## A. Appendix - Implementation

### A.1. Concurrent Inference with Dynamic Loading

The proposed MOVD tackles the challenge of limited device memory in text-to-video generation, which restricts the on-device inference of large diffusion-based models, by introducing Concurrent Inference with Dynamic Loading (CI-DL), which partitions models and executes them in a concurrent and dynamic manner.

**Concurrent Inference.** The model components of Open-Sora [69], *i.e.*, STDiT [69] and T5 [39], easily exceed the available memory of many mobile devices, *e.g.*, 3.3 GB RAM of iPhone 15 Pro, as shown in Fig. 3. Given that the Transformer architecture [57], which is the backbone for both T5 [39] and STDiT [69], we partition these models into smaller blocks (segments) and load them into memory accordingly for model inference.

To execute video model inference using the model partitioning, each block must be loaded onto memory sequentially before execution, increasing the overall latency of video generation by incurring block loading time. Fig. 10-(a) shows the sequential block load and inference cycle of STDiT [69], where GPU remains idle intermittently, waiting for each block to be loaded into memory, and only begins execution after the loading is complete. This sequential loading and execution process significantly increases the overall latency of model inference.

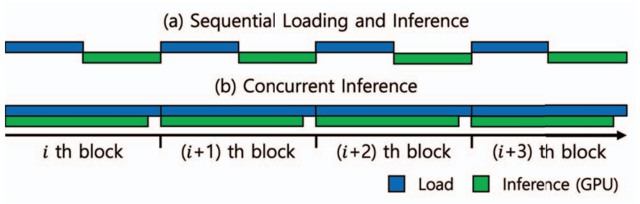


Figure 10. The block loading and inference cycles for (a) sequential loading and inference, and (b) concurrent inference.

To minimize the increase in model inference latency caused by sequential block loading and execution, we propose Concurrent Inference, which leverages both the CPU and GPU for parallel block loading and execution; CPU loads the $(i + 1)$th block, while GPU concurrently executes the $i$th block. Initially, the first and second blocks are loaded into memory concurrently, with the first block completing its loading first. Subsequently, the inference of the first block and the loading of the second block occur simultaneously. This process continues such that the inference of the $i$th block and the loading of the $(i + 1)$th block overlap, ensuring continuous parallelism until the final block. Fig. 10-(b) depicts the load and inference cycle of STDiT with Concurrent Inference, which shows that GPU is active without almost no idle time by performing block loading and inference in parallel.

Given the number of model blocks $b$, block loading latency $l$, and inference latency of block $e$, the latency reduction $r$ achieved through Concurrent Inference is given by:

$$r = b \cdot \min(l, e) - \alpha \tag{8}$$

where $\alpha$ is the overhead caused by the block loading.

Given the large number of denoising steps performed by STDiT, which is partitioned into multiple blocks for execution, similar to T5 [39], the number of blocks $b$ is expected to be large, leading to a significant reduction in latency. It is expected to accelerate the overall model inference effectively regardless of the device's memory capacity. When the available memory is limited, then $b$ increases, while with larger memory, both $l$ and $e$ increase. In either case, it can result in an almost constant latency reduction $r$ in Eq. (8).

**Dynamic Loading.** To further enhance the model inference latency, we propose Dynamic Loading, which is applied in conjunction with Concurrent Inference. It maintains a subset of model blocks in memory without unloading them, with the selection for the subset of blocks to be retained in memory dynamically determined based on the device's available memory at runtime.

The available memory of the device can vary at runtime based on the system status and configurations of applications running on the mobile device. By retaining a subset of model blocks in memory, the overhead of reloading these blocks during subsequent steps of Concurrent Inference can be eliminated, enabling reductions in model inference latency. To achieve this, we measure the device's run-time memory capacity and the memory required for inferring a single model block during the initial step of Concurrent Inference. Next, the memory allocated for retaining certain model blocks is dynamically determined as the difference between the available memory and the memory required for inferring a model block. Then, a series of model blocks that fit within this allocated memory is loaded in a retained state.

Fig. 11 depicts Dynamic Loading; the first four model blocks are loaded in a retrained state. Unlike other blocks, *e.g.*, the 5th block, these blocks are not unloaded to memory after the initial step, reducing block loading overhead.

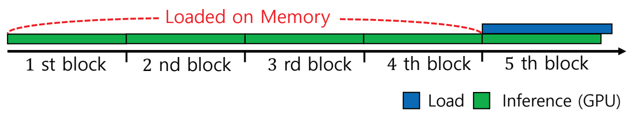


Figure 11. The block loading and inference cycle for Dynamic Loading applied with Concurrent Inference.

Applying Dynamic Loading, the latency reduction $r$ in Eq. (8) for Concurrent Inference is updated as:

$$r = b \cdot \min(l, e) + d \cdot \max(0, l - e) - \alpha \cdot (1 - d/b) \tag{9}$$

where $d$ is the number of blocks maintained in memory. As the number of model blocks retained in memory increases

| Category | Method | Temporal Quality↑ | | | | | Frame-Wise Quality↑ | | Quality Score↑ |
|---|---|---|---|---|---|---|---|---|---|
| | | Subject Consistency | Background Consistency | Temporal Flickering | Motion Smoothness | Dynamic Degree | Aesthetic Quality | Imaging Quality | |
| Animal | Open-Sora | 0.97 | 0.98 | 0.99 | 0.99 | 0.15 | 0.51 | 0.56 | 0.79 |
| | Open-Sora (MOVD) | 0.95 | 0.97 | 0.99 | 0.99 | 0.28 | 0.48 | 0.55 | 0.79 |
| Architecture | Open-Sora | 0.99 | 0.98 | 0.99 | 0.99 | 0.05 | 0.53 | 0.60 | 0.78 |
| | Open-Sora (MOVD) | 0.98 | 0.98 | 0.99 | 0.99 | 0.12 | 0.49 | 0.56 | 0.78 |
| Food | Open-Sora | 0.97 | 0.97 | 0.99 | 0.99 | 0.26 | 0.52 | 0.60 | 0.81 |
| | Open-Sora (MOVD) | 0.95 | 0.97 | 0.99 | 0.99 | 0.38 | 0.48 | 0.53 | 0.81 |
| Human | Open-Sora | 0.96 | 0.97 | 0.99 | 0.99 | 0.38 | 0.48 | 0.57 | 0.81 |
| | Open-Sora (MOVD) | 0.96 | 0.96 | 0.99 | 0.99 | 0.43 | 0.48 | 0.55 | 0.82 |
| Lifestyle | Open-Sora | 0.97 | 0.97 | 0.99 | 0.99 | 0.23 | 0.45 | 0.56 | 0.79 |
| | Open-Sora (MOVD) | 0.96 | 0.97 | 0.99 | 0.99 | 0.25 | 0.45 | 0.53 | 0.78 |
| Plant | Open-Sora | 0.98 | 0.98 | 0.99 | 0.99 | 0.15 | 0.50 | 0.58 | 0.79 |
| | Open-Sora (MOVD) | 0.97 | 0.98 | 0.99 | 0.99 | 0.16 | 0.46 | 0.55 | 0.78 |
| Scenery | Open-Sora | 0.98 | 0.98 | 0.99 | 0.99 | 0.10 | 0.50 | 0.50 | 0.77 |
| | Open-Sora (MOVD) | 0.97 | 0.98 | 0.99 | 0.99 | 0.17 | 0.48 | 0.47 | 0.77 |
| Vehicles | Open-Sora | 0.97 | 0.97 | 0.99 | 0.99 | 0.37 | 0.48 | 0.54 | 0.81 |
| | Open-Sora (MOVD) | 0.94 | 0.96 | 0.98 | 0.99 | 0.44 | 0.47 | 0.49 | 0.80 |

Table 6. The VBench [23] evaluation by category: 68 frames, 256×256 resolution.

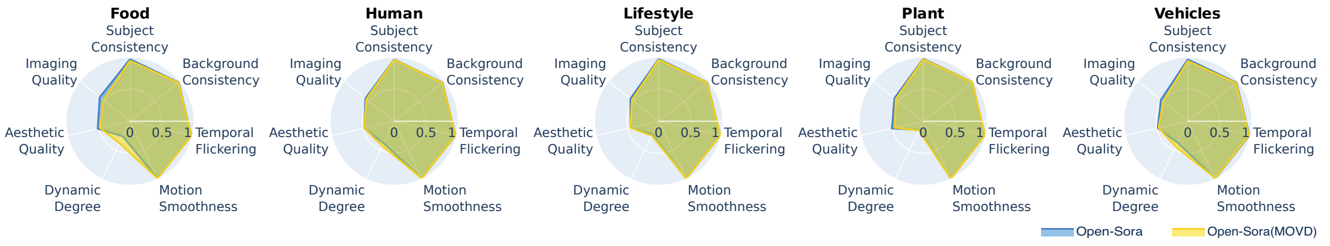


Figure 12. A visual comparison of the VBench [23] performance between Mobile-Oriented Video Diffusion and Open-Sora [69].

(*i.e.*, a larger $d$), the overhead of loading unloading blocks is further minimized, fully accelerating the overall model inference under the device's run-time memory constraints. Dynamically keeping some model blocks in memory with a retrained state is particularly advantageous for STDiT [69] that is iteratively executed for denoising steps, which entails loading and reusing the same blocks for each step.

## B. Appendix - Additional Experiments

### B.1. Video Generation Performance

In Sec. 7.1, we evaluate the quality of videos generated on an iPhone 15 Pro [1] and compare them to videos produced by Open-Sora running on NVIDIA A6000 GPUs using VBench [23] as the evaluation benchmark. The evaluation is conducted on 68-frame videos at 256×256 resolution, using text prompts provided by VBench [23], consisting of 100 examples each across eight categories: animals, architecture, food, humans, lifestyle, plants, scenery, and vehicles. Tab. 6 and Fig. 12 present the comprehensive comparison of generated videos on all categories.

### B.2. Concurrent Inference with Dynamic Loading

Fig. 13-(b) illustrates the model block loading and inference cycles of STDiT [69] when applying the proposed Concurrent Inference (Sec. A.1), whereas Fig. 13-(a) depicts the case without its application. It can be observed that, with Concurrent Inference, the GPU executes each model block for inference without almost no idle time in parallel with the model block loading, indicated by the overlap between the red (loading) and black (inference) boxes in Fig. 13-(b), resulting in a block inference latency reduction from 29s to 23s. In contrast, without Concurrent Inference, each model block inference is executed only after the corresponding block is fully loaded to memory, indicated by the lack of overlap between the red (loading) and black (inference) boxes in Fig. 13-(a). Consequently, the total latency of the full denoising process using multiple executions of STDiT [69] is reduced by approximately 25%, decreasing from 1,000 to 750 seconds for 30 denoising steps. Given that STDiT is executed multiple times to perform numerous denoising steps, it significantly accelerates the STDiT's overall inference. In addition, when applied with Dynamic Loading (Sec. A.1), it achieves an additional average speed improvement of 17 seconds as reloading is not required for certain model blocks that are retained in memory, provided in Eq. (9).

Fig. 14 shows the case of T5 [39]. Unlike STDiT, which exhibits similar latencies for both block loading and inference execution, T5 exhibits a much longer block loading la-

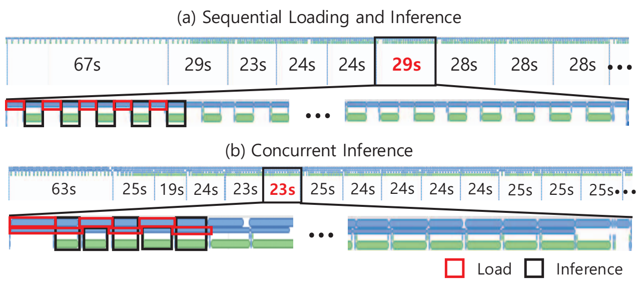


Figure 13. The block loading and inference cycles of STDiT [69] without (a) and with (b) Concurrent Inference. The red box represents the loading cycle, while the black box indicates the model block inference on the iPhone 15 Pro's GPU.

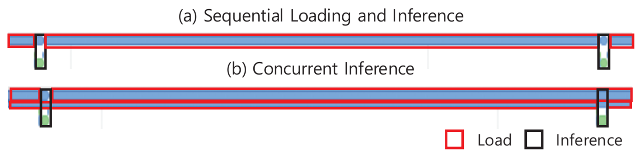


Figure 14. The block loading and inference cycles of T5 [39] without (a) and with (b) Concurrent Inference.

tency relative to inference latency. Consequently, the overlap between the model loading and inference is minimal, as shown by the small region of overlap between the red (loading) and black (inference) boxes. As a result, the latency improvement is expected to be less substantial, as in Eq. (8). Nevertheless, the inference latency is reduced from 164 to an average of 137 seconds, achieving a reduction of 16%. This result implies the effectiveness of concurrent loading and inference, even in cases when there is an imbalance between the latencies of model block loading and inference.

## B.3. Impact of LPL and TDTM: Ablation Studies on Larger Text-to-Video Models

The primary constraint for applying Linear Proportional Leap (LPL) is that the diffusion model must be a flow-matching–based model trained with a rectified flow target [33]. Since Pyramidal Flow [24], one of the state-of-the-art open-source video generative models, satisfies this condition, we conduct experiments on it.

In a manner similar to Open-Sora [69] with Rectified Flow [33], Pyramidal Flow consistently demonstrates a high degree of straightness in its generation process following the initial generation unit (as illustrated in Fig. 17). As demonstrated in Tab. 7, the application of LPL during generation reduces the overall generation time by almost 38%, decreasing from 264.60 seconds to 165.15 seconds. Notably, Pyramidal Flow employs a cascading model procedure in which subtle variances are treated as noise to be subsequently denoised. Since these variances become negligible in the final output, Linear Proportional Leap is particularly well-suited to this architecture. Our experiments involving multiple prompts (as shown in Fig. 15) confirm that removing nearly half of the total denoising steps does not compromise performance while preserving both visual quality and temporal consistency. Furthermore, we observe that additional steps at later units can also be omitted due to the high straightness of Pyramidal Flow. Future work will further investigate this phenomenon by exploring the compatibility of flow-based models with the current LPL algorithm, developing the algorithm to be more compatible with other types of models.

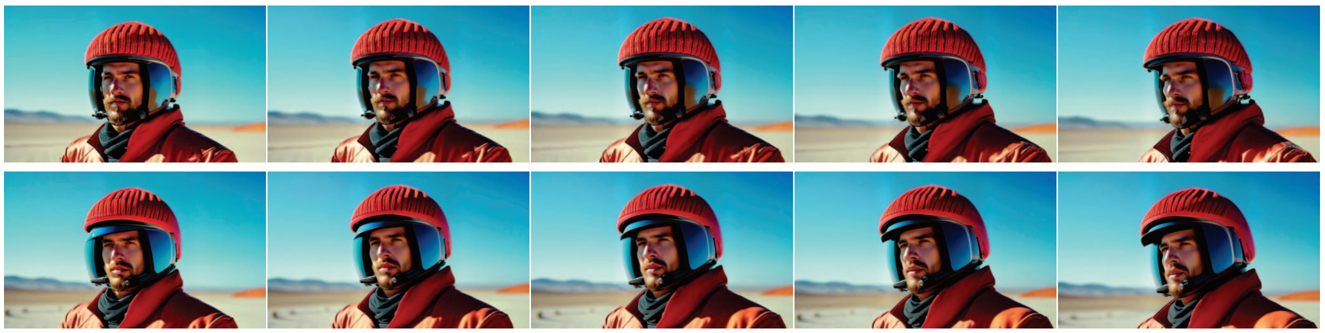
A key requirement for applying TDTM is that the model's attention mechanism must operate over a multiple temporal dimension. HunyuanVideo [26] is an open-source text-to-video generative model that meets the necessary conditions for both TDTM and LPL. Similar to Open-Sora [69], experimental results demonstrating the effectiveness of TDTM and LPL on HunyuanVideo are presented in Tab. 7 and Fig. 16. Applying TDTM to only half of the DIT steps results in a 1.48× speed-up. Likewise, using LPL to reduce the total number of steps by half yields a 1.83× speed-up. Most notably, when both TDTM and LPL are combined—reducing the steps from 50 to 25 via LPL and applying TDTM to 12 of those steps—a maximum speed-up of **2.56**× can be achieved.

| Models | Time (s) | Resolution | DiT Steps | Speedup |
|---|---|---|---|---|
| Pyramidal Flow - 2B | 262.03 | 1280 × 768 | 270 | 1.00× |
| Pyramidal Flow - 2B - LPL (2) | 165.15 | 1280 × 768 | 159 | 1.59× |
| Pyramidal Flow - 2B | 48.88 | 640 × 384 | 270 | 1.00× |
| Pyramidal Flow - 2B - LPL (2) | 30.54 | 640 × 384 | 159 | 1.60× |
| HunyuanVideo - 13.2B | 2434.64 | 544 × 960 | 50 | 1.00× |
| HunyuanVideo - 13.2B - TDTM | 1644.94 | 544 × 960 | 50 | 1.48× |
| HunyuanVideo - 13.2B - LPL (2) | 1325.08 | 544 × 960 | 50 | 1.83× |
| HunyuanVideo - 13.2B - TDTM + LPL (2) | 952.34 | 544 × 960 | 50 | **2.56**× |

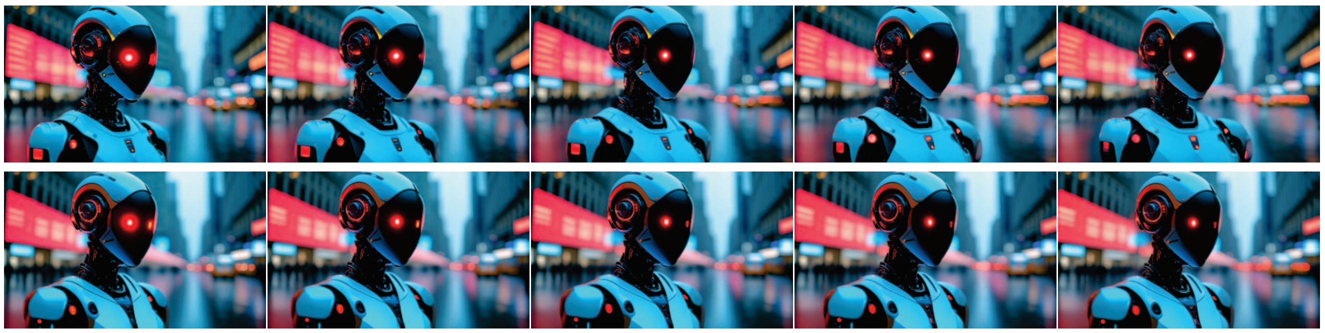
Table 7. Ablation study conducted on Pyramidal Flow [24] based on the miniFLUX [27] architecture and HunyuanVideo [26]. 'DiT steps' represent the total number of DiT forward computations required for the denoising process. Speedups are measured relative to the corresponding model, with execution time computed as the average of three independent runs. Notably, LPL and TDTM does not degrade video quality, even when using nearly half of its total steps.

## B.4. Comparison and Combination with Accelerated Attention Methodology

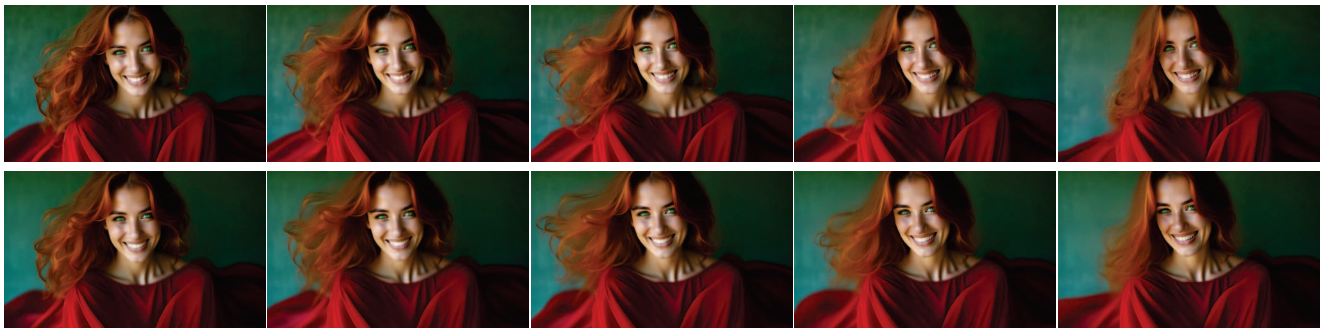
In this section, we compare and combine our methods (LPL, TDTM) with a fine-tuned Sliding Tile Attention (STA). Experiments on HunyuanVideo [26] employed the current STA implementation [64], and the results are reported in Tab. 8. As the STA implementation currently supports only H100 GPUs, all experiments in this section were conducted on an H100 at a resolution of 1280×768. Individually, LPL and TDTM each reduce denoising wall-clock time relative to the naive baseline, although their improvements are smaller than those achieved by STA. When combined (LPL+TDTM), however, they yield shorter wall-clock times



Prompt: *"A movie trailer featuring the adventures of a 30-year-old spaceman wearing a red wool-knitted motorcycle helmet, blue sky, salt desert, cinematic style, shot on 35mm film, vivid colors."*



Prompt: *"A cybernetic humanoid scans the streets with red eyes as holographic screens flicker around its head, displaying futuristic data."*



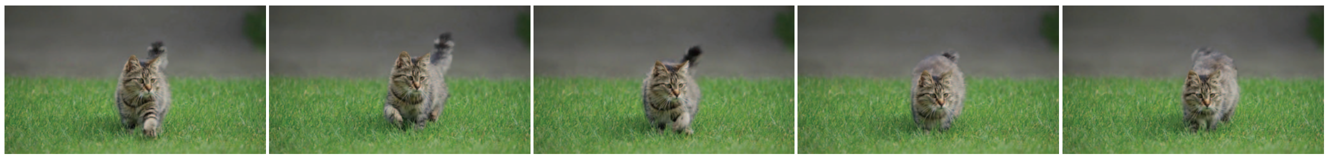
Prompt: *"A green-eyed woman with auburn hair and a mysterious smile, wearing a flowing red dress."*

Figure 15. Videos generated using naive Pyramidal Flow [24] (top) and Pyramidal Flow with the proposed Linear Proportional Leap (bottom) under identical prompts. Linear Proportional Leap (LPL) reduces the total number of denoising steps by nearly half (*i.e.*, from 270 to 159), while maintaining video generation results that are highly comparable to those achieved using the full set of denoising steps and preserving most details.

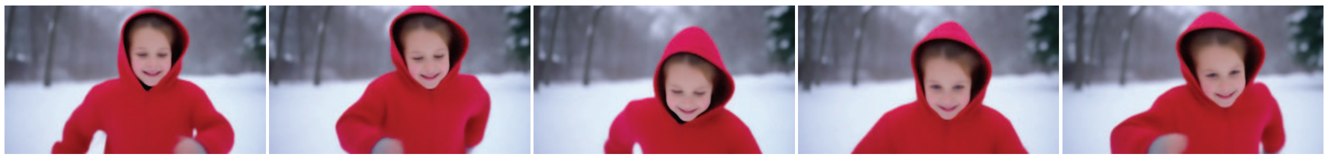
| Combination/Time | Naive | TDTM (25/50) | LPL (25/50) | Sliding Tile Attention | TDTM (12/25) + LPL(25/50) | Sliding Tile Attention + TDTM(12/25) + LPL(25/50) |
|---|---|---|---|---|---|---|
| Pure Denoising Loop Time | 1653 ± 0.00s | 1073 ± 1.15s | 863 ± 0.00s | 699 ± 0.00s | 585 ±0.58s | 352 ± 0.00s |
| End-To-End Time | 1699.04 ± 0.33s | 1122.53 ± 3.32s | 902.30 ± 0.35s | 738.21 ± 0.09s | 623.83 ± 0.42s | 389.72 ± 0.05s |

Table 8. Evaluation of computation time with Sliding Tile Attention (STA) on HunyuanVideo.

than STA (699 s vs. 585 s). Specifically, LPL reduces the number of denoising steps, TDTM decreases the number of tokens processed by attention, and STA accelerates attention computation. Because these optimizations target different stages of the pipeline without conflict, their joint application results in clear synergy, producing a substantial reduction in denoising wall-clock time (Naive: 1653 s; Combined: 352 s).



Prompt: *“A cat walks on the grass, realistic style.”*



Prompt: *“Close-up, A little girl wearing a red hoodie in winter running toward camera.”*

Figure 16. Videos generated using HunyuanVideo [26] with Temporal Dimension Token Merging (TDTM) in the upper row and with the proposed Linear Proportional Leap (LPL) in the lower row. LPL reduces the number of denoising steps by nearly half (from 50 to 25), and TDTM halves the input size of the attention mechanism, while preserving most of the visual details and overall video quality.

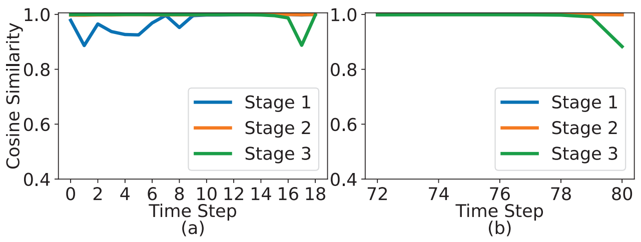


Figure 17. Cosine similarities between adjacent drifts estimated by Pyramidal Flow [24] at each stage. (a) and (b) show cosine similarities computed at the first and last units, respectively. The values approach 1.0 at the last unit across all stages. Computations follow the experimental settings in [24], using 20 denoising steps in (a) and 10 in (b).

## C. Appendix - Discussions and Limitations

**Latency Improvement.** Although MOVD enables efficient video generation, the generation latency remains higher compared to utilizing high-end GPUs; it requires several minutes to generate a video, whereas an NVIDIA A6000 GPU takes one minute. This discrepancy is evident due to the substantial disparity in computational resources between them. For instance, the iPhone 15 Pro’s GPU features up to 2.15 TFLOPS with 3.3 GB of available memory, compared to the NVIDIA A6000, which offers up to 309 TFLOPS and 48 GB of memory. Despite this significant resource gap, MOVD achieves exceptional efficiency in video generation. Currently, it utilizes only the iPhone 15 Pro’s GPU. We anticipate that the latency could be significantly enhanced if it can leverage NPU (Neural Processing Unit), *e.g*., the iPhone 15 Pro’s Neural Engine [1], which delivers a peak performance of 35 TOPS. However, the current limitations in Apple’s software and SDK support for state-of-the-art diffusion-based models [69] make the iPhone’s NPU challenging to utilize effectively. We look forward to the development of software support for NPUs and leave the exploration of this for future work. Also, we plan to investigate the potential of NPUs on a variety of mobile devices, such as Android smartphones.

**Model Optimization.** In MOVD, only T5 [39] is quantized to int8, reducing its size from 18 GB to 4.64 GB, while STDiT [69] and VAE [12] are executed with float32 due to their performance susceptibility, which has a significant impact on video quality. Additionally, we do not apply pruning [40] or knowledge distillation [16], as these methods also drop visual fidelity. In particular, we observe that naively shrinking STDiT [69] leads to significant visual loss, caused by iterative denoising steps, where errors propagate and accumulate to the final video. Another practical difficulty in achieving lightweight model optimization is the lack of resources required for model optimization; both retraining and fine-tuning state-of-the-art diffusion-based models typically demand several tens of GPUs, and the available datasets are often limited to effectively apply optimization methods. To tackle these challenges, we propose model acceleration techniques without retraining, compression, or pruning for video generation in this work, *i.e*., Linear Proportional Leap (Sec. 4) and Temporal Dimension Token Merging (Sec. 5).

**Straightness Constraints.** For video generation models that do not exhibit straightness during their denoising procedures, such as CogVideoX [62], which employs DPM-Solver [68], Linear Proportional Leap (LPL) is currently inapplicable. In such cases, as described in previous work [63], an alternative approach involves predicting modifications to the model’s noise schedule by employing additional methodologies, such as reinforcement learning. However, these kind of methods require additional training which may invoke extra costs, and necessitates retraining whenever the target model architecture changes. By contrast, LPL is a plug-and-play algorithm that can be applied in a model-agnostic manner, as long as a sufficient level of straightness is maintained during the denoising process, thereby eliminating the need for additional training. In future work, we plan to investigate the optimal reduction point achievable with this algorithm, and we anticipate that it will evolve into a widely applicable methodology for all Rectified Flow-based models [33].

**Limited Scope of TDTM Application.** Temporal Dimension Token Merging (TDTM) is applicable across all stages of the denoising process. As shown in Tab. 4, the more stages TDTM is applied to, the greater the speedup achieved, with a maximum improvement of up to 1.7×. In contrast, Linear Proportional Leap (LPL) terminates the denoising process early—typically around step 15 of 30—and transitions directly to the final stage. While this reduces the available window for applying TDTM, it also enables a more concentrated use of TDTM in the remaining steps. As a result, when TDTM is applied to half of the steps after LPL (e.g., 12 out of 25), the combined use of TDTM and LPL achieves a maximum speedup of up to 2.56× in HunyanVideo, surpassing the individual contributions of each. This demonstrates the potential for synergistic gains, even under early-stopping scenarios.