task in Equation 3, they use a low-rank projection $\tilde{x}_t = \theta_K x_t$, where θ_K is a matrix that is learnable in the outer loop.

Moreover, perhaps not all the information in x_t is worth remembering, so the reconstruction label can also be a low-rank projection $\theta_V x_t$ instead of x_t. In summary, the self-supervised loss in [43] is:

$$\ell(W; x_t) = \|f(\theta_K x_t; W) - \theta_V x_t\|^2. \tag{4}$$

Lastly, since $\theta_K x_t$ has fewer dimensions than x_t, [43] can no longer use the output rule in Equation 2. So they make another projection $\theta_Q x_t$, and change the output rule to:

$$z_t = f(\theta_Q x_t; W_t). \tag{5}$$

Note that in the inner loop, only W is optimized, therefore written as an argument of ℓ; the θs are "hyper-parameters" of this inner-loop loss function. $\theta_K, \theta_V, \theta_Q$ are optimized in the outer loop, analogous to the Query, Key, and Value parameters of self-attention.

2.3. TTT-MLP Instantiation

Following [43], we instantiate the inner-loop model f as a wrapper around $f_{\texttt{MLP}}$: a two-layer MLP similar to those in Transformers. Specifically, the hidden dimension is $4\times$ the input dimension, followed by a GELU activation [16]. For better stability during TTT, f always contains a Layer Norm and residual connection. That is,

$$f(x) = x + \texttt{LN}(f_{\texttt{MLP}}(x)).$$

A TTT layer with this f is called TTT-MLP, which is the default instantiation throughout this paper. In Section 4 we also instantiate TTT-Linear (the f above wrapping around a linear model) as a baseline.

3. Approach

At a high level, our approach simply adds TTT layers to a pre-trained Diffusion Transformer and fine-tunes it on long videos with text annotations. At a practical level, making this approach work involves many design choices.

3.1. Architecture

Pre-trained Diffusion Transformer. Our approach of adding TTT layers then fine-tuning can, in principle, work with any backbone architecture. We choose Diffusion Transformers [32] for our initial demonstration because it is the most popular architecture for video generation. Since the cost of pre-training a Diffusion Transformer on videos is prohibitive, we start from a pre-trained checkpoint called CogVideo-X 5B [19].

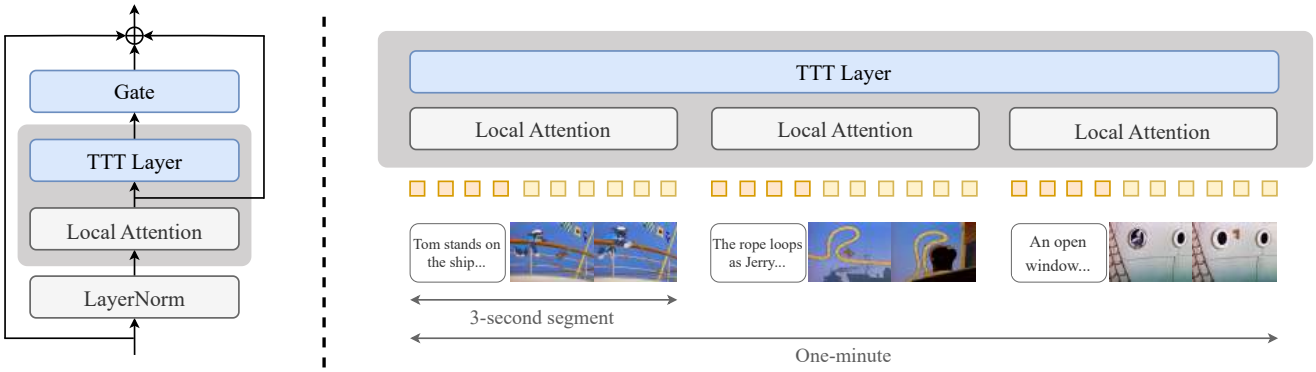


Figure 3. Overview of our approach. **Left**: Our modified architecture adds a TTT layer with a learnable gate after each attention layer. See Subsection 3.1. **Right**: Our overall pipeline creates input sequences composed of 3-second segments. This structure enables us to apply self-attention layers locally over segments and TTT layers globally over the entire sequence. See Subsection 3.2.

Gating. Given an input sequence $X = (x_1, \ldots, x_T)$ where each token $x_t \in \mathbb{R}^d$, a TTT layer produces an output sequence $Z = (z_1, \ldots, z_T) = \texttt{TTT}(X)$. Each $z_t \in \mathbb{R}^d$ follows the recurrence described by Equations 1, 4 and 5 in Section 2. Naively inserting TTT layers into a pre-trained network would dramatically worsen its predictions at the beginning of fine-tuning, when the TTT layers are randomly initialized. To avoid this degradation, we gate TTT with a learned vector $\alpha \in \mathbb{R}^d$ following standard practice [1]:

$$\texttt{gate}(\texttt{TTT}, X; \alpha) = \tanh(\alpha) \otimes \texttt{TTT}(X) + X, \tag{6}$$

where $\tanh(\alpha) \in (-1, 1)^d$ is multiplied element-wise with each z_t in $Z = \texttt{TTT}(X)$. We initialize all values in α to 0.1, so the values in $\tanh(\alpha)$ are close to 0 (≈ 0.1) at the beginning of fine-tuning. This initialization of α allows TTT to still contribute to $\texttt{gate}(\texttt{TTT}, X; \alpha)$ without significantly overwriting X.

Bi-direction. Diffusion models, including CogVideo-X, are non-causal, meaning that an output token z_t can condition on all of $x_1, \ldots, x_T$ instead of only the past tokens $x_1, \ldots, x_t$. To use TTT layers in a non-causal manner, we apply a standard trick called bi-direction [30]. Given an operator $\texttt{rev}(X) = (x_T, \ldots, x_1)$ that reverses $X = (x_1, \ldots, x_T)$ in time, we define

$$\texttt{TTT}'(X) = \texttt{rev}(\texttt{TTT}(\texttt{rev}(X))). \tag{7}$$

Since rev is applied twice, $\texttt{TTT}'(X)$ is still in chronological order. But the TTT layer inside it now scans through X in reverse-chronological order.

Modified architecture. Standard Transformers, including CogVideo-X, contain interleaving sequence modeling blocks and MLP blocks. Specifically, a standard sequence modeling block takes an input sequence X and produces

$$X' = \texttt{self\_attn}(\texttt{LN}(X)) \tag{8}$$
$$Y = X' + X, \tag{9}$$

where LN is Layer Norm[1] and $X' + X$ forms a residual connection. We only modify the sequence modeling blocks, leaving everything else in the architecture unchanged. Each modified block, illustrated in the left panel of Figure 3, continues from the X' in Equation 8 and produces

$$Z = \texttt{gate}(\texttt{TTT}, X'; \alpha), \tag{10}$$
$$Z' = \texttt{gate}(\texttt{TTT}', Z; \beta), \tag{11}$$
$$Y = Z' + X. \tag{12}$$

Note that $\texttt{TTT}'$ only makes another call to TTT, so they share the same underlying parameters $\theta_K, \theta_V, \theta_Q$. But for gating, Equation 10 and 11 use different parameters α and β.

3.2. Overall Pipeline

In this subsection, we discuss how to create the input sequence of tokens to our architecture and how each sequence is processed in segments. Except for the first two text formats in the upcoming discussion, everything applies to both fine-tuning and inference. Our pipeline is illustrated in the right panel of Figure 3.

Scenes and segments. We structure our videos to contain multiple scenes,[2] and each scene contains one or more 3-second *segments*. We use a 3-second segment as the atomic unit of text-to-video pairing for three reasons:

- The maximum length of generation for the original pre-trained CogVideo-X is 3 seconds.
- The length of most scenes in the *Tom and Jerry* episodes is at least 3 seconds.
- Building a dataset with multiple stages (Subsection 3.3) is most convenient given 3-second segments.

Formats of text prompts. At inference time, a user can write the text prompt for a long video in any of the three

[1] Diffusion Transformers such as CogVideo-X use adaptive LN [32].

[2] A scene is loosely defined as "a part of a film in which the action happens in one place or is of one particular type." (Oxford Dictionary)

formats listed below in the order of increasing detail. See Figure 8 in Appendix for examples of each format.

- **Format 1**: A short summary of the plot in 5-8 sentences. Some of the examples are shown in Figure 1.
- **Format 2**: A more detailed plot in roughly 20 sentences, with each sentence roughly corresponding to a 3-second segment. Sentences can be labeled as belonging to certain scenes or groups of scenes, but these labels will be treated only as suggestions.
- **Format 3**: A storyboard. Each 3-second segment is described by a paragraph of 3-5 sentences, containing details such as background colors and camera movements. Groups of one or more paragraphs are strictly enforced as belonging to certain scenes with the keywords `<scene start>` and `<scene end>`.

The actual input to our text tokenizer is always in Format 3 during both fine-tuning and inference. Conversion between the formats is performed by Claude 3.7 Sonnet in the order of $1 \to 2 \to 3$.[3] For fine-tuning, our human annotations are already in Format 3, as discussed in Subsection 3.3.

From text to sequences. After the original CogVideo-X tokenizes the input text for each video, it concatenates the text tokens with noisy video tokens to form the input sequence to the Transformer. To generate a long video, we apply the same procedure independently for each 3-second segment. Specifically, given a storyboard in Format 3 with n paragraphs, we first produce n *sequence segments*, each containing text tokens extracted from the corresponding paragraph followed by video tokens. Then we concatenate all n sequence segments together to form the input sequence, which now has interleaved text and video tokens.

Local attention, global TTT. CogVideo-X uses self-attention layers to process the entire input sequence globally for each video of maximum length 3 seconds, but global attention becomes inefficient for long videos. To avoid increasing the context length of self-attention layers, we make them local to each 3-second segment, attending to each of the n sequence segments independently.[4] The TTT layers process the entire input sequence globally because they are efficient in long context.

3.3. Fine-Tuning Recipe and Dataset

Multi-stage context extension. Following standard practice for LLMs [51], we extend the context length of our modified architecture to one minute in five stages. First, we fine-tune the entire pre-trained model on 3-second segments of *Tom and Jerry* to adapt it to this domain. New parameters (specifically those in TTT layers and gates) are assigned a higher learning rate during this stage. Over the next four stages, we fine-tune on videos of 9, 18, 30, and eventually 63 seconds. To avoid forgetting too much of the world knowledge from pre-training, we only fine-tune the TTT layers, gates, and self-attention layers, using a lower learning rate during these four stages. See Appendix A for the detailed recipe.

Super-resolution on original videos. We start with 81 episodes of *Tom and Jerry* released between 1940 and 1948. Each episode is about 5 minutes, adding up to about 7 hours for all episodes. The original videos vary in resolution, which is uniformly poor by modern standards. We run a video super-resolution model [49] on the original videos, producing visually enhanced videos with shared resolution of 720×480 for our dataset.

Multi-stage dataset. Following the structure discussed in Subsection 3.2, we first have human annotators break down each episode into scenes, then extract 3-second segments from each scene. Next we have human annotators write a detailed paragraph for each 3-second segment.[5] Stage 1 fine-tunes directly on these segments. To create data for the last four stages, we concatenate contiguous 3-second segments into videos of 9, 18, 30 and 63 seconds together with their text annotations. Scene boundaries are marked by the same keywords in Subsection 3.2. As a result, annotations for all training videos are in Format 3.

3.4. Parallelization for Non-Causal Sequences

The update rule discussed in Section 2 cannot be naively parallelized across tokens in a sequence, since computing W_t requires $\nabla \ell(W_{t-1}; x_t)$, which in turn requires W_{t-1}. To enable parallelization, we update W on b tokens at a time, which [43] calls an inner-loop mini-batch. Throughout this paper, we set $b = 64$.

Concretely, for mini-batch $i = 1, \ldots, T/b$ (assuming T is an integer multiple of b),

$$W_{ib} = W_{(i-1)b} - \frac{\eta}{b} \sum_{t=(i-1)b+1}^{ib} \nabla \ell \left(W_{(i-1)b}; x_t \right). \quad (13)$$

Because the sequence is non-causal, we then use W_{ib} to produce the output tokens for all timesteps in mini-batch i:

$$z_t = f(W_{ib}; x_t), \qquad \text{for } t = (i-1)b + 1, \ldots, ib. \quad (14)$$

Note that $W_{(i-1)b+1}, \ldots, W_{ib-1}$ are no longer needed.

After this modification, f can process an (inner-loop) mini-batch of tokens in parallel, similar to how a regular MLP processes an (outer-loop) mini-batch of training data. As a side benefit, we observe that averaging gradients across tokens reduces variance and stabilizes each update to W.

[3] We observe that converting from Format 1 directly to Format 3 results in worse ability to follow the style of the human annotations in Format 3 in the fine-tuning dataset.

[4] As an artifact of our pre-processing step, the sequence segments actually have an overlap of 1 latent frame (1350 tokens).

[5] Each paragraph includes 1–2 sentences describing the background, 1–2 sentences describing the characters, and 2 sentences describing actions and camera movements. On average, each paragraph contains 98 words, which corresponds to 132 tokens.

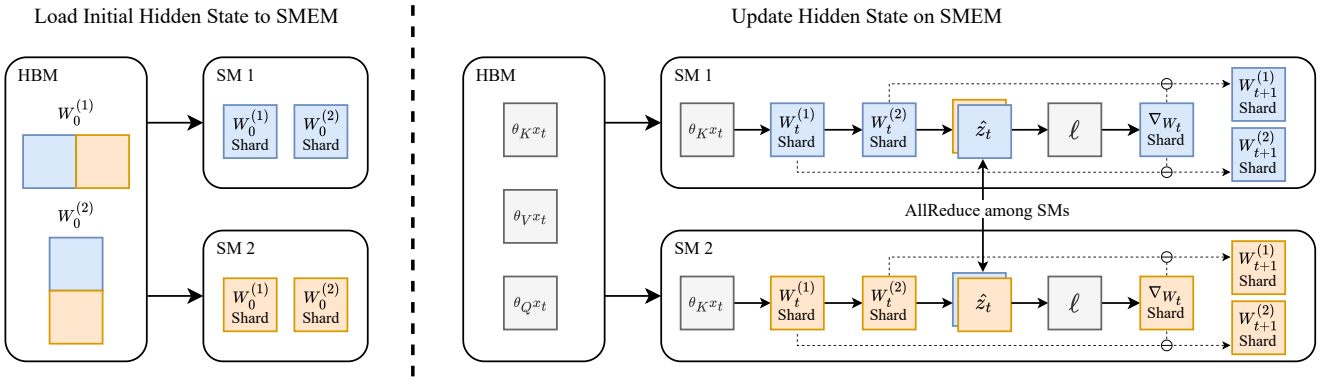


Figure 4. On-chip Tensor Parallel, discussed in Subsection 3.5. **Left:** To reduce the memory required on each SM for TTT-MLP, we shard the hidden state $W^{(1)}$ and $W^{(2)}$ across SMs, transferring them between HBM and SMEM only during initial loading and final output. **Right:** We update the hidden state entirely on-chip and use the DSMEM feature on the NVIDIA Hopper GPU architecture to `AllReduce` intermediate activations among SMs.

3.5. On-Chip Tensor Parallel

Implementing TTT-MLP efficiently for GPUs requires special designs to take advantage of their memory hierarchy. A chip on a GPU is called a Streaming Multiprocessor (SM), analogous to a core on a CPU. All SMs on a GPU share a relatively slow but large global memory called HBM, then each SM has a fast but small on-chip memory called SMEM. Frequent data transfers between the SMEMs and HBM on a GPU can significantly hurt overall efficiency.

Efficient implementations of Mamba and self-attention layers (Flash Attention [9]) use kernel fusion to minimize this kind of transfer. The high-level idea of these implementations is to load inputs and initial states into each SMEM, perform computations entirely on-chip, and write only the final outputs back to HBM. However, the hidden state for TTT-MLP, namely the weights $W^{(1)}$ and $W^{(2)}$ of the two-layer MLP f, is too large to be stored in the SMEM of a single SM (when combined with inputs and activations).

To reduce the memory required on each SM, we use Tensor Parallelism [39] to shard $W^{(1)}$ and $W^{(2)}$ across SMs, as shown in Figure 4. Similar to how large MLP layers can be sharded and trained across the HBMs of multiple GPUs, we apply the same idea now across the SMEMs of multiple SMs, treating each SM as the analogy of a GPU. We use the DSMEM feature on the NVIDIA Hopper GPU architecture to implement `AllReduce` among SMs. More details of our kernel are discussed in Appendix B.

Our implementation significantly improves efficiency, since hidden states and activations are now read from and written to HBMs only during initial loading and final output. As a general principle, if a model architecture f can be sharded with standard Tensor Parallelism across GPUs, then the same sharding strategy can be applied across SMs when f is used as the hidden state.

4. Evaluation

We perform human evaluation on a multi-axis benchmark for TTT-MLP and five baselines, all with linear complexity: local attention, TTT-Linear, Mamba 2, Gated DeltaNet, and sliding window attention layers.

4.1. Baselines

Except for local attention, all baselines are added to the same pre-trained CogVideo-X 5B using the approach in Subection 3.1; their modified architectures all have 7.2B parameters. All baselines use the same fine-tuning recipe in Subsection 3.3 and Appendix A. Next we discuss the baselines in detail.

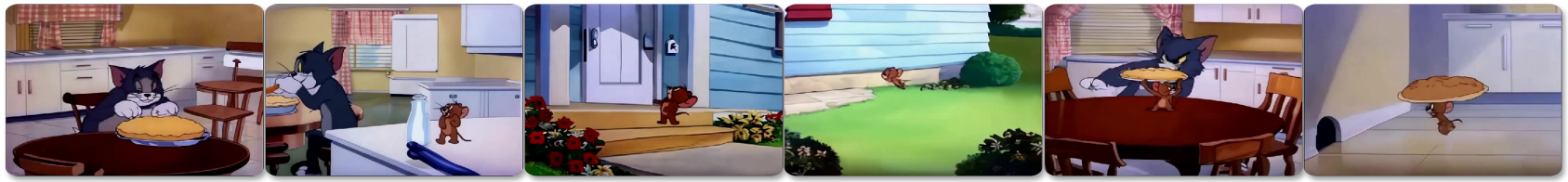
- **Local attention**: No modification to the original architecture, which performs self-attention on each 3-second segment independently.
- **TTT-Linear** [43]: A TTT layer that instantiates $f(x) = x + \texttt{LN}(f_{\texttt{Linear}}(x))$, where $f_{\texttt{Linear}}$ is a linear model.
- **Mamba 2** [8]: A modern RNN layer with a matrix hidden state, which is $\approx 4\times$ larger than the hidden state in TTT-Linear but $\approx 2\times$ smaller than that in TTT-MLP.
- **Gated DeltaNet** [53]: An extension of DeltaNet [52] and Mamba 2 with an improved update rule.
- **Sliding-window attention** [3]: Self-attention with a fixed window of 8192 tokens (about 1.5 seconds of video).

4.2. Evaluation Axes and Protocol

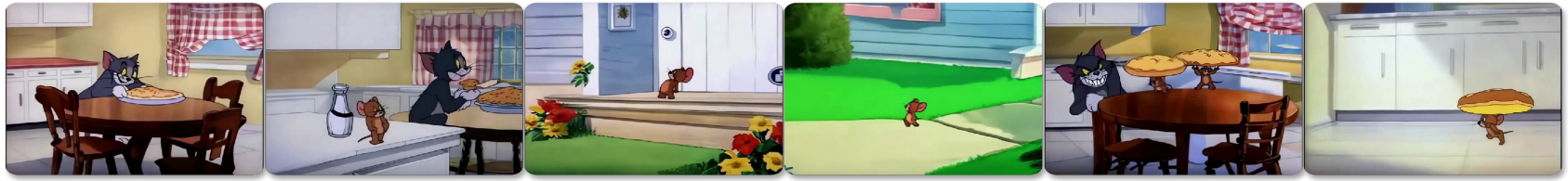
From the six evaluation axes in MovieGen [44], we adopt the four relevant to our domain for human evaluation.[6]

[6]Out of the six axes in MovieGen, we omit "realness" which does not apply to cartoons. We also omit "motion completeness" which "measures whether the output video contains enough motion", because all videos in our domain have highly dynamic motion. We adapt "frame consistency" to "temporal consistency" to also include consistency across scenes.

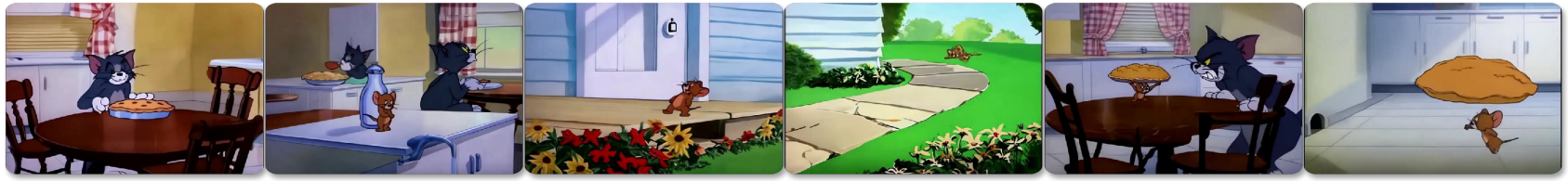
Tom is happily eating an apple pie at the kitchen table. Jerry looks longingly wishing he had some. Jerry goes outside the front door of the house and rings the doorbell. While Tom comes to open the door, Jerry runs around the back to the kitchen. Jerry steals Tom's apple pie. Jerry runs to his mouse hole carrying the pie, while Tom is chasing him. Just as Tom is about to catch Jerry, he makes it through the mouse hole and Tom slams into the wall.



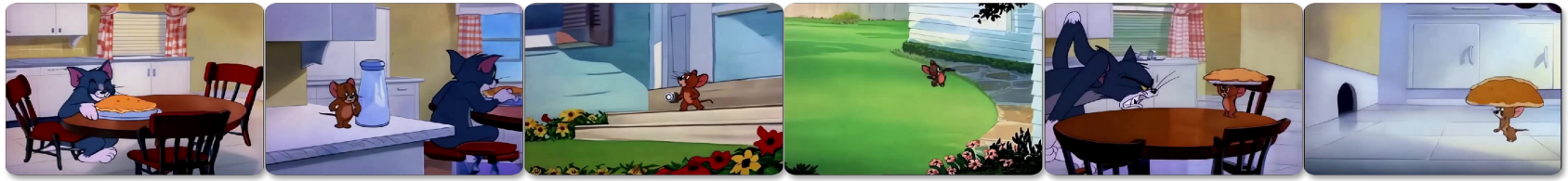
TTT-MLP preserves temporal consistency over scene changes and across angles, producing smooth, high-quality actions.



Sliding-window attention alters the kitchen environment, changes the house color, and duplicates Jerry stealing the pie.



Gated DeltaNet lacks temporal consistency across different angles of Tom but maintains the kitchen environment in later frames.



Mamba 2 distorts Tom's appearance as he growls and chases Jerry but maintains a similar kitchen environment throughout the video.

Figure 5. Video frames comparing TTT-MLP against Gated DeltaNet and sliding-window attention, the leading baselines in our human evaluation. TTT-MLP demonstrates better scene consistency by preserving details across transitions and better motion naturalness by accurately depicting complex actions.

	Text following	Motion naturalness	Aesthetics	Temporal consistency	Average
Mamba 2	985	976	963	988	978
Gated DeltaNet	983	984	993	1004	991
Sliding window	**1016**	1000	1006	975	999
TTT-MLP	1014	**1039**	**1037**	**1042**	**1033**

Table 1. Human evaluation results for one-minute videos. TTT-MLP improves over the second best method by 34 Elo points on average. Axes with the most improvements are scene consistency (+38) and motion smoothness (+39). For context, GPT-4 scores 46 Elo points over GPT-3.5 Turbo, and GPT-4o scores 29 over GPT-4 Turbo in Chatbot Arena [6].

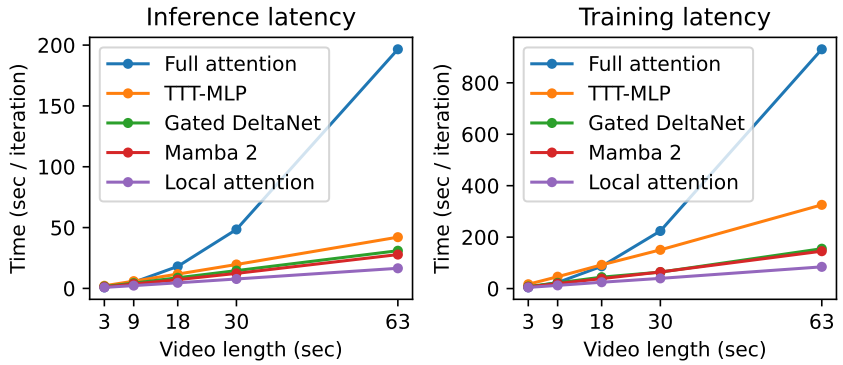


Figure 6. For 63-second videos, inference with full attention (over 300k tokens) would have taken 11× longer than local attention, and training 12× longer, as discussed in Section 1. TTT-MLP takes 2.5× and 3.8× respectively – significantly more efficient than full attention, but still less efficient than, for example, Gated DeltaNet, which takes 1.8× longer than local attention in both inference and training.

- **Text following**: "aligment with the provided prompt."
- **Motion naturalness**: "natural limb movements, facial expressions, and adherence to physical laws. Motion that appears unnatural or uncanny will be penalized."
- **Aesthetics**: "interesting and compelling content, lighting, color, and camera effects."
- **Temporal consistency**: both inside and across scenes.

The quoted descriptions are from MovieGen [44].

Our evaluation is based on pairwise preferences in blind comparisons, because directly rating long videos or ranking many of them at once is challenging. Specifically, an evaluator is given a random axis from the four above and a random pair of videos sharing the same plot, then asked to indicate the better video for that axis. To collect the pool of videos, we first sample 100 plots using Claude 3.7 Sonnet (in Format 1 → 2 → 3 as discussed in Subsection 3.2), then generate one video per method per plot. The methods generating the videos are always unknown to the evaluators.

Our evaluators were recruited on `prolific.com` with the filters: living in the U.S., English as a first language, aged 18 to 35 years, with at least 100 previous submissions and an approval rate of at least 98%. The demographics of our evaluators, disclosed on the website, are as follows.

- **Gender**: 50.78% male, 47.66% female, 1.56% other.
- **Ethnicity**: 57.03% White, 23.44% Black, 10.94% Mixed, 5.47% Asian, and 3.12% other.

Based on this information, we believe that our evaluators constitute a representative sample of the U.S. population.

4.3. Results

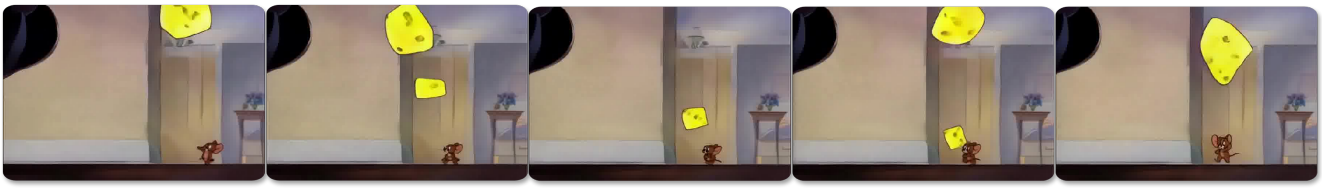
We aggregate the pairwise preferences using the Elo system in LMSys Chatbot Arena [6]. The Elo scores are shown in Table 1.

TTT-MLP improves over the second-best method by 34 Elo points on average. For context, GPT-4 scores 46 Elo points over GPT-3.5 Turbo (1163 vs. 1117), and GPT-4o scores 29 over GPT-4 Turbo (1285 vs. 1256) in LMSys Chatbot Arena [6], so our improvement by 34 is practically meaningful.[7] Figure 5 compares frames of sample videos generated by TTT-MLP and the baselines. The videos illustrated in Figure 5 can be accessed on the project website: `https://test-time-training.github.io/video-dit`

18-second elimination round. Note that local attention and TTT-Linear do not appear in Table 1. To avoid the much higher cost of evaluating longer videos on every method, we first conducted an elimination round using 18-second videos following the same procedure discussed in Subsection 4.2. This round eliminated local attention, which performed worst, and also TTT-Linear, which performed worse than TTT-MLP. Results of the elimination round are shown in Table 3 in the Appendix.

[7] `https://lmarena.ai/`, accessed on March 20, 2025. The models considered are GPT-4o-2024-05-13, GPT-4-Turbo-2024-04-09, GPT-4-0613, and GPT-3.5-Turbo-0613.

Temporal consistency: The boxes morph between 3-second segments of the same scene.



Motion naturalness: The cheese hovers in mid-air rather than falling naturally to the ground.

Aesthetics: The lighting in the kitchen becomes dramatically brighter as Tom turns around.

Figure 7. Artifacts in videos generated by TTT-MLP. **Temporal consistency**: Objects sometimes morph at the boundaries of 3-second segments, potentially because the diffusion model samples from different modes across the segments. **Motion naturalness**: Objects sometimes float unnaturally because gravitational effects are not properly modeled. **Aesthetics**: Lighting changes do not consistently align with actions unless explicitly prompted. Complex camera movements, such as parallax, are sometimes depicted inaccurately.

4.4. Limitations

Short context. For the 18-second elimination round discussed above, Gated DeltaNet performs the best on average, leading Mamba 2 by 27 Elo points and TTT-MLP by 28 (see Table 3 in the Appendix). For 18-second videos, the context length is roughly 100k tokens. This evaluation shows the scenario where RNN layers with linear (matrix) hidden states, such as Gated DeltaNet and Mamba 2, are still the most effective. Moreover, evaluation results for both 18 and 63-second videos indicate that Gated DeltaNet improves meaningfully on Mamba 2.

Wall-clock time. Even after applying our improvements in Subsection 3.4 and 3.5, the efficiency of TTT-MLP is still worse than Gated DeltaNet and Mamba 2. This limitation is highlighted in Figure 6, where inference and training with TTT-MLP are $1.4\times$ and $2.1\times$ slower than with Gated DeltaNet, for example. Section 6 discusses two potential improvements of our TTT-MLP kernel for better efficiency. Note that training efficiency is not a significant concern in our application because the RNN layers are integrated after pre-training, which constitutes most of the overall training budget. Training efficiency of the RNN layers is only relevant during fine-tuning, which is a small part of the budget to begin with. In contrast, inference efficiency is much more meaningful.

Video artifacts. The generated 63-second videos demonstrate clear potential as a proof of concept, but still contain notable artifacts, especially in motion naturalness and aesthetics. Figure 7 illustrates examples of artifacts corresponding to three of our evaluation axes. We observe that videos with these kinds of artifacts are not particular to TTT-MLP, but common among all methods. The artifacts might have been a consequence of the limited capability of the pre-trained CogVideo-X 5B model. For example, videos (link) generated by the original CogVideo-X also seem to have limited motion naturalness and aesthetics.

5. Related Work

Modern RNN layers, especially linear attention variants [23, 37], such as Mamba [8, 12] and DeltaNet [35, 52], have demonstrated impressive performance in natural language tasks. Inspired by their success and ideas from Fast Weight Programmers [7, 21, 24, 36], [43] proposes scalable and practical ways to make the hidden states large and nonlinear, therefore more expressive. Recent work [2] develops even larger and more nonlinear hidden states, and updates them with more sophisticated optimization techniques. The related work section in [43] contains a detailed discussion of inspirations for TTT layers. [48] gives a good overview of recent developments in RNN layers.

Long video modeling. Some early work [40] generates long videos by training GAN [11, 22] to predict the next frame based on the current frame and the motion vector. Generation quality has improved significantly due to recent progress in auto-regression (AR) and diffusion-based approaches [13, 25, 44, 54]. TATS [10] proposes the sliding window attention on the Transformer to generate videos longer than the training length. Phenaki [45] works in a similar auto-regressive way, but each frame is generated by MaskGIT [4]. Pre-trained diffusion models can be extended to generate longer videos by using cascade [15, 50, 55], streaming [17], and adding transitions [5].

Story synthesis methods such as [20, 26, 28, 29, 31, 33] generate sequences of images or videos corresponding to individual sentences in a text story. For example, Craft [14] generates videos of complex scenes through retrieval, and StoryDiffusion [56] uses diffusion to improve the smoothness of transitions between frames. While related to text-to-video generation, story synthesis methods usually need additional components in their pipeline to maintain coherence across scenes, which are not processed end-to-end.

6. Future Work

We outline several promising directions for future work.

Faster implementation. Our current TTT-MLP kernel is bottlenecked by register spills and suboptimal ordering of asynchronous instructions. Efficiency could probably be further improved by minimizing register pressure and developing a more compiler-aware implementation of asynchronous operations.

Better integration. Using bi-direction and learned gates is only one possible strategy for integrating TTT layers into a pre-trained model. Better strategies should further improve generation quality and accelerate fine-tuning. Other video generation backbones, such as autoregressive models, might require different integration strategies.

Longer videos with larger hidden states. Our approach can potentially be extended to generate much longer videos with linear complexity. The key to achieving that goal, we believe, is to instantiate the hidden states as much larger neural networks than our two-layer MLP. For example, f itself can be a Transformer.

Acknowledgements. We thank Hyperbolic Labs for compute support, Yuntian Deng for help with running experiments, and Aaryan Singhal, Arjun Vikram, and Ben Spector for help with systems questions. Yue Zhao would like to thank Philipp Krähenbühl for discussion and feedback. Yu Sun would like to thank his PhD advisor Alyosha Efros for the insightful advice of looking at the pixels when working on machine learning.

Note on authorship. Gashon Hussein and Youjin Song joined the team after an initial version of this project was submitted to CVPR, and have made major contributions to the final version. Because CVPR does not allow us to add authors after submission, their names could not appear on OpenReview and the conference webpage. However, we all agree that the official author list should include their names, as presented in our released PDFs. This project would not be possible without their work.

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *NeurIPS*, 2022. 4

[2] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024. 9

[3] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. 6

[4] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *CVPR*, 2022. 10

[5] Xinyuan Chen, Yaohui Wang, Lingjun Zhang, Shaobin Zhuang, Xin Ma, Jiashuo Yu, Yali Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. Seine: Short-to-long video diffusion model for generative transition and prediction. In *ICLR*, 2023. 10

[6] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael Jordan, Joseph E Gonzalez, et al. Chatbot arena: An open platform for evaluating llms by human preference. In *ICML*, 2024. 2, 8

[7] Kevin Clark, Kelvin Guu, Ming-Wei Chang, Panupong Pasupat, Geoffrey Hinton, and Mohammad Norouzi. Meta-learning fast weight language models. *EMNLP*, 2022. 9

[8] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. In *ICML*, 2024. 2, 6, 9

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*, 2022. 6

[10] Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic vqgan and time-sensitive transformer. In *ECCV*, 2022. 10

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 2020. 10

[12] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *COLM*, 2024. 2, 9

[13] Agrim Gupta, Lijun Yu, Kihyuk Sohn, Xiuye Gu, Meera Hahn, Fei-Fei Li, Irfan Essa, Lu Jiang, and José Lezama.

Photorealistic video generation with diffusion models. In *ECCV*, 2024. 10

[14] Tanmay Gupta, Dustin Schwenk, Ali Farhadi, Derek Hoiem, and Aniruddha Kembhavi. Imagine this! scripts to compositions to videos. In *ECCV*, 2018. 10

[15] Yingqing He, Tianyu Yang, Yong Zhang, Ying Shan, and Qifeng Chen. Latent video diffusion models for high-fidelity long video generation. *arXiv preprint arXiv:2211.13221*, 2022. 10

[16] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 3

[17] Roberto Henschel, Levon Khachatryan, Daniil Hayrapetyan, Hayk Poghosyan, Vahram Tadevosyan, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Streamingt2v: Consistent, dynamic, and extendable long video generation from text. *arXiv preprint arXiv:2403.14773*, 2024. 10

[18] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. 1

[19] Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. Cogvideo: Large-scale pretraining for text-to-video generation via transformers. In *ICLR*, 2023. 2, 3

[20] Ting-Hao Huang, Francis Ferraro, Nasrin Mostafazadeh, Ishan Misra, Aishwarya Agrawal, Jacob Devlin, Ross Girshick, Xiaodong He, Pushmeet Kohli, Dhruv Batra, et al. Visual storytelling. In *NAACL*, 2016. 10

[21] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *NeurIPS*, 2021. 9

[22] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020. 10

[23] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, 2020. 2, 9

[24] Louis Kirsch and Jürgen Schmidhuber. Meta learning backpropagation and improving it. *NeurIPS*, 34:14122–14134, 2021. 9

[25] Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, Kathrina Wu, Qin Lin, Junkun Yuan, Yanxin Long, Aladdin Wang, Andong Wang, Changlin Li, Duojun Huang, Fang Yang, Hao Tan, Hongmei Wang, Jacob Song, Jiawang Bai, Jianbing Wu, Jinbao Xue, Joey Wang, Kai Wang, Mengyang Liu, Pengyu Li, Shuai Li, Weiyan Wang, Wenqing Yu, Xinchi Deng, Yang Li, Yi Chen, Yutao Cui, Yuanbo Peng, Zhentao Yu, Zhiyu He, Zhiyong Xu, Zixiang Zhou, Zunnan Xu, Yangyu Tao, Qinglin Lu, Songtao Liu, Dax Zhou, Hongfa Wang, Yong Yang, Di Wang, Yuhong Liu, Jie Jiang, and Caesar Zhong. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv 2412.03603*, 2025. 10

[26] Yitong Li, Zhe Gan, Yelong Shen, Jingjing Liu, Yu Cheng, Yuexin Wu, Lawrence Carin, David Carlson, and Jianfeng Gao. Storygan: A sequential conditional gan for story visualization. In *CVPR*, 2019. 10

[27] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common diffusion noise schedules and sample steps are flawed. In *WACV*, 2024. 1

[28] Chang Liu, Haoning Wu, Yujie Zhong, Xiaoyun Zhang, Yanfeng Wang, and Weidi Xie. Intelligent grimm-open-ended visual storytelling via latent diffusion models. In *CVPR*, 2024. 10

[29] Adyasha Maharana, Darryl Hannan, and Mohit Bansal. Storydall-e: Adapting pretrained text-to-image transformers for story continuation. In *ECCV*, 2022. 10

[30] Shentong Mo and Yapeng Tian. Scaling diffusion mamba with bidirectional ssms for efficient image and video generation. *arXiv preprint arXiv:2405.15881*, 2024. 4

[31] Xichen Pan, Pengda Qin, Yuhong Li, Hui Xue, and Wenhu Chen. Synthesizing coherent story with auto-regressive latent diffusion models. In *WACV*, 2024. 10

[32] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *CVPR*, 2023. 3, 4

[33] Tanzila Rahman, Hsin-Ying Lee, Jian Ren, Sergey Tulyakov, Shweta Mahajan, and Leonid Sigal. Make-a-story: Visual memory conditioned consistent story generation. In *CVPR*, 2023. 10

[34] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2022. 1

[35] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *ICML*, 2021. 2, 9

[36] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. 9

[37] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. 2, 9

[38] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024. 1

[39] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. 6

[40] Ivan Skorokhodov, Sergey Tulyakov, and Mohamed Elhoseiny. Stylegan-v: A continuous video generator with the price, image quality and perks of stylegan2. In *CVPR*, 2022. 10

[41] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021. 1

[42] Benjamin F Spector, Simran Arora, Aaryan Singhal, Daniel Y Fu, and Christopher Ré. Thunderkittens: Simple, fast, and adorable ai kernels. In *ICLR*, 2025. 1

[43] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024. 2, 3, 5, 6, 9, 1

[44] The Movie Gen team. Movie gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024. 2, 6, 8, 10

[45] Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan. Phenaki: Variable length video generation from open domain textual description. In *ICLR*, 2023. 10

[46] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008. 3

[47] Hongjie Wang, Chih-Yao Ma, Yen-Cheng Liu, Ji Hou, Tao Xu, Jialiang Wang, Felix Juefei-Xu, Yaqiao Luo, Peizhao Zhang, Tingbo Hou, Peter Vajda, Niraj K. Jha, and Xiaoliang Dai. Lingen: Towards high-resolution minute-length text-to-video generation with linear computational complexity, 2024. 2

[48] Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *arXiv preprint arXiv:2501.12352*, 2025. 9

[49] Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. Real-esrgan: Training real-world blind super-resolution with pure synthetic data. In *ICCVW*, 2021. 5

[50] Yaohui Wang, Xinyuan Chen, Xin Ma, Shangchen Zhou, Ziqi Huang, Yi Wang, Ceyuan Yang, Yinan He, Jiashuo Yu, Peiqing Yang, et al. Lavie: High-quality video generation with cascaded latent diffusion models. *IJCV*, 2024. 10

[51] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. In *NAACL*, 2024. 5

[52] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *NeurIPS*, 2024. 6, 9

[53] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *ICLR*, 2025. 2, 6

[54] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. In *ICLR*, 2025. 2, 10, 1

[55] Shengming Yin, Chenfei Wu, Huan Yang, Jianfeng Wang, Xiaodong Wang, Minheng Ni, Zhengyuan Yang, Linjie Li, Shuguang Liu, Fan Yang, et al. Nuwa-xl: Diffusion over diffusion for extremely long video generation. *arXiv preprint arXiv:2303.12346*, 2023. 10

[56] Yupeng Zhou, Daquan Zhou, Ming-Ming Cheng, Jiashi Feng, and Qibin Hou. Storydiffusion: Consistent self-attention for long-range image and video generation. In *NeurIPS*, 2024. 10

Video len.	Ctx. len	Trainable parameters	Learning rate	Schedule	Steps
3 sec	18048	TTT / Pre-trained Params	1×10^{-4} / 1×10^{-5}	Cosine / Constant	5000
9 sec	51456	TTT + Local Attn (QKVO)	1×10^{-5}	Constant	5000
18 sec	99894	TTT + Local Attn (QKVO)	1×10^{-5}	Constant	1000
30 sec	168320	TTT + Local Attn (QKVO)	1×10^{-5}	Constant	500
63 sec	341550	TTT + Local Attn (QKVO)	1×10^{-5}	Constant	250

Table 2. Hyper-parameters for multi-stage fine-tuning. First, the entire pre-trained model is fine-tuned on 3-second segments of *Tom and Jerry*, with higher learning rates assigned to the newly introduced TTT layers and gates. Then, only TTT layers, gates, and self-attention parameters are fine-tuned at reduced learning rates.

	Text following	Motion naturalness	Aesthetics	Temporal consistency	Average
Local Attention	965	972	969	944	962
TTT-Linear	1003	995	1007	1001	1001
Mamba 2	**1023**	987	1008	1004	1005
Gated DeltaNet	1020	**1039**	**1044**	1026	**1032**
SWA	995	1004	993	980	993
TTT-MLP	994	1002	1002	1019	1004

Table 3. Human evaluation results for 18-second videos, discussed in Subsection 4.3 and 4.4.

A. Experiment Details

Diffusion schedule. Following CogVideoX [54], we fine-tune our model using v-prediction [34], which includes a diffusion noise schedule with 1000 steps and Zero-SNR [27] enforced at the final step.

Training configurations. We use the following hyper-parameters for all stages of training:

- **Optimizer:** AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$
- **Learning Rate:** Linear warmup over 2% of training steps
- **Batch Size:** 64
- **Gradient Clipping:** 0.1
- **Weight Decay:** 10^{-4} applied to all params except biases and normalization layers
- **VAE Scale Factor**: 1.0
- **Dropout:** Zero-out text prompt with probability 0.1
- **Precision:** Mixed Precision with PyTorch FSDP2

TTT configurations. A key hyperparameter for TTT layers is the inner-loop learning rate η, which we set $\eta = 1.0$ for TTT-Linear and $\eta = 0.1$ for TTT-MLP.

Sampling schedule. We follow the DDIM sampler [41] with 50 steps, applying dynamic classifier-free guidance (CFG) [18] that increases CFG magnitude from 1 to 4 and utilizing negative prompts to further enhance video quality.

B. On-Chip Tensor Parallel Details

We use ThunderKittens [42] to implement the TTT-MLP kernel, described in Subsection 3.5.

Hidden state sharding. We follow the standard strategy for Tensor Parallel, sharding the first layer column-wise and the second layer row-wise. As the GeLU non-linearity is elementwise, the forward pass of the TTT-layer requires a single reduction for computing the inner loss used to update the hidden state.

Further latency optimizations. We incorporate several techniques from FlashAttention-3 [38] to further reduce I/O latency on NVIDIA Hopper GPUs. In particular, we implement a multi-stage pipelining scheme that asynchronously prefetches future mini-batches from HBM, overlapping data transfers with computation on the current mini-batch. This approach, known as *producer-consumer asynchrony*, involves dedicating specialized warpgroups to either data loading (producer) or computation (consumer).

Gradient checkpointing. We integrate gradient checkpointing along the sequence dimension [43] directly into our fused kernel. To reduce I/O-induced stalls and CUDA thread workloads, we use the Tensor Memory Accelerator (TMA) to perform asynchronous memory stores.

Format 1

Tom is happily eating an apple pie at the kitchen table. Jerry looks longingly wishing he had some. Jerry goes outside the front door of the house and rings the doorbell. While Tom comes to open the door, Jerry runs around the back to the kitchen. Jerry steals Tom's apple pie. Jerry runs to his mouse hole carrying the pie, while Tom is chasing him. Just as Tom is about to catch Jerry, Jerry makes it through the mouse hole and Tom slams into the wall.

Format 2

Segment 1-2: Tom walks into the kitchen carrying an apple pie. He sits at the table and begins eating.

Segment 3-5: The viewpoint shifts behind the countertop, revealing Jerry hiding behind a salt shaker. Jerry steps out, watches Tom eating the pie, and eagerly rubs his tummy. He then darts off-screen to the right.

Segment 6-8: Outside the house, Jerry approaches the front door, jumps to press the doorbell, and quickly runs away.

The story continues...

Format 3

<start_scene>The kitchen has soft yellow walls, white cabinets, and a window with red-and-white checkered curtains letting in gentle sunlight. In the middle, there's a round wooden table with matching chairs, sitting on a clean white-tiled floor. Tom, the blue-gray cat, walks in from the left holding a warm, golden-brown pie on a shiny silver tray. He moves calmly across the room toward the table, carefully places the pie down, pulls out a chair, and sits comfortably. The camera smoothly follows Tom from left to right, clearly showing each of his movements.

The kitchen has soft yellow walls, white cabinets, and a window with red-and-white checkered curtains letting in gentle sunlight. In the middle, there's a round wooden table with matching chairs, sitting on a clean white-tiled floor. Tom, the blue-gray cat, sits comfortably at the table with the golden-brown pie resting on its shiny silver tray directly in front of him. He carefully uses his paw to pick up a slice from the tray, lifts it toward his mouth, and takes a large bite. The camera slowly moves closer, clearly showing Tom enjoying his pie as crumbs lightly fall onto the table.<end_scene>

<start_scene>The kitchen has soft yellow walls, white cabinets, and a window with red-and-white checkered curtains letting in gentle sunlight. The cabinets have white countertops, on which a tall glass salt shaker is sitting. In the background, Tom, the blue-gray cat, sits at the round wooden table, eating the golden-brown pie. Jerry, the brown mouse, stands on the white countertop, hidden behind the salt shaker. Jerry glances around briefly, then steps out from behind the salt shaker. The camera captures Jerry as he emerges from behind the salt shaker and stands on the countertop.

The kitchen has soft yellow walls, white cabinets, and a window with red-and-white checkered curtains letting in gentle sunlight. The cabinets have white countertops, on which a tall glass salt shaker is sitting. In the background, Tom, the blue-gray cat, sits at the round wooden table, eating the golden-brown pie. Jerry, the brown mouse, stands on top of the countertop next to the salt shaker. Jerry rubs his stomach with his paws and looks at Tom, the blue-gray cat. The camera remains in position slightly to the side of Jerry, capturing his hungry expression.

The kitchen has soft yellow walls, white cabinets, and a window with red-and-white checkered curtains letting in gentle sunlight. The cabinets have white countertops, on which a tall glass salt shaker is sitting. In the background, Tom, the blue-gray cat, sits at the round wooden table, eating the golden-brown pie. Jerry, the brown mouse, stands on top of the countertop next to the salt shaker. Jerry gives his belly a final rub, then turns to the left and quickly begins running along the countertop toward the right side of the scene. The camera captures Jerry as he disappears off-screen.<end_scene>

<start_scene>The front of the house has light blue walls, a white wooden front door, a small round white doorbell button beside it, and a small porch with steps leading down to a neat green lawn. Bright flowers in red and yellow line the walkway, and sunlight warmly fills the area. Jerry, the brown mouse, calmly walks up onto the porch from the right side, moving toward the front door and doorbell at the center of the scene. The camera smoothly tracks Jerry's steps, capturing clearly as he crosses the porch and comes to a gentle stop near the steps, glancing cautiously upward at the doorbell.

The story continues...

Figure 8. Illustration of the three prompt formats discussed in Subsection 3.2: (1) a short summary of the plot, (2) sentence-level descriptions of the segments, and (3) a detailed storyboard.