OmniCache: Multidimensional Hierarchical Feature Caching for Diffusion Models

Anonymous authors
Paper under double-blind review

Abstract

Recent high-resolution image and video diffusion models, e.g., SD3, FLUX, Sora, have advanced generative intelligence but remain computationally expensive due to quadratic attention and multi-step inference. In this paper, we address the challenge of computational inefficiency in image & video generation by exploiting the inherent redundancy in the processed token content. We identify four primary types of redundancies: intra-frame, inter-frame, motion, and step redundancy. To mitigate these, we propose *OmniCache*, a novel mechanism that employs multidimensional hierarchical feature caching techniques: Frame Cache and Block Cache, together with incorporating Token Cache across transformer layers. These strategies enable us to compress spatial features in the temporal layers and temporal features in the spatial layers, significantly enhancing generation efficiency without the need for additional training. Moreover, we also study the improvements introduced by the orthogonal layered caching technique with *OmniCache*. *OmniCache* is evaluated on state-of-the-art diffusion models for both image and video generation, including SD3, SVD-XT, and Latte. It achieves up to 35% reduction in inference latency on Stable Diffusion 3 (SD3), 25% on SVD-XT, and 28% on Latte, while maintaining high visual fidelity.

1 Introduction

With the rising popularity of Sora (Brooks et al., 2024), generating minute-long high-resolution videos has enabled users to bring their imagination to life. Producing long and coherent videos is a crucial step toward Artificial General Intelligence (AGI). Sora can generate 1800 frames of 1080p video in one run. However, as the resolution and frame rate increase, the required memory and computation rise exponentially, greatly elevating deployment costs.

Unlike image generation, video generation must ensure spatial continuity and vivid motion. To achieve this, methods such as Stable Video Diffusion (SVD) (Blattmann et al., 2023), VideoCrafter (Chen et al., 2024a), MicroCinema (Wang et al., 2023) and AnimateDiff (Guo et al., 2024) extend image diffusion models with temporal layers to ensure temporal consistency while generating each frame. Models like Latte (Ma et al., 2024a) and Open-Sora (Tech, 2023) utilize transformer-based Diffusion alternating between temporal attention and spatial attention at each layer. Consequently, the computational burden of generating N frames is linearly proportional to N, and doubling the video resolution quadruples the computational costs. However, this increase in frames and resolution leads to significant informational redundancy. For instance, as shown in Table 1, increasing frame rate from 1 FPS to 30 FPS raises bitrate threefold, while the frame rate increases thirtyfold; similarly, raising resolution from 360p to 1080p enlarges pixel area ninefold but bitrate only 3.55×. Thus, there exists substantial redundancy in the video generation process, and exploiting this redundancy to accelerate video production can significantly enhance efficiency.

We categorize the inefficiencies in diffusion-based video generation into four complementary types of redundancy:

- *Intra-frame redundancy*: Analogous to a codec's Intra Frame Compression, each frame inherently contains numerous repetitive patterns. By consolidating computations of similar regions within

Resolution & Frame Rate	Bit Rate (Kbps)
640×360, 1 FPS	192
640×360, 30 FPS	576
1920×1080, 30 FPS	2048

Table 1: Approximate H.265 recommended bit rates (Kbps) for various resolutions and frame rates (Hikvision, 2024).

a frame, we can reduce the number of tokens required for spatial attention, thereby increasing computational efficiency.

- *Inter-frame redundancy*: Similar to a codec's Inter Frame Compression, the areas that change between frames are actually quite limited, with a substantial amount of redundant information. Consequently, codecs encode primarily the key frames, while other frames are derived using motion vectors to transmit information. Taking advantage of inter-frame redundancy, we can significantly enhance the generation efficiency via merging similar content across frames.
- *Motion redundancy*: In codecs, motion vectors are calculated on a block basis, primarily because the motion itself is inherently sparse. By merging temporal attention computations across different spatial positions, we can make the generation of motion more efficient.
- *Step redundancy*: In diffusion models, sampling cost increases with the number of denoising steps. We propose to exploit feature similarity across adjacent steps to cache computations, thereby improving denoising efficiency.

Prior studies have partially leveraged such observations. Ma et al. (2023); Wimbauer et al. (2024) reuse feature maps across denoising steps in U-Net architectures, while Chen et al. (2024b); Zou et al. (2025); Ma et al. (2024b) apply caching to transformer-based diffusion models. Other works explored low-resolution optical-flow-like representations (He et al., 2024) or image-to-video generation from single images (Yu et al., 2024; Ni et al., 2023), demonstrating the benefit of exploiting spatial and motion consistency. While effective, these approaches address only one redundancy dimension and lack a unified mechanism to balance computation across spatial, temporal, and step domains. In particular, ToMeSD (Bolya & Hoffman, 2023) accelerates Stable Diffusion by merging similar tokens via feature averaging, which will disrupt positional encoding in video diffusion models.

To address these limitations, we propose *OmniCache*, a unified hierarchical caching framework that removes redundancy across multiple levels of diffusion models. As shown in Figure 1, it integrates three feature-reuse modules—Token Cache, Frame Cache, and Block Cache, targeting intra-frame, inter-frame, and motion redundancy, respectively. It performs merging and unmerging through caching rather than averaging, preserving token order and positional consistency. Throughout this paper, "merging" and "unmerging" denote caching-based feature reuse. In addition, a complementary Layered Cache captures cross-step redundancy at the model-layer level. Together, these modules enable hierarchical compression and efficient computation reuse without modifying the model architecture or requiring retraining.

We formulate caching as an adaptive resource-allocation problem: Given the redundancy pattern across spatial, temporal, and diffusion dimensions, determine where caching yields the largest efficiency gain while maintaining perceptual quality. By combining lightweight similarity computation, patch-level hierarchical caching, and GPU-optimized Triton kernels, *OmniCache* achieves substantial inference acceleration with minimal perceptual quality loss.

The core contributions of *OmniCache* are:

1. *Where are the redundancies:* We introduce a unified optimization perspective that models feature redundancy in diffusion models along four axes: intra-frame, inter-frame, motion, and step redundancy. By framing caching as an *adaptive resource allocation problem* across these dimensions and model layers, we provide a principled foundation for reducing redundant computations in diffusion models.

2. *How to compress better:* We introduce four dimensions of feature caching methods specifically designed for modern diffusion models: Frame Cache, Block Cache, Token Cache, and Layered Cache. By adjusting

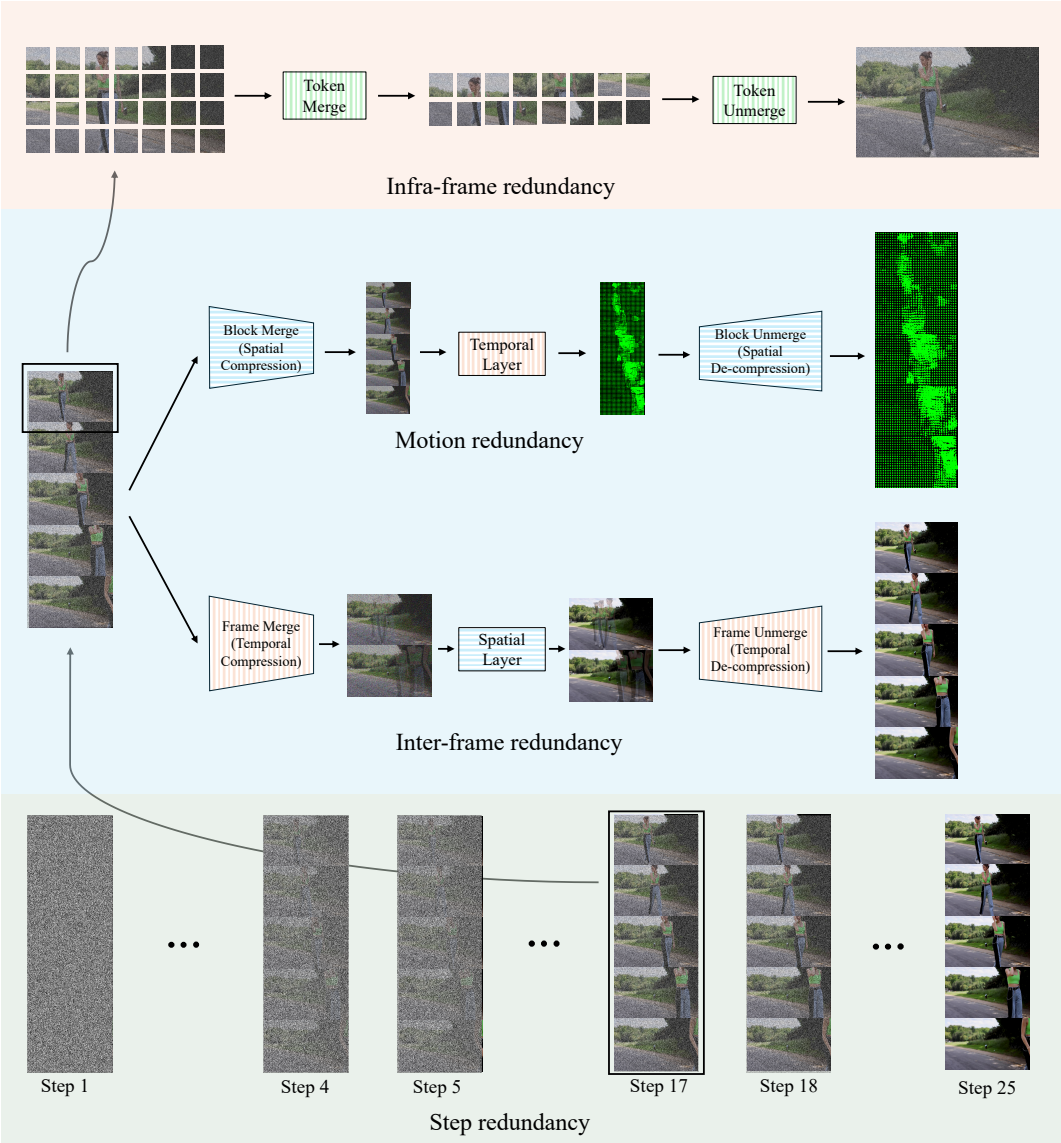

Figure 1: *OmniCache* reduces redundancy in a hierarchical cascade across time and space. First, it reduces step redundancy by caching features across consecutive denoising steps. For the remaining non-cached features, it exploits motion redundancy (repeated spatio-temporal patterns) and inter-frame redundancy (similarity across adjacent frames). Finally, within each frame, it identifies intra-frame redundancy by detecting repetitive tokens.

the hyper-parameters r_f, r_b, and r_t at inference time, we can significantly adjust efficiency and generation quality. Moreover, we introduce two additional controls: M, which denotes the number of transformer or U-Net blocks and allows us to adjust the caching scope to avoid compressing sensitive layers; and T, which controls the caching interval across denoising timesteps. By developing a hierarchical caching, we further enhance efficiency and provide more flexibility.

3. *Validation on 3 SOTA open-sourced image & video diffusion models:* We implement triton-based CUDA kernels for Frame Cache, Block Cache, Token Cache, and validate our approach on video diffusion models like SVD-XT (UNet-based) and Latte (Transformer-based), and image diffusion models like SD3. Our methods allow for training-free improvements, significantly reducing latency and GPU usage with minimal impact on quality and motion.

2 Related Works

2.1 Video Diffusion Models

Stable Video Diffusion (SVD) (Blattmann et al., 2023) extends Stable Diffusion with temporal layers for video generation, achieving high-quality image-to-video results through staged pretraining and finetuning. SV3D (Voleti et al., 2024) is finetuned from the image-to-video SVD model on multi-view datasets, enabling it to take a single image and output a sequence of multi-view images. Latte (Ma et al., 2024a) introduces a Latent Diffusion Transformer that models spatio-temporal tokens in latent space. Sora (Brooks et al., 2024) demonstrates remarkable video synthesis but remains closed-source; Open-Sora (Tech, 2023) offers an open implementation supporting 2–15s videos at 144p–720p.

2.2 Efficient Video Generation

CMD (Yu et al., 2024) proposes a content-motion latent diffusion model that encodes a video as a combination of a content frame (like an image) and a low-dimensional motion latent representation. This approach enables efficient video generation, as it only requires generating a single content frame and a motion latent to reconstruct a vivid video. LFDM (Ni et al., 2023) use a diffusion network that predicts optical flow to generate video by warping a user-given image through a specific decoder. These ideas are similar to the concepts in this paper; Our proposed frame merge idea involves merging similar frames in spatial layers, then using the temporal layers' original frame rate motion information to reconstruct each frame.

2.3 Token Merge

ToMe (Bolya et al., 2023) proposed a token merging mechanism in Vision Transformers (ViTs) (Dosovitskiy et al., 2020) to save computational resources. It introduced an efficient bipartite graph matching mechanism to compute the similarity between tokens, merging a fixed number of r tokens at each layer. However, in diffusion models, features for denoising need to be computed for each position, preventing the reduction of feature map sizes. ToMeSD (Bolya & Hoffman, 2023) addresses this issue by adding token merge and unmerge operations before and after each attention layer, ensuring the resolution remains unchanged. Nevertheless, ToMeSD is only applied to image Stable Diffusion. ToDo (Smith et al., 2024) employs merge operations specifically for key and value in attention layers to enhance performance for Stable Diffusion. However, the modified attention mechanism is challenging to apply directly to flash attention (Dao et al., 2022) or xformer (Lefaudeux et al., 2022). VidTome (Li et al., 2024) uses token merging in video editing scenarios, designing local and global token merge mechanisms to ensure multi-scale spatio-temporal consistency. Their approach focuses mainly on image Stable Diffusion (Rombach et al., 2021) to achieve video editing and has not been verified for modern video diffusion tasks like text-to-video and image-to-video.

In contrast, *OmniCache* is designed for both modern image and video diffusion models, combining redundancy-specific strategies for image and video data. It features a rich set of caching strategies and has demonstrated efficiency improvements in state-of-the-art diffusion models such as Stable Diffusion (SD3), Stable Video Diffusion (SVD-XT), and Latte.

2.4 Layer-wise Token Caching

To improve the poor performance of diffusion based generative models (for both images and videos), recent works have tried exploiting the structural similarity of features across diffusion denoising steps and resorted to caching and reusing those feature maps, thereby reducing the computational overhead in the generation process. DeepCache (Ma et al., 2023) explored this idea in UNet-based image diffusion models i.e., Stable Diffusion 1.5 by observing that high-level computationally expensive features, representing abstract, semantic information, do not change significantly across consecutive denoising steps. This allows them to cache and reuse those features in subsequent steps. Similarly, Δ-DiT (Chen et al., 2024b), and ToCa (Zou et al., 2025) explored caching to speedup generation in transformer-based diffusion models. Δ-DiT exploited the isotropic architecture of DiT models (i.e., Pixart-α, DiT-XL) and cached the offset between input and output features rather than raw feature maps, while also discovering block-specific roles in DiT. ToCa studied method to

adaptively select which tokens to cache based on individual characteristics such as temporal redundancy, error propagation sensitivity and layer depth. For *OmniCache*, we also explored idea of layer-wise token caching and its utilization in DiT-based models (i.e., SD3), however, our method utilizes a data-centric approach to determine the caching pattern via computing latency and output deterioration per transformer block in DiT models across denoising schedule.

3 Methods

3.1 Background

Diffusion Models. Diffusion models generate data by iteratively denoising Gaussian noise through learned transformations. Given a random timestep $t \in [1, T]$ and original (or VAE-sampled latent) data $x_0 \in \mathbb{R}^{C\times H\times W}$ for images or $x_0 \in \mathbb{R}^{N\times C\times H\times W}$ for videos (N=number of frames, C=channels, H=height, W=width), the forward process is $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ where $\bar{\alpha}_t$ is a constant across t and corresponds to noise variance schedule, and ϵ represent noise sampled from standard normal distribution.

The noise estimation network $\epsilon_\theta(x_t, c, t)$, parameterized by θ, takes as input the diffused latent x_t, conditioning signal c (e.g., text prompt or reference image), and diffusion timestep t. It predicts the noise component in x_t to align with the ground truth, and is trained by minimizing the denoising objective introduced by Ho et al. (2020).

$$\mathcal{L}(\theta) = \mathbb{E}_{t,x_0\sim q(x),c,\epsilon\sim\mathcal{N}(0,1)}[||\epsilon - \epsilon_\theta(x_t, c, t)||_2^2]$$

where $q(x)$ represents dataset distribution, and $\mathcal{N}$ being standard Gaussian distribution.

To generate a new image/video during inference, random Gaussian noise is sampled x_T and provided as input to the network ϵ_θ to estimate the noise. Then, the predicted noise $\epsilon_\theta(x_T, \dots)$ is used to denoise the input sample using specific sampling solvers to get x_{T-1}. This process is then iterated across T times to generate the final sample (image/video) $\tilde{x}_0$. If DDPM (Ho et al., 2020) is used during denoising, then the iterative process can be formulated as:

$$x_{t-1} = \frac{1}{\sqrt{\hat{\alpha}_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, c, t)) + \sigma_t z$$

where σ_t, $\beta_t = (1-\alpha_t)$, and $\bar{\alpha_t} = \prod_{k=1}^{t}\alpha_k$ are constant w.r.t t and $z \in \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Recent research have introduced various numerical solvers (Song et al., 2020; Lu et al., 2022; 2025; Karras et al., 2022) which might have different formulations but all are functions of x_t, and ϵ_θ.

Unlike image diffusion, video diffusion must also ensure frame-to-frame coherence. Modern large models alternate spatial and temporal attention within U-Net or transformer blocks, capturing both detailed textures and temporal motion while avoiding prohibitively large attention windows.

Token Merge and Unmerge. Token Merging (ToMe) Bolya et al. (2023) improves Vision Transformer throughput by merging redundant tokens inside transformer blocks via bipartite similarity matching, reducing token count and compute. For diffusion models, however, every spatial position must estimate noise, so ToMeSD (Bolya & Hoffman, 2023) adds merge and unmerge steps around attention and feed-forward layers to lower intermediate cost while keeping the feature-map size unchanged.

The design goal of ToMeSD Bolya & Hoffman (2023) is to merge some of the feature map tokens to reduce computation. It divides the target feature map into **source (src)** and **destination (dst)** parts. The algorithm first partitions the tokens into a **src** and **dst** set and computes the pairwise cosine similarity between the two sets. Then, **src** tokens are linked to their most similar token in **dst**. Next, the most similar edges are selected, and connected tokens are merged. Afterwards, the remaining merged **src** tokens are concatenated with **dst** tokens to serve as input for the target layers. To ensure a certain level of diversity and uniformity, ToMeSD Bolya & Hoffman (2023) randomly selects a feature point from every 2x2 area within each image as the **src** set. Regarding the unmerge operation, if $x^*_{1,2} = \frac{x_1+x_2}{2}$, we can "unmerge" them back into x'_1 and x'_2 by setting $x'_1 = x^*_{1,2}$ and $x'_2 = x^*_{1,2}$. While this operation results in some information loss, the minimal error is justifiable as the tokens were sufficiently similar initially.

Figure 2: Limitation of token merging.

Nevertheless, ToMeSD's averaging operation introduces a reordering of elements within the sequence, which disrupts compatibility with order-sensitive encoding schemes such as positional embeddings e.g., commonly used RoPE embeddings (Heo et al., 2024; Su et al., 2023) as visualized in Figure 2. Moreover, ToMeSD's unmerging strategy simply reuses the averaged features through uniform padding, which lacks position.

To address these limitations, we propose Token Cache, a method that also applies bipartite graph matching to identify similar token pairs, but instead of averaging, it deterministically retains one token while discarding its matched counterpart in a pair of source and destination tokens to preserve sequence order. For the unmerging process, Token Cache leverages step redundancy by caching positionally consistent tokens from previous denoising steps and reusing them in the current step, thereby maintaining spatial coherence and reducing redundant computations

3.2 *OmniCache*

In this section, we present *OmniCache*, a unified hierarchical caching framework that removes redundancy across multiple levels of diffusion models. As shown in Figure 1, it integrates three feature-reuse modules, Token Cache, Frame Cache, and Block Cache, to address intra-frame, inter-frame, and motion redundancy, respectively, along with a complementary Layered Cache for cross-step redundancy. Together, these modules enable hierarchical compression and efficient feature reuse without modifying the model architecture or requiring retraining.

3.2.1 Motion Redundancy and Block Cache

In video diffusion models, motion features are typically extracted by dedicated temporal transformer layers. For an input feature map of dimensions `(B, T, C, H, W)`, we reshape it to `(B*H*W, T, C)` to compute temporal features. This reshaping allows us to calculate temporal relationships across T frames at each H and W coordinate of the video. We refer to this sequence of `1*1*T` attention input as a block. We believe that a significant portion of motion in a video is repetitive, such as background shifts due to camera movement or rigid body motion. This redundancy is why codecs like H.265 use block-wise motion vectors to encode movement across entire regions. Therefore, within the temporal transformer's input blocks, there is substantial potential for computation aggregation.

We cannot directly use ToMeSD's method here, as it only calculates similarity and merges tokens within a single image. However, inspired by its efficient bipartite graph matching to find the closest edges, we can apply a similar approach to compute similarity between blocks for Block Cache.

To calculate the similarity between blocks, the simplest method is to concatenate the features of all `T` elements in a block. As shown in Figure 3, we transform the input features to `(B, H*W, T*C)`, isolating these `H*W` blocks, each containing `T*C` features. Following ToMeSD, we randomly select one element from a `2*2` block as the source block, with the remaining elements serving as destination blocks. We then compute the similarity between the source and destination blocks by multiplying the source vector by the transpose of the destination vector. Instead of averaging similar blocks, we skip the r_b most similar **src** blocks during the merging process and reuse the cached features from the previous denoising step to reconstruct the features

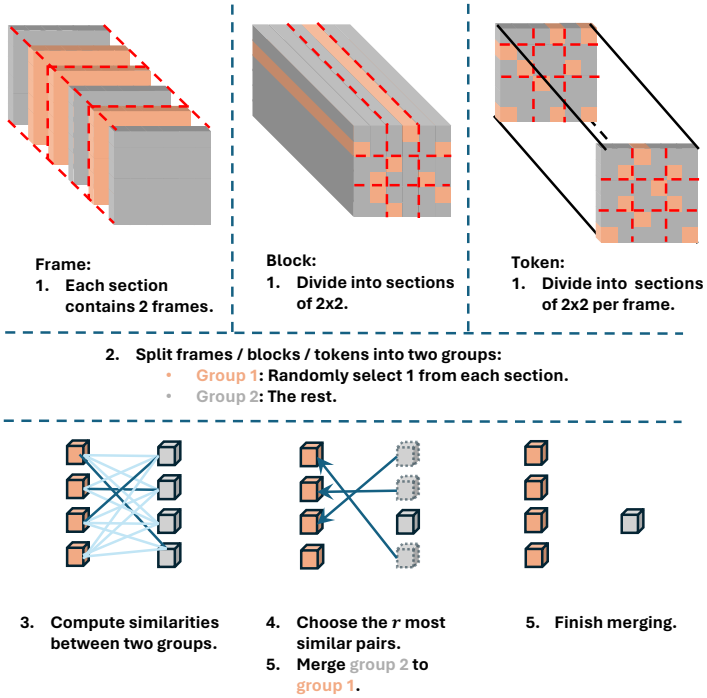

Figure 3: Illustration of *OmniCache*'s *Frame Cache*, *Block Cache*, and *Token Cache* operations. We designed three dimensions of merge operations for modern video diffusion models to more flexibly reduce feature redundancy and enhance efficiency.

in the current step during unmerging. This strategy effectively reduces motion redundancy while preserving positional consistency.

3.2.2 Inter-frame redundancy and Frame Cache

As we know, adjacent frames in a video often contain very close content, such as a static background or minor motion differences. In works like Ni et al. (2023), the warp method is used to efficiently generate videos based on overlapping images. This inspires us to consider whether merging some similar feature maps in the spatial layer during the generation process, allowing them to share computation, could take advantage of this characteristic of videos.

Again, we utilize the bipartite graph matching algorithm to find edges between closest spatial content between frames for Frame Cache. For an input feature map of dimensions `(B, T, H, W, C)`, we reshape it to `(B, T, H*W*C)` to concatenate all features of each frame as the feature of that frame. In this case, we divide these T frames into **src** and destination **dst** sets. To balance randomness and uniformity, we select one frame between every two adjacent frames to enter the **dst** frame set, and then select the r_f most similar frames from the **src** frames. Similar to Block Cache, the most similar r_f frames from **src** set are skipped, and the cached features from the previous denoising step are reused.

A direct concern is whether this approach reduces the frame rate of video generation. In fact, our computation in the motion layers still uses the original frame rate input, only merging some similar feature map calculations in the spatial layers. The independent motion relationships generated in the motion layers at the original frame rate can still ensure the richness of frame-to-frame transitions in the generated video. Therefore, the similar frames averaged in the spatial layer will regain diversity after being processed in the motion layer.

3.2.3 Intra-frame redundancy and Token Cache

Despite performing Frame Cache, there remains considerable redundancy among tokens within each frame. For instance, in two similar frames depicting the sky and the beach, there will still be substantial spatial

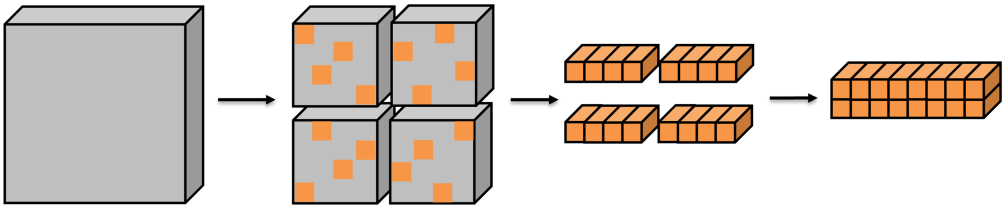

Figure 4: Illustration of Hierarchical Caching

redundancy after Frame Cache. Therefore, we apply Token Cache to each frame before entering the spatial layers.

The input to Token Cache is the feature map generated from Frame Cache, but this leads to a reduced frame rate for video generation where the generated video has fewer motions. To address this, we first calculate the similarity between all frames without merging the similar ones, then split them into merged and unmerged feature maps. For merged feature maps, we identify r_f pairs of frames and merge similar input tokens in each pair to reduce intra-frame redundancy. The processed feature map will have the shape `(B,` r_f`*2, H, W, C)` which we reshape to `(B*`r_f`, 2*H*W, C)`. Within `2*H*W` tokens, we select two mutually exclusive elements from each `2*2*2` cube and merge them into `H*W-`r_t tokens. For unmerged feature maps, within the `H*W` region of each frame, we randomly select one element from each `2*2` block as **src** tokens, while the remaining elements serve as **dst** tokens. We then merge the r_t most similar tokens. Finally, the results from merged and unmerged feature maps are concatenated. This approach maintains the original frame rate while merging similar tokens, minimizing temporal information loss compared to the sequential operation of Frame Cache.

3.3 Hierarchical Caching

For Frame Cache, the input feature map has the shape `(B, T, H*W*C)`, which results in an excessively large dimension when computing similarity due to the large size of the spatial content. To reduce overhead, we divide the spatial content into multiple patches, resulting in a shape of `(B, T, k, H/k, k, W/k, C)` where `k*k` is the number of patches, shown in Figure 4. We then permute and reshape the feature map to `(B*k*k, T, (H/k)*(W/k)*C)` for similarity computation. Using patches for Frame Cache reduces overhead by over 20× for Stable Video Diffusion (SVD) without degrading quality or motion. Similarly, patches can be applied to Block Cache and Token Cache, as adjacent pixels are more likely to be merged. This process reduces the overhead of Block Cache and Token Cache by about 18× and 15×, respectively, in SVD. The usage of patches also enhances caching performance by enabling patch-to-patch caching, which allows for more flexible and diverse caching options across different patches between frames.

After performing caching with patches, we can increase the patch size to allow for further caching or revert the patches back to frames for global caching. This enables a hierarchical caching strategy, where we adjust the caching ratio and hierarchical level to optimize performance.

3.4 Triton Kernel Acceleration

To improve the efficiency of the merging and unmerging operations, we replace the original implementation with custom Triton kernels. These kernels leverage fine-grained GPU parallelism and memory coalescing to optimize the gather–scatter pattern inherent to our token permutation process. Specifically, we design block-level parallel kernels that operate on batched token indices, performing atomic accumulation and permutation across the spatial and temporal dimensions. By controlling tile sizes, warp scheduling, and staged memory access, the Triton kernels achieve high throughput while avoiding redundant global memory synchronization. This design enables direct in-GPU reduction (e.g., mean aggregation) without host round-trips, significantly reducing latency. In our profiling, the Triton-based implementation yields substantial speedups over the original version, demonstrating the benefit of custom kernel fusion and memory-efficient accumulation in our caching pipeline. Additional experiments evaluating the performance of our Triton kernels compared to PyTorch implementations on SD3-medium, across varying token group sizes in the hierarchical caching setup, are provided in Appendix A.

3.5 Step-redundancy and Layered caching

Previous methods such as DeepCache, Δ-DiT (Ma et al., 2023; Chen et al., 2024b) have provided solutions to accelerate diffusion-based generation by resorting to feature reuse (caching) in UNet-based and isotropic DiT-based models. Both of these methods studied that the distribution of feature maps across a given block in the diffusion model across a window of denoising steps remain consistent and thus, can be cached. We also utilize this idea to complement our proposed hierarchical frame, block and token cache methods to propose a holistic approach. However, unlike DeepCache (Ma et al., 2023) and Δ-DiT (Chen et al., 2024b) that utilized feature similarity observations to determine layers/blocks to cache, we propose a data-centric algorithm to compute the caching schedule.

The details of our approach are as follows. For image and video diffusion models, we use prompts (captions) of varying lengths from MS-COCO'17 dataset to compute average latency per transformer layer over defined denoising steps. Moreover, we compute the output loss (using visual quality metrics such as FID, Inception Score, FVD) each transformer layer brings by caching individual layer and generating output over the denoising steps. For each (layer l, denoising timestep t) pair, we assign a weight $w_{(l,t)}$ using the latency and output loss: $w_{(l,t)} = \frac{\text{output loss}}{\text{latency}}$. We utilize computed weights (in sorted order), latencies and user-defined speedup s_r to define the initial caching strategy containing those components which have a higher compute time (higher latency) or a higher impact on output generation as shown in algorithm 1

Once the initial cache schedule is generated, we optimize the cache proposal using the computation graph of the model by tracking the dependencies between layers and their cached outputs at various timesteps in the proposal. This is done so that redundant computations can be avoided while ensuring that each node's output are computed only when needed, based on the model's graph dependencies. The cache strategy refinement algorithm is provided in algorithm 3. Full detials of 3 can be found in Appendix B.

Algorithm 1 Cache Schedule Proposal

Require: weights $w_{(l,t)}$, latencies, target speedup s_r, caching interval N

function GET_CACHE_STRATEGY
  sort(w) by $w_{(l,t)}$
  $TimeSave \leftarrow 1 - \frac{1}{s_r}$; $sum \leftarrow 0$; $i \leftarrow 0$
  while $sum < TimeSave$ **do**
    $sum \leftarrow sum + w[i][l]$
    $CacheStrategy[w[i][layer], w[i][timestep]] \leftarrow 1$
    $i \leftarrow i + 1$
  end while
  return $CacheStrategy$
end function

Algorithm 2 Cache Strategy Refinement

Require: Model graph $\mathcal{G}$, CacheStrategy $\mathcal{C}$

Preprocess: Initialize metadata for nodes
Build Dependencies: Set child/parent relations
Count Dependencies: Compute child_num, parent_num
Prune Redundancies: Remove unnecessary cache steps
Prune Consecutive Steps: Drop identical outputs
Recalculate Counters: Update dependency counts
Align Steps: Match with parent cache intervals
return Optimized strategy $\hat{\mathcal{C}}$

Figure 5: **(Left)** Algorithm for generating cache schedule proposals. **(Right)** Refinement process for optimizing cache strategies.

4 Experiments

4.1 Experimental Settings

Model Configurations. We conduct experiments employing commonly used image and video diffusion models, including Stable-Video-Diffusion (SVD-XT) for image-to-video generation and Stable-Diffusion-3-medium (SD3-medium) Esser et al. (2024) for text-to-image generation using NVIDIA A100 with 40 GB VRAM. Similar to ToMe, we utilize these models directly with our method without training. Each model uses its default scheduler for sampling: Flow Matching Euler Discrete with 28 sampling steps for SD3-medium and Euler Discrete Sampler with 25 sampling steps for SVD. In the SVD inference experiments, the model is utilized to generate 25 frames with a resolution of 576×1024. In the Latte inference experiments, the Latte-1 T2V model is employed to produce 16 frames with a resolution of 512×512.

Evaluation and Metrics. To evaluate performance of image-to-video setting, we utilize SVD to produce 10,000 videos using the UCF101 dataset and measure FVD scores. For text-to-image generation, we use 8,000

randomly generated MS-COCO'14 captions to generate images and 20,000 real dataset images to compute FID score to assess image generation quality, while CLIP score is used to access alignment of images and captions.

4.2 Design Choices

We investigate various ratios for each caching option and determine the appropriate layers for application in our approach. Our thorough analysis reveals that the initial and final spatial transformer layers are particularly sensitive to caching. For instance, in SVD, we exclude the last layer of the decoder block from frame caching. For the Latte experiments, we exclude caching in the first and last transformer layers, and assign the ratios r_f, r_b, and r_t as 30%, 40%, and 20%, respectively, across the remaining 26 transformer layers. We further observe that applying frame caching or token caching to fully connected (FFN) layers degrades generation quality. Accordingly, in the SVD configuration, the ratios r_f, r_b, and r_t are set to 50%, 40%, and 50%, respectively, while excluding the last layer from frame caching and all FFN layers from both frame and token caching. For SD3-medium, we apply hierarchical token caching to both joint attention and FFN layers. However, our experiments indicate we cannot apply caching on the top and bottom 4 transformer blocks as they are sensitive to caching which in turn impacts the generated image quality. We hypothesize that this is because the top blocks generate low-level (coarse) features, while the bottom blocks generate high-level (fine) features, as shown in previous studies Chen et al. (2024b). We experimented with different settings of merge ratios for SD3-medium as can be seen in Table 2

4.3 Main results

Training-free improvements on state-of-the-art image and video Diffusion models. We validate the effectiveness of *OmniCache* on UNet–based image-to-video models (e.g., SVD), as well as DiT–based models for text-to-video tasks (e.g., Latte) and text-to-image tasks (e.g., Stable Diffusion 3 Medium). As shown in Figures 6 and 7, we achieve nearly lossless performance, and achieving latency reduction of SVD by 25%, Latte by 28%, and SD3 by 35%, respectively. Moreover, Figure 6, 7, and 8 visualize the performance of *OmniCache* on SVD, SD3, and Latte, respectively. They show that, at the same compression rates, *OmniCache* produces significantly higher-quality generations compared to TomeSD-based methods.

Method	**Latency and Speedup**			**Evaluation Metrics**	
	$\mathbf{S_{DiT}} \uparrow$	$\mathbf{L_{DiT}}$ **(ms)**	**Retrain**	**FID** $\downarrow$	**CLIP** $\uparrow$
SD3-medium	1.00	138.36	✗	31.57	18.60
Omnimerge (cache) 40%	1.20	114.96	✗	31.42	20.26
Omnimerge (cache) 50%	1.27	109.00	✗	31.94	20.32
Omnimerge (cache) 60%	1.35	102.88	✗	32.45	20.38
Layer-cache	1.75	79.00	✗	31.36	20.11
Omnimerge (cache)+Layer-cache 40%	2.28	60.55	✗	31.14	20.28
Omnimerge (cache)+Layer-cache 50%	2.32	59.71	✗	31.95	20.30

Table 2: **Quantitative comparison of text-to-image generation** on MS-COCO2014 with SD3-medium. DiT latency is reported in milliseconds.

Quantitative Metrics for SVD. For the *OmniCache* experiments, we adopt the best configuration of 50% r_f, 40% r_b, and 50% r_t. As shown in Table 3, *OmniCache* achieves nearly identical FVD metrics to SVD while reducing latency by 25% compared to the ground-truth (non-caching) baseline, and importantly, it does so in a training-free manner.

For ToMeSD, which performs single-dimension token merging, we also select its best setting of 40% token merging to ensure a fair comparison. Our results show that *OmniCache* not only preserves FVD scores more effectively but also delivers an additional 11% speedup. This indicates that *OmniCache* has a greater

Method	Ratio (%)	FVD ↓	s/im ↓
Ground Truth	0	502.68	110
Token Merge (ToMeSD)	40	503.8	92.6
OmniCache	/	503.12	82.4
Frame Cache	20	489.14	105
	40	495.89	99.3
	60	535.07	94.8
Token Cache	20	497.49	102
	40	502.5	94.3
	60	517.64	84
Block Cache	10	499.84	107
	30	502.71	103
	50	528.24	98.8

Table 3: Quantitative evaluation for different caching ratios for SVD.

capacity to maintain video quality while enhancing inference efficiency, compared to existing token-merging approaches.

Spatially Compressing Temporal and Temporally Compressing Spatial Layers Minimizing Information Loss. As shown in Figure 1, we hypothesize that compressing the spatial dimension in the temporal layer and the temporal dimension in the spatial layer results in less information loss. We conducted experiments to test this hypothesis by swapping the positions of frame caching (caching 30% of frames) and block caching (caching 30% of blocks). As depicted in Figure 9, the results of Spatially compressing Temporal layer (S2T) and Temporally compressing Spatial layer (T2S) closely resemble the original SVD results. However, Spatially compressing the Spatial layer (S2S) and Temporally compressing the Temporal layer (T2T) resulted in a noticeable decrease in frame rate, as well as the appearance of blurriness and artifacts.

***OmniCache* Enables More Efficient Feature Compression.** We also aim to highlight the strength of our contributions in feature compression. Under the appropriate S2T and T2S compression strategies, *OmniCache* leverages three dimensions of caching, Frame Cache, Block Cache, and Token Cache, to effectively minimize information loss. To demonstrate the effectiveness of *OmniCache*, we compared its three caching operations with direct interpolation methods at a 40% compression ratio, as shown in Figure 10. The results of Frame Interpolation (FI) show a significant drop in frame rate, almost to the point of stalling. Even when the temporal dimension is compressed in the Spatial layer, direct interpolation still results in considerable information loss. Block Interpolation (BI) yields very blurry results, with direct Spatial interpolation in the temporal layer leading to a noticeable loss of motion information, causing incorrect frame-to-frame information transmission. Similarly, Token Interpolation (TI) used to compress intra-frame information in the Spatial layer also results in blurred outcomes. In contrast, all three of our caching methods yield results that remain very close to the original SVD. The carefully designed *OmniCache* effectively exploits video redundancy to enhance efficiency while preserving quality.

4.4 Ablation

Analysis of Different Caching Compression Ratios. As we analyze in Table 3, the effects of frame caching, block caching, and token caching are quite interesting. We observe that within 40% for frame caching, 40% for token caching, and 30% for block caching, there is no decrease in the FVD scores; in fact, there is even a slight improvement. This demonstrates the effectiveness of our method in removing video redundancy.

Analysis of Speedup Ratios & Output Quality with Layered Caching. To analyze the effectiveness of our Layered Caching algorithm, we set up a caching interval $N = 5$ and computed cache schedules for

various speedups for Stable-Diffusion-3. For baseline, we used DeepCache's approach with caching intervals ($N = 2, 3$). We measured the model's end-to-end speedup for computational efficiency while perceptual (FID,CLIP) and Pixel-wise (LPIPS, SSIM, PSNR) for output quality by generating 2000 images using random image-text pairs from MS-COCO 2017 dataset.

From our analysis, we can observe that for same speedups, our approach outperforms baseline's performance. Moreover, with increasing speedup values, the model's output quality remains quite close to the original model's performance.

Method	**Speedup**	**Perceptual**		**Pixel-wise**		
		FID ↓	**CLIP ↑**	**LPIPS ↓**	**SSIM ↑**	**PSNR ↑**
SD3 (w/o caching)	1.000x	39.187	32.169	0.000	1.000	inf
DeepCache (int=2)	1.8928x	39.0282	–	–	–	–
DeepCache (int=3)	2.5552x	39.6342	–	–	–	–
Our Method	1.645x	37.546	32.096	0.329	0.634	14.801
	1.811x	37.222	32.250	0.279	0.681	16.333
	1.889x	36.598	32.027	0.367	0.597	13.955
	1.969x	36.852	31.993	0.367	0.598	13.968
	2.184x	35.353	31.676	0.363	0.596	14.680
	2.348x	35.560	31.579	0.362	0.596	14.617
	2.506x	35.619	31.676	0.346	0.615	15.053
	2.551x	35.979	31.511	0.364	0.596	14.672
	2.655x	35.991	31.207	0.374	0.584	14.626
	2.778x	36.202	31.178	0.376	0.583	14.597

Table 4: Performance metrics of Stable-Diffusion-3 for different speedups. Image metrics are computed using 2000 random image–text pairs from the MS-COCO 2017 dataset.

5 Limitations

While our work has successfully endeavored to enhance the efficiency of video diffusion models by eliminating redundancy in a training-free setting, the analysis of redundancy we present holds potential value in the training and fine-tuning processes as well. Techniques akin to Progressive Distillation Salimans & Ho (2022) may be synergistically combined with the approaches outlined in this paper to further amplify the achievable compression rates.

Our work introduces four granular levels of caching strategies, namely Frame Cache, Block Cache, Token Cache, and Layered Cache. The essence of *OmniCache* lies in determining the ratios of these four types of caching at each layer. Although our analysis of spatially compressing the temporal dimension and temporally compressing the spatial dimension has improved compression rates, we acknowledge that an attentive determination of caching ratios at each layer could lead to further enhancements in performance.

6 Conclusion

In summary, *OmniCache* presents a novel and effective approach to addressing the computational inefficiencies inherent in the video generation process by targeting four specific types of redundancies: intra-frame, inter-frame, motion, and step redundancies. By introducing a suite of feature caching techniques, Frame Cache, Block Cache, Token Cache, and Layered Cache, *OmniCache* allows for significant improvements in the efficiency of video diffusion models without the need for retraining.

Our experimental evaluations demonstrate the robustness and versatility of *OmniCache*. On state-of-the-art open-source diffusion models such as SVD-XT, Latte, and SD3, we have successfully validated the performance enhancements offered by *OmniCache*. With our approach, users can achieve a training-free increase in inference speed by 25% for SVD-XT, 28% for Latte, and 35% for SD3, all while maintaining the original quality of the generated content.

References

Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023.

Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. *CVPR Workshop on Efficient Deep Learning for Computer Vision*, 2023.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations*, 2023.

Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024. URL `https://openai.com/research/video-generation-models-as-world-simulators`.

Haoxin Chen, Yong Zhang, Xiaodong Cun, Menghan Xia, Xintao Wang, Chao Weng, and Ying Shan. Videocrafter2: Overcoming data limitations for high-quality video diffusion models, 2024a.

Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. δ-dit: A training-free acceleration method tailored for diffusion transformers, 2024b. URL `https://arxiv.org/abs/2406.01125`.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis, 2024. URL `https://arxiv.org/abs/2403.03206`.

Yuwei Guo, Ceyuan Yang, Anyi Rao, Zhengyang Liang, Yaohui Wang, Yu Qiao, Maneesh Agrawala, Dahua Lin, and Bo Dai. Animatediff: Animate your personalized text-to-image diffusion models without specific tuning. *International Conference on Learning Representations*, 2024.

Zhaoyuan He, Yifan Yang, Lili Qiu, Kyoungjun Park, and Yuqing Yang. Nerve: Real-time neural video recovery and enhancement on mobile devices. *Proc. ACM Netw.*, 2(CoNEXT1), mar 2024. doi: 10.1145/3649472. URL `https://doi.org/10.1145/3649472`.

Byeongho Heo, Song Park, Dongyoon Han, and Sangdoo Yun. Rotary position embedding for vision transformer, 2024. URL `https://arxiv.org/abs/2403.13298`.

Hikvision. H.264 and H.265 Recommended Bit Rate at General Resolutions, 2024. URL `https://www.hikvision.com/content/dam/hikvision/ca/faq-document/H.2645-&-H.2645-Recommended-Bit-Rate-at-General-Resolutions.pdf`. [Online; accessed 18-May-2024].

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models, 2022. URL `https://arxiv.org/abs/2206.00364`.

Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang, Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xformers: A modular and hackable transformer modelling library. `https://github.com/facebookresearch/xformers`, 2022.

Xirui Li, Chao Ma, Xiaokang Yang, and Ming-Hsuan Yang. Vidtome: Video token merging for zero-shot video editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps, 2022. URL https://arxiv.org/abs/2206.00927.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, June 2025. ISSN 2731-5398. doi: 10.1007/s11633-025-1562-4. URL http://dx.doi.org/10.1007/s11633-025-1562-4.

Xin Ma, Yaohui Wang, Gengyun Jia, Xinyuan Chen, Ziwei Liu, Yuan-Fang Li, Cunjian Chen, and Yu Qiao. Latte: Latent diffusion transformer for video generation. *arXiv preprint arXiv:2401.03048*, 2024a.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free, 2023. URL https://arxiv.org/abs/2312.00858.

Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. *Advances in Neural Information Processing Systems*, 37:133282–133304, 2024b.

Haomiao Ni, Changhao Shi, Kai Li, Sharon X Huang, and Martin Renqiang Min. Conditional image-to-video generation with latent flow diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18444–18455, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Ethan Smith, Nayan Saxena, and Aninda Saha. Todo: Token downsampling for efficient generation of high-resolution images. *arXiv preprint arXiv:2402.13573*, 2024.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

HPC-AI Tech. Open-sora: Democratizing efficient video production for all, 2023. URL https://github.com/hpcaitech/Open-Sora/tree/main. Accessed: 2024-05-21.

Vikram Voleti, Chun-Han Yao, Mark Boss, Adam Letts, David Pankratz, Dmitry Tochilkin, Christian Laforte, Robin Rombach, and Varun Jampani. Sv3d: Novel multi-view synthesis and 3d generation from a single image using latent video diffusion, 2024. URL https://arxiv.org/abs/2403.12008.

Yanhui Wang, Jianmin Bao, Wenming Weng, Ruoyu Feng, Dacheng Yin, Tao Yang, Jingxu Zhang, Qi Dai Zhiyuan Zhao, Chunyu Wang, Kai Qiu, et al. Microcinema: A divide-and-conquer approach for text-to-video generation. *arXiv preprint arXiv:2311.18829*, 2023.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, et al. Cache me if you can: Accelerating diffusion models through block caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6211–6220, 2024.

Sihyun Yu, Weili Nie, De-An Huang, Boyi Li, Jinwoo Shin, and Anima Anandkumar. Efficient video diffusion models via content-frame motion-latent decomposition. *ICLR 2024*, abs/2403.14148, 2024. URL https://api.semanticscholar.org/CorpusID:268554134.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching, 2025. URL https://arxiv.org/abs/2410.05317.

Appendix

A Performance Evaluation of Triton Kernels

To further analyze the performance of our custom Triton kernels, we compare them against the baseline PyTorch implementation on the SD3-medium model under varying number of token groups $(1, 2, 4, 8)$ configurations in the hierarchical caching pipeline. Moreover, as described in Section 4.2, we omitted the inital and final 4 transformer blocks as they are sensitive to hierarchical caching which in turns impacts the generated image quality. As shown in Figure 11, the Triton-based kernels consistently outperform the PyTorch counterparts achieving $> 2.0\times$ speedup for merge and unmerge routines, demonstrating superior scalability and reduced latency as the number of token groups increases.

B Cache Strategy Refinement Algorithm

For completeness, we include the full pseudocode of the cache strategy refinement procedure introduced in Section 3.5. This algorithm refines the preliminary cache schedule proposed in Algorithm 1 by pruning redundant cache steps, realigning dependencies, and optimizing the caching intervals for computational efficiency.

Algorithm 3 Cache Strategy Refinement Algorithm

```
Require: Model Compute Graph G, CacheStrategy C
  Preprocess:
  for each node in cachelist do
      Initialize metadata (children, parents, flags, counters) for the node
  end for
  Build Dependency Graph:
  for each node do
      Add child and parent relationships based on model structure
  end for
  Calculate Dependency Counters:
  for each node do
      Set child_num and parent_num based on relationships
  end for
  Remove Redundant Cache Steps:
  for each node in topological order do
      Use flags to identify and remove unnecessary cache steps
      Update flags for child nodes
  end for
  Remove Consecutive Cache Steps with Identical Outputs:
  for each eligible node in reverse topological order do
      for each pair of consecutive cache steps do
          if outputs are identical then
              Remove the redundant cache step
          end if
      end for
  end for
  Recalculate Dependency Counters
  Align Cache Steps with Parent Dependencies:
  for each node in topological order do
      for each cache step not in interval do
          Update to minimum parent cache step ≥ current step
      end for
  end for
  Return Optimized CacheStrategy Ĉ
```

This refinement process enforces structural consistency within the model's compute graph, eliminating redundant computations while maintaining temporal and dependency integrity across cached layers.

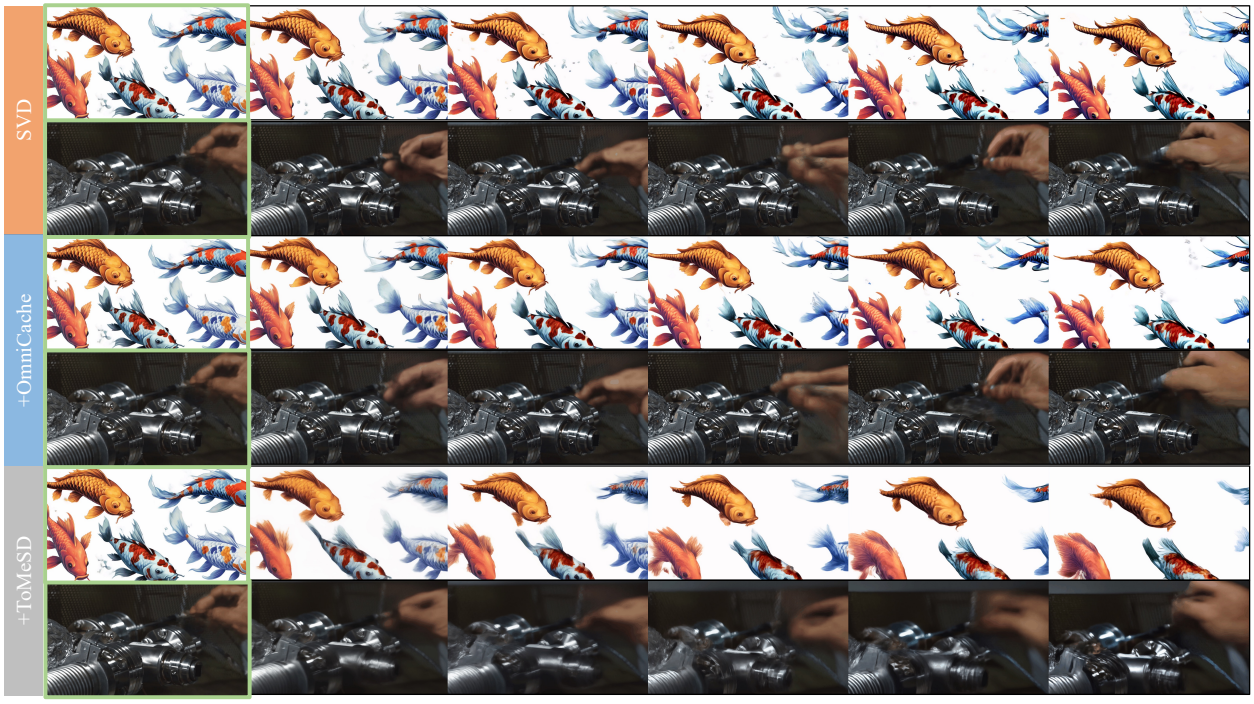

Figure 6: Performance of *OmniCache* on SVD. With r_f, r_b, r_t set to (50%, 30%, 50%), compared against TomeSD with $r_t = 65\%$ at the same compression rate. We achieve similar performance to the original SVD while improving inference time by 25%.

Figure 7: Visualization examples for *OmniCache*, i.e., hierarchical token caching (with use of token cache for unmerging) for merge ratios $(40\%, 50\%, 60\%)$, layer-wise caching, and the integration of both approaches. Prompts used to generate these images are provided in appendix

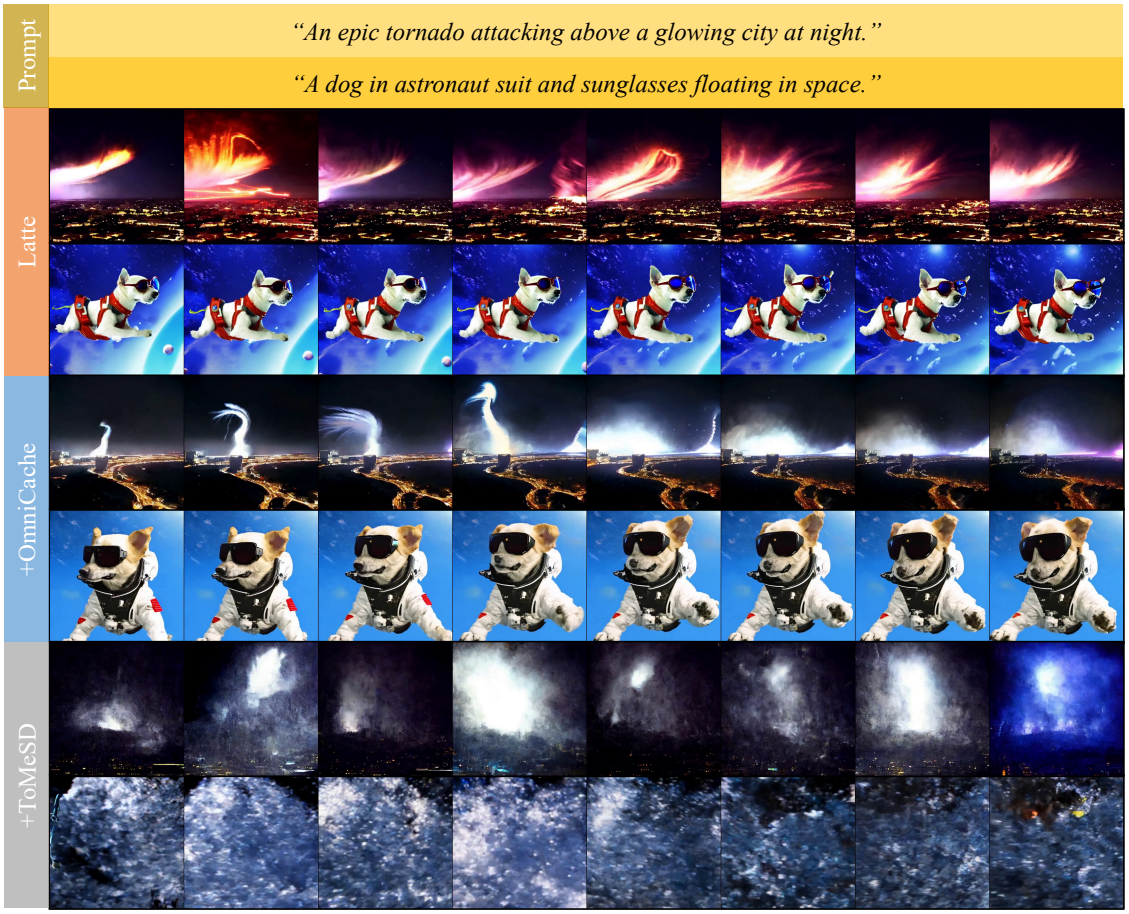

Figure 8: Performance of *OmniCache* on Latte. With r_f, r_b, r_t set to (30%, 40%, 20%), compared against ToMeSD with $r_t = 25\%$ at the same compression rate. We achieve similar performance to the original Latte while improving inference time by 28%.

Figure 9: *OmniCache* proposes Spatially compress temporal (S2T) and temporally compress spatial (T2S) vs. Spatially compress temporal (S2S) and temporally compress spatial (T2T). r_f, r_b, r_t are set to (30%, 30%, 0%). To ensure a fair comparison, Token Merge is not included since it can only be applied in the spatial layer and is irrelevant to this issue. Experiments demonstrate that our proposed S2T and T2S retain more information.

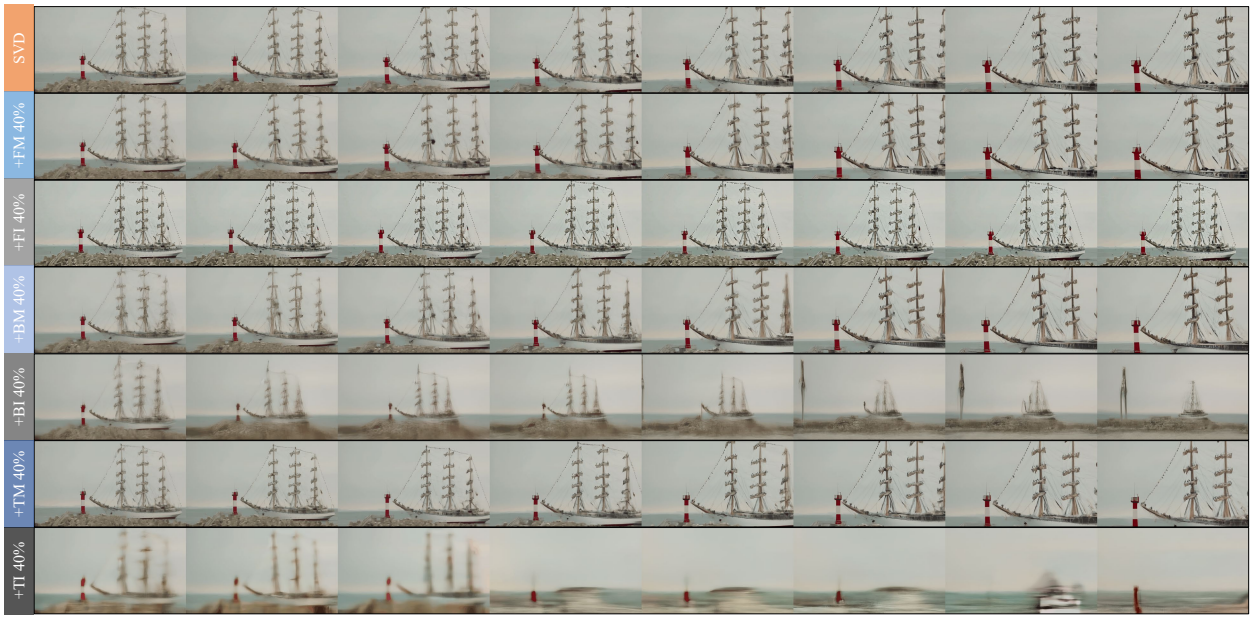

Figure 10: Comparing Frame Cache, Block Cache (BM), and Token Cache (TM) with frame interpolation (FI), block interpolation (BI), and token interpolation (TI). To ensure fairness, all compression ratios are set to 40%. We conduct experiments on the image-to-video model of SVD.

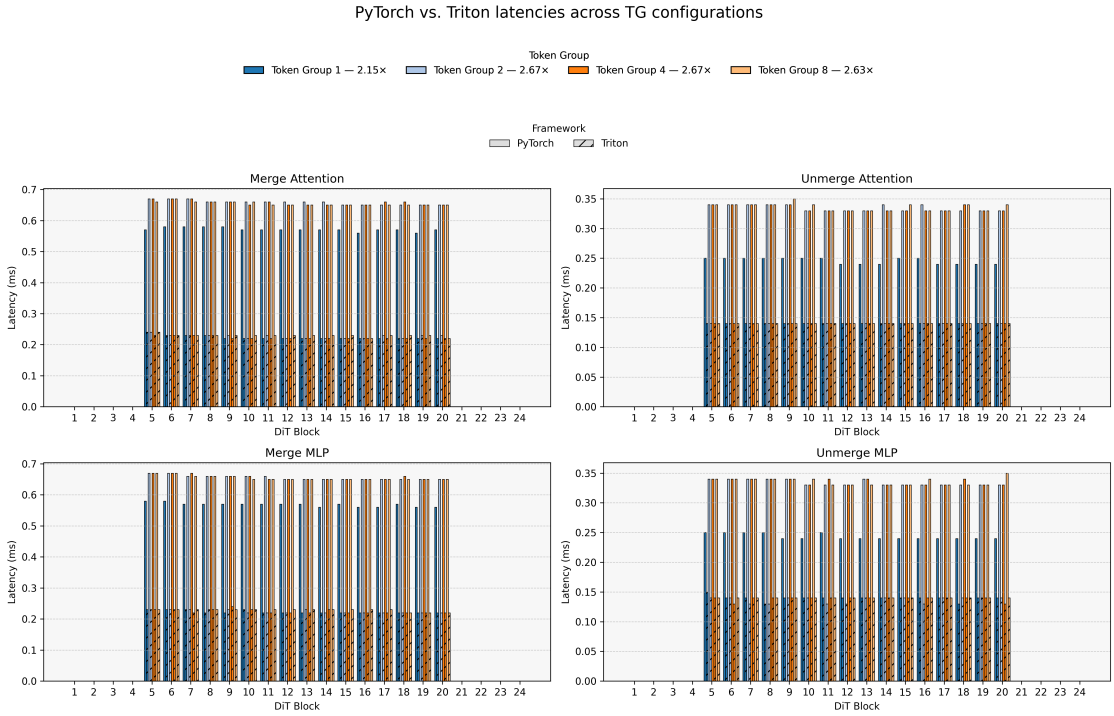

Figure 11: Comparison of our custom Triton kernel implementation against the PyTorch baseline on SD3-medium under different token group sizes in hierarchical caching. The Triton kernels achieve higher throughput and lower latency due to optimized memory access and fused operations. Average speedups for different token groups is provided alongside the Token Group legend