PipeDiT: Accelerating Diffusion Transformers in Video Generation with Task Pipelining and Model Decoupling

Sijie Wang, Qiang Wang, Shaohuai Shi*

School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen
25b951105@stu.hit.edu.cn, {qiang.wang, shaohuais}@hit.edu.cn

Abstract

Video generation has been advancing rapidly, and diffusion transformer (DiT) based models have demonstrated remarkable capabilities. However, their practical deployment is often hindered by slow inference speeds and high memory consumption. In this paper, we propose a novel pipelining framework named PipeDiT to accelerate video generation, which is equipped with three main innovations. First, we design a pipelining algorithm (PipeSP) for sequence parallelism (SP) to enable the computation of latent generation and communication among multiple GPUs to be pipelined, thus reducing inference latency. Second, we propose DeDiVAE to decouple the diffusion module and the variational autoencoder (VAE) module into two GPU groups, whose executions can also be pipelined to reduce memory consumption and inference latency. Third, to better utilize the GPU resources in the VAE group, we propose an attention co-processing (Aco) method to further reduce the overall video generation latency. We integrate our PipeDiT into both OpenSoraPlan and HunyuanVideo, two state-of-the-art open-source video generation frameworks, and conduct extensive experiments on two 8-GPU systems. Experimental results show that, under many common resolution and timestep configurations, our PipeDiT achieves 1.06× to 4.02× speedups over OpenSoraPlan and HunyuanVideo.

Introduction

Video generation models (Li et al. 2024a; Cho et al. 2024; Sun et al. 2024a; Blattmann et al. 2023) have progressed rapidly in recent years, enabling the synthesis of continuous videos from textual or visual inputs. Among existing approaches, Diffusion Transformers (DiTs) (Fan et al. 2025; Ma et al. 2024; Yang et al. 2024) have become the dominant paradigm due to their superior visual quality and temporal coherence. DiT models generate videos through an iterative reverse diffusion process (Ho, Jain, and Abbeel 2020; Sohl-Dickstein et al. 2015; Song and Ermon 2019), progressively refining noise into a clean video signal (Fig. 1). However, this inherently sequential denoising procedure limits parallelism and results in high inference latency (Li et al. 2023b; Shih et al. 2023; Chen et al. 2024a).

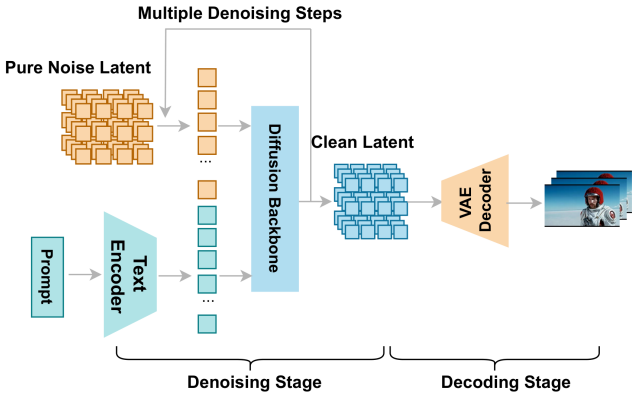

Figure 1: Text-to-video generation starts with encoding the input text and a pure noise latent into a semantic representation, which guides a diffusion model to iteratively refine a latent. The refined latent is then upsampled by a VAE decoder to generate the final video.

Recent work proposes various inference optimizations for both image and video generation. For images, methods such as DistriFusion (Li et al. 2024b) and PipeFusion (Fang et al. 2024b) partition inputs into patches and distribute computation across GPUs to improve throughput and alleviate memory bottlenecks. For video, techniques including Teacache (Liu et al. 2025) and related approaches (Chen et al. 2024c; Selvaraju et al. 2024) exploit temporal feature redundancy to reduce effective timesteps. While effective, these methods may introduce quality degradation.

As a result, most state-of-the-art video generation systems rely instead on system-level parallelization strategies, particularly sequence parallelism (SP) (Li et al. 2023a; Sun et al. 2024b; Zhao et al. 2024), which accelerate inference while maintaining generation quality.

Currently, two main SP paradigms have been proposed. The first is DeepSpeed-Ulysses (referred to as Ulysses) (Jacobs et al. 2023), which parallelizes attention by splitting heads and forming full-sequence $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$. Its communication cost is relatively low, requiring three All-to-All ex-

*Corresponding author.

changes before attention and one after. However, its scalability is bounded by the number of heads, and current implementations do not overlap computation with communication, leaving GPUs partially idle. The second is Ring-Attention (Li et al. 2021), which performs local attention on partial sequences and circulates $\mathbf{K}$ and $\mathbf{V}$ through P2P communication to enable global attention. This supports higher parallelism, but the heavier communication cost often offsets the benefit, making Ulysses preferable when parallelism is not strictly constrained. Recent efforts combine both approaches into Unified Ulysses-Ring SP (USP) (Fang and Zhao 2024; Fang et al. 2024a), expanding parallelism at the expense of additional communication.

While existing system-level optimizations aim to accelerate DiT-based video generation without degrading video quality, two key challenges remain: (1) inter-GPU communication during denoising frequently becomes the bottleneck, and (2) decoding with a VAE easily triggers out-of-memory (OOM) issues, making the decoding phase inefficient.

We propose PipeDiT, an optimized inference framework that reduces end-to-end video generation latency while preserving output quality. First, we develop PipeSP, a pipelined SP algorithm that overlaps computation and communication within Ulysses, effectively hiding communication delays and improving GPU utilization. Second, to mitigate memory explosion caused by colocating the diffusion module and VAE decoder, we introduce DeDiVAE, which decouples diffusion denoising and VAE decoding into two GPU groups. This reduces peak memory consumption and enables pipelined execution between denoising and decoding. Third, to address idle GPUs in the decoupled setup, we propose Aco, an attention co-processing strategy that splits DiT into linear layers and attention kernels, allowing attention computation to be executed cooperatively across both GPU groups for higher utilization.

Our contributions are summarized as follows:

- We analyze the computation–communication behavior of Ulysses and introduce a pipelined variant that overlaps communication with denoising computation.
- We propose DeDiVAE, a module-level pipeline parallelism method that decouples denoising and VAE decoding across GPUs, significantly reducing peak memory usage and improving generation efficiency.
- We further enhance GPU utilization under the decoupled architecture by introducing Aco, a fine-grained decomposition of DiT that distributes attention computation across all GPUs.
- Built upon OpenSoraPlan (PKU-YuanGroup 2025) and HunyuanVideo (Tencent-Hunyuan 2025), we evaluate PipeDiT on two 8-GPU systems. Experiments show substantial improvements in single-timestep runtime and multi-prompt generation latency, demonstrating its effectiveness and scalability.

Preliminaries & Motivations

Diffusion-based video generation comprises two stages as shown in Fig. 1: a Diffusion-based denoising stage that refines a latent representation over multiple timesteps and a variational autoencoder (VAE)-based decoding stage that upsamples the latent into a full-resolution video. The denoising stage is computationally heavy due to attention operations, while the decoding stage is highly memory-intensive due to upsampling to target resolution and frame rate. We conduct a preliminary benchmark using two state-of-the-art video generation frameworks, OpenSoraPlan (PKU-YuanGroup 2025) and HunyuanVideo (Tencent-Hunyuan 2025) as shown in Fig. 2. This indicates that the diffusion stage takes significantly longer than the other two stages, but its memory usage is relatively small. Without offloading enabled, the VAE Decoder peaks at 44GB of memory when decoding the 256×128×33 latent, which is largely because the model parameters occupy a substantial amount of GPU memory. It is evident that during the entire diffusion-based video generation process, the denoising of the latent becomes the time bottleneck, while the decoding of the latent is the memory bottleneck.

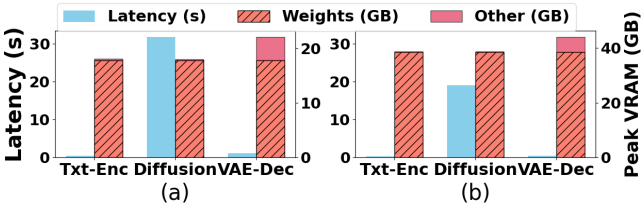

Figure 2: Latency and peak GPU memory usage of each component during inference (using eight GPUs with SP) for a single prompt in (a) OpenSoraPlan (PKU-YuanGroup 2025) model with a resolution of 480×352×65 and 50 timesteps (b) HunyuanVideo (Kong et al. 2024) model with a resolution of 256×128×33 and 50 timesteps.

In current mainstream video generation models, the diffusion backbone and the VAE decoder are typically colocated, which leads to serialized execution of diffusion computation and VAE upsampling. Without employing offloading or other memory-saving techniques, this design results in significant additional and inefficient memory consumption. Therefore, colocating the diffusion model and the VAE decoder is unfavorable for parallel video generation and hinders the generation of higher-resolution videos. The experimental results show that under the single-GPU memory constraint of 48 GB, OpenSoraPlan is unable to generate videos with resolutions larger than 1024×576×97 without offloading. Due to its larger model weights, HunyuanVideo cannot generate videos beyond 256×128×33 in resolution (see the experimental section for details).

Offloading is a commonly used strategy to reduce GPU memory consumption during inference (Abul-Fazl, Dina, and Fairuza 2025; Chen et al. 2024b). This strategy saves GPU memory by dynamically transferring model weights between the CPU and GPU. The primary advantage of this strategy is its implementation simplicity and its effectiveness in enabling higher-resolution video generation with limited GPU memory. Accordingly, offloading is adopted by several large-scale video generation systems—such as HunyuanVideo (Tencent-Hunyuan 2025), Wan (Wan et al.

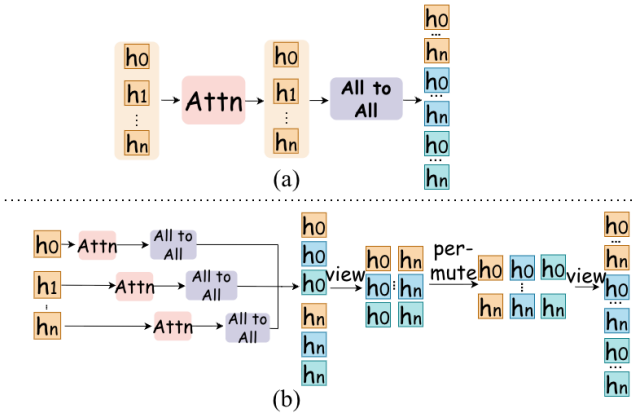

Figure 3: (a) The execution process of Ulysses, where computation and communication are executed sequentially. (b) Our optimized SP (PipeSP) by pipelining communication and computation. The subsequent post-processing resolves the misalignment issue introduced by the pipelining.

2025), and OpenSoraPlan (PKU-YuanGroup 2025). *However, offloading introduces significant CPU-GPU data transfer overhead, which depends on model size and bandwidth.* The offloading overhead may easily dominate and hurt efficiency, while it cannot run the inference without offloading due to the high memory consumption of video generation.

Sequence parallelism (SP) (Li et al. 2023a; Wang et al. 2025; Wu et al. 2024) such as Ulysses accelerates long-sequence processing by splitting computation along the attention-head dimension, allowing different GPUs to process different heads in parallel. As shown in Fig. 3(a), each GPU first computes its local $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, followed by three All-to-All operations that exchange these tensors across GPUs. This produces full-length sequences but only partial attention heads per GPU. After all heads are computed, a final All-to-All redistributes results to obtain full-head hidden states.

However, in the original Ulysses design, this final All-to-All is triggered only after *all* heads finish computing, leaving GPUs idle while waiting for communication and causing noticeable inefficiency.

Methodology

Pipelining Computation and Communication in SP

To address the issue of serial communication and computation in Ulysses, we propose a pipelined SP (PipeSP) algorithm that partitions the computation of attention along the head dimension and issues an All-to-All immediately after each head is processed, as illustrated in Fig. 3(b), thus overlapping communication with computation to improve the computation efficiency. Specifically, in the attention layer that has n heads, each head is processed independently. Thus, the n heads can be partitioned into n independent attention operations, each of which has only one head. After each head has been computed at its attention, its result can be communicated with other GPUs via an All-to-All operation.

Algorithm 1: PipeSP: Overlapping Computation and Communication in SP

Require: $Q \in \mathbb{R}^{B\times h\times S\times D}$, K, V, $attention\_mask$

1: Initialize $chunks$, $results$, $event\_lst$
2: **for** $j \leftarrow 0$ to $h-1$ **do**
3: $\quad result \leftarrow \texttt{attention}(Q[:,j,:,:], K[:,j,:,:], V[:,j,:,:], attention\_mask[:,j,:,:])$
4: $\quad$ Append $result$ to $results$
5: $\quad$ Record event $event\_lst[j]$
6: $\quad$ Wait on CUDA stream for $event\_lst[j]$
7: $\quad hidden\_states \leftarrow \texttt{All\_to\_All}(results[j])$
8: $\quad$ Append $hidden\_states$ to $chunks$
9: **end for**
10: $hidden\_states \leftarrow \texttt{concat}(chunks, \text{dim} = 1)$
11: $hidden\_states \leftarrow \texttt{view}(-1, h, n, D)$
12: $hidden\_states \leftarrow \texttt{permute}(0, 2, 1, 3)$
13: $hidden\_states \leftarrow \texttt{view}(-1, h \times n, D)$

Thus, the operations of attention (computation) and All-to-All (communication) form a pipeline, which keeps GPU resources be fully utilized during the inference process. After the results of all heads have been gathered, a layout transformation is performed to align the result be identical with that without pipelining.

The PipeSP algorithm is shown in Algorithm 1. In lines 1–9, the $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ tensors are partitioned along the attention head dimension, and for each head, attention is computed over the full sequence. An event is recorded to mark the completion of this computation, and once all GPUs have completed the corresponding event, an All-to-All is triggered. In this step, each GPU receives the portion of the attention output corresponding to its local sequence slice for a single head. Lines 10–13 perform post-processing to reorder the collected results. This reordering is necessary because the optimized method collects attention outputs one head at a time, whereas the original method gathered them in a total different way, resulting in a misalignment shown in Fig. 3(b). To resolve this, the tensor must be reshaped and permuted using the sequence `view` → `permute` → `view`. Specifically, `view(-1, h, n, D)` reshapes the head dimension into a 2D layout of $[h, n]$, `permute(0, 2, 1, 3)` swaps the GPU and head axes, and the final `view(-1, nh, D)` restores the expected layout. Mathematically, this process ensures the final tensor matches the original layout expected by the attention module, while enabling efficient communication–computation overlap. A formal proof of the correctness of the `view–permute–view` transformation is provided in the Supplementary Material.

Memory-Efficient Diffusion–VAE Decoupling

To address the issues of low computational efficiency and poor GPU memory utilization caused by colocating the diffusion model and the VAE decoder, we propose **Diffusion–VAE Module Decoupling** (DeDiVAE) by breaking down the Diffusion module and the VAE module to two disjoint GPU groups: *Denoising Group* and *Decoding Group*. Specifically, for a given N-GPU system for video generation

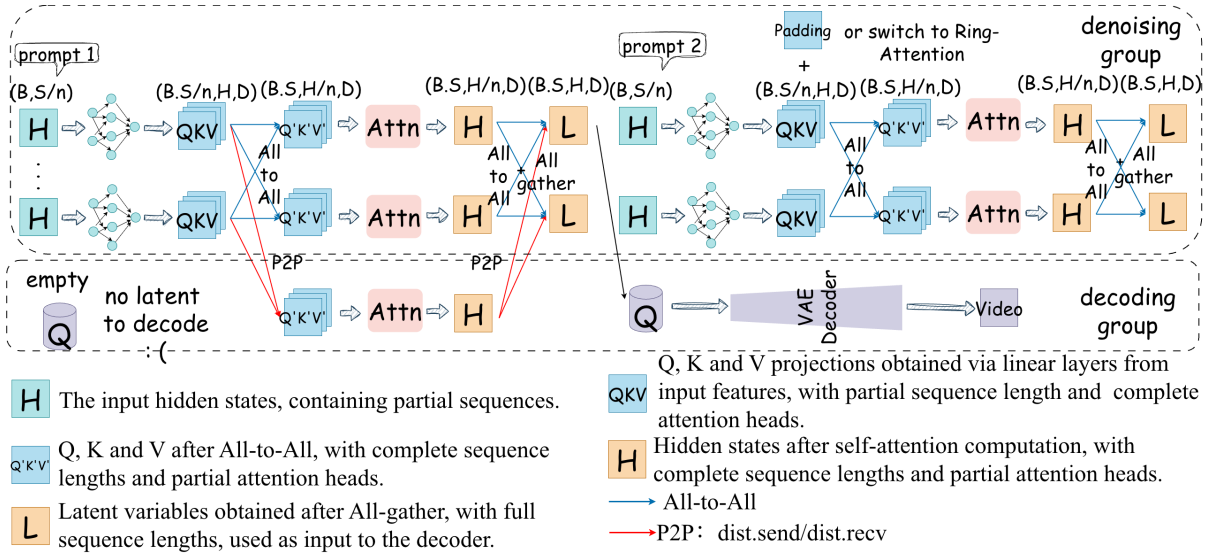

Figure 4: In the prompt 1 stage, the Denoising GPUs transmit the computed **Q**, **K**, and **V** tensors to the Decoding GPUs, enabling parallel attention computation across both groups. In the prompt 2 stage, the Denoising GPUs perform attention computation independently, while the Decoding GPUs execute decoding in parallel.

with DiT, DeDiVAE splits the N GPUs to N_{denoise} GPUs as the *Denoising Group* and the other $N_{\text{decode}} = N - N_{\text{denoise}}$ GPUs as the *Decoding Group*. Accordingly, full video generation model is split into the Diffusion backbone stored in the *Denoising Group*, and the VAE decoder stored in the *Decoding Group*. The decoupling effectively avoids the OOM problem for large models and high-resolution video generation. The latent outputs of the *Denoising Group* should be sent the *Denoising Group* to generate the video.

At first glance, DeDiVAE might also lead to idle periods due to the data dependency between the two groups, potentially affecting inference efficiency. This issue can be addressed by implementing a pipeline execution with multiple prompts. Given that a video generation service typically handles multiple ongoing queries, multiple prompts (or queries) can be pipelined within our decoupled structure as demonstrated in Fig. 4. The decoding execution with VAE of the first prompt can be overlapped with the denoising execution with diffusion of the second prompt, which allows both GPU groups to keep busy. To maximize the utilization of GPU resources, we provide an effective analysis on how many GPUs should be assigned to the two groups.

Optimal GPU partitioning. Given N GPUs for inference, there are N_{denoise} GPUs in the *Denoising Group* and N_{decode} GPUs in the *Decoding Group*. Let T_{denoise} denote the time of denoising one prompt on a single GPU and T_{decode} denote the time of decoding one latent on a single GPU. During inference we accelerate the denoising with SP across N_{denoise} GPUs, while the decoder uses data parallelism across N_{decode} GPUs. Since the total workloads of inference are unchanged, to enable an maximal overlap is to make the execution time of the groups be identical. Thus, the first-order balance condition is

$$\left(\frac{T_{\text{denoise}}}{N_{\text{denoise}}} + T_{\text{comm}}\right) N_{\text{decode}} \approx T_{\text{decode}},$$

$$\Longrightarrow \quad N_{\text{decode}} \approx \left\lceil \frac{T_{\text{decode}}}{T_{\text{decode}} + T_{\text{denoise}}} N \right\rceil,$$

which yields the optimal N_{decode} that maximizes GPU utilization: both stages finish a micro-batch in approximately the same time, preventing either group of GPUs from idling while the other is still computing. Since intra-node GPU communication is very fast, and PipeSP overlaps communication with computation to hide most of the communication overhead, omitting T_{comm} does not affect the resulting resource allocation.

In practice, though we assign the GPUs in a balance way, the execution time of the diffusion stage may still dominate the overall execution time. To improve the efficiency, we design a new co-processing approach in DeDiVAE as introduced in the following section.

Attention Co-processing

When denoising process is much slower than a VAE decoding, the Decoding GPUs idle during most of the generation window, and the pipeline cannot achieve a full overlap. Therefore, we propose **Attention Co-processing** (Aco) to utilize the idle time of the *Decoding Group*. We further split the DiT block into two disjoint kernels:

- *Linear projections*: $\mathbf{Q} = \mathbf{X}W_Q$, $\mathbf{K} = \mathbf{X}W_K$, $\mathbf{V} = \mathbf{X}W_V$,
- *Attention kernel*: $\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$,

and assign them to the two GPU groups. The Denoising GPUs keep the DiT weights and compute the linear pro-

jections; immediately afterwards they transmit the resulting $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors via point-to-point links to the Decoding GPUs, when Decoding GPUs are not decoding latents. Because the attention kernel depends only on $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, and the computations of different attention heads are independent in multi-head attention, the Decoding GPUs can execute it autonomously.

As shown in Fig. 4, in the prompt 1 stage, since there is no latent requiring for decoding, the idle Decoding GPUs can be leveraged to assist with attention computation. Specifically, the Denoising GPUs first perform intra-group All-to-All communication, followed by P2P communication to send the $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors to the Decoding GPUs. Both groups then execute attention computations in parallel. Afterward, the Denoising GPUs aggregate the results via intra-group All-to-All and inter-group P2P communication, obtaining latents with partial sequence length but complete attention heads. Afterwards, the Denoising GPUs push the latent outputs into a shared queue that is accessible to all processes. The Decoding GPUs then retrieve them from the queue for subsequent decoding.

In the prompt 2 stage, as the decoding queue becomes non-empty, the Decoding GPUs are occupied with decoding latents from the queue. Consequently, the Denoising GPUs must perform attention computation independently. During this stage, denoising and decoding proceed in parallel. It is worth noting that if the number of attention heads is not divisible by the number of Denoising GPUs, there are two possible strategies to handle this. For models that only adopt Ulysses, such as OpenSoraPlan (PKU-YuanGroup 2025), head dimension padding must be introduced to ensure balanced workload distribution. In contrast, models like HunyuanVideo (Tencent-Hunyuan 2025) and Wan (Wan et al. 2025) adopt Unified Sequence Parallelism (USP) (Fang and Zhao 2024), which allows flexible configuration of both the Ulysses degree and the Ring-Attention degree. When the number of heads is not divisible, we can switch Denoising GPUs from Ulysses to Ring-Attention, effectively changing the parallelism from head-wise splitting to sequence-wise splitting, thereby avoiding the overhead of padding and improving GPU utilization.

Performance Analysis. Let t_L and t_A denote the wall-clock time of one linear-projection and one attention kernel on Denoising GPUs, respectively, and let N_{denoise} and N_{decode} be the two GPU group sizes ($N_{\text{denoise}} + N_{\text{decode}} = N$). In the baseline decoupling only the denoising group participates:

$$T_{\text{baseline}} = t_L + t_A. \tag{1}$$

With Aco, the linear part still costs t_L, but the attention time scales inversely with the total number of GPUs that now share the work:

$$T_{\text{coop}} = t_L \; + \; t_A \, \frac{N_{\text{denoise}}}{N_{\text{denoise}} + N_{\text{decode}}}. \tag{2}$$

Hence the theoretical speed-up is

$$S = \frac{T_{\text{baseline}}}{T_{\text{coop}}} = \frac{t_L + t_A}{t_L + t_A \frac{N_{\text{denoise}}}{N}}. \tag{3}$$

Note that the above analysis assumes the number of attention heads H is divisible by the number of Denoising GPUs N_{denoise}. If not divisible, and switching between Ulysses and Ring-Attention as in HunyuanVideo is not supported, then padding is required to balance the workload, leading to wasted GPU resources. For example, if $H = 24$ and 7 GPUs are used for denoising, padding increases the head count to 28 so that each GPU handles 4 heads. However, only 6 GPUs are effectively needed, and one GPU performs redundant computations. Our Attention Co-processing solves this issue by avoiding padding and ensuring all GPUs perform meaningful work, even when H is not divisible by N_{denoise}.

Evaluation

Experimental Setups

Baselines. We implement our PipeDiT on two state-of-the-art open-source video generation systems OpenSoraPlan at v1.3.0[1] and HunyuanVideo[2]. As the generation algorithm of PipeDiT is identical with OpenSoraPlan and HunyuanVideo, the generated videos are identical, so we mainly compare the time and memory efficiency. Note that OpenSoraPlan uses Ulysses, while HunyuanVideo integrates USP in xDiT (Fang et al. 2024a). The model sizes for OpenSoraPlan and HunyuanVideo are 2B and 13B parameters, respectively.

Performance Metrics. The performance metrics are video generation efficiency and GPU memory consumption. The efficiency metrics consist of two aspects. The first is the latency per timestep, which measures the optimization effect of PipeSP. The second metric is the overall latency for generating multiple videos from consecutive prompts, which measures the overall optimization effect of PipeDiT.

Testbeds. All experiments are conducted on two 8-GPU systems: 1) eight NVIDIA RTX A6000 48GB GPUs and 2) eight NVIDIA L40 48GB GPUs. More environment information can be found in the extended version.

End-to-End Performance

We configure different video resolutions, commonly used diffusion timesteps (10, 30, and 50), and 10 prompts[3] to compare generation latency as shown in Table 1, where "base" indicates the baseline using offloading, "opt" indicates our PipeDiT, and "spd" indicates the speedup of PipeDiT over the baseline. Since Aco does not always achieve improvement especially on low workload video generation, bold numbers indicate the results generated using PipeDiT w/ Aco.

From the results in Table 1, our PipeDiT always achieves faster inference speed by $1.08\times$-$3.27\times$ than baseline both OpenSoraPlan and HunyuanVideo. Particularly, PipeDiT yields the most notable speedups under lower resolutions, fewer frames, and shorter timesteps, reaching up to $3.27\times$ (up to $4.02\times$ as shown in Supplementary Material). As the resolution, frame count, and timesteps increase, the benefit diminishes since the computation time dominates and the

[1] https://github.com/PKU-YuanGroup/Open-Sora-Plan

[2] https://github.com/Tencent-Hunyuan/HunyuanVideo

[3] Due the page limit, we put the results with more comprehensive configurations in the extended version.

Resolution	OpenSoraPlan (A6000)									OpenSoraPlan (L40)								
	10			30			50			10			30			50		
	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑
$480 \times 352 \times 97$	227	107	2.12×	420	304	1.38×	622	502	1.24×	252	154	1.64×	492	**407**	1.21×	738	**657**	1.12×
$640 \times 352 \times 97$	257	135	1.90×	522	389	1.34×	786	643	1.22×	303	**206**	1.47×	650	**545**	1.19×	983	**883**	1.11×
$800 \times 592 \times 97$	520	397	1.31×	1257	**1097**	1.15×	1994	**1766**	1.13×	646	**517**	1.25×	1609	**1441**	1.12×	2570	**2373**	1.08×
$1024 \times 576 \times 97$	555	430	1.29×	1360	**1144**	1.19×	2162	**1832**	1.18×	731	**591**	1.24×	1836	**1639**	1.12×	2940	**2689**	1.09×
Resolution	HunyuanVideo (A6000)									HunyuanVideo (L40)								
	10			30			50			10			30			50		
	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑	base	opt	spd↑
$480 \times 352 \times 97$	540	**165**	3.27×	767	**445**	1.72×	965	**726**	1.33×	676	**229**	2.95×	992	**649**	1.53×	1350	**1068**	1.26×
$640 \times 352 \times 97$	593	**191**	3.10×	865	**531**	1.63×	1142	**907**	1.26×	760	**295**	2.58×	1231	**843**	1.46×	1702	**1392**	1.22×
$800 \times 592 \times 97$	1082	**506**	2.14×	1880	**1492**	1.26×	2686	**2470**	1.09×	1694	**923**	1.84×	3291	**2702**	1.22×	4898	**4482**	1.09×
$1024 \times 576 \times 97$	1399	**729**	1.92×	2545	**2090**	1.22×	3726	**3453**	1.08×	2237	**1333**	1.68×	4576	**3894**	1.18×	6952	**6453**	1.08×

Table 1: Latency and speedup for generating 10 videos with the baseline system and our optimized PipeDiT. Bold numbers indicate the results obtained with PipeDiT w/ Aco.

OpenSoraPlan (A6000)								
	480×352×65	480×352×129	640×352×65	640×352×129	800×592×65	800×592×129	1024×576×65	1024×576×129
Baseline (s)	1.67	1.30	1.21	2.10	2.21	4.98	2.57	6.86
PipeSP(s)	1.73	1.20	1.69	1.83	2.05	4.74	2.41	6.54
Speedup	0.97×	1.08×	0.72×	1.15×	1.08×	1.05×	1.07×	1.05×
OpenSoraPlan (L40)								
	480×352×65	480×352×129	640×352×65	640×352×129	800×592×65	800×592×129	1024×576×65	1024×576×129
Baseline (s)	1.36	1.66	1.05	2.44	2.87	7.34	3.61	10.30
PipeSP (s)	1.42	1.57	1.01	2.34	2.74	7.05	3.44	9.95
Speedup	0.96×	1.06×	1.04×	1.04×	1.05×	1.04×	1.05×	1.04×

Table 2: Improvement in per-timestep latency with our PipeSP.

relative impact of data transfer decreases, making offloading less of a bottleneck. Despite this, our PipeDiT on OpenSoraPlan with the A6000 platform still achieves $1.18\times$ improvement over the baseline in the highest setting.

For different models, HunyuanVideo has more parameters, so its offloading takes longer time, making PipeDiT more effective to HunyuanVideo under lower resolutions and timesteps. In contrast, under higher resolutions and timesteps, the optimizations of PipeDiT in HunyuanVideo bring greater gains to OpenSoraPlan due to its shorter computation time. For different hardware, because the A6000 GPUs are connected with NVLink which delivers higher communication speed than L40. Thus, PipeDiT yields shorter per-timestep computation times compared to L40, and thus it has higher improvement on A6000 than L40.

Effectiveness of PipeSP

To evaluate the effectiveness of PipeSP, we use eight representative resolution and frame configurations and measure the per-timestep latency, as shown in Table 2. Notably, PipeSP achieves a 15% performance improvement under the 640×352×129 configuration. The results also indicate that the optimization achieves the best performance under moderate resolutions, as it strikes a balance between overly short and excessively long computation times. When the resolution is low, the computation time is short, and the communication overhead introduced by overlap can offset its benefits. Conversely, at high resolutions, the proportion of communication time becomes less significant relative to the overall computation time, thereby reducing the optimization gains.

Memory Efficiency of DeDiVAE

For the memory optimization comparison, we compare our DeDiVAE with the original implementation w/o offloading as the baseline. The second row in Table 4 presents the original implementation with offloading, while the third row shows the results of our DeDiVAE approach. As shown in the table, both our method and the offloading strategy significantly reduce memory consumption during inference. The baseline implementation fails with an OOM error under the highest resolution setting, indicating that without any memory optimization strategies, the model is unable to gener-

A	B	C	D	480×352×65 T(s)↓	480×352×65 Spd↑	480×352×129 T(s)↓	480×352×129 Spd↑	640×352×65 T(s)↓	640×352×65 Spd↑	640×352×129 T(s)↓	640×352×129 Spd↑	800×592×65 T(s)↓	800×592×65 Spd↑	800×592×129 T(s)↓	800×592×129 Spd↑	1024×576×65 T(s)↓	1024×576×65 Spd↑	1024×576×129 T(s)↓	1024×576×129 Spd↑
OpenSoraPlan (A6000)																			
✓	✗	✗	✗	314	1×	529	1×	368	1×	665	1×	777	1×	1875	1×	851	1×	1995	1×
✗	✓	✗	✗	217	1.45×	452	1.17×	234	1.57×	500	1.33×	649	1.20×	1872	1.00×	702	1.21×	2138	0.93×
✗	✓	✓	✗	200	1.57×	390	1.36×	250	1.47×	509	1.31×	649	1.20×	1847	1.02×	717	1.19×	1936	1.03×
✗	✓	✓	✓	261	1.20×	414	1.28×	296	1.24×	507	1.31×	645	1.20×	1652	1.14×	683	1.25×	1690	1.18×
HunyuanVideo (A6000)																			
✓	✗	✗	✗	636	1×	911	1×	695	1×	1104	1×	1294	1×	2681	1×	1676	1×	3733	1×
✗	✓	✗	✗	340	1.87×	681	1.34×	403	1.72×	824	1.34×	984	1.32×	2501	1.07×	1374	1.22×	3680	1.01×
✗	✓	✓	✗	345	1.84×	701	1.30×	404	1.72×	824	1.34×	983	1.32×	2499	1.07×	1374	1.22×	3675	1.02×
✗	✓	✓	✓	327	1.94×	595	1.53×	396	1.76×	741	1.49×	942	1.37×	2259	1.19×	1242	1.35×	3090	1.21×

Table 3: Efficiency improvement of different optimization methods.

Methods	480×352×129 Mem	480×352×129 ↓%	640×352×129 Mem	640×352×129 ↓%	1024×576×129 Mem	1024×576×129 ↓%
OpenSoraPlan						
Baseline	26.5	–	29.4	–	OOM	–
Offloading	18.4	30.6%	18.4	37.4%	28.3	41.0%
DeDiVAE	18.0	32.1%	18.2	38.1%	28.1	41.5%
HunyuanVideo						
Baseline	OOM	–	OOM	–	OOM	–
Offloading	29.37	38.8%	29.38	38.8%	33.01	31.2%
DeDiVAE	41.44	13.6%	41.43	13.7%	42.12	12.2%

Table 4: Peak GPU memory usage (GB) and reduction ratio.

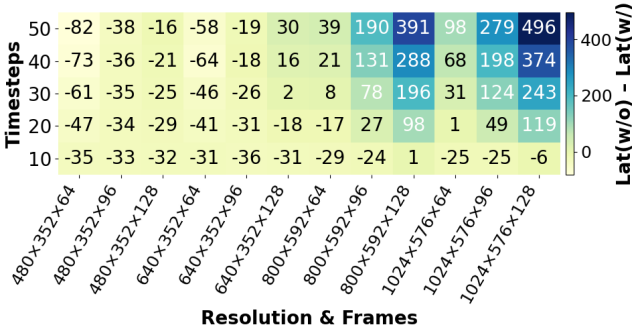

Figure 5: The heatmap of the latency difference between the two methods: (1) PipeDiT w/o Aco and (2) PipeDiT w/ Aco.

ate videos beyond $1024 \times 576 \times 129$ in OpenSoraPlan and $480 \times 352 \times 129$ in HunyuanVideo.

It should be noted that in HunyuanVideo, our DeDiVAE demonstrates greater peak memory consumption than the offloading method. This occurs because we colocated the text encoder with the VAE decoder. Given the substantial size of the text encoder in HunyuanVideo, colocating it with the DiT module, as done in OpenSoraPlan, is not feasible. This demonstrates that our method allows adaptable management of module positioning based on the attributes of the model.

Ablation Study

To evaluate the performance improvements of various optimization methods, we fixed the number of timesteps to 30 and selected eight different resolutions and frame settings for ablation studies. The results are shown in Table 3, where "A" indicates the baseline offloading method, "B" indicates DeDiVAE, "C" indicates PipeSP, and "D" refers to Aco.

The results show that PipeSP demonstrates significant performance improvements on OpenSoraPlan. This is primarily because OpenSoraPlan incurs longer All-to-All communication times, and due to its non-modular design, we were able to partially overlap the computation of **Q**, **K**, and **V** with the three All-to-All communications that occur before the attention computation.

For DeDiVAE, it achieves substantial efficiency gains under lower resolutions. However, as the resolution increases, the drawback of having fewer GPUs allocated to denoising becomes more apparent. Introducing Aco helps to address this limitation. Even under the highest resolution setting, the combined approach still delivers considerable performance benefits. The performance difference of PipeDiT w/ Aco and w/o Aco is shown in Fig. 5, which indicates that Aco improves performance on high workload tasks.

Conclusion

In this study, we propose PipeDiT, a system-level optimization framework for accelerating DiT-based video generation. PipeDiT introduces three key techniques: (1) PipeSP, a pipelined sequence-parallel algorithm that overlaps communication with computation; (2) DeDiVAE, a module-level decoupling method that splits the diffusion and VAE modules across two GPU groups to reduce memory usage; and (3) Aco, an attention co-processing mechanism that utilizes idle decoding GPUs to assist denoising. Experiments on two multi-GPU systems show that PipeDiT significantly reduces memory consumption and greatly improves generation efficiency.

Acknowledgments

The research was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 62302123, and the Shenzhen Science and Technology Program under Grant No. KJZD20240903104103005, KJZD20230923115113026, and KQTD20240729102154066.

References

Abul-Fazl, S.; Dina, R.; and Fairuza, H. 2025. Diffusion Models at Scale: Techniques, Applications, and Challenges.

Blattmann, A.; Dockhorn, T.; Kulal, S.; Mendelevitch, D.; Kilian, M.; Lorenz, D.; Levi, Y.; English, Z.; Voleti, V.; Letts, A.; et al. 2023. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*.

Chen, J.; Ge, C.; Xie, E.; Wu, Y.; Yao, L.; Ren, X.; Wang, Z.; Luo, P.; Lu, H.; and Li, Z. 2024a. Pixart-σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In *European Conference on Computer Vision*, 74–91. Springer.

Chen, M.; Mei, S.; Fan, J.; and Wang, M. 2024b. Opportunities and challenges of diffusion models for generative AI. *National Science Review*, 11(12): nwae348.

Chen, P.; Shen, M.; Ye, P.; Cao, J.; Tu, C.; Bouganis, C.-S.; Zhao, Y.; and Chen, T. 2024c. Delta-DiT: A Training-Free Acceleration Method Tailored for Diffusion Transformers. *arXiv preprint arXiv:2406.01125*.

Cho, J.; Puspitasari, F. D.; Zheng, S.; Zheng, J.; Lee, L.-H.; Kim, T.-H.; Hong, C. S.; and Zhang, C. 2024. Sora as an agi world model? a complete survey on text-to-video generation. *arXiv preprint arXiv:2403.05131*.

Fan, W.; Si, C.; Song, J.; Yang, Z.; He, Y.; Zhuo, L.; Huang, Z.; Dong, Z.; He, J.; Pan, D.; et al. 2025. Vchitect-2.0: Parallel transformer for scaling up video diffusion models. *arXiv preprint arXiv:2501.08453*.

Fang, J.; Pan, J.; Sun, X.; Li, A.; and Wang, J. 2024a. xDiT: an Inference Engine for Diffusion Transformers (DiTs) with Massive Parallelism. *arXiv preprint arXiv:2411.01738*.

Fang, J.; Pan, J.; Wang, J.; Li, A.; and Sun, X. 2024b. Pipefusion: Patch-level pipeline parallelism for diffusion transformers inference. *arXiv preprint arXiv:2405.14430*.

Fang, J.; and Zhao, S. 2024. Usp: A unified sequence parallelism approach for long context generative ai. *arXiv preprint arXiv:2405.07719*.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33: 6840–6851.

Jacobs, S. A.; Tanaka, M.; Zhang, C.; Zhang, M.; Song, S. L.; Rajbhandari, S.; and He, Y. 2023. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*.

Kong, W.; Tian, Q.; Zhang, Z.; Min, R.; Dai, Z.; Zhou, J.; Xiong, J.; Li, X.; Wu, B.; Zhang, J.; et al. 2024. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*.

Li, C.; Huang, D.; Lu, Z.; Xiao, Y.; Pei, Q.; and Bai, L. 2024a. A survey on long video generation: Challenges, methods, and prospects. *arXiv preprint arXiv:2403.16407*.

Li, D.; Shao, R.; Xie, A.; Xing, E.; Gonzalez, J. E.; Stoica, I.; Ma, X.; and Zhang, H. 2023a. Lightseq: Sequence level parallelism for distributed training of long context transformers.

Li, M.; Cai, T.; Cao, J.; Zhang, Q.; Cai, H.; Bai, J.; Jia, Y.; Li, K.; and Han, S. 2024b. Distrifusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7183–7193.

Li, S.; Xue, F.; Baranwal, C.; Li, Y.; and You, Y. 2021. Sequence parallelism: Long sequence training from system perspective. *arXiv preprint arXiv:2105.13120*.

Li, Y.; Wang, H.; Jin, Q.; Hu, J.; Chemerys, P.; Fu, Y.; Wang, Y.; Tulyakov, S.; and Ren, J. 2023b. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *Advances in Neural Information Processing Systems*, 36: 20662–20678.

Liu, F.; Zhang, S.; Wang, X.; Wei, Y.; Qiu, H.; Zhao, Y.; Zhang, Y.; Ye, Q.; and Wan, F. 2025. Timestep Embedding Tells: It's Time to Cache for Video Diffusion Model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 7353–7363.

Ma, X.; Wang, Y.; Jia, G.; Chen, X.; Liu, Z.; Li, Y.-F.; Chen, C.; and Qiao, Y. 2024. Latte: Latent diffusion transformer for video generation. *arXiv preprint arXiv:2401.03048*.

PKU-YuanGroup. 2025. Open-Sora-Plan. https://github.com/PKU-YuanGroup/Open-Sora-Plan. Accessed: 2025-07-19.

Selvaraju, P.; Ding, T.; Chen, T.; Zharkov, I.; and Liang, L. 2024. Fora: Fast-forward caching in diffusion transformer acceleration. *arXiv preprint arXiv:2407.01425*.

Shih, A.; Belkhale, S.; Ermon, S.; Sadigh, D.; and Anari, N. 2023. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36: 4263–4276.

Sohl-Dickstein, J.; Weiss, E.; Maheswaranathan, N.; and Ganguli, S. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, 2256–2265. pmlr.

Song, Y.; and Ermon, S. 2019. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32.

Sun, R.; Zhang, Y.; Shah, T.; Sun, J.; Zhang, S.; Li, W.; Duan, H.; Wei, B.; and Ranjan, R. 2024a. From sora what we can see: A survey of text-to-video generation. *arXiv preprint arXiv:2405.10674*.

Sun, W.; Qin, Z.; Li, D.; Shen, X.; Qiao, Y.; and Zhong, Y. 2024b. Linear attention sequence parallelism. *arXiv preprint arXiv:2404.02882*.

Tencent-Hunyuan. 2025. HunyuanVideo: A Systematic Framework for Large Video Generation Model. https://github.com/Tencent-Hunyuan/HunyuanVideo. Accessed: 2025-07-19.

Wan, T.; Wang, A.; Ai, B.; Wen, B.; Mao, C.; Xie, C.-W.; Chen, D.; Yu, F.; Zhao, H.; Yang, J.; et al. 2025. Wan: Open

and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*.

Wang, Y.; Wang, S.; Zhu, S.; Fu, F.; Liu, X.; Xiao, X.; Li, H.; Li, J.; Wu, F.; and Cui, B. 2025. Flexsp: Accelerating large language model training via flexible sequence parallelism. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 421–436.

Wu, B.; Liu, S.; Zhong, Y.; Sun, P.; Liu, X.; and Jin, X. 2024. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 640–654.

Yang, Z.; Teng, J.; Zheng, W.; Ding, M.; Huang, S.; Xu, J.; Yang, Y.; Hong, W.; Zhang, X.; Feng, G.; et al. 2024. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*.

Zhao, X.; Cheng, S.; Chen, C.; Zheng, Z.; Liu, Z.; Yang, Z.; and You, Y. 2024. Dsp: Dynamic sequence parallelism for multi-dimensional transformers. *arXiv preprint arXiv:2403.10266*.