db-SP: Accelerating Sparse Attention for Visual Generative Models with Dual-Balanced Sequence Parallelism

Siqi Chen [*1] **Ke Hong** [*1] **Tianchen Zhao** [1] **Ruiqi Xie** [1] **Zhenhua Zhu** [1] **Xudong Zhang** [1] **Yu Wang** [1]

Abstract

Scaling Diffusion Transformer (DiT) inference via sequence parallelism is critical for reducing latency in visual generation, but is severely hampered by workload imbalance when applied to models employing block-wise sparse attention. The imbalance stems from the inherent variation in sparsity across attention heads and the irregular distribution of dense blocks within the sparse mask, when sequence parallelism is applied along the head dimension (as in Ulysses) or the block dimension (as in Ring Attention). In this paper, we formalize a *sparse imbalance ratio* to quantify the imbalance, and propose *db*-SP, a sparsity-aware sequence parallelism technique that tackles the challenge. *db*-SP contains a dual-level partitioning approach that achieves near-perfect workload balance at both the head and block levels with negligible overhead. Furthermore, to handle the evolving sparsity patterns across denoising steps and layers, *db*-SP dynamically determines the parallel degrees for the head and block dimensions at runtime. Experimental results demonstrate that *db*-SP delivers an end-to-end speedup of 1.25× and an attention-specific speedup of 1.40× over state-of-the-art sequence parallel methods on average. Code is available at https://github.com/thu-nics/db-SP.

1 Introduction

Recent advances in visual generative models, particularly those based on Diffusion Transformers (DiTs), have revolutionized image and video generation ability. The computational flow of a DiT involves feeding a noised latent and conditioning information into a Transformer (Vaswani et al., 2023) backbone, which predicts the noise to denoise the latent over multiple denoising steps. A Transformer layer is primarily composed of attention and linear computations. Notably, the attention computation suffers from quadratic computational complexity with respect to the token number, and hence becomes a significant bottleneck in DiT inference, occupying over half of the end-to-end latency in multi-frame video generation tasks.

Sparse attention (Zhang et al., 2025c; Xi et al., 2025; Yang et al., 2025a; Xia et al., 2025; Zhao et al., 2025; Ribar et al., 2024; Yuan et al., 2024; Fu et al., 2025) is an effective technique to enhance attention efficiency while maintaining generation quality. Among them, the block-wise sparse pattern (Zhao et al., 2025; Zhang et al., 2025c; Yang et al., 2025a; Xia et al., 2025) becomes promising and is widely adopted in visual generation models due to its efficiency advantages. This efficiency stems from its ability to maintain local computational contiguity and assign adequately sized data to distinct Streaming Multiprocessors (SMs), thereby improving hardware utilization. Nevertheless, even with

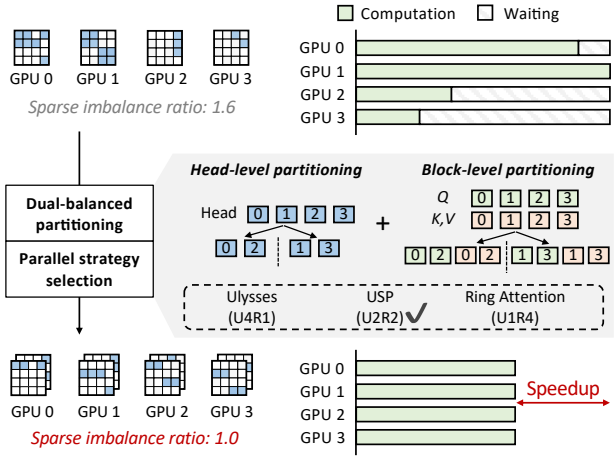

Figure 1. db-SP achieves speedup via dual-balanced partitioning and parallel strategy selection. *Top*: Sparse attention is partitioned across GPUs by attention heads, leading to workload imbalance. *Bottom*: *db*-SP balances the workload at both head-level and block-level partitioning. Ulysses, USP, and Ring Attention are different sequence parallelism methods, where UxRy denotes that x GPUs perform Ulysses and y GPUs perform Ring Attention.

[*]Equal contribution [1]Department of Electronic Engineering, Tsinghua University, Beijing, China. Correspondence to: Yu Wang <yu-wang@mail.tsinghua.edu.cn>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

block-wise sparse attention, the generation latency on a single GPU remains prohibitively high, *e.g.*, exceeding 15 minutes per video using the Wan2.1-T2V-14B model on an A800 GPU with PAROAttention, which is still impractical for real-world use.

Sequence parallelism (SP) plays a critical role in scaling DiT inference. Given that DiT models typically process a large number of tokens but have relatively small model weights, partitioning the activations is more critical than partitioning the weights. Moreover, the compute-intensive nature ensures that partitioning the activations across multiple GPUs still provides each with sufficient workload to saturate the computational capabilities. Consequently, by applying SP, the latency of DiT inference can be scaled down almost proportionally across multiple GPUs.

However, existing SP methods, including Ulysses (Jacobs et al., 2023), Ring Attention (Liu et al., 2023), and USP (Fang & Zhao, 2024), overlook the workload imbalance introduced by sparse attention, failing to fully leverage the potential of multi-GPU parallelism. Specifically, Ulysses partitions different attention heads across GPUs, while Ring Attention partitions the sequence (and thus the dense blocks in sparse attention). USP combines the two aforementioned methods by organizing each GPU into a Ulysses group and a Ring Attention group, allowing the respective parallel method to be applied orthogonally within the dedicated group. Therefore, the workload imbalance manifests at two distinct levels. (1) **Head-level**: The sparsity pattern varies significantly across different attention heads, leading to unequal assignments to each GPU. (2) **Block-level**: The irregular distribution of dense blocks within the sparse attention mask results in an uneven number of blocks across different GPUs.

In this paper, we formalize such workload imbalance by introducing a ***sparse imbalance ratio*** (ρ_s), which indicates the ratio of the workload on the most heavily-loaded GPU to the average number across all GPUs. Our measurements reveal that the *sparse imbalance ratio* achieves over 1.5 in visual generation tasks (1.513 with Wan2.1-T2V-14B model and SpargeAttn using Ulysses on eight A800 GPUs), indicating substantial potential for improvement.

Although mitigating the workload imbalance potentially brings acceleration, there remain several challenges. (1) **Dual-level interplay**: The two levels of workload imbalance are coupled, making the co-optimization a non-trivial challenge that often results in prohibitively intricate partitioning plans. (2) **Reorganizing overhead**: Workload balancing requires data replacement, leading to intra-GPU reorderings and inter-GPU exchanges. The challenge is to mitigate the overhead for higher performance gain. (3) **Parallel strategy selection**: Selecting an SP strategy (*i.e.*, the parallel degrees for Ulysses and Ring Attention) presents a challenge when integrated with sparse attention, as the workload balance is highly dependent on both the sparse pattern and the parallel strategy.

To address the challenges, we propose *db*-SP, a dual-balanced sequence parallelism that balances the workload of sparse attention across GPUs at both levels. At both levels, *db*-SP employs a greedy algorithm for workload partitioning. To reduce overhead, *db*-SP exploits sparse mask similarity across denoising steps to reduce partitioning executions, and introduces a reward factor at the block level to create a biased greedy algorithm that mitigates inter-GPU exchange costs. Given the interplay between the two levels, our design is guided by a key prior: near-perfect load balance is achievable independently at each level. Consequently, *db*-SP simplifies the joint optimization problem into two sequential sub-problems: it first performs head-level partitioning, then conducts block-level partitioning under the assumption of an ideally balanced head-level workload distribution. Furthermore, *db*-SP incorporates a runtime mechanism that dynamically switches between parallel strategies through latency prediction to select the optimal strategy and the corresponding partitioning plan.

The proposed approach is straightforward yet effective, spanning multi-level techniques. (1) **Kernel implementation**: *db*-SP utilizes sorting-based partitioning for workload balancing, avoiding the degraded computation-to-memory-access ratio that results from decomposing dense blocks into small tiles. (2) **Parallelism optimization**: *db*-SP yields a near-perfectly balanced workload across GPUs and introduces dynamic selection of Ulysses and Ring Attention parallel degrees at runtime, as SP does not affect model weight placement. (3) **Application characteristics**: *db*-SP exploits the temporal similarity and initial data residence to reduce overhead.

In summary, this paper makes the following contributions:

- We identify and systematically analyze the dual-level (head and block) workload imbalance that arises from applying block-wise sparse attention with existing sequence parallelism methods.
- We introduce a dual-balanced workload partitioning approach that decouples the joint optimization of the two levels, employing distinct algorithms for the head-level and block-level workload balancing at a negligible overhead.
- We design a sparsity-aware strategy selection mechanism that predicts the latency of different parallelism plans, enabling dynamic selection of the optimal parallel strategy for each attention computation.
- We implement *db*-SP and experiments shown that *db*-SP achieves up to 1.25× end-to-end speedup and 1.40× attention acceleration on average over state-of-

the-art sequence parallel methods on typical DiT models and tasks.

2 BACKGROUND

In this section, we introduce the background of Diffusion Transformer inference in visual generative tasks, and the related works.

2.1 Visual Generative Models

Diffusion Transformer (DiT) (Peebles & Xie, 2023) has emerged as the mainstream model architecture for visual generative tasks such as image and video generation (Wan et al., 2025; Yang et al., 2025b), incorporating the core denoising steps of diffusion models and the scalable and powerful Transformer (Vaswani et al., 2023; Dosovitskiy et al., 2021) architecture. The core of the Transformer architecture is the attention mechanism. Taking the Multi-Head Attention (MHA) in DiT models as an example, the query (Q), key (K), and value (V) matrices derived through linear projection are divided into multiple attention heads along the feature dimension, and then the output (O) is calculated by $O = PV, P = \text{softmax}(QK^T/\sqrt{d})$ for each attention head, where d is head dimension size.

Early DiT models in visual generative tasks, such as OpenSora series (Zheng et al., 2024; Peng et al., 2025), employ a factorized spatial-temporal attention mechanism, computing self-attention along the spatial dimensions (*i.e.,* height and width) and the temporal dimension (*i.e.,* frame) independently. Nevertheless, recent progress including CogVideoX (Yang et al., 2025b) and Wan (Wan et al., 2025) utilizes a more integrated 3-dimensional self-attention (Kong et al., 2025; Arnab et al., 2021) that treats a video clip as a unified sequence of spatial-temporal tokens, allowing the model to jointly attend to visual features across both space and time simultaneously. Such a mechanism further prolongs the attention latency in the inference procedure, degrading the generation efficiency, particularly for high-resolution image or long-video generation tasks.

2.2 Block-wise Sparse Attention

In visual generative models, the attention map $P = \text{softmax}(QK^T/\sqrt{d})$ exhibits inherent sparsity, and previous works (Zhao et al., 2025; Zhang et al., 2025c; Yang et al., 2025a; Xia et al., 2025) demonstrate that the block-wise sparse attention reduces computational costs while maintaining acceptable generation quality. The block-wise pattern is hardware-friendly, avoiding irregular memory accesses within blocks and ensuring sufficient data for parallel computation on GPU Streaming Multiprocessors (SMs).

Following FlashAttention2 (Dao, 2024), Q is partitioned along the sequence dimension into sub-sequences, with the workload corresponding to different sub-sequences and attention heads distributed to different SMs for parallel processing. Consequently, for a block-wise sparse attention kernel, if the block size is an integer multiple of the sub-sequence length, it can reuse the underlying implementation of the dense attention kernel without introducing redundant computation. Therefore, for block-wise sparse attention, ***the total number of dense blocks in the sparse mask across all heads represents the workload on a single GPU***.

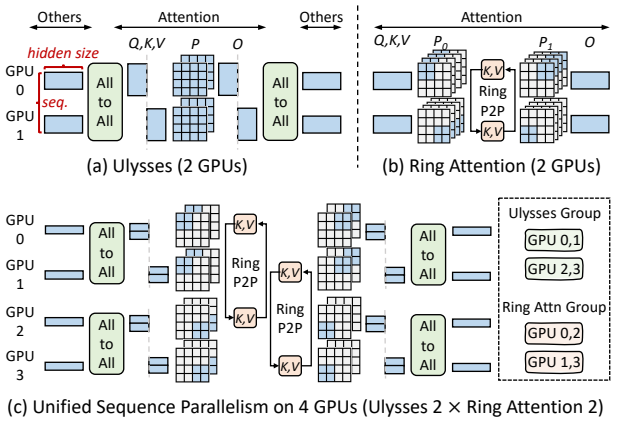

Figure 2. Comparison of sequence parallelism variants.

2.3 Sequence Parallelism

Sequence parallelism (SP) shows advantages in DiT inference over the commonly used Tensor Parallelism (TP), as it avoids the time-consuming AllReduce communication primitive. There are two typical paradigms for sequence parallelism, *i.e.*, Ulysses (Jacobs et al., 2023) and Ring Attention (Liu et al., 2023). Based on that, Unified Sequence Parallelism has been proposed to incorporate the two paradigms to form a hybrid one.

As depicted in Fig. 2(a), Ulysses (Jacobs et al., 2023) partitions the input along the sequence dimension across multiple GPUs for all the other parts except for the attention computation, and dispatches different attention heads onto different GPUs for the attention computation. Specifically, the inputs are reorganized via a pair of all-to-all communication primitives before and after the attention, respectively.

As depicted in Fig. 2(b), Ring Attention (Liu et al., 2023) partitions the input along the sequence dimension for all the parts, including the attention computation. To achieve that, Ring Attention introduces a ring-based communication pattern, *i.e.*, each GPU holds a Q chunk along the sequence dimension, iteratively computing partial attention over each K,V chunk, and then passes the K,V chunk to the neighboring GPU via peer-to-peer (P2P) communication. The procedure is repeated until all K,V chunks have been traversed on each GPU. Such a pattern enables the overlap between the K,V communication and the partial attention

computation, yielding an opportunity for near-zero communication overhead.

Unified Sequence Parallelism (USP) (Fang & Zhao, 2024; Gu et al., 2024) is a hybrid design on top of the two former paradigms, addressing the parallelism degree limitation by the attention head number in Ulysses, and the inefficiency caused by chunked attention computation in Ring Attention. As depicted in Fig. 2(c), USP organizes the GPUs into orthogonal Ulysses and Ring Attention groups. GPUs in the same Ulysses group compute different attention heads, whereas the GPUs in the same Ring Attention group partition Q along the sequence dimension and utilize the ring-based communication to traverse K, V.

2.4 Related Works

In this section, we discuss the related works including the sparse attention methods and parallel computing designs, as well as the studies attempting to incorporate both. Moreover, we also list the related works involving other efficient DiT inference techniques, which are orthogonal to this work.

2.4.1 Sparse Attention Methods

Prior works have explored various attention mask patterns tailored for visual generation, including window-based pattern (Fu et al., 2025; Yuan et al., 2024), spatial-temporal pattern (Xi et al., 2025), hybrid mask combination (Jiang et al., 2024), and block-wise pattern (Zhao et al., 2025; Zhang et al., 2025c; Yang et al., 2025a). Regarding the block-wise pattern, PAROAttention (Zhao et al., 2025) abstracts static masks against varying inputs, whereas SpargeAttn (Zhang et al., 2025c) and Sparse VideoGen2 (Yang et al., 2025a) utilize the dynamic mechanism to selectively compute part of the attention map online. The block-wise pattern enables practical single-GPU acceleration via customized kernels.

2.4.2 Parallel Computing Designs

DistriFusion (Li et al., 2024) and PipeFusion (Fang et al., 2024b) are designed specifically for DiT inference in multi-GPU systems, reusing features to unveil the opportunity for layer-wise parallelism. Nevertheless, the feature reusing brings uncertainty to the generation quality.

Sequence parallelism, including Ulysses (Jacobs et al., 2023), Ring Attention (Liu et al., 2023), and USP (Fang & Zhao, 2024) is the commonly used parallel computing design for accelerating DiT inference, and has been implemented in the open-source inference engines, such as xDiT (Fang et al., 2024a) and ParaAttention (WaveSpeedAI, 2025). However, when combined with the sparse attention, existing sequence parallelism paradigms suffer from severe workload imbalance across GPUs.

BurstAttention (Sun et al., 2024; 2025) attempts to mitigate the imbalance in Ring Attention with block-wise sparsity, by uniformly partitioning a block onto different GPUs to achieve balance. We observe that such a method is not suitable for visual generation, as the sparse block size is necessarily small to retain generation quality. Such a small block size leads to the degradation of the attention kernel performance after partitioning, which is detailed in §6.2.

2.4.3 Efficient DiT Inference

Quantization reduces the memory and computational amounts using low-bit precision of model weights or activations, *e.g.*, SageAttention(Zhang et al., 2025b;a) has explored quantizing QK^T to INT4/8 while employing FP8 (8-bit floating-point) for PV. Caching is the technique that exploits the temporal redundancy across denoising steps, reusing intermediate features (Zou et al., 2025; 2024) or complete outputs (Liu et al., 2025; Lv et al., 2025) for less computation. Besides, DiffServe (Ahmad et al., 2025) constructs model cascades for routing different requests based on the demand, leading to lower latency violation rates. DDiT (Huang et al., 2025) proposes a dynamic resource allocation mechanism for efficiently deploying DiT and encoder modules. Those methods are orthogonal to this work.

3 MOTIVATION

In this section, we first clarify the motivation for using sequence parallelism together with block-wise sparse attention, and then define a theoretical *sparse imbalance ratio* to demonstrate the consequential workload imbalance problem. Finally, we pose the challenges in solving the workload imbalance issue in the real system.

3.1 Sparse Attention with Sequence Parallelism

Sparse attention is still the primary bottleneck, and applying sequence parallelism is rewarding. The self-attention computation in DiT is computationally expensive, growing quadratically with the input token number. As depicted in Fig. 3, when applied with the block-wise sparsity, the sparse attention still occupies over 50% of the total inference latency. The major components in DiT, including the linear layers and self-attention, are computation-intensive operations with large enough input size (>70k tokens), and thus are scalable with sequence parallelism. As depicted in Fig. 3, the end-to-end latency decreases with more GPUs. However, due to workload imbalance, existing sequence parallelism methods fall short of fully realizing their scaling potential, achieving only a 6.09× (Ulysses) and 5.81× (Ring Attention) reduction in latency when scaling from single-GPU inference to eight GPUs.

Directly applying sequence parallelism to sparse attention leads to workload imbalance across GPUs. The workload

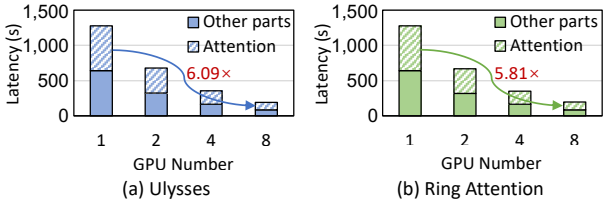

Figure 3. The latency of Wan2.1-T2V-14B with PAROAttention as the GPU number increases.

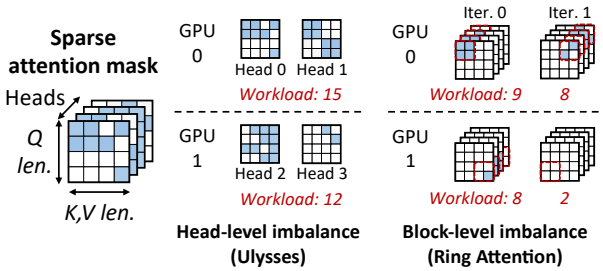

Figure 4. Dual-level workload imbalance when applying sequence parallelism to sparse attention.

imbalance arises from two perspectives: (1) Head-level imbalance. Each attention head owns a single attention mask, and the sparsity distinguishes significantly among different heads. When applied with Ulysses, heads are distributed across GPUs, and thus the sparsity among GPUs is different. (2) Block-level imbalance. The sparse attention mask is inherently irregular rather than uniform. When applied with Ring Attention, each GPU holds one Q chunk, meanwhile iteratively computing the corresponding K, V chunks. At every iteration, the dense block number within the K, V chunk for different Q chunk distinguishes, thereby leading to workload imbalance among GPUs.

3.2 Sparse Imbalance Ratio (ρ_s)

We introduce a *sparse imbalance ratio* ρ_s to quantify such workload imbalance. Workload imbalance brings latency degradation due to synchronization across GPUs. Specifically, Ulysses synchronizes GPUs in the All-to-All communications after attention, and Ring Attention requires synchronization at each iteration via Ring P2P communication. Therefore, we consider the synchronization in the formulation by decomposing the layer-wise latency into N periods based on the synchronization points. Suppose the GPU set is $\mathcal{G}$, ρ_s is defined as follows.

$$\rho_s = \frac{\sum_{i\in\{0,..,N-1\}}(\max_{g\in\mathcal{G}}(b_{i,g}(\mathbf{s},\mathbf{p})))}{\sum_{i\in\{0,..,N-1\},g\in\mathcal{G}} b_{i,g}(\mathbf{s},\mathbf{p})/|\mathcal{G}|}, \quad (1)$$

where $b_{i,g}(\mathbf{s},\mathbf{p})$ denotes the dense block number in the attention masks on GPU g during the i-th synchronization period, which is a function of the sparse pattern $\mathbf{s}$ and parallelism strategy $\mathbf{p}$. Thus, ρ_s is defined as the ratio of the layer-wise latency given $\mathbf{s}$ and $\mathbf{p}$ to the ideal latency under a perfectly balanced workload, *i.e.*, the theoretical speedup through workload balancing.

Table 1. The sparse imbalance ratio (ρ_s) of different models across various sparse attention methods and sequence parallelism paradigms. A larger ρ^s represents a more imbalanced workload. USP (UxRy) denotes x GPUs to form a Ulysses group and y GPUs for a Ring Attention group. SpargeAttn has not been implemented with Ring Attention.

Model	Sparse Attention Method	Sequence Parallelism	ρ_s
Wan2.1	PAROAttention	Ulysses	1.355
		Ring Attention	1.255
		USP (U4R2)	1.253
		USP (U2R4)	1.246
	SpargeAttn	Ulysses	1.428
CogVideoX1.5	PAROAttention	Ulysses	1.447
		Ring Attention	1.269
		USP (U4R2)	1.201
		USP (U2R4)	1.159
	SpargeAttn	Ulysses	1.513

As shown in Tab. 1, we measure the sparsity imbalance ratio ρ_s in real-world scenarios with the sparse attention method PAROAttention (Zhao et al., 2025) and SpargeAttn (Zhang et al., 2025c). ρ_s ranges from 1.159 to 1.513, indicating potentials for acceleration through workload balancing.

3.3 Challenges in Real Systems

Although the workload imbalance can be theoretically addressed, there remain several challenges.

- **How to design a general dual-balanced parallelism method.** The sparse mask exhibits workload imbalance at both head and block levels. A key challenge is to decouple their interplay and design a general dual-balanced method that is agnostic to any block-wise sparse mask pattern.

- **How to mitigate the overhead brought by workload balancing.** Workload balancing inevitably introduces overhead from extra reordering or data exchange. The challenge lies in mitigating the overhead to prevent it from negating the performance benefits.

- **How to choose parallelism strategy against sparse dynamics.** As discussed in §2, Ulysses and Ring Attention each has distinct advantages, and the choice of their degrees in constructing sequence parallelism impacts performance. The impact is more pronounced with sparse attention, where the sparse mask varies across denoising steps or even individual transformer layers, and efficiently determining the parallelism strategy presents a notable challenge.

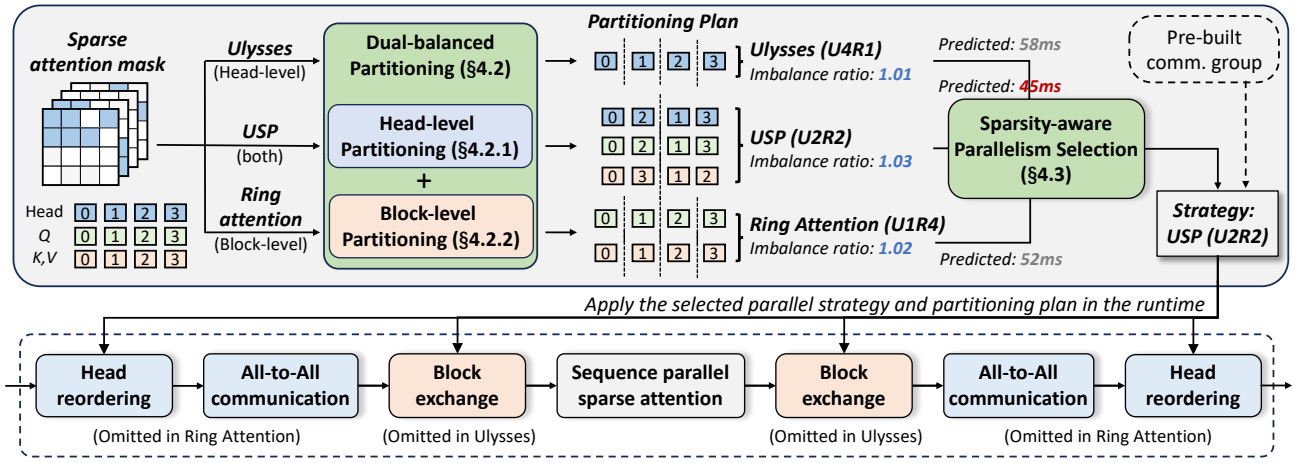

Figure 5. The overview of *db*-SP. For every attention computation, each parallel strategy applies its specific partitioning method to mitigate the workload imbalance, achieving a post-balancing sparse imbalance ratio close to 1. Furthermore, *db*-SP leverages a sparsity-aware selection mechanism that considers the parallel strategy $\mathbf{p}$, sparse pattern $\mathbf{s}$, and *sparse imbalance ratio* ρ_s to predict and apply the optimal parallel strategy and the partitioning plan at runtime.

4 METHODOLOGY

To address the challenges in building such a dual-balanced system that incorporates both sequence parallelism and sparse attention, we propose *db*-SP. In this section, we first introduce the dual-balanced method in *db*-SP that efficiently balances the workload for Ulysses, Ring Attention, and USP with negligible overhead, and then propose a sparsity-aware parallelism selection mechanism to determine the parallelism strategy for each transformer layer.

4.1 Overview

The overview of *db*-SP is shown in the Fig. 5. When computing attention across GPUs, sequence parallelism strategies including Ulysses, Ring Attention, or the hybrid USP (UxRy) are available, where UxRy denotes that every x GPUs form a Ulysses group (head-level parallelism), whereas every y GPUs form a Ring Attention group (Block-level parallelism). Each strategy applies its specific partitioning to mitigate the dual-level (*i.e.*, head and block) workload imbalance, achieving a post-balancing *sparse imbalance ratio* ρ_s close to 1. Taking the parallel strategy $\mathbf{p}$, the sparse attention pattern $\mathbf{s}$, and the resulting ρ_s as inputs, *db*-SP employs a sparsity-aware parallelism selection mechanism to predict the optimal strategy and apply its corresponding partitioning scheme for runtime deployment.

4.2 Dual-Balanced Partitioning

Given a random block-wise sparse mask, the generated dual-balanced partitioning plan results in a *sparse imbalance ratio* ρ_s close to 1. In this section, we first introduce the head-level partitioning for Ulysses and the mask-level partitioning for Ring Attention, respectively. For each, we detail the partitioning approach and the corresponding over-

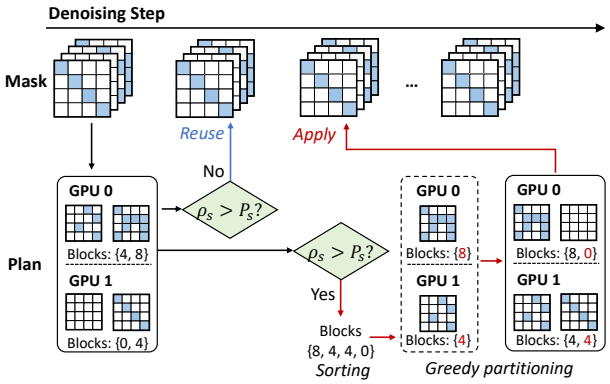

Figure 6. Head-level partitioning in *db*-SP.

head mitigation method. Based on that, we clarify how to incorporate both for USP.

4.2.1 *Head-level Partitioning*

To achieve balance, head-level partitioning distributes the attention heads to different GPUs, so that on every GPU, the total number of dense blocks in the sparse masks corresponding to its attention heads is approximately the same. Since there is only one synchronization point across GPUs (*i.e.*, the All-to-All after attention), the formulation of the *sparse imbalance ratio* ρ_s (Eq. 1) can be specified as follows.

$$\rho_s = \frac{\max_{g\in\mathcal{G}}(b_{0,g}(\mathbf{s}))}{\sum_{g\in\mathcal{G}} b_{0,g}(\mathbf{s})/|\mathcal{G}|}, b_{0,g}(\mathbf{s}) = \sum_{j\in\mathcal{H}_g} b_j^{\text{head}}(\mathbf{s}), \quad (2)$$

where $\mathcal{H}_g$ denotes the partitioned head set on GPU g, and $b_j^{\text{head}}(\mathbf{s})$ is the block number of j-th head's sparse mask.

Partitioning Approach. We apply a straightforward greedy algorithm for head-level partitioning. Specifically, we first sort the heads in descending order based on the number of

dense blocks. Then, we iteratively assign each sorted head to a GPU according to the following principle: each head is always placed onto the GPU that currently has the least cumulative number of blocks. Such a loop continues until all heads are assigned. The sufficiently large number of heads and number of blocks per head in visual generative models make the simple partitioning method highly effective, resulting in a $\rho_s \leq 1.1$ with Ulysses, which is detailed in §6.3.

Overhead Mitigation. Attention masks in each head pose similarity across denoising steps, thereby creating considerable potential for minimizing the overhead of the head-level partitioning. For a dynamic sparse attention method such as SpargeAttn (Zhang et al., 2025c), the sparse masks are generated online, and thus the partitioning plan is determined at runtime for each attention computation, with the number of partitioning executions equaling the number of layers multiplied by the number of denoising steps (*e.g.*, 2,000 times in Wan2.1-T2V-14B). To this end, we propose a ***partitioning plan reusing mechanism by exploiting the similarity across denoising steps***.

Based on the observation, we introduce a threshold P_s to automatically determine the timing for generating a new partitioning plan. For a certain transformer layer, we reuse the partitioning plan from the previous denoising step if the current ρ_s falls below the predefined threshold P_s, otherwise, a new plan is generated based on the current sparse mask. With P_s set to 1.10, partitioning is only necessary in 5 out of 50 denoising steps with Wan2.1-T2V-14B.

4.2.2 *Block-level Partitioning*

Block-level partitioning is responsible for distributing the Q, K, V chunks onto different GPUs, so that on every GPU at every iteration, the computed number of dense blocks is approximately the same. As discussed in §3, Ring Attention performs synchronization across GPUs via Ring P2P communication at every iteration, the formulation of the *sparse imbalance ratio* ρ_s (Eq. 1) is specified by

$$\rho_s = \frac{\sum_{i\in\{0,..,|\mathcal{G}|-1\}}(\max_{g\in\mathcal{G}}(b_{i,g}(\mathbf{s})))}{\sum_{i\in\{0,..,|\mathcal{G}|-1\},g\in\mathcal{G}} b_{i,g}(\mathbf{s})/|\mathcal{G}|}, \tag{3}$$

where $b_{i,g}(\mathbf{s})$ is the dense block number in the computation of the g-th Q chunk and the $((i+g) \bmod |\mathcal{G}|)$-th K, V chunk accumulated across all attention heads.

Partitioning Approach. We quantify the workload of Ring Attention by the number of dense blocks in the 2D computation between Q and K, V. Each GPU is assigned a specific Q chunk, while each iteration processes a specific K, V chunk. Therefore, the goal is to partition the sparse mask as evenly as possible into $|\mathcal{G}| \times |\mathcal{G}|$ regions, each defined by a pair of Q chunk and K, V chunk. To achieve that, we also apply a greedy algorithm to partition Q across GPUs

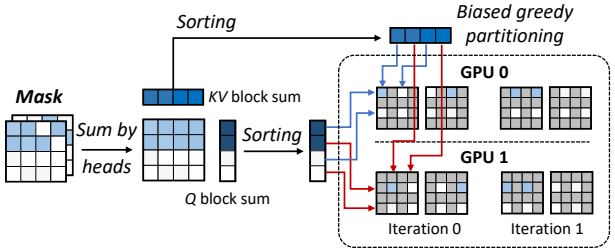

Figure 7. Block-level partitioning in *db*-SP.

and K, V across iterations, respectively. Given a block-wise sparse mask, we partition Q, K, V at the block-size granularity (*e.g.*, 64 as in PAROAttention). For Q, we sort each block in descending order by the total count of its corresponding K, V blocks and then greedily assign them across GPUs, putting every Q block onto the GPU with the least accumulated block number. Similarly, for K, V, we sort each block by its number of corresponding Q blocks and greedily assign them across iterations. Such an approach yields a $\rho_s \leq 1.05$ with Ring Attention, as detailed in §6.4.

Overhead Mitigation. Compared to head-level partitioning, block-level partitioning involves not only the intra-GPU data reorderings but also the inter-GPU data exchanges. Such data exchange is necessitated by the workload balancing. Although the Q and K, V for a given token are generated on a certain GPU, they are subsequently transferred to other GPUs for computation to achieve a balanced workload. Inter-GPU communication is much expensive than intra-GPU data reordering, and thus the key of overhead mitigation lies in minimizing the inter-GPU data exchanges.

We introduce ***a reward factor R_b to encourage the Q and K, V to reside on the initial GPU***. Taking partitioning Q blocks as an example, the modified greedy algorithm proceeds as follows for each Q block. First, the accumulated K, V block count on the Q block's initial GPU is artificially reduced by the reward R_b multiplied by the current Q block's K, V block count. Then, the algorithm selects the GPU with the least resulting accumulated count for assignment. Similarly, the K, V blocks are partitioned in the biased greedy manner.

4.2.3 *Dual-Balancing Partitioning Algorithm*

The optimization of head-level partitioning and block-level partitioning exhibits interdependence. Specifically, the partitioning plan of one determines the total number of blocks and the sparse pattern for the other. To avoid the prohibitive search cost associated with joint optimization, we simplify the problem based on a key insight: both the proposed head-level and block-level partitioning approaches individually yield a near-perfectly balanced workload across GPUs. Consequently, ***the dual-level optimization problem can be decomposed into two subproblems***, each targeting the im-

Algorithm 1 Dual-Balancing Partitioning

Input: sparse pattern $\mathbf{s}$, parallel strategy $\mathbf{p}$ (UxRy), reusing threshold P_s, partitioning reward R_b
head_partitioning_plan ← *old_head_partitioning_plan*
// Head-level partitioning
if $(x > 1)$ and (cal_imbalance_ratio($\mathbf{s}$) $> P_s$) **then**
  blocks_per_head ← get_blocks_per_head($\mathbf{s}$)
  sorted_head_id ← desc_argsort(*blocks_per_head*)
  sum_per_gpu ← $\{0\}_x$
  for *head_id* in enumerate(*sorted_head_id*) **do**
    gpu_id ← argmin(*sum_per_gpu*)
    head_partitioning_plan [*gpu_id*].append(*head_id*)
    sum_per_gpu.update()
  end for
end if
q_partitioning_plan,*kv_partitioning_plan* ← $\emptyset$
// Block-level partitioning
if $y > 1$ **then**
  // Assume workload is balanced across heads
  blocks_accum_by_head ← sum_by_head($\mathbf{s}$)
  blocks_per_q ← sum_by_kv(*blocks_accum_by_head*)
  sorted_q_id ← desc_argsort(*blocks_per_q*)
  sum_per_gpu ← $\{0\}_y$
  for *q_id* in enumerate(*sorted_q_id*) **do**
    // Perform biased greedy partitioning with reward
    biased_sum_per_gpu ← [$s - R_b$ if g is initial_gpu for g, s in enumerate(*sum_per_gpu*)]
    gpu_id ← argmin(*biased_sum_per_gpu*)
    q_partitioning_plan[*gpu_id*].append(*q_id*)
    sum_per_gpu.update()
  end for
  // Repeat to partition K, V blocks
  blocks_per_kv ← sum_by_q(*blocks_accum_by_head*)
  ...
end if
Output: *head_partitioning_plan*, *q_partitioning_plan*, *kv_partitioning_plan*

balance minimization at its respective level based on the assumption that the other is well-balanced.

In practice, we choose to perform head-level partitioning first. Under the assumption that the workload is perfectly balanced across all GPUs within the same Ulysses group, an identical block-level partitioning plan can be optimally derived for all Ring Attention groups. The detailed dual-balancing partitioning algorithm is presented in Alg. 1. At the head level, if the partitioning plan from the last denoising step fails to be reused, the algorithm performs greedy head-level partitioning to generate a new one (Lines 4-13). At the block level, we assume workload is perfectly balanced across different heads, and eliminate the influence of the head dimension through summation. During the partitioning procedure, the reward R_b is introduced to address the initial GPU residence preference of the Q and K, V (Lines 23-25), in order to mitigate the inter-GPU data exchange overhead.

The general principles of hyperparameter selection are as follows. First, on platforms with lower interconnect bandwidth, it is beneficial to use a larger R_b to reduce inter-GPU data exchange overhead. Moreover, with higher sparsity, the imbalance across GPUs is more pronounced. Accordingly, R_b and P_s can be adjusted to smaller values for better workload balancing. In our experiments, the hyperparameters are empirically selected with R_b=5 and P_s=1.05. Based on the sensitivity analysis presented in §6.4, performance remains largely unaffected across ranges of R_b=5∼10 and P_s=1.03∼1.10.

4.3 Sparsity-Aware Parallel Strategy Selection

As discussed in §2, the employment of USP (Fang & Zhao, 2024) forms a parallel strategy $\mathbf{p}$ (UxRy) design space to incorporate Ulysses and Ring Attention. Given hardware, model settings, and inputs, a fixed parallel strategy can be optimal for dense attention under static conditions. Nevertheless, the sparse attention exhibits dynamic sparsity patterns $\mathbf{s}$ that vary per transformer layer and denoising step, which influences both the *sparse imbalance ratio* ρ_s and the efficiency of a parallel strategy $\mathbf{p}$ itself, leading to performance degradation with a fixed parallel strategy.

To further investigate the influence of the sparse pattern $\mathbf{s}$, given parallel strategy $\mathbf{p}$ (UxRy), we formulate the latency of the multi-GPU attention part $\mathcal{L}(\mathbf{s}, \mathbf{p})$ by

$$\begin{aligned}\mathcal{L}(\mathbf{s},\mathbf{p}) =& \mathcal{L}^{\text{all2all}}(x(\mathbf{p})) + \mathcal{L}^{\text{attn}}(\mathbf{s},\mathbf{p}),\\ \mathcal{L}^{\text{attn}}(\mathbf{s},\mathbf{p}) =& \big[\max\big(\mathcal{L}^{\text{comp}}(\mathbf{s}), \mathcal{L}^{\text{p2p}}(y(\mathbf{p}))\big)\\ &\times(y(\mathbf{p})-1) + \mathcal{L}^{\text{comp}}(\mathbf{s})\big] \times \rho_s(\mathbf{s},\mathbf{p}),\\ \mathcal{L}^{\text{comp}}(\mathbf{s}) =& (\frac{\mathcal{L}^{\text{dense}}}{|\mathcal{G}|} \times \text{density}(\mathbf{s})) + \mathcal{L}^{\text{launch}}.\end{aligned} \tag{4}$$

In Eq. 4, $\mathcal{L}^{\text{all2all}}(x(\mathbf{p}))$ denotes the All-to-All communication latency, and hence is determined by the degree of Ulysses. Note that the sparse pattern $\mathbf{s}$ does not affect the communication volume. $\mathcal{L}^{\text{attn}}(\mathbf{s},\mathbf{p})$ represents the attention latency, which considers the computation-communication overlap in Ring Attention, where $\mathcal{L}^{\text{comp}}(\mathbf{s})$ is the attention computation latency and $\mathcal{L}^{\text{p2p}}(y(\mathbf{p}))$ is the Ring P2P communication latency at one iteration. $\mathcal{L}^{\text{attn}}(\mathbf{s},\mathbf{p})$ also includes the performance degradation brought by workload imbalance via the multiplicative factor $\rho_s(\mathbf{s},\mathbf{p})$. Moreover, the computation latency $\mathcal{L}^{\text{comp}}(\mathbf{s})$ is determined by the sparsity/density, and captures the kernel launch overhead $\mathcal{L}^{\text{launch}}$.

On top of that, *db*-SP utilizes a layer-wise sparsity-aware parallel strategy selection mechanism, enabling the strategy switch for every attention computation. The core insight is that, sequence parallelism avoids partitioning model

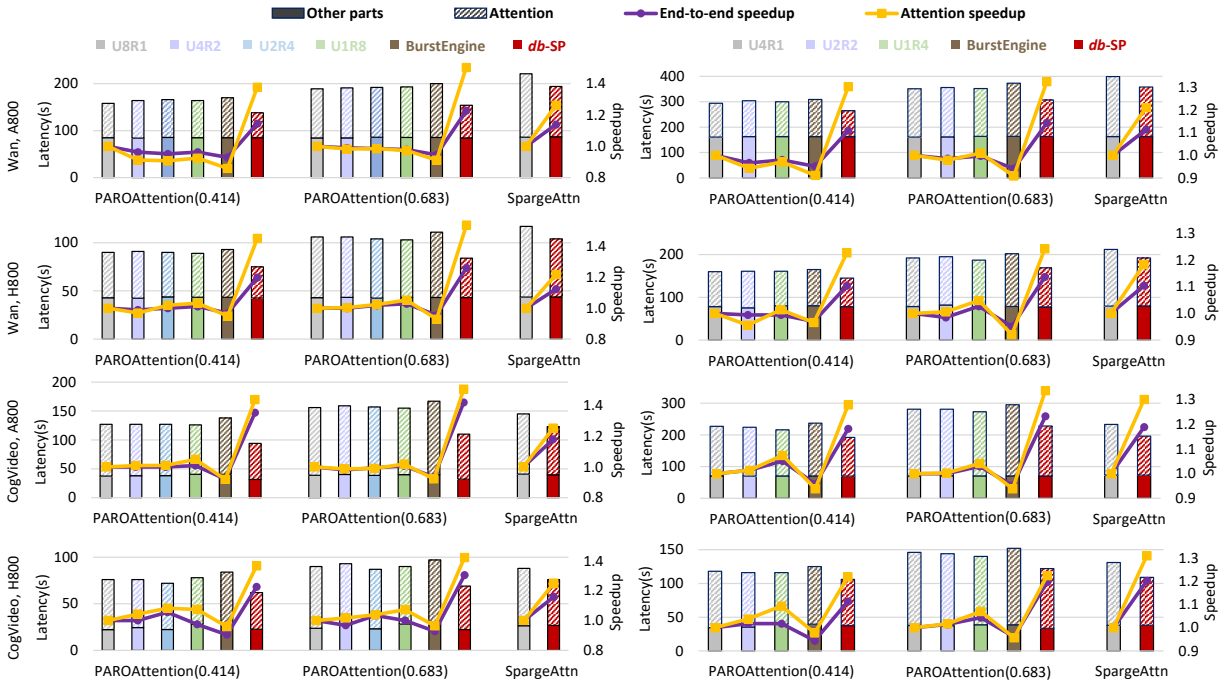

Figure 8. The end-to-end latency comparison on eight (*left*) and four (*right*) GPUs. SpargeAttn only supports Ulysses due to the attention kernel implementation.

weights across GPUs, ***switching between different sequence parallel strategies requires no model reloading, enabling dynamic strategy selection at runtime***. The switching necessitates changes in the inter-GPU communication patterns, which require the reconfiguration of communication groups. To eliminate the overhead, *db*-SP introduces pre-built communication groups, constructing all potential communication groups ($\log_2(|\mathcal{G}|) + 1$ in total) for different parallel strategies before runtime.

Given a sparse pattern $\mathbf{s}$, we predict the latency of each parallel strategy $\mathbf{p}$ using Eq. 4. Specifically, $\mathcal{L}^{\text{all2all}}(x)$, $\mathcal{L}^{\text{p2p}}(y)$, $\mathcal{L}^{\text{dense}}$, and $\mathcal{L}^{\text{launch}}$ are fitted offline using profiling data collected from dense attention computations. Notably, both $\mathcal{L}^{\text{all2all}}(x)$ and $\mathcal{L}^{\text{p2p}}(y)$ are platform-specific and must be individually fitted for each distinct hardware configuration, including its network topology. These two functions are independent of sparsity, which allows them to be fitted offline. In the event of network fabric congestion, the two functions require refitting, and to mitigate prediction errors, periodic fitting can be introduced to track real-time network conditions. At runtime, the parallel strategy with the lowest predicted latency is selected and executed using its pre-built communication groups (x GPUs for Ulysses and y GPUs for Ring Attention), as illustrated in Fig. 5.

5 Implementation

We implement *db*-SP based on the open-source code of USP (Fang & Zhao, 2024), and replace the attention kernel backend with the implementation of PAROAttention (Zhao et al., 2025) or SpargeAttn (Zhang et al., 2025c) depending on the sparse method, to support block-wise sparse masks as input. *db*-SP contains interfaces to integrate into DiT inference engines for sequence parallel attention computation. Currently, we have integrated *db*-SP into the mainstream inference engines, including xDiT (Fang et al., 2024a) and ParaAttention (WaveSpeedAI, 2025), to support the end-to-end inference of different models. *db*-SP involves intra-GPU data reorderings and inter-GPU data exchanges in the partitioning approach, which are implemented by `index_select` and `all_to_all` with NCCL (NVIDIA, 2025) backend in PyTorch (Paszke et al., 2019), respectively.

6 Evaluation

We evaluate *db*-SP with various sparse masks, models, and hardware platforms. The evaluation demonstrates that *db*-SP yields an attention speedup of 1.40× and an end-to-end speedup of 1.25× on average. Meanwhile, *db*-SP introduces merely less than 5% partitioning overhead.

6.1 Setup

Testbed. We conduct experiments on two distinct GPU servers. The first is equipped with eight NVIDIA A800 80GB SXM4 GPUs, and the second has eight NVIDIA H800 80GB GPUs. The GPUs in both servers are fully connected via the NVLink high-speed interconnect fabric. The corresponding software environment includes CUDA

12.1 (Nvidia, 2024), PyTorch 2.5.1 (Paszke et al., 2019), and NCCL 2.21.5 (NVIDIA, 2025).

Benchmark. We evaluate *db*-SP with two models, Wan2.1-T2V-14B (Wan et al., 2025) and CogVideoX1.5-5B (Yang et al., 2025b), abbreviated as Wan2.1 and CogVideoX1.5 in the following. We evaluate by generating 81-frame 1280P videos with 50 sampling steps, and 161-frame 1280P videos with 50 sampling steps, for the Wan2.1-T2V-14B model and the CogVideoX1.5-5B model, respectively. The 3D-VAE tokenizer processes the input into a sequence of tokens, upon which Wan2.1-T2V-14B generates 21 frames at 3600 tokens per frame, and CogVideoX1.5-5B produces 21 frames at 4080 tokens per frame.

For the sparse attention method, we use PAROAttention (Zhao et al., 2025) to represent the ones with static masks across denoising steps, and use SpargeAttn (Zhang et al., 2025c) for the dynamic ones. Specifically, for PAROAttention (Zhao et al., 2025), we use two sparse masks with sparsity of 0.414 and 0.683, denoted as PAROAttention(0.414) and PAROAttention(0.683) in the following, respectively. Note that the attention kernel of SpargeAttn only supports Ulysses.

Baseline. We compare *db*-SP with the typical sequence parallelism methods including Ulysses (Jacobs et al., 2023), Ring Attention (Liu et al., 2023), and USP (Fang & Zhao, 2024). The parallel strategy employed in USP is specified as UxRy, indicating that the parallel degrees for Ulysses and Ring Attention are x and y, respectively. Additionally, we employ BurstEngine (Sun et al., 2025) as our sparsity-aware sequence parallelism baseline, as it specifically optimizes workload imbalance for models utilizing block-wise sparse attention with Ring Attention.

Evaluation Metrics. We record the end-to-end inference latency and the attention latency for comparison. Also, we use the *sparsity imbalance ratio* to measure the extent of workload balance across GPUs.

6.2 End-to-end Performance

Single-Node Results. Fig. 8 shows the end-to-end latency and attention latency comparison on four and eight GPUs. *db*-SP consistently outperforms all baselines, achieving an average end-to-end speedup of 1.12-1.26× for Wan2.1 and 1.16-1.42× for CogVideoX1.5 on eight GPUs. On four GPUs, the end-to-end speedup ranges from 1.10× to 1.15× and from 1.10× to 1.15× for Wan2.1 and CogVideoX1.5, respectively. *db*-SP achieves higher speedup on eight GPUs, as the workload imbalance exacerbates with the increase in GPU count. The sparse attention computation still occupies most of the end-to-end latency, and *db*-SP delivers an average of 1.26×/1.38× attention speedup over Ulysses and 1.22×/1.42× over Ring Attention on four/eight GPUs. Notably, BurstEngine fails to outperform other baselines due to the performance degradation of the attention kernel. Taking a block size of 256 as an example, BurstEngine uses a tile size of 32 for the attention kernel on eight GPUs, which is 1.25× slower than using a tile size of 64. Moreover, we observe that the baselines with different parallel strategies perform similarly without workload balancing. However, once the workload imbalance is addressed, the performance gap becomes pronounced, which is detailed in §6.3.

Table 2. The latency comparison on multi-node settings. U16R1 is not feasible due to the head number.

	U16R1	U8R2	U4R4	U2R8	U1R16	*db*-SP
End-to-end	–	237s	128s	178s	Too slow	116s (**1.10**×)
Attention	–	192s	84.6s	136s	Too slow	73.6s (**1.15**×)

Table 3. Performance comparison between H100 and H800 GPUs. Results are collected with Wan2.1 and PAROAttention (0.683).

	8 GPUs					
	U8R1	U4R2	U2R4	U1R8	*db*-SP	Speedup
H800	106s	106s	104s	103s	84.0s	1.26×
H100	60.7s	60.6s	61.1s	60.7s	40.2s	**1.51**×
	4 GPUs					
	U4R1	U2R2	U1R4	-	*db*-SP	Speedup
H800	192s	195s	187s	-	169s	1.14×
H100	108s	108s	110s	-	86.3s	**1.26**×

Multi-Node Results. To demonstrate the scalability of *db*-SP, we conduct experiments on two nodes, each equipped with eight A100 GPUs and connected via a 200 Gbps cross-node interconnect, using the PAROAttention sparse pattern. As we scale from single-node to multi-node deployment, the relative latency overhead of non-optimized communication becomes increasingly dominant. Nevertheless, the opportunity for improvement remains, as the workload imbalance across GPUs grows with the number of devices. As shown in Tab. 2, *db*-SP achieves a 1.10× end-to-end speedup even in the multi-node setting.

Impact of GPU types. Given the compute-intensive nature of long-sequence DiT workloads, the end-to-end performance gains delivered by *db*-SP are governed by the interplay between the GPU's interconnect bandwidth and its peak computational throughput (*i.e.*, TFLOPS). *db*-SP accelerates the attention computation by dynamically balancing the workload across GPUs. Therefore, the achievable speedup is positively correlated with the proportion of attention computation time. As evidenced in Fig. 8, A800 GPUs yield a higher speedup than H800 GPUs, which can be attributed to A800 GPUs' lower computational throughput, which increases the relative proportion of time spent on

Table 4. Benefits breakdown with dynamic sparse attention, measured with SpargeAttn and Wan2.1.

	U8R1	U4R2	U2R4	U1R8	*db*-SP	
					w/o dynamic parallel strategy (U4R2)	overall
End-to-end	230s	238s	282s	306s	214s (1.11×)	203s (1.13∼1.51×)
Attention	145s	144s	165s	205s	124s (1.16×)	117s (1.24∼1.75×)

Table 5. The sparse imbalance ratio ρ_s before and after dual-level partitioning, ranging across layers and steps.

	U8R1	U4R2	U2R4	U1R8
ρ_s before	1.10∼1.69	1.08∼1.63	1.08∼1.42	1.05∼1.39
ρ_s after	1.00∼1.06	1.00∼1.01	1.00∼1.01	1.00∼1.01

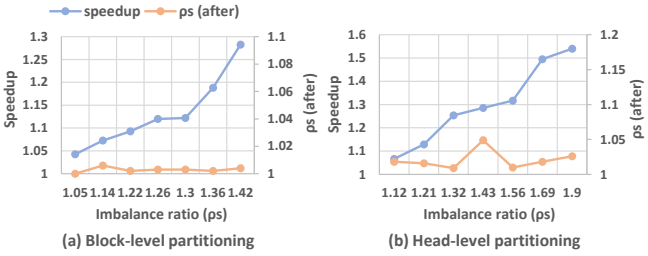

Figure 9. Attention speedup as sparse imbalance ratio grows.

computation, while the two platforms exhibit comparable interconnect performance.

A similar trend can be observed when comparing H100 and H800 GPUs. As presented in Tab.3, *db*-SP attains a higher speedup on H100 GPUs, as the higher interconnect bandwidth on the H100 server reduces the portion of the unoptimized inter-GPU communication latency.

6.3 Ablation Study

Dual-balanced Partitioning. We evaluate the attention speedup brought by head-level partitioning and block-level partitioning separately. Specifically, we use typical sparse masks with varying sparse imbalance ratios ranging across layers and steps and measure the corresponding attention speedups. As depicted in Fig. 9, the speedup achieved by the partitioning approach increases with the sparse imbalance ratio of the input mask, exhibiting a close-to-linear growth trend. Tab. 5 shows the corresponding sparse imbalance ratios (ρ_s) after applying the proposed dual-level balancing approach. The achieved ρ_s is within 6% to the perfectly-balanced upper bound (*i.e.*, an imbalance ratio of 1).

Benefits breakdown. To further understand the benefits of *db*-SP, we modify the attention kernel of SpargeAttn (Zhang et al., 2025c) to support dynamic sparisty. Specifically, we adapt the kernel to Ring Attention computation by enabling

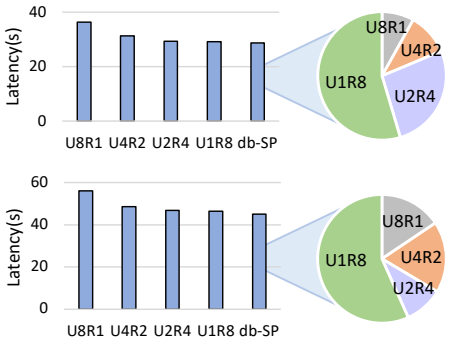

Figure 10. Latency comparison to static parallel strategies.

`log_sum_exp` output. Experiments are conducted on the server with eight A100 GPUs. As shown in Tab. 4, the results demonstrate the effectiveness of both dual-level balancing and dynamic parallel strategy selection for dynamic sparse workloads.

Sparsity-Aware Parallel Strategy Selection. We evaluate the benefits of the parallel strategy selection mechanism by comparing *db*-SP with different parallel strategies with workload balancing. The dynamic parallel strategy switching brings 1.02-1.27× end-to-end speedup compared to all the static strategies. To visualize the selection distribution, we present the detailed ratios of all the strategies in Fig. 10.

6.4 Overhead analysis

Since head-level partitioning only involves intra-GPU reorderings, the overhead ratio in the end-to-end latency is less than 1%. For further investigation, we quantify the effect of using different reusing thresholds P_s on the *sparse imbalance ratio* and the times of new plan generation, as shown in Fig. 11. Although block-level partitioning needs inter-GPU data exchanges, the communication can be overlapped with the preceding computation, such as sorting and indexing. As a result, the overhead can be mitigated to within 5% of the end-to-end latency. Moreover, we also quantify the effect of the reward factor R_b choice, as depicted in Fig. 12. A trade-off exists between the *sparse imbalance ratio* and the associated overhead. We observe that the performance remains relatively stable when R_b is set between 5 and 10 and P_s between 1.03 and 1.10. Based on empirical observations, we select $R_b = 5$ and $P_s = 1.05$ as the default hyperparameters in our experiments.

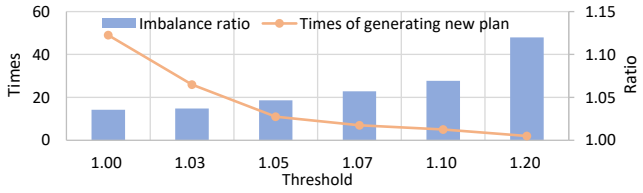

Figure 11. Effect of threshold P_s choice.

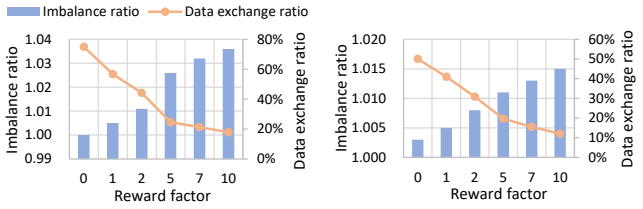

Figure 12. Effect of reward factor R_b choice.

To provide an intuitive understanding of the overhead, we present a typical end-to-end latency breakdown of Wan2.1 with PAROAttention in Tab. 6. The generation of the partitioning plan and parallel strategy incurs negligible overhead, as both are based on lightweight heuristics. The intra-GPU reorderings (`index_select`) and inter-GPU data exchange introduce overheads on the order of seconds, which are further mitigated through overlap and remain minor compared to the overall speedup achieved in the attention.

7 Discussion

In this section, we discuss whether *db*-SP can be generalized to other scenarios.

Irregular Sparsity. The irregular sparse pattern can be viewed as a form of block-wise sparsity with an extremely small block size (*e.g.*, 1×1), which is theoretically manageable by *db*-SP. However, as the block size decreases, the associated overhead becomes significant. Furthermore, smaller block sizes degrade the computation-to-memory-access ratio and introduce irregular memory access within the attention kernel. Consequently, current sparse attention methods (Zhang et al., 2025c; Zhao et al., 2025) for DiT inference adopt block sizes of 64/128, which balances generation quality with kernel efficiency. Compared to the sequence length exceeding 50k, block sizes of 64/128 remain sufficiently fine-grained to support diverse sparse patterns.

DiT Training. *db*-SP mainly focuses on DiT inference, while it is applicable for workload balancing in training. However, it requires additional considerations for the implementation complexity arising from backward computation and other parallelism strategies (*e.g.*, tensor and data parallelism), as well as corresponding modifications to the latency prediction model for parallel strategy selection.

LLMs. The application to mainstream autoregressive LLMs presents several challenges. (1) Attention heads. LLMs tend to employ Grouped Query Attention (GQA) or Multi-head Latent Attention (MLA), which have fewer heads than the Multi-head Attention (MHA) commonly used in DiTs. A reduced head number enlarges workload imbalance under the head-level partitioning strategy of *db*-SP. (2) Limited applicability of SP. While DiTs feature relatively small model weights and inherently long sequence lengths, which are well-suited for SP, the advantages of SP for LLMs are confined to scenarios involving long contexts (Wu et al., 2024).

Table 6. A typical end-to-end latency breakdown of Wan2.1 and PAROAttention on eight A800 GPUs. O1,O2,..,O4 are overheads.

	Baseline	Ours
Attention	104.5s	63.5s (**1.65**×)
O1: Plan and strategy generation	–	0.013s (static sparsity)
O2: Head-level `index_select`	–	1.334s (Q) + 1.342s (K) + 1.518s (V) + 1.534s (O) = 5.728s (*2.860s overlapped*)
O3: Block-level `index_select`	–	1.044s (Q) + 1.124s (K) + 1.116s (V) = 3.284s
O4: Inter-GPU data exchange	–	*1.024s (Q) + 1.002s (K)* + 0.956s (V) + 1.050s (O) = 4.032s (*2.026s overlapped*)
Others	84.5s	84.4s
End-to-end	189s	156s (**1.21**×)

8 Conclusion

In this paper, we introduce *db*-SP, a dual-balanced sequence parallelism method for accelerating visual generative tasks. *db*-SP applies tailored partitioning algorithms at the head level and block level to achieve a near-perfectly balanced workload across GPUs. Moreover, *db*-SP includes a parallel strategy selection mechanism that selects the optimal based on the sparse pattern at runtime. The techniques enable *db*-SP to outperform the existing sequence parallelism methods by 1.25× on average in end-to-end latency.

Acknowledgements

We sincerely thank the anonymous shepherd and reviewers for their insightful suggestions. This work is supported by the Ministry of Industry and Information Technology of China (No. 2440STCZB2589), the National Natural Science Foundation of China (No. 62325405, U24B6015), Beijing Natural Science Foundation (No. L242018, L257010), Beijing Municipal Science and Technology Project (No. Z241100004224013), Beijing National Research Center for Information Science, Technology (BNRist), Beijing Innovation Center for Future Chips, and State Key Laboratory of Space Network and Communications.

REFERENCES

Ahmad, S., Yang, Q., Wang, H., Sitaraman, R. K., and Guan, H. Diffserve: Efficiently serving text-to-image diffusion models with query-aware model scaling. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., and Schmid, C. Vivit: A video vision transformer, 2021. URL https://arxiv.org/abs/2103.15691.

Dao, T. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Fang, J. and Zhao, S. Usp: A unified sequence parallelism approach for long context generative ai, 2024. URL https://arxiv.org/abs/2405.07719.

Fang, J., Pan, J., Sun, X., Li, A., and Wang, J. xdit: an inference engine for diffusion transformers (dits) with massive parallelism. *arXiv preprint arXiv:2411.01738*, 2024a.

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

Fu, Z., Song, W., Wang, Y., Wu, X., Zheng, Y., Zhang, Y., Xu, D., Wei, X., Xu, T., and Zhao, X. Sliding window attention training for efficient large language models, 2025. URL https://arxiv.org/abs/2502.18845.

Gu, D., Sun, P., Hu, Q., Huang, T., Chen, X., Xiong, Y., Wang, G., Chen, Q., Zhao, S., Fang, J., Wen, Y., Zhang, T., Jin, X., and Liu, X. Loongtrain: Efficient training of long-sequence llms with head-context parallelism, 2024. URL https://arxiv.org/abs/2406.18485.

Huang, H., Hu, C., Zhu, J., Gao, Z., Xu, L., Shan, Y., Bao, Y., Ninghui, S., Zhang, T., and Wang, S. Ddit: Dynamic resource allocation for diffusion transformer model serving, 2025. URL https://arxiv.org/pdf/2506.13497.

Jacobs, S. A., Tanaka, M., Zhang, C., Zhang, M., Song, S. L., Rajbhandari, S., and He, Y. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models, 2023. URL https://arxiv.org/abs/2309.14509.

Jiang, H., Li, Y., Zhang, C., Wu, Q., Luo, X., Ahn, S., Han, Z., Abdi, A. H., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024. URL https://arxiv.org/abs/2407.02490.

Kong, W., Tian, Q., Zhang, Z., Min, R., Dai, Z., Zhou, J., Xiong, J., Li, X., Wu, B., Zhang, J., Wu, K., Lin, Q., Yuan, J., Long, Y., Wang, A., Wang, A., Li, C., Huang, D., Yang, F., Tan, H., Wang, H., Song, J., Bai, J., Wu, J., Xue, J., Wang, J., Wang, K., Liu, M., Li, P., Li, S., Wang, W., Yu, W., Deng, X., Li, Y., Chen, Y., Cui, Y., Peng, Y., Yu, Z., He, Z., Xu, Z., Zhou, Z., Xu, Z., Tao, Y., Lu, Q., Liu, S., Zhou, D., Wang, H., Yang, Y., Wang, D., Liu, Y., Jiang, J., and Zhong, C. Hunyuanvideo: A systematic framework for large video generative models, 2025. URL https://arxiv.org/abs/2412.03603.

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

Liu, H., Zaharia, M., and Abbeel, P. Ring attention with blockwise transformers for near-infinite context, 2023. URL https://arxiv.org/abs/2310.01889.

Liu, J., Geddes, J., Guo, Z., Jiang, H., and Nandwana, M. K. Smoothcache: A universal inference acceleration technique for diffusion transformers, 2025. URL https://arxiv.org/abs/2411.10510.

Lv, Z., Si, C., Song, J., Yang, Z., Qiao, Y., Liu, Z., and Wong, K.-Y. K. Fastercache: Training-free video diffusion model acceleration with high quality, 2025. URL https://arxiv.org/abs/2410.19355.

Nvidia. Cuda toolkit. [Online], June 2024. https://developer.nvidia.com/cuda-toolkit.

NVIDIA. Nvidia collective communication library. [Online], 2025. https://docs.nvidia.com/deeplearning/nccl.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Peebles, W. and Xie, S. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Peng, X., Zheng, Z., Shen, C., Young, T., Guo, X., Wang, B., Xu, H., Liu, H., Jiang, M., Li, W., Wang, Y., Ye, A., Ren, G., Ma, Q., Liang, W., Lian, X., Wu, X., Zhong, Y., Li, Z., Gong, C., Lei, G., Cheng, L., Zhang, L., Li, M., Zhang, R., Hu, S., Huang, S., Wang, X., Zhao, Y., Wang, Y., Wei, Z., and You, Y. Open-sora 2.0: Training a commercial-level video generation model in $200k. *arXiv preprint arXiv:2503.09642*, 2025.

Ribar, L., Chelombiev, I., Hudlass-Galley, L., Blake, C., Luschi, C., and Orr, D. Sparq attention: Bandwidth-efficient llm inference, 2024. URL https://arxiv.org/abs/2312.04985.

Sun, A., Zhao, W., Han, X., Yang, C., Liu, Z., Shi, C., and Sun, M. Burstattention: An efficient distributed attention framework for extremely long sequences, 2024. URL https://arxiv.org/abs/2403.09347.

Sun, A., Zhao, W., Han, X., Yang, C., Liu, Z., Shi, C., and sun, M. Burstengine: an efficient distributed framework for training transformers on extremely long sequences of over 1m tokens, 2025. URL https://arxiv.org/abs/2509.19836.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Wan, T., Wang, A., Ai, B., Wen, B., Mao, C., Xie, C.-W., Chen, D., Yu, F., Zhao, H., Yang, J., Zeng, J., Wang, J., Zhang, J., Zhou, J., Wang, J., Chen, J., Zhu, K., Zhao, K., Yan, K., Huang, L., Feng, M., Zhang, N., Li, P., Wu, P., Chu, R., Feng, R., Zhang, S., Sun, S., Fang, T., Wang, T., Gui, T., Weng, T., Shen, T., Lin, W., Wang, W., Wang, W., Zhou, W., Wang, W., Shen, W., Yu, W., Shi, X., Huang, X., Xu, X., Kou, Y., Lv, Y., Li, Y., Liu, Y., Wang, Y., Zhang, Y., Huang, Y., Li, Y., Wu, Y., Liu, Y., Pan, Y., Zheng, Y., Hong, Y., Shi, Y., Feng, Y., Jiang, Z., Han, Z., Wu, Z.-F., and Liu, Z. Wan: Open and advanced large-scale video generative models, 2025. URL https://arxiv.org/abs/2503.20314.

WaveSpeedAI. Paraattention. [Online], 2025. https://github.com/chengzeyi/ParaAttention.

Wu, B., Liu, S., Zhong, Y., Sun, P., Liu, X., and Jin, X. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism, 2024. URL https://arxiv.org/abs/2404.09526.

Xi, H., Yang, S., Zhao, Y., Xu, C., Li, M., Li, X., Lin, Y., Cai, H., Zhang, J., Li, D., et al. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025.

Xia, Y., Ling, S., Fu, F., Wang, Y., Li, H., Xiao, X., and Cui, B. Training-free and adaptive sparse attention for efficient long video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Yang, S., Xi, H., Zhao, Y., Li, M., Zhang, J., Cai, H., Lin, Y., Li, X., Xu, C., Peng, K., et al. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation. *arXiv preprint arXiv:2505.18875*, 2025a.

Yang, Z., Teng, J., Zheng, W., Ding, M., Huang, S., Xu, J., Yang, Y., Hong, W., Zhang, X., Feng, G., Yin, D., Zhang, Y., Wang, W., Cheng, Y., Xu, B., Gu, X., Dong, Y., and Tang, J. Cogvideox: Text-to-video diffusion models with an expert transformer, 2025b. URL https://arxiv.org/abs/2408.06072.

Yuan, Z., Zhang, H., Lu, P., Ning, X., Zhang, L., Zhao, T., Yan, S., Dai, G., and Wang, Y. Ditfastattn: Attention compression for diffusion transformer models, 2024. URL https://arxiv.org/abs/2406.08552.

Zhang, J., Huang, H., Zhang, P., Wei, J., Zhu, J., and Chen, J. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. In *International Conference on Machine Learning (ICML)*, 2025a.

Zhang, J., Wei, J., Zhang, P., Zhu, J., and Chen, J. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *International Conference on Learning Representations (ICLR)*, 2025b.

Zhang, J., Xiang, C., Huang, H., Wei, J., Xi, H., Zhu, J., and Chen, J. Spargeattn: Accurate sparse attention accelerating any model inference. In *International Conference on Machine Learning (ICML)*, 2025c.

Zhao, T., Hong, K., Yang, X., Xiao, X., Li, H., Ling, F., Xie, R., Chen, S., Zhu, H., Zhang, Y., and Wang, Y. Paroattention: Pattern-aware reordering for efficient sparse and quantized attention in visual generation models, 2025. URL https://arxiv.org/abs/2506.16054.

Zheng, Z., Peng, X., Yang, T., Shen, C., Li, S., Liu, H., Zhou, Y., Li, T., and You, Y. Open-sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024.

Zou, C., Zhang, E., Guo, R., Xu, H., He, C., Hu, X., and Zhang, L. Accelerating diffusion transformers with dual feature caching, 2024. URL https://arxiv.org/abs/2412.18911.

Zou, C., Liu, X., Liu, T., Huang, S., and Zhang, L. Accelerating diffusion transformers with token-wise feature caching, 2025. URL https://arxiv.org/abs/2410.05317.

A ARTIFACT APPENDIX

A.1 Abstract

This section is mainly a guideline to perform artifact evaluation for the paper *db-SP: Accelerating Sparse Attention for Visual Generative Models with Dual-Balanced Sequence Parallelism*. The code of *db*-SP includes a dual-level balanced sequence parallel implementation and a sparsity-aware parallel strategy selection mechanism. The following artifacts demonstrate the functionality and efficiency of *db*-SP, including four experiments corresponding to the major claims in the paper: (1) The end-to-end speedup with *db*-SP. (2) The attention speedup against varying sparse imbalance ratios. (3) The effectiveness of the proposed parallel strategy selection method. (4) The negligible overhead against different hyperparameters.

A.2 Artifact check-list (meta-information)

- **Algorithm:** Inference implementation of sparse attention on multi-GPU platforms.
- **Program:** CUDA and Python code.
- **Run-time environment:** Ubuntu 20.04 with CUDA SDK12.1 installed.
- **Hardware:** Any Nvidia GPUs with computational capability $\geq$ 8.0 (Recommended GPU: NVIDIA A800 80GB or NVIDIA H800 80GB).
- **How much disk space required (approximately)?:** 30GB.
- **How much time is needed to prepare workflow (approximately)?:** 2 hours.
- **How much time is needed to complete experiments (approximately)?:** 3 hours.
- **Publicly available?:** Yes.

A.3 Description

A.3.1 How delivered

The source code is available in the form of Github repository (`https://github.com/thu-nics/db-SP`). The sparse mask used for evaluation can be downloaded from Tsinghua Cloud (`https://cloud.tsinghua.edu.cn/d/458ccdacdf9c4edf8548/`).

A.3.2 Hardware dependencies

The implementation works on Intel x86 CPUs and Nvidia GPUs with the computational capability $\geq$ 8.0. For the convenience of evaluation, we provide a server equipped with Nvidia A800 GPUs. Please refer to `README.md` in the GitHub repository.

A.3.3 Software dependencies

- Python 3.8 (version not mandated)
- PyTorch 2.5.1
- NCCL 2.21.5
- Transformers $\geq$ 4.30.0

A.4 Installation

Please follow the `README.md` in the repository to install *db*-SP. The specific dependency is listed in `requirements.txt`, and a conda environment can be created via `environment.yaml`.

A.5 Experiment workflow

Before the evaluation, please prepare the environments and download the models. On the provided server, we have already done the preparation steps. For evaluation, run the provided scripts according to `README.md` for each experiment.

A.6 Evaluation and expected result

For each experiment, the results will be saved into the corresponding `.csv` files under the main directory. The latency and speedup numbers will be shown for comparison.

- End-to-end experiment shows about 1.25$\times$ speedup with *db*-SP on average.
- Attention experiment exhibits a close-to-linear growth trend as the sparse imbalance ratio increases.
- Parallel strategy selection enhances the overall performance by using different strategies for different layers.
- Overhead can be negligible ($<$5%) and provides an opportunity for a trade-off.