RADiT: Redundancy-Aware Diffusion Transformer Acceleration Leveraging Timestep Similarity

Youngjun Park†, Sangyeon Kim‡, Yeonggeon kim‡, Gisan Ji‡, and Sungju Ryu*
†Department of Semiconductor Engineering, ‡Department of Electronic Engineering,
*Department of System Semiconductor Engineering,
Sogang University, Seoul, Republic of Korea
{youngjun, sangyeonkim, yeonggeonkim, gisanji, sungju}@sogang.ac.kr

2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11133190

***Abstract*—Diffusion Transformers (DiTs) have demonstrated unprecedented performance across various generative tasks including image and video generation. However, a large amount of computations on the inference process and iterative sampling steps in the DiT models result in high computational costs, leading to substantial latency and energy consumption challenges. To address these issues, we propose a redundancy-aware DiT (RADiT), a novel software-hardware co-optimization accelerator for DiTs that minimizes redundant operations in the iterative sampling stages. We identify data redundancy by evaluating blockwise input features and skip redundant computations by reusing results from consecutive timesteps.**

Furthermore, to minimize accuracy degradation and maximize computational efficiency, the Dynamic Threshold Scaling Module (DTSM) and Compress and Compare Unit (CCU) are employed in the redundancy detection process. This approach enables DiTs to achieve up to 1.8× and 1.7× faster speeds for image and video generation, respectively, without compromising quality, along with 41% and 45.5% reductions in energy consumption. Our RADiT scheme improves throughput by 1.67× and 1.76× for image and video generation tasks, respectively, while maintaining output quality and significantly reducing energy consumption.

***Index Terms*—Diffusion model, transformer, neural network, timestep similarity, hardware accelerator.**

I. Introduction

Diffusion models [1], [2] have achieved significant advancements in generative models and are widely adopted in various tasks including image [3], text-to-image [4], and video generation [5], [6]. These models generate high-quality samples by progressively denoising data during the inference process. Diffusion models have advantages over traditional generative models, such as generative adversarial networks (GANs) [7] and variational autoencoders (VAEs) [8] in terms of stable training and high sample quality, thereby leading to active research and increased attention in the generative models.

Recently, diffusion transformers (DiTs) [9] such as OpenAI's Sora [10] utilize transformers [11] as a backbone by integrating transformer-based architectures into diffusion models to enable more sophisticated and complex generative tasks instead of convolutional approaches. However, the computational complexity of DiT models is much larger than previous U-net based models [1], [4], [12], requiring substantial processing costs for large-scale data, and hence the real-time computation is one of the most important challenges. Recent studies on accelerating diffusion models [13]–[16] focus on leveraging the similarity in iterative sampling processes. However, conventional approaches have been targeting U-net based diffusion models and not well-suited for emerging DiT architectures. To overcome such challenges, we propose a software-hardware co-design method called RADiT which leverages timestep similarity in the DiT neural networks. Our RADiT first evaluates the block-level redundant computation on the input features in the consecutive timesteps. For the similar output features in the consecutive timesteps, we only compute once and reuse the result of the previous timestep for the following timestep computation. However, there is still a concern for the accuracy drop with such an approximate computing method. To minimize the accuracy drop, dynamic threshold scaling module (DTSM) flexibly adjusts the redundancy threshold criteria for every input data in real time. Furthermore, we design the compress and compare unit (CCU) to reduce the computing overhead of timestep similarity using low-bit compression of the feature map buffer. Our main contributions are as follows.

1) We analyze the redundancy in block-level operations across consecutive timesteps in DiTs and propose a redundancy-aware algorithm to skip the redundant computations.
2) To dynamically adjust the thresholding level for the evaluation of the timestep similarity, We propose a dynamic threshold scaling which efficiently accesses the redundancy across various timesteps, transformer blocks, and input data.
3) We introduce a compress and compare scheme to enable fast redundancy detection and minimize the computing overhead with low-bit data.
4) We finally propose a novel hardware architecture, RADiT, optimized with the proposed redundancy detection algorithm to accelerate DiTs effectively.

II. Background and Motivation

A. Diffusion Transformers

The diffusion model's fundamental idea is to gradually add noise to the original image. Then, the original image can be sampled by removing the noise step by step. Adding noise to

*Corresponding Author

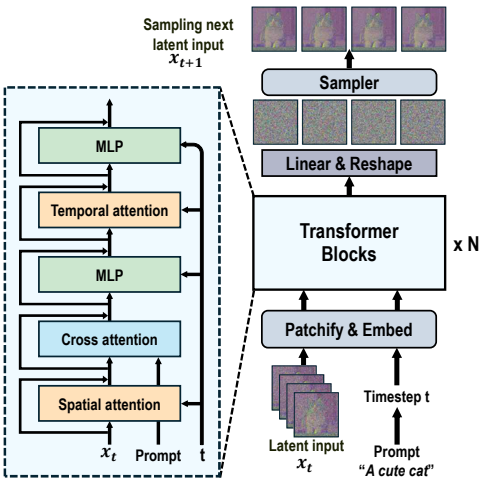

Fig. 1. DiT architecture for image and video generation.

the original image x_0 over T timesteps is called the forward process. The forward process is defined as follows.

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right) \quad (1)$$

Where β_t is a variable that controls the noise, increasing as t approaches T. Conversely, removing the added noise step by step is called the reverse process. The reverse process is defined as follows.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right) \quad (2)$$

μ_θ and Σ_θ represent the mean and variance of the predicted noise as estimated by the model. Previously, a U-Net-based architecture was adopted as the model for predicting noise. However, transformer-based architecture DiT has emerged, achieving state-of-the-art performance across various generative models.

Fig. 1 illustrates the DiT architecture for video generation [17]. DiT comprises three main components: an embedding module for input data, timesteps, and text prompts; transformer blocks for noise prediction at specific timesteps; and a sampler that generates the latent input for the next timestep based on the predicted noise distribution. The transformer blocks are composed of attention blocks and multi-layer perceptrons (MLPs) to capture relationships between tokens embedded at the patch level. Through transformer blocks, DiT can capture global relationships in the data, which helps it recognize overall patterns. However, since DiT uses an attention mechanism, it has higher computational complexity and memory requirements compared to U-Net-based models. Therefore, we have focused on reducing computational load and accelerating the processing speed of DiT.

B. Observations on the Redundancy of DiT

The most important characteristic of the diffusion model is that it iterates the model over multiple timesteps. In other words, because it removes noise gradually, we predicted that features in the same block would be similar across adjacent timesteps. In this work, we analyzed whether this similarity exists during the inference process of DiT. We examined how much the input features of each block differs from (similar to) the input of previous timestep at all timesteps. The results are shown in Fig. 2a. Additionally, we analyzed the attention scores of two blocks at adjacent timesteps with similar input features (Fig. 2b). From this analysis, we observed the following characteristics. 1) During the inference process of the DiT models, redundancy in the input of blocks appears across adjacent timesteps. 2) This redundancy varies depending on the block and timestep, as well as on the characteristics of the input data. 3) Attention/MLP blocks with similar input tend to have similar output.

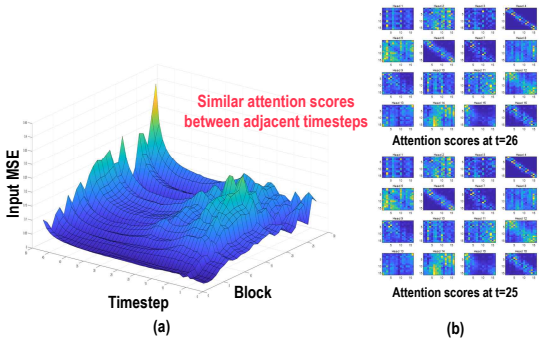

Fig. 2. The results of analyzing the redundancy inherent in DiTs. (a) Input MSE of transformer blocks over timestep. (b) Attention scores across adjacent timesteps.

Based on these observations, we aim to accelerate DiT-based models by skipping computation where each block's input is similar to that of the previous timestep and by reusing the results from the previous timestep. However, simply skipping and reusing in regions with frequent similar inputs significantly degrades model performance. This degradation is caused by errors from reusing previous results, which are amplified by the residual connections in DiT as the inference progresses. To mitigate such an accuracy drop, it is necessary to check the similarity of each block and verify if there are any errors caused by the reuse of the result data. As a result, we introduce software-hardware co-optimization approach called RADiT which will be explained in the following Section III.

III. Proposed Work: RADiT

In this section, we introduce our proposed method called RADiT, software-hardware co-design approach. First, we provide an overview of the overall operation of RADiT in Section III-A. Next, we explain the detailed methods employed by RADiT for redundancy-aware network compression in Section III-B. Finally, in Section III-C, we describe the overall hardware architectural design of RADiT.

A. Design Overview of RADiT

Fig. 3 illustrates the overall operation of RADiT. The main concept of RADiT is to reduce redundant computations during iterative sampling by analyzing the input to a DNN operation block such as self-attention, cross-attention, and MLPs. If the input is considered similar to the previous timestep, the computation is skipped, and the output from the block in the previous timestep is reused. This approach minimizes redundant operations and accelerates the sampling process. To enable this functionality, RADiT satisfies the following two requirements. First, we need to set a threshold to determine

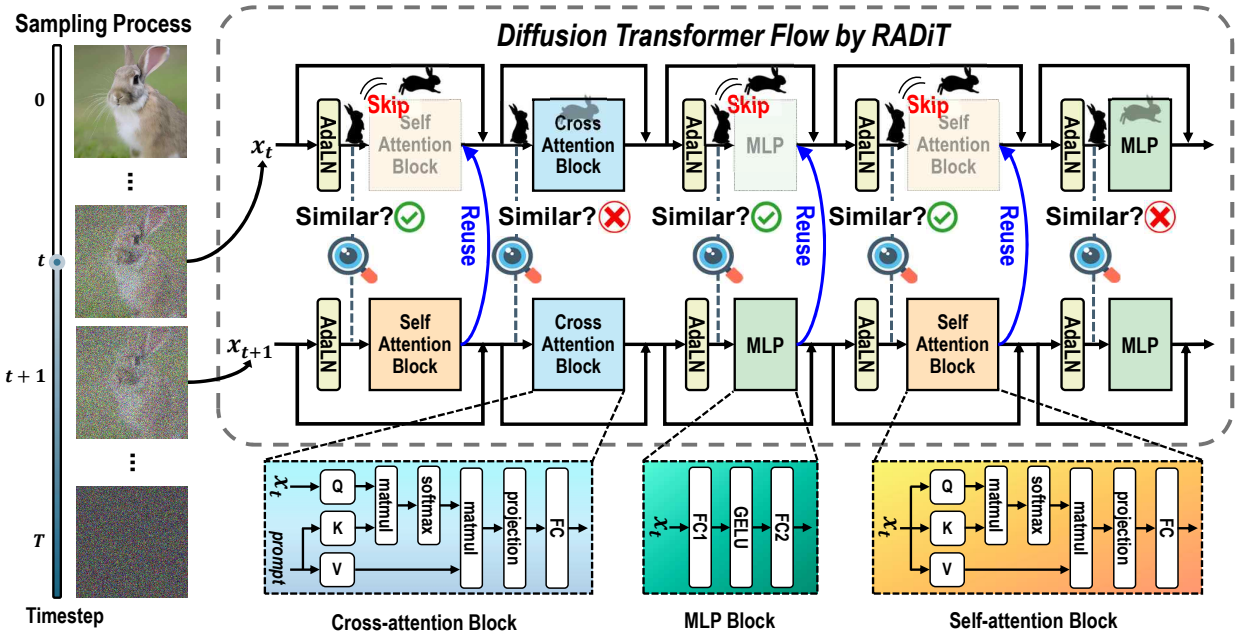

Fig. 3. An overview of the timestep similarity-based redundancy-aware DiT inference process with RADiT.

whether the input is similar to the previous timestep. As shown in Fig. 2a, differences in input across adjacent timesteps vary for every block, and these differences change dynamically during each inference depending on the input data. Therefore, it is crucial to dynamically adjust the similarity threshold for each block during inference to ensure consistent samples that closely resemble the original. This aspect will be explained in more detail in Section III-B2. Second, the comparison between the current and previous timestep inputs must be fast enough. This comparison occurs during the DNN operation at each timestep of the diffusion models. If the latency of the comparison process is significantly large, the overhead may outweigh the benefits of skipping computations. Therefore, directly loading raw previous timestep inputs and calculating differences using metrics like mean squared error (MSE) is not practical. Instead, RADiT employs a method that first compresses the data bits, and then it detects differences by comparing the compressed lower bits. This approach reduces the size of the previous timestep data that needs to be loaded from memory by 4× for FP16. Further details on this mechanism are provided in Section III-B3.

B. Redundancy Aware Network Compression

This Section introduces our various techniques for compressing diffusion models through the redundancy-aware algorithm optimized by RADiT.

1) Redundancy-Aware Algorithm: We analyze the redundancy in the DiT models and propose a redundancy-aware algorithm (Algorithm 1) to minimize redundant computations.

Let the input to the n-th DNN operation block at an arbitrary timestep t be denoted as x_t^n and its output as $F(x_t^n)$, and please note that the denoising process of the diffusion model starts from timestep T to 0. If the input x_{t+1}^n to the same n-th block at the previous timestep $t+1$ is similar to x_t^n, we can approximate $F(x_t^n)$ to $F^*(x_t^n) = F(x_{t+1}^n)$. A key aspect to note here is that we skip not individual operations like matrix multiplication but entire blocks, such as attention and MLP blocks. Thus, the redundancy-aware algorithm significantly reduces latency by skipping DNN computation, weight loading required for computation, memory read/write operation for activation and attention scores, and additional functions like reshaping and normalization.

Algorithm 1: Redundancy-Aware Computing

Input: Total timesteps T, Depth of block N, Block input x_t^n, Block threshold θ^n, DNN operation block $F(\cdot)$, Timestep difference function $f(\cdot)$

Output: Updated outputs $F(x_t^n)$ for all t and n

```
for t ← T to 1 do
    for n ← 1 to N do
        Δ(x_t^n) ← f(x_{t+1}^n, x_t^n);
        if Δ(x_t^n) < θ^n then
            Reuse: F(x_t^n) ← F*(x_t^n) = F(x_{t+1}^n);
        else
            Compute: F(x_t^n) ← F(x_t^n);
        end
    end
end
```

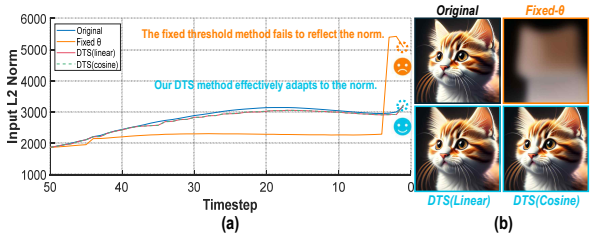

Fig. 4. (a) Changes in input L2 norm across timesteps. (b) The image sampling results by each method. θ is threshold value.

2) Dynamic Threshold Scaling Method: Reusing computation results from the previous block is a highly effective method for reducing latency, but it introduces errors in similarity evaluation, which worsen with increasing block depth.

Algorithm 2: Dynamic Threshold Update for Timestep Differences

Input: Total timesteps T, Depth of blocks N, Timestep differences Δx_t^n, Parameter α_t
Output: Updated thresholds θ^n for all n

```
for t ← T down to 1 do
    for n ← 1 to N do
        if θ^n is None then
            Initialize: θ^n ← Δx_t^n;
        else
            Δ(Δx_t^n) ← Δx_t^n − Δx_{t+1}^n;
            if Δ(Δx_t^n) > 0 and Δ(Δx_{t+1}^n) < 0 then
                Update: θ^n ← Δx_{t+1}^n + α_t · Δ(Δx_t^n);
            else
                Keep: θ^n;
            end
        end
    end
end
```

Therefore, using a fixed threshold exacerbates this issue in later blocks (Fig. 4), leading to excessive reuse and compounding errors. To address this, we propose the dynamic threshold scaling (DTS) method, as described in Algorithm 2.

For subsequent timesteps, the algorithm calculates the change in timestep difference, $\Delta(\Delta x_t^n) = \Delta x_t^n - \Delta x_{t+1}^n$. If the change indicates a significant variation ($\Delta(\Delta x_t^n) > 0$) and the previous difference is negative (($\Delta(\Delta x_{t+1}^n) < 0$), (this condition identifies a local minimum in Δx_t^n) the threshold is updated.

By using DTS, we can mitigate errors caused by reuse of block output. If significant reuse-induced errors occur in the previous step, the threshold is lowered, enforcing stricter similarity criteria in the subsequent timestep. Conversely, if errors are small, the threshold is increased, allowing for more aggressive data reuse. The parameter α_t in the DTS algorithm starts at 0 and increases to 1 as the timestep t decreases from T to 0. This design is inspired by β_t (Equation 1), which determines the amount of noise added during the forward process. At the initial timestep ($t = 0$), less noise is added, and as t approaches T, the noise decreases further. Similarly, during the inference (reverse process), redundancy is generally lower at the beginning and increases toward the end as in our analysis (Fig. 4). Based on this observation, we define α_t using two approaches: linear scheduling (Equation 3) and cosine scheduling (Equation 4).

$$\alpha_t = \frac{t - T}{T} \tag{3}$$

$$\alpha_t = \frac{1}{2}\left(1 - \cos\pi\left(\frac{T - t}{T}\right)\right) \tag{4}$$

Fig. 5a shows the timestep difference when similarity is computed using a **fixed threshold** in the later blocks. Due to the reuse of previous block results with the fixed threshold, the step difference becomes small in most timesteps, leading to excessive reuse. However, as shown in Fig. 4, this results in a significant loss of original consistency, causing a severe degradation in the accuracy.

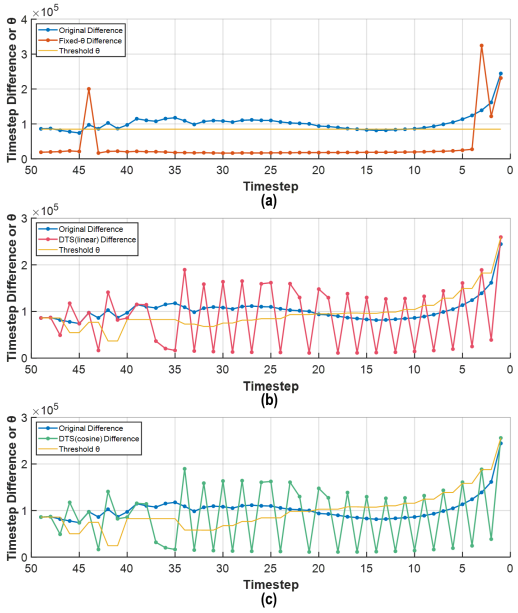

Fig. 5. Variation in differences of block outputs between adjacent timesteps. Original and reused results using (a) fixed threshold, (b) linear scheduling, and (c) cosine scheduling. θ is threshold.

On the other hand, Figs. 5b-c illustrate the timestep difference of the inputs when each of the two **dynamic threshold** scaling method is applied depending on the input similarity. These dynamic thresholding approaches track the original difference better than the fixed thresholding method. Consistently, Fig. 4 demonstrates that the DTS method preserves the original consistency, allowing for faster inference without significantly compromising model performance.

3) Efficient Bit Compression and Comparing: To apply the algorithms discussed so far, the input x_{t+1}^n of the previous timestep must be read before each DNN operation, leading to a latency overhead. To minimize this overhead, we propose an effective compression method for DiTs, which reduces input bits to 4 bits while maintaining comparability.

The adaptive layer normalization [18] in front of each DNN operation block of DiTs (Fig. 3) normalizes the input data and scales and shifts it based on the current timestep information. Due to the normalization layers, our evaluation (Fig. 6a) shows that input activation values range from -16-to-16. Based on the analysis, we propose an efficient compression method tailored to the distribution of input values.

Fig. 6b illustrates the proposed compression method. Assuming the input data is in FP16 format as baseline DiT used, the first and second bits of the exponent field are used as indicator bits. Based on these indicator bits, the 16-bit data is compressed into 4 bits using one of the following three cases: **Case 1 (Indicator Bits = 00):** When the indicator bits are 00, the compressed result is derived from input_data$[14:12]$, and the most significant bit (MSB) is set to 0. This case uses 2 compressed levels: 2^{-11}-to-2^{-7} are mapped to 0001, and values below 2^{-11} are mapped to 0000. **Case 2 (Indicator Bits = 01):** When the indicator bits are 01, the compressed result is

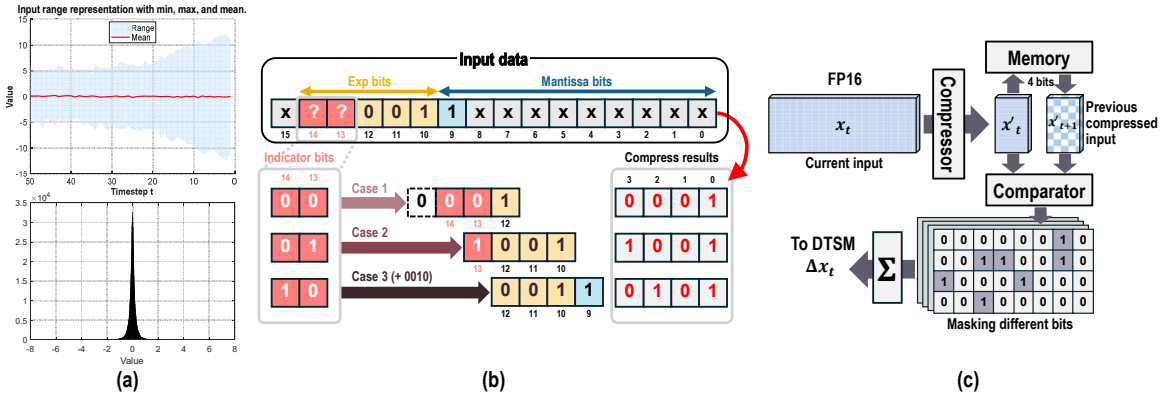

Fig. 6. Input compression method and comparing flow. (a) The range and distribution of input after adaptive layer normalization. (b) A method to compress input by dividing cases based on indicator bits. (c) A description of the process to compress the input, perform a comparison, and estimate the difference.

derived from input_data[13 : 10]. This case uses 8 compressed levels: 2^{-7}-to-2^{-6}, 2^{-6}-to-2^{-5}, $\cdots$, 2^{0}-to-2^{1} are mapped to 1000, 1001, $\cdots$, 1111, respectively. **Case 3 (Indicator Bits = 10):** When the indicator bits are 10, the compressed result is derived from input_data[12 : 9]+0010, where the input_data[9] is MSB of the mantissa bits. Therefore, this case uses 6 compressed levels: 2^{1}-to-3, 3-to-2^{2}, 2^{2}-to-6, 6-to-2^{3}, 2^{3}-to-12, and 12-to-2^{4} are mapped to 0010, 0011, 0100, 0101, 0110, and 0111, respectively. Sign is not considered, as it is rare for the sign to change significantly between adjacent timesteps.

Fig. 6c demonstrates the process of calculating the difference based on the proposed bit compression technique. The input x_t at the current timestep t is compressed into x'_t, and it is compared with the compressed x_{t+1} from the previous timestep. The differing bits are masked as 1, while the same bits are set to 0. By summing all masked 1s, the difference Δx_t between the inputs of two adjacent timesteps is obtained.

C. RADiT Hardware Accelerator Design

Fig. 7 illustrates the architecture of the DiT accelerator, RADiT, which incorporates the methods discussed above. RADiT consists of the following key components: 1) A *dynamic threshold scaling module (DTSM)* that applies the DTS method to appropriately scale the threshold value. 2) A *compress and compare unit (CCU)* that efficiently compresses input data and compares feature similarity between adjacent timesteps. 3) A systolic array [19] consisting of 64×64 processing elements (PEs) for performing DNN operations. These components work together to accelerate DiT operations by achieving speedup, reducing computational overhead, and maintaining

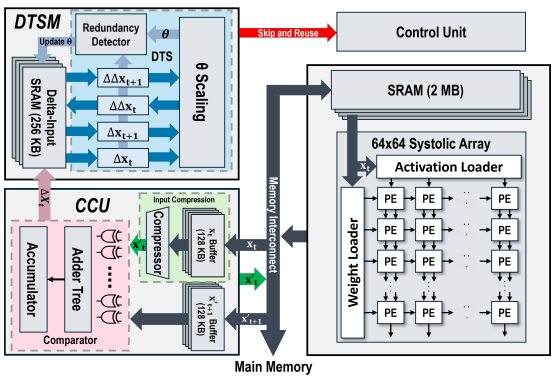

Fig. 7. Hardware architecture of RADiT.

TABLE I
COMPARED TO THE PREVIOUS U-NET-BASED DIFFUSION ACCELERATORS, OUR DESIGN SUPPORTS DIT ACCELERATION AND MULTIPLE TASKS.

Previous Work	Supporting Model	Evaluation Task
ISSCC'24 [15]	U-Net	Image only
VLSI'24 [16]	U-Net	Image only
ISCA'24 [14]	U-Net	Image only
DAC'24 [20]	U-Net	Image only
Ours	Transformer (DiT)	Image and video

model consistency. While the systolic array performs DNN operations for the DiT models, the CCU compresses the current input feature and compares it with the compressed input feature from the previous timestep. The comparison results are passed to the DTSM, which sets the threshold using Algorithm 2. If the similarity between two timesteps calculated by the CCU is smaller than the threshold determined by the DTSM, the redundancy detector in DTSM generates a *skip-and-reuse* signal, which is sent to the control unit. Based on this signal, the control unit skips the current operation and reuses the result from the previous computation. This mechanism enables RADiT to continuously identify redundancy between adjacent timesteps and skip unnecessary computations, thus improving operational efficiency.

IV. RESULTS

A. Experimental Setup

In this section, we present the evaluation results of the RADiT hardware architecture. The image generation capability of RADiT was assessed using the DiT [9] model, while its video generation capability was analyzed with the Latte [21] model. We implemented behavioral models of both the baseline and RADiT hardware accelerators in Verilog HDL. The design was synthesized in a 28nm targeting 500MHz. An LPDDR3 model was used for the DRAM performance evaluation. Previous diffusion hardware accelerators were dedicated for convolution-based DNNs only (Table I), so we compared our performance with vanilla DiT hardware baseline without timestep similarity.

B. Results

1) Area: Fig. 8 provides detailed area breakdown on the RADiT architecture. A large portion of the RADiT area is occupied by the systolic array [19], which is designed for DNN operations and consists of 64×64 processing element (PE) arrays. In the systolic array, the weight and psum(+input)

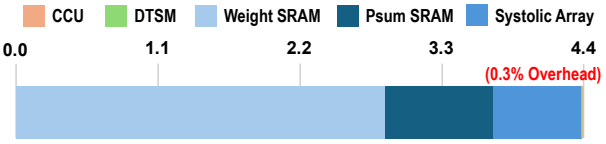

Fig. 8. The area breakdown of the proposed RADiT architecture. CCU and DTSM are overheads compared to the baseline.

SRAMs account for approximately 85% of the total area, while the logic components take up around 15%. The CCU and DTSM responsible for the redundancy aware algorithm require area overhead, but the additional area occupies only 0.3% of the total area, which is negligible.

2) Accuracy: To evaluate the performance of RADiT's redundancy-aware algorithm on the DiTs, we conducted experiments on the following three tasks: image generation, video generation, and text-to-video generation. For image generation, we used the DiT-XL/2 [9] model trained on the ImageNet [22] dataset to generate 50,000 images with a resolution of 256×256 for comparison. We evaluated image quality using widely used metrics such as FID [23] and Inception Score [24]. For video generation, we utilized the Latte [21] model trained on four public datasets-Faceforensics [25], SkyTimelapse [26], UCF101 [27], and Taichi-HD [28]-to generate 2,000 videos with a duration of 2 seconds, resolution of 256×256, and a frame rate of 16 fps. The performance was measured using the FVD [29]. For text-to-video generation, we used the Latte-1 model and evaluated the video quality with VBench [30], which measures the alignment between the generated video and the given prompt. 945 prompts were used to generate 5 videos per prompt, each with a duration of 2 seconds, resolution of 512×512, and a frame rate of 16 fps. The detailed results for each task are presented in Tables II, III, and IV. As a result, RADiT achieved approximately 1.7x Tflops reduction in image generation, 1.8x in video generation, and 1.6x in text-to-video generation, with minimal degradation in quality in most cases. In some cases, our RADiT approach shows better accuracy value than the baseline. We carefully analyze that non-critical redundant features have been eliminated via our redundancy-aware computing, thereby slightly improving the accuracy value in the cases.

3) Latency and Energy: Fig. 9 and Fig. 10 illustrates the normalized latency and energy results for three tasks: image generation, video generation, and text-to-video tasks, evaluated across three models. RADiT achieves significant latency reductions of 40%, 43.1%, and 36.5% (Fig. 9) for the three tasks, respectively, due to its operation-skipping mechanism. The additional DRAM accesses required for comparisons with previous steps introduce an overhead of only 1.5%, which is negligible. So, our design still shows much smaller latency than the baseline thanks to the block-level computational skipping.

TABLE II
256X256 IMAGE GENERATION RESULTS.

Method	FID↓	sFID↓	IS↑	Precision↑	Recall↑	Gflops↓
DiT-XL/2 [9]	2.27	4.60	278.24	0.83	0.57	119
RADiT(linear)	2.57	5.38	264.17	0.81	0.59	70.2
RADiT(cosine)	2.68	5.49	261.92	0.81	0.59	71.4

TABLE III
FVD(↓) OF 256X256 VIDEO GENERATION ON DIFFERENT DATASETS.

Method	FaceForensics	SkyTimelapse	UCF101	Taichi-HD	Gflops↓
Latte [21]	30.69	46.43	150.74	108.14	5572
RADiT(linear)	30.72	47.98	146.75	116.64	3036.7
RADiT(cosine)	31.08	48.68	162.08	120.17	3176.0

TABLE IV
VBENCH SCORES OF 512X512 TEXT-TO-VIDEO GENERATION RESULTS.

Method	Quality score↑	Semantic score↑	Total score↑	Tflops↓
Latte-1 [21]	79.72	67.58	77.29	3439.47
RADiT(linear)	80.29	67.49	77.73	2235.65
RADiT(cosine)	80.47	68.02	77.98	2270.05

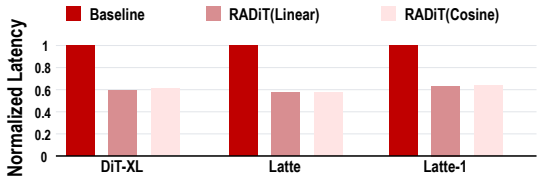

Fig. 9. The comparison of normalized latency between the baseline and RADiT for three DiT-based models.

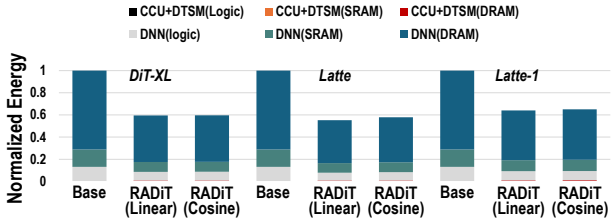

Fig. 10. The comparison of normalized energy consumption between the baseline and RADiT for three DiT-based models.

Energy consumption shows a similar trend (Fig. 10), with reductions of 41%, 45.5%, and 38.4% for the three tasks, respectively, and energy consumption due to the DRAM access is dominant part. These results demonstrate that RADiT effectively reduces inference time while maintaining low overhead, making it a highly efficient solution for latency and energy optimization.

V. CONCLUSION

We proposed RADiT, a redundancy-aware diffusion transformer hardware accelerator. RADiT introduces a software-hardware co-optimization approach that dynamically identifies and skips redundant computations, significantly reducing latency and energy consumption during inference. Our experimental results demonstrated that RADiT achieves substantial speedups of 1.6-1.8×, respectively, with negligible accuracy drop and hardware overhead.

ACKNOWLEDGMENT

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00266, Development of Ultra-Low Power Low-Bit Precision Mixed-mode SRAM PIM, 30%), National Research Foundation (NRF) funded by the Korean government (MSIT) (No. RS-2024-00439520, 30%), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (RS-2025-02263669, Development of server-level DRAM-stacked PIM solution for accelerating ultra-large AI models, 30%), and Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE) (P0017011, HRD Program for Industrial Innovation, 10%). The EDA tool was supported by the IC Design Education Center(IDEC), Korea.

REFERENCES

[1] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[2] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*. pmlr, 2015, pp. 2256–2265.

[3] B. Kawar, M. Elad, S. Ermon, and J. Song, "Denoising diffusion restoration models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 23 593–23 606, 2022.

[4] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

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

[6] J. Ho, W. Chan, C. Saharia, J. Whang, R. Gao, A. A. Gritsenko, D. P. Kingma, B. Poole, M. Norouzi, D. J. Fleet, and T. Salimans, "Imagen video: High definition video generation with diffusion models," *ArXiv*, vol. abs/2210.02303, 2022.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[8] D. P. Kingma, M. Welling *et al.*, "Auto-encoding variational bayes," 2013.

[9] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4195–4205.

[10] Y. Liu, K. Zhang, Y. Li, Z. Yan, C. Gao, R. Chen, Z. Yuan, Y. Huang, H. Sun, J. Gao, L. He, and L. Sun, "Sora: A review on background, technology, limitations, and opportunities of large vision models," 2024. [Online]. Available: https://arxiv.org/abs/2402.17177

[11] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[12] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[13] X. Ma, G. Fang, and X. Wang, "Deepcache: Accelerating diffusion models for free," *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15 762–15 772, 2023.

[14] W. Kong, Y. Hao, Q. Guo, Y. Zhao, X. Song, X. Li, M. Zou, Z. Du, R. Zhang, C. Liu, Y. Wen, P. Jin, X. Hu, W. Li, Z. Xu, and T. Chen, "Cambricon-d: Full-network differential acceleration for diffusion models," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 903–914.

[15] R. Guo, L. Wang, X. Chen, H. Sun, Z. Yue, Y. Qin, H. Han, Y. Wang, F. Tu, S. Wei, Y. Hu, and S. Yin, "20.2 a 28nm 74.34tflops/w bf16 heterogenous cim-based accelerator exploiting denoising-similarity for diffusion models," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 362–364.

[16] Y. Qin, Y. Wang, X. Yang, Z. Zhao, S. Wei, Y. Hu, and S. Yin, "A 52.01 tflops/w diffusion model processor with inter-time-step convolution-attention-redundancy elimination and bipolar floating-point multiplication," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2024, pp. 1–2.

[17] H. Lu, G. Yang, N. Fei, Y. Huo, Z. Lu, P. Luo, and M. Ding, "Vdt: General-purpose video diffusion transformers via mask modeling," 2023. [Online]. Available: https://arxiv.org/abs/2305.13311

[18] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin, "Understanding and improving layer normalization," 2019. [Online]. Available: https://arxiv.org/abs/1911.07013

[19] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[20] Y. Jing, M. Wu, J. Zhou, Y. Sun, Y. Ma, R. Huang, T. Jia, and L. Ye, "Aig-cim: A scalable chiplet module with tri-gear heterogeneous compute-in-memory for diffusion acceleration," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657373

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

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[23] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[24] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[25] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "Faceforensics: A large-scale video dataset for forgery detection in human faces," *ArXiv*, vol. abs/1803.09179, 2018.

[26] W. Xiong, W. Luo, L. Ma, W. Liu, and J. Luo, "Learning to generate time-lapse videos using multi-stage dynamic generative adversarial networks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2364–2373, 2017.

[27] K. Soomro, A. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *ArXiv*, vol. abs/1212.0402, 2012.

[28] A. Siarohin, S. Lathuilière, S. Tulyakov, E. Ricci, and N. Sebe, "First order motion model for image animation," in *Neural Information Processing Systems*, 2020.

[29] T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly, "Fvd: A new metric for video generation," in *DGS@ICLR*, 2019.

[30] Z. Huang, Y. He, J. Yu, F. Zhang, C. Si, Y. Jiang, Y. Zhang, T. Wu, Q. Jin, N. Chanpaisit, Y. Wang, X. Chen, L. Wang, D. Lin, Y. Qiao, and Z. Liu, "Vbench: Comprehensive benchmark suite for video generative models," *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21 807–21 818, 2023.