CoDM: A Co-design Framework for Efficient Sparse Diffusion Models

Xiaolong Wu [1] **Xiang Gao** [2] **Xiyun Song** [3] **Zhongfang Lin** [3] **Heather Yu** [3] **Xianfeng David Gu** [2]

Abstract

Diffusion models have emerged as a powerful class of generative models that excel at capturing complex data distributions and producing realistic, high-fidelity samples. However, these benefits come at the cost of expensive computation and memory requirements due to their iterative denoising process. The cost is especially significant for high-resolution images, videos, 3D data, or long sequences. In this paper, we propose CoDM, a co-design framework that seamlessly integrates model compression techniques with the sparse tensor cores of NVIDIA Hopper H100 GPUs. By leveraging specialized hardware capabilities and jointly optimizing the model compression scheme and storage format, CoDM achieves significant model speedup while maintaining data generation quality. Specifically, our approach enhances diffusion models through several key strategies, namely reducing inference steps and model weights through a novel hierarchical pruning scheme, improving memory efficiency via a new sparse storage format, and leveraging TensorRT optimization and the specialized cores of GPU hardware accelerators. This codesign approach addresses the computational challenges of diffusion models, making them more accessible for real-world applications. Experimental results in a Text-to-Image application demonstrate that our approach surpasses the state-of-the-art, achieving a 7.4-fold speedup on the ImageNet (256×256) dataset and an 11.5-fold speedup on the CIFAR-10(32×32) dataset, all while preserving the quality of the generated images with a similar or lower Fr echet Inception Distance (FID) score.

1. Introduction

Diffusion models (DM) have emerged as a powerful class of generative frameworks, excelling at capturing complex data distributions and producing high-fidelity and realistic samples. However, these benefits come at the cost of expensive computation and memory requirements due to their iterative denoising process. DMs typically operate in two stages: a forward (diffusion) process that perturbs the data distribution to learn time-dependent score functions and a reverse (sampling) process that iteratively generates data samples from a prior distribution. Their large number of model parameters and iterative processing steps pose significant challenges for practical applications of DMs in various domains, especially for high-resolution videos, 3D data, or long sequences, even when using multiple GPUs (Li et al., 2024).

To address these challenges, researchers have proposed a variety of model compression techniques, memory-efficient attention mechanisms, and other optimizations. On the algorithm side, one category of methods aims to accelerate DM inference by reducing the number of inference steps. These include training-free samplers (Bao et al., 2022), (Zhao et al., 2024), (Lu et al., 2022), (Zheng et al., 2023), (Chen et al., 2023) and model distillation approaches (Salimans & Ho, 2022), (Luo et al., 2023). However, sample quality remains suboptimal when sampling steps are reduced, as a limited number of iterations often fail to accurately reconstruct the high-dimensional data space, such as images or videos.

Model sparsity through pruning algorithms offers a promising alternative to reduce the computational burden while maintaining or even improving model performance. However, existing pruning algorithms overlook GPU-specific architecture optimization features (Wang), (Ma et al., 2024) or produce low-quality and inefficient results (Pool & Yu, 2021), (Mishra et al., 2021). Parallel research exploration to reduce Multiply-Accumulate operations per inference step ((Bolya & Hoffman, 2023), (Wang)) often lacks GPU acceleration support.

On the hardware side, recent advances, such as NVIDIA's Ampere and Hopper GPU architectures, have introduced specialized acceleration techniques for model inference by incorporating fine-grained structured sparsity capabilities. One key feature of these architectures is the 2:4 sparse mode, where only two out of every four adjacent weights in a pre-trained model are retained, enforcing a 50% sparsity rate. This sparsity enables acceleration methods to focus exclusively on non-zero values in matrix multiplications, theoretically achieving up to a 2x speedup. In the

context of diffusion models, leveraging the 2:4 sparsity can effectively halve the computational load, a valuable optimization for the iterative refinement process in generative tasks. Furthermore, NVIDIA's Hopper architecture offers FP8 (floating-point 8-bit) precision calibration to optimize tensor operations for lower precision computations (Luo et al., 2024). While prior research work has applied 2:4 structured sparsity to DMs (Wang et al., 2024a), it suffers from two main drawbacks: 1) it neglects the distinctive structural features of diffusion models, leading to reduced sample quality, and 2) it achieves only 1.2x inference acceleration, failing to fully utilize the performance potential of the GPU architecture.

Previous work have primarily focused on either model compression or hardware acceleration in isolation, thus fail to fully exploit the maximum potential speedup of DMs for practical applications. In this paper, we propose CoDM, a co-deign framework that seamlessly integrates diffusion model compression techniques with the sparse tensor core features of NVIDIA Hopper H100 GPUs. Specifically, our approach enhances diffusion models through several key strategies: 1) reducing inference steps and weights through a novel hierarchical pruning algorithm based on the structural properties of diffusion models, 2) utilizing TensorRT optimization and the specialized architecture of GPU hardware accelerators, and 3) improving memory efficiency via a new all-in-one Hierarchical Blocked ELLPACT (Rice & Boisvert, 2012) (HB-ELL) sparse storage format that integrates diffusion model structural pruning with the n:m hardware structured sparsity capability. By leveraging specialized hardware capabilities and jointly optimizing model compression algorithms as well as novel efficient memory format, CoDM achieves significant model speedup while maintaining data generation quality.

We evaluate CoDM with several diffusion models on NVIDIA's H100 GPU with ImageNet and CIFAR10 datasets by testing FID and MACs. Our results with a UNet-based diffusion model demonstrate that our method surpasses the state-of-the-art, delivering a $7.4\times$ speedup on the ImageNet (256×256) dataset and an $11.5\times$ speedup on the CIFAR-10 (32×32) dataset. Furthermore, our model achieves a lower Fr´echet Inception Distance (FID) score than existing methods, reflecting superior image generation quality.

The key contributions of this work are as follows:

- To the best of our knowledge, little has been done to explore speedup of DM inference with joint consideration of algorithms and hardware. We propose CoDM, a co-design approach that bridges the gap between algorithm-focused and hardware-focused solutions, enabling more effective speedup.
- We introduce a novel three-layer hierarchical pruning algorithm that efficiently incorporates sparsity into diffusion models based on their structural properties.
- A new sparse tensor storage format is proposed to seamlessly connect diffusion model pruning with GPU sparsity capabilities.
- We conduct a comprehensive evaluation of CoDM's computational efficiency improvements across multiple benchmark tasks and analyze the impacts of individual components within the CoDM framework, providing insight for future research.

2. Related Work and Background

Diffusion Model. Diffusion models have evolved rapidly, with foundational works such as DDPM (Denoising Diffusion Probabilistic Models) and DDIM (Denoising Diffusion Implicit Models) setting the stage for various applications. These models leverage a forward and reverse process to generate samples, with the reverse process trained to denoise inputs at each step. The existing methodologies for enhancing the efficiency of diffusion models primarily focus on three strategies: optimizing network architectures (Rombach et al., 2022), (Yang et al., 2023) , (Nichol & Dhariwal, 2021), refining training processes (Hang et al., 2023), (Wang et al., 2024b), and accelerating sampling methods (Ho et al., 2020). Many diffusion models utilize U-Net structures as denoisers, and their efficiency can be improved by incorporating hierarchical designs (Ramesh et al., 2022) or performing training within a latent space (Rombach et al., 2022). Recent research advocates for integrating more efficient layers or architectures within the U-Net denoiser to enhance performance, thereby enabling the model to learn higher-quality image representations during training (Yang et al., 2023). Additionally, significant efforts are directed towards boosting training efficiency, with some studies showing that diffusion training can be accelerated by adjusting the weight distribution across timesteps (Salimans & Ho, 2022). Training can also be optimized by applying diffusion models at the patch level (Wang et al., 2024b). Finally, several approaches focus on sampling efficiency, which often does not require retraining the model (Liu et al., 2022). In this area, various techniques reduce the number of steps, using strategies like early stopping (Lyu et al., 2022) and distillation (Salimans & Ho, 2022).

Model Pruning. In parallel, the concept of model sparsity has gained traction, with techniques like pruning, quantization, and low-rank factorization applied across different neural networks. Sparse models are not only computationally efficient but also often exhibit better generalization properties. Sparse model are particularly achieved through the application of network pruning techniques (Liu et al., 2017), (He et al., 2017), (Luo et al., 2017), (He et al., 2019). Pruning methodologies generally fall into two main cate-

gories: structural pruning (Filters'Importance, 2016), (Ding et al., 2019), (You et al., 2019), (Liu et al., 2021) and unstructured pruning (Park et al., 2020), (Dong et al., 2017), (Sanh et al., 2020). Structural pruning distinguishes itself by physically removing parameters and substructures from networks, while unstructured pruning effectively masks parameters by setting them to zero (Fang et al., 2023).

Most network pruning research to date has focused on discriminative tasks, especially classification tasks (He et al., 2017). Few studies have explored the potential of pruning techniques in generative tasks, such as GAN compression (Li et al., 2020), (Vo et al., 2022). Recently, applying structural pruning techniques to diffusion Models has introduced unique challenges, prompting a re-evaluation of traditional approaches (Ma et al., 2024),(Wang). While existing pruning techniques generally overlook critical hardware features, other work (Wang et al., 2024a) has integrated modern GPU capabilities but fail to consider the distinct structural characteristics of diffusion models. In this work, we introduce the first dedicated pruning method that effectively combines diffusion model features with GPU hardware optimizations to achieve efficient, high-quality model inference. CoDM advances this research by exploring the intersection of diffusion models, sparsity, and GPU hardware capabilities. Whereas previous works have focused mainly on either sparse pruning algorithms for diffusion models or exploiting GPU hardware features individually, our approach is novel in its unified integration of diffusion model properties and GPU hardware optimizations to enhance both efficiency and performance.

Sparse Tensor Cores of NVIDIA GPUs. NVIDIA Hopper GPUs have expanded their Tensor Core Units (TCUs) to support row-wise 2:4 sparsity, incorporating hardware capabilities for sparse computation. These enhanced TCUs, known as Sparse Tensor Cores (SPTCs), allow for efficient processing of sparsely structured data. To leverage SPTCs, the first operand in tensor operations must be formatted in NVIDIA's N:M sparsity format, where N represents the maximum number of non-zero elements in a block of M values. This structure is illustrated in Figure 1. On the left, the figure shows an uncompressed sparse matrix following the 2:4 row-wise sparsity pattern. Compressing this $M \times N$ matrix involves two main components: (1) an $M \times N/2$ matrix that stores the non-zero values, and (2) a metadata structure that encodes the positions of non-zero elements within each group of four values. On the right side of Figure 1, the mapping of a 2:4 sparse operation onto SPTCs is depicted. The metadata structure enables the hardware to efficiently select the corresponding elements in the dense matrix B for Matrix Multiply-Accumulate (MMA) operations, optimizing performance for sparse computations.

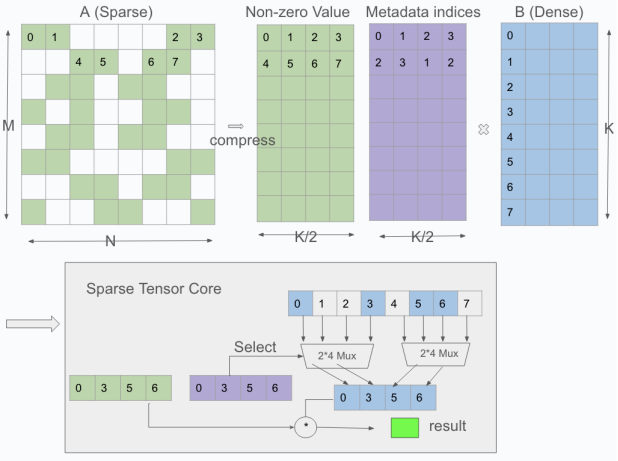


Figure 1. The 2:4 format and its mapping to SPTCs

3. Sparse Diffusion Model

In this section, we first present the theoretical foundations underlying sparse diffusion model pruning, followed by an introduction to the CoDM framework.

3.1. Theoretical Foundations

3.1.1. Diffusion Models

We delve into the mathematical framework underlying CoDM. The diffusion process is traditionally represented as a series of iterative updates, with each step involving the application of a learned denoising function. By introducing sparsity, we modify this process to selectively activate only a subset of the model's parameters during each step.

Consider the forward diffusion process as a Markov chain, where each step involves adding Gaussian noise to the input:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t, \tag{1}$$

where β_t represents the noise schedule, and ϵ_t is the Gaussian noise.

The reverse process involves denoising:

$$x_{t-1} = \frac{1}{\sqrt{1-\beta_t}}(x_t - \sqrt{\beta_t}\epsilon_\theta(x_t, t)), \tag{2}$$

where ϵ_θ is the neural network with parameters θ trained to predict noise.

In CoDM, we introduce sparsity into the denoising network ϵ_θ by modifying its architecture to include sparse layers. Specifically, we employ pruning techniques to remove less important parameters, resulting in a model that is both computationally efficient and effective.

3.1.2. Sparse Pruning

In the context of a sparse diffusion model, the objective is to reduce the number of parameters in the denoising network ϵ_θ

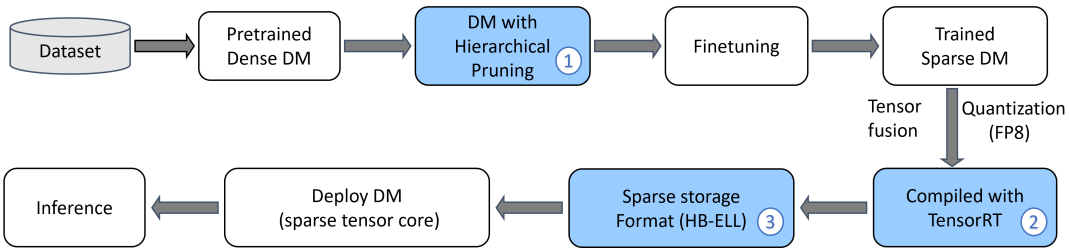


Figure 2. CoDM Framework Overview

by pruning unimportant weights while retaining the model's performance. This involves introducing a sparse structure into ϵ_θ by removing a fraction of its parameters.

Given a neural network ϵ_θ with weight matrix W, we introduce a binary mask matrix M_t at step t of the diffusion process to indicate the retained weights. The pruned weight matrix $\tilde{W}_t$ at step t is then given by:

$$\tilde{W}_t = M_t \odot W \tag{3}$$

where $\odot$ denotes element-wise multiplication, and each element of the binary mask M_t is defined as:

$$m_{i,j}^t = \begin{cases} 1 & \text{if } w_{i,j} \text{ is kept at step } t \\ 0 & \text{if } w_{i,j} \text{ is pruned at step } t \end{cases} \tag{4}$$

In this case, the mask matrix M_t evolves over the steps of the diffusion process, progressively pruning smaller weights at each step.

The goal of sparse pruning is to minimize the number of nonzero weights while retaining model performance, leading to the following optimization problem:

$$\min_M \sum_{i,j} m_{i,j} \quad \text{subject to} \quad \mathcal{L}(\tilde{W}) \leq \epsilon \tag{5}$$

where $\mathcal{L}(\tilde{W})$ is the loss function of the pruned diffusion model, and ϵ is an acceptable threshold for tolerable performance degradation.

For an N:M sparsity network, the sparse DM inference is expressed as:

$$y_t^{(N:M)} = F(W_t^{(N:M)}; x_t) \tag{6}$$

where $F(\cdot)$ is the forward propagation function, and $W_t^{(N:M)}$ is the pruned weight matrix adhering to the N:M sparsity pattern.

To achieve 2:4 sparse GPU acceleration, the standard sparse training method involves training from a dense model with 0% sparsity to a model with 50% sparsity, which often leads to significant information loss. Additionally, this approach is time-consuming. Applying this method directly to our hierarchical pruning approach, which combines block sparsity with 2:4 sparsity, would also result in information loss. To address this, we introduce two novel strategies: 1) a progressive sparse training process, and 2) adaptive sparsity adjustment. These strategies enable us to achieve a 50% sparse model while minimizing information loss.

(1)Block Sparsity with Progressive Pruning:

Let B denote the block structure of the model, where each block consists of a subset of the model weights. Initially, we set a small block sparsity rate γ_{init}, which **progressively increases** through the training process. This allows the model to adapt to sparsity levels without significant information loss.

The block sparsity optimization can be written as:

$$\min_B \sum_{b \in B} \sum_{i,j \in b} m_{i,j} \quad \text{subject to} \quad \mathcal{L}(\tilde{W}_B) \leq \epsilon_B, \tag{7}$$

where B represents the blocks, $\mathcal{L}(\tilde{W}_B)$ is the loss for the pruned model with block sparsity, and ϵ_B is the corresponding performance threshold for block sparsity.

For the 2:4 sparsity pattern, we ensure that within each group of 4 weights, 2 are retained and 2 are pruned. This pattern is well-suited for specialized hardware such as NVIDIA Hopper architecture GPUs, which accelerate matrix multiplications with **2:4 sparse operators**.

The corresponding optimization for 2:4 sparsity, applied within each block b, is as follows:

$$\min_B \sum_{b \in B} \sum_{i,j \in b} m_{i,j} \quad \text{subject to} \quad \mathcal{L}(\tilde{W}_B) \leq \epsilon_B, \tag{8}$$

where the nonzero weights within each block b follow the 2:4 sparsity pattern. Inference for the pruned model is defined as:

$$y_t^{(2:4)} = F(W_t^{(2:4)}; x_t), \tag{9}$$

(2) Adaptive Sparsity Adjustment:

To further reduce information loss, we introduce an **adaptive sparsity adjustment** mechanism that dynamically adjusts the sparsity during training based on the gradient information. For each block b, we define a function $\gamma(t)$ that

adjusts the sparsity rate based on the performance at time t:

$$\gamma(t) = \gamma_{init} + \Delta\gamma(t), \tag{10}$$

where $\Delta\gamma(t)$ is the rate of sparsity increase at time t, based on the gradients and model performance.

Thus, the optimization for the adaptive sparsity process becomes:

$$\min_{M,B} \sum_{i,j} m_{i,j} + \sum_{b \in B} \Delta\gamma(t) \quad \text{subject to} \quad \mathcal{L}(\tilde{W}) \leq \epsilon. \tag{11}$$

After pruning, we fine-tune the model to recover performance. The fine-tuning objective is to train the pruned model on the denoising task, minimizing the loss over the predicted noise $\epsilon_\theta(x_t, t)$ compared to the ground truth noise ϵ_t:

$$\mathcal{L}_{\text{fine-tune}}(\theta) = \mathbb{E}_{t,x_0,\epsilon_t}\left[\left\|\epsilon_t - \epsilon_\theta(\tilde{W}, x_t, t)\right\|^2\right] \tag{12}$$

The fine-tuning process aims to adjust the remaining weights in $\tilde{W}$ to minimize this objective, effectively restoring the model's denoising accuracy.

In practice, a sparsity-inducing regularization term $R(M)$ can be added to the fine-tuning loss to encourage a desired sparsity pattern, giving the combined objective:

$$\mathcal{L}_{\text{combined}}(\theta, M) = \mathbb{E}_{t,x_0,\epsilon_t}\left[\left\|\epsilon_t - \epsilon_\theta(\tilde{W}, x_t, t)\right\|^2\right] + \lambda R(M) \tag{13}$$

where λ is a regularization coefficient, and $R(M)$ may represent an ℓ_1-norm penalty or other sparsity constraints.

3.2. CoDM Framework

Figure 2 shows an overview of the proposed framework. Starting with a pretrained dense diffusion model, we introduce a three-layer hierarchical pruning algorithm that operates at the step layer, U-Net layer, and 4-element layer, tailored for 2:4 Sparse Tensor Core compatibility. We then fine-tune the diffusion model (DM) using this hierarchical pruning approach to reduce multiply-accumulate operations (MACs) and prepare it for accelerated inference on sparse GPUs. Additionally, we compile the pruned sparse DM using TensorRT (ten, 2023) with tensor fusion and FP8 quantization optimizations. To further enhance efficiency, we employ a novel sparse storage format, which enables us to deploy the optimized sparse DM on the 2:4 Sparse Tensor Core for acceleration. We present the detailed description of the key stages in the following sections.

4. Pruning algorithm

We build a three-layer hierarchical pruning algorithm by integrating step pruning, taylor pruning with 2:4 pruning, enforcing two out of every four consecutive elements to be zero.

For simplicity, we assume θ is represented as a 2-D matrix, where each sub-structure $\theta_i = [\theta_{i0}, \theta_{i1}, \ldots, \theta_{iK}]$ is a row vector with K scalar parameters. Structural pruning aims to produce a sparse parameter matrix θ' that retains as much of the model's original performance as possible. Therefore, we can frame the pruning objective as minimizing the loss disruption due to pruning:

$$\min_{\theta'} |L(\theta') - L(\theta)|, \quad \text{s.t.} \quad \|\theta'\|_0 \leq s. \tag{14}$$

Here, $|\theta'|_0$ represents the L_0 norm, counting the number of non-zero row vectors, and s denotes the sparsity level of the pruned model. However, given the iterative nature of diffusion models, the training objective L can be seen as a composition of T interdependent tasks: $\{L_1, L_2, \ldots, L_T\}$. Each task both influences and depends on the others, creating a unique challenge distinct from traditional pruning tasks that generally focus on a single objective.

First Layer: step pruning on T. We prune the steps based on two observations of diffusion Model:

Observation 1: Early steps of denosing procedure focus on local details like edges and color and later ones pay more attention to contents such as object and shape.

Observation 2: There is a strong temporal locality between adjacent steps in the denoising process.

Building on the first observation, we set a low threshold in the early steps to reduce their weights, using a heuristic algorithm to determine the number of initial steps. For the second observation, we apply direct pruning to every other step, targeting adjacent steps in each pair of connected steps.

We then examine the individual impact of each remaining step and its loss component L_t in the pruning process.

Second Layer: Taylor Expansion at L_t. To assess the contribution of each L_t for structural pruning, we apply a Taylor expansion to L_t to approximate the loss disruption linearly:

$$\begin{aligned} L_t(\theta') &= L_t(\theta) + \nabla L_t(\theta) \cdot (\theta' - \theta) + O(\|\theta' - \theta\|^2) \\ &\Rightarrow L_t(\theta') - L_t(\theta) = \nabla L_t(\theta) \cdot (\theta' - \theta) + O(\|\theta' - \theta\|^2). \end{aligned} \tag{15}$$

This formulation enables us to quantify and optimize the impact of each pruning step on the diffusion model's performance. By calculating the importance score for each weight, taylor pruning can identify and prune the least important weights. This importance score is computed using a combination of the weight values and their gradients. Weights with scores below a set threshold are pruned, aiming to

maintain the model's overall functionality while reducing parameter count.

Third layer: 2:4 Pruning. We use weight magnitude to choose weights to prune for each four weights, so it is natural to maximize the total magnitude after the 2:4 constraint's application. We apply a permutation pruning technique (Pool & Yu, 2021) to ensure that an optimal permutation preserves the full magnitude of the input matrix.

Algorithm 1 Hierarchical Pruning

Input: Pretrained diffusion model θ, dataset X, threshold $\mathcal{T}$, early step number N

Output: Pruned diffusion model θ'

1. $\mathcal{L}_{\text{max}} \leftarrow 0$
2. $x \leftarrow$ mini-batch(X)
3. $\epsilon \sim \mathcal{N}(0, 1)$
4. *Accumulate gradients over partial steps with* $\mathcal{T}$
5. **for** t in $[0, 1, 2, \ldots, T]$ **do**
   (a) **if** $t \mod 2 = 0$ **then**
      i. $\mathcal{L}_t \leftarrow \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\, x + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t)\|^2$ *Equation 2*
      ii. $\mathcal{L}_{\text{max}} \leftarrow \max(\mathcal{L}_{\text{max}}, \mathcal{L}_t)$
      iii. **if** $t \leq N$ **then** $\mathcal{T} \leftarrow \mathcal{T}/10$ **end if**
      iv. **if** $\mathcal{L}_t / \mathcal{L}_{\text{max}} \leq \mathcal{T}$ **then break**
      v. $\nabla_{\theta_{i,k}} \mathcal{L}_t(\theta, x) \leftarrow$ back-propagation$(\mathcal{L}_t(\theta, x))$
   (b) **end if**
6. **end for**
7. *Estimate the importance of sub-structure* θ_i *with accumulated* t*-step gradients*
8. $\mathcal{I}(\theta_i, x) \leftarrow \sum_k |\theta_i| \cdot \left|\sum_{s=0}^{t} \nabla_{i,k} \mathcal{L}_s(\theta, x)\right|$
9. *Pruning and fine-tuning*
10. **Remove** 50% **of channels in each group of 4 elements to obtain** θ'
11. **Fine-tune the pruned model** θ' **on** X
12. **return** θ'

5. Sparse Storage Format

To support hierarchical sparsity of our hierarchical pruning, we propose a novel storage format on sparse tensor cores (STC).

Our work proposes , an all-in-one format being able to support hierarchical sparsity. , the hierarchical ELLPACK format, saves element-wise and vector-wise sparse data in ELL format (Rice & Boisvert, 2012) inspired by the work (Zhu et al., 2019). The format is shown in Figure 3. In particular, sparse data is first split into blocks, where only non-zero

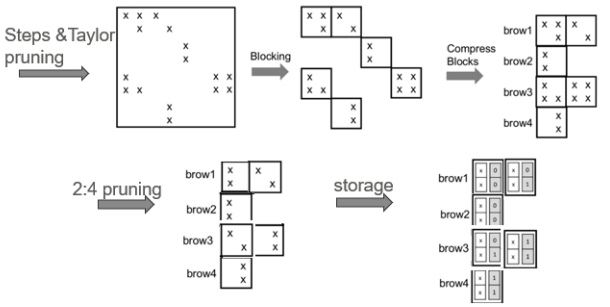


Figure 3. formats for an example 8×8 sparse matrix using 2×2 blocks.

blocks (at least one non-zero entry shown in a block) are saved. In Figure 3, only six non-zero blocks are saved in a 8×8 sparse matrix. All non-zero blocks are organized by blocked rows (*brow1-4* in Figure 3, sized 2). For each block's sparsity pattern, we choose different storage strategies underneath.

ELL is chosen as the format for both element-wise operation and structural blocks because: 1) It has been proven to be efficient on both the two types (Zhu et al., 2019) for computation and storage; 2) Limited by block size, data in a small block does not have a high divergence of the amount of non-zeros among rows, which makes ELL more suitable; 3) ELL format is well-suited for massive parallel processing on GPUs. Sparse blocks need to save these non-zeros in two arrays: $value$ and $offset$ in the ELL format. We observe different numbers of columns, Cs, in the ELL representation of blocks. The ELL representations of three blocks only have one ELL column ($C = 1$), while those of two blocks have two ELL columns ($C = 2$). Thus, an extra array $bcols$ is stored to save this information. We save all blocks in a row-major order, e.g., blocks in *brow1*, blocks in *brow2*, etc. Using ELL for small blocks not only saves storage but also limits the offset index range thus can be represented in less bits. For example, only 1 bit is needed in Figure 3, as indices are in the range [0,1].

However, due to the pruning and the randomness of sparsity, this uniformed number of columns has to be the maximum row length to cover all non-zeros (as mentioned in (Zhu et al., 2019)). Thus, zero-filling is necessary for the ELL format to store and compute these zeros, which increases memory footprints and time complexity. supports non-uniform number of ELL columns (Cs), and every matrix block has its affiliated column length saved in $bcols$, reducing the zero-filling ratio.

6. TensorRT Optimization

Tensor Fusion is an optimization technique used by TensorRT to combine multiple tensor operations into a single kernel launch. This reduces overhead from repeated memory reads and writes, allowing data to remain in the GPU's high-speed memory longer and minimizing data transfer between GPU cores. This optimization is crucial for models with multiple small operations or layers, which are common in our spare diffusion model.

FP8 Precision Calibration is a newer precision level supported by TensorRT on H100 GPUs that handle 8-bit floating-point operations. We explore this optimization on the NVIDIA Hopper architecture with TensorRT 10.0 (ten, 2024). FP8 (floating-point 8-bit) calibration allows models trained in higher precision (like FP16 or FP32) to be run at 8-bit precision, reducing both memory usage and computational cost while maintaining acceptable accuracy.

7. Experiments

7.1. Experimental Setup

To demonstrate the effectiveness of our method, we evaluate it on a widely adopted diffusion model: LDM (Rombach et al., 2022). Our experiments are conducted on the NVIDIA DGX H100 system, which features 8x NVIDIA H100 Tensor Core GPUs with 16,896 CUDA cores and 528 4th-generation Tensor Cores per GPU and 80GB of GPU memory each, totaling 640GB. The system is powered by dual 6th-generation Intel Xeon CPUs. To validate the effectiveness of our solution, we conduct experiments on two benchmark datasets, CIFAR10 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009). A total of 50,000 images are generated and FID score (Kynkäänniemi et al., 2019) is used to evaluate the quality of the generated images.

Baselines We choose SparseDM (Wang et al., 2024a) as the main baseline for our method since SparseDM also leverage 2:4 sparsity structure. We also compare our work with step pruning (Ma et al., 2024), Taylor pruning (Wang), and ASP (Pool & Yu, 2021).

7.2. Pruning Results

The results highlight the computational efficiency and performance of CoDM, with detailed comparisons to other models.

Figure 4 in Appendix A presents the pruning results for the LDM pre-trained on ImageNet at 256 × 256 resolution and CIFAR-10 at 32 × 32 resolution. A LDM consists of an encoder, a decoder, and a U-Net model, with approximately 400M parameters attributed to the U-Net and only 55M to the autoencoder. Therefore, we primarily focus on pruning the U-Net model. We set a threshold T=0.1 to exclude converged layers, making the pruning process more efficient. With this threshold, only 534 steps are involved in pruning. After estimating importance, we apply a fixed channel sparsity of 30% across all layers, reducing the U-Net to 189.43M parameters. Finally, we fine-tune the pruned model for just 4 epochs using the official training scripts, with a learning rate scaled to 0.1 times the base rate.

The results of Hierarchical pruning on CIFAR10 (32×32) and ImageNet (256×256) are shown in Table 2 and 3 respectively. The two tables present the performance and image quality of various pruning methods on diffusion models. Each table compares the methods based on four metrics: MACs, throughput, speed, and FID score. In both tables, "Ours" offers a well-rounded performance, achieving competitive MACs, throughput, speed, and FID scores, showing effective pruning for efficient model performance.

7.3. TensorRT Optimization

The fourth column of Table 1 reports the inference performance of a pruned model on the ImageNet (256×256) and CIFAR-10 (32×32) dataset with TensorRT Tensor fusion and FP8 quantization optimization. The results show that our pruning algorithm reduces MACs to 71.96G. Using PyTorch, the pruned model achieves a speedup of 1.8x compared to the original model, while with TensorRT optimization further accelerates performance with a $3.7\times$ speedup. This table demonstrates the combined effect of model pruning and TensorRT optimizations in significantly reducing computational cost and inference time, highlighting the potential of these techniques for deploying efficient sparse diffusion models.

7.4. Sparse Tensor Core

The fifth column of Table 1 presents the performance results of a pruned diffusion model with Sparse Tensor Core optimizations. For ImageNet, the pruned model achieves a 3.7x speedup with TensorRT and a substantial $7.4\times$ speedup with Sparse Tensor Core optimization using HB-ELL format. Similarly, on CIFAR-10, Hierarchical pruning with TensorRT optimization delivers a $5.9\times$ speedup, while further with Sparse Tensor Core optimization significantly boosts performance with an $11.5\times$ speedup. These results illustrate the efficiency gains possible through pruning and hardware optimization, especially with Sparse Tensor Core technology, in reducing inference time while maintaining effective model performance. In Table 5 of Appendix B, we list the performance speedup breakdown for different optimizations.

Dataset	MACs (Value)	Hierarchical pruning (Value / speedup)	TensorRT w/ Hierarchical pruning (Value / speedup)	Sparse Tensor Core w/ Hierarchical pruning + TensorRT (Value / speedup)
ImageNet (256 × 256)	71.96G	6.13s / 1.8×	3.02s / 3.7×	1.49s / 7.4×
CIFAR-10 (32 × 32)	3.6G	272.6ms / 2.9×	129.1ms / 5.9×	65.3ms / 11.5×

Table 1. Sparse Tensor Core Optimization

Method	MACs ↓	Throughput ↑	Speed ↑	FID ↓
SparseDM	5.67G	0.10	1.1×	4.23
Taylor-Pruning	6.1G	0.09	1×	**4.17**
ASP	6.1G	0.09	1×	4.35
DeepCache	**3.5G**	0.13	1.38×	5.31
Ours	3.6G	**0.13**	**1.41**×	4.20

Table 2. Hierarchical Pruning on CIFAR-10 (32 × 32)

Method	MACs ↓	Throughput ↑	Speed ↑	FID ↓
SparseDM	87.3G	1.43	1.2×	5.21
Taylor-Pruning	99.8G	1.34	1×	**4.10**
ASP	87.3G	1.44	1×	5.12
DeepCache	**37.6G**	**1.47**	2.65×	6.21
Ours	71.96G	1.35	1.71×	4.42

Table 3. Hierarchical Pruning on ImageNet (256 × 256)

7.5. Ablation Study

We employ a heuristic approach to evaluate the pruning steps of the diffusion model. Table 4 illustrates our selection of the optimal early pruning step in Algorithm 1 on the ImageNet (256×256) dataset. We test step N in the range of 5 to 45, presenting the results for different step counts, along with the corresponding MACs and FID scores. The highlighted row for 20 steps shows a balance between computational efficiency and FID score, suggesting that this configuration offers an optimal trade-off between efficiency and quality. As the number of steps decreases, MACs decrease, indicating reduced computational load, while FID scores initially remain low, then gradually increase, reflecting a degradation in image quality with fewer steps.

Steps (N)	MACs$_\downarrow$ (Ours)	FID$_\downarrow$ (Ours)
5	102.13	3.37
10	96.49	3.45
15	83.51	3.58
20	**71.96**	**4.42**
21	69.31	7.19
23	66.34	8.56
25	52.71	9.27
30	49.03	9.37
35	33.48	9.95
40	23.50	10.12
45	13.97	11.88

Table 4. MACs and FID trade-off with different steps.

8. Conclusion

In this paper, we introduce CoDM, a novel approach that integrates sparsity into diffusion models. By seamlessly combining diffusion model structure pruning with NVIDIA H100 GPU performance optimization, our method significantly improves the computational efficiency of these models while maintaining image quality. Future work will explore further optimizations and extensions of CoDM to support distributed and heterogeneous platform (e.g., ASIC) and multi-GPU architecture.

References

Nvidia/tensorrt. 2023.

Nvidia/tensorrt. https://developer.nvidia.com/blog/nvidia-tensorrt-10-0-upgrades-usability-performance-and-ai-model-support/. 2024.

Bao, F., Li, C., Zhu, J., and Zhang, B. Analytic-dpm: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. *arXiv preprint arXiv:2201.06503*, 2022.

Bolya, D. and Hoffman, J. Token merging for fast stable diffusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4599–4603, 2023.

Chen, Z., He, G., Zheng, K., Tan, X., and Zhu, J. Schrodinger bridges beat diffusion models on text-to-speech synthesis. *arXiv preprint arXiv:2312.03491*, 2023.

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

Ding, X., Ding, G., Guo, Y., and Han, J. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4943–4953, 2019.

Dong, X., Chen, S., and Pan, S. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in neural information processing systems*, 30, 2017.

Fang, G., Ma, X., Song, M., Mi, M. B., and Wang, X. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16091–16101, 2023.

Filters'Importance, D. Pruning filters for efficient convnets. 2016.

Hang, T., Gu, S., Li, C., Bao, J., Chen, D., Hu, H., Geng, X., and Guo, B. Efficient diffusion training via min-snr weighting strategy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7441–7451, 2023.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4340–4349, 2019.

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

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kynkäänniemi, T., Karras, T., Laine, S., Lehtinen, J., and Aila, T. Improved precision and recall metric for assessing generative models. *Advances in neural information processing systems*, 32, 2019.

Li, M., Lin, J., Ding, Y., Liu, Z., Zhu, J.-Y., and Han, S. Gan compression: Efficient architectures for interactive conditional gans. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5284–5294, 2020.

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

Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J.-H., Wang, X., Chen, Y., Yang, W., Liao, Q., and Zhang, W. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pp. 7021–7032. PMLR, 2021.

Liu, L., Ren, Y., Lin, Z., and Zhao, Z. Pseudo numerical methods for diffusion models on manifolds. *arXiv preprint arXiv:2202.09778*, 2022.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pp. 2736–2744, 2017.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

Luo, S., Tan, Y., Patil, S., Gu, D., von Platen, P., Passos, A., Huang, L., Li, J., and Zhao, H. Lcm-lora: A universal stable-diffusion acceleration module. *arXiv preprint arXiv:2311.05556*, 2023.

Luo, W., Fan, R., Li, Z., Du, D., Wang, Q., and Chu, X. Benchmarking and dissecting the nvidia hopper gpu architecture. *arXiv preprint arXiv:2402.13499*, 2024.

Lyu, Z., Xu, X., Yang, C., Lin, D., and Dai, B. Accelerating diffusion models via early stop of the diffusion process. *arXiv preprint arXiv:2205.12524*, 2022.

Ma, X., Fang, G., and Wang, X. Deepcache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15762–15772, 2024.

Mishra, A., Latorre, J. A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., and Micikevicius, P. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pp. 8162–8171. PMLR, 2021.

Park, S., Lee, J., Mo, S., and Shin, J. Lookahead: A farsighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*, 2020.

Pool, J. and Yu, C. Channel permutations for n: m sparsity. *Advances in neural information processing systems*, 34: 13316–13327, 2021.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

Rice, J. R. and Boisvert, R. F. *Solving elliptic problems using ELLPACK*, volume 2. Springer Science & Business Media, 2012.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

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

Sanh, V., Wolf, T., and Rush, A. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in neural information processing systems*, 33:20378–20389, 2020.

Vo, D. M., Sugimoto, A., and Nakayama, H. Ppcd-gan: Progressive pruning and class-aware distillation for large-scale conditional gans compression. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp. 2436–2444, 2022.

Wang, G. F. X. M. X. Structural pruning for diffusion models—supplementary materials—.

Wang, K., Chen, J., Li, H., Mi, Z., and Zhu, J. Sparsedm: Toward sparse efficient diffusion models. *arXiv preprint arXiv:2404.10445*, 2024a.

Wang, Z., Jiang, Y., Zheng, H., Wang, P., He, P., Wang, Z., Chen, W., Zhou, M., et al. Patch diffusion: Faster and more data-efficient training of diffusion models. *Advances in neural information processing systems*, 36, 2024b.

Yang, X., Zhou, D., Feng, J., and Wang, X. Diffusion probabilistic model made slim. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 22552–22562, 2023.

You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

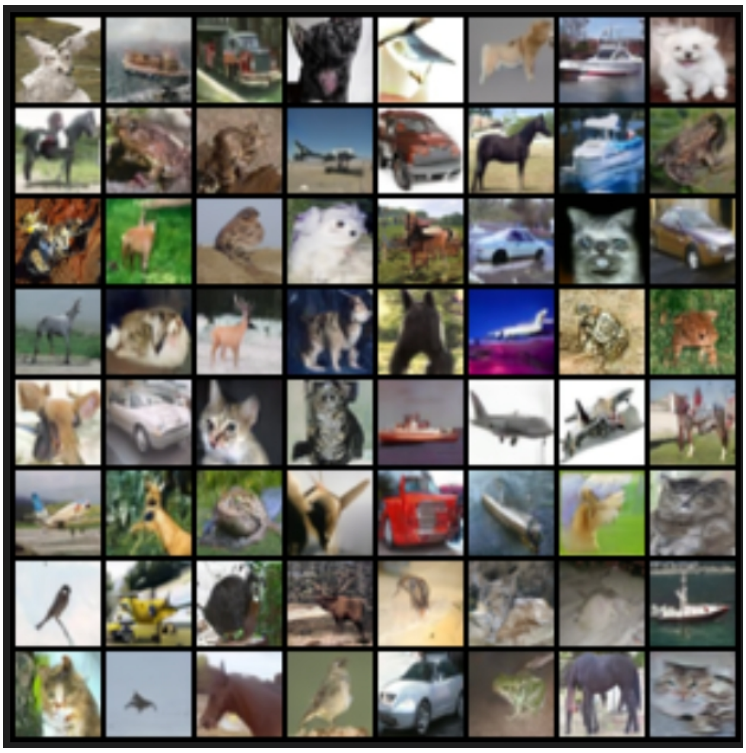
Zhao, W., Bai, L., Rao, Y., Zhou, J., and Lu, J. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

Zheng, K., Lu, C., Chen, J., and Zhu, J. Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. *Advances in Neural Information Processing Systems*, 36:55502–55542, 2023.

Zhu, M., Zhang, T., Gu, Z., and Xie, Y. Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 359–371, 2019.

A. Images sampled

(a) ImageNet (256 × 256).



(b) CIFAR-10 (32 × 32).

Figure 4. Images sampled from the pruned conditional LDM on ImageNet and CIFAR-10 datasets.

B. Performance speedup breakdown for different optimizations

In Table 5, we present the individual performance contributions of three key components in our proposed framework: TensorRT, the pruning algorithm, and the sparse storage format. TensorRT, an optimized inference engine, achieves a 33.5% speedup by enhancing computation efficiency. The pruning algorithm contributes a 34.2% speedup by reducing the computational workload, enabling faster inference. Lastly, the sparse storage format provides a 32.3% performance gain by improving memory efficiency, which further accelerates model processing. These improvements demonstrate the effectiveness of each component in optimizing the overall framework performance.

Ours	**Performance speedup (%)**
TensorRT	33.5%
Hierarchical pruning algorithm	34.2%
Sparse Storage Format	32.3%

Table 5. Performance speedup for different components.