LiteDiffusion for CIFAR-10

Atif Abedeen
aabedeen@umass.edu

Darsh Gondalia
dgondalia@umass.edu

Abstract

Diffusion models have revolutionized image generation, but their significant computational demands and large model sizes limit practical deployment on resource-constrained devices. This paper introduces Lite-Diffusion (L-Diff), a compressed diffusion model designed for efficient and rapid inference on edge devices. By leveraging a combination of modern model compression techniques—including pruning variants, quantization, knowledge distillation, and the novel application of Low-Rank Adaptation (LoRA)—we reduce model size and computational requirements without substantial loss in image quality. A real-world example illustrates the utility of deploying DDPMs on edge devices: enabling offline content generation for augmented reality applications on smartphones, where quick and efficient image synthesis is crucial. We evaluate various combinations of these techniques to develop the best-performing compressed model.

1. Introduction

Diffusion models have emerged as a leading approach in generative modeling, particularly for producing high-quality images. However, the computational demands and large sizes of state-of-the-art (SOTA) models like Denoising Diffusion Probabilistic Models (DDPMs) [10] pose significant challenges for deployment on resource-constrained devices. Edge devices such as smartphones, drones, and IoT gadgets often lack the computational power and memory to run these models efficiently.

Consider a real-life scenario where a smartphone app provides augmented reality experiences by generating context-specific images in real-time. Deploying a full-sized DDPM on the device would be impractical due to latency issues and battery consumption. An efficient, compressed model would enable quick inferencing without relying on constant cloud communication, preserving user privacy and reducing operational costs.

To address this challenge, we investigate various model compression techniques to reduce the size and computational requirements of diffusion models while maintaining high image quality. Our approach explores pruning variants [6], quantization [12], and knowledge distillation [9]—both individually and in combination—to identify the most effective strategies for model compression.

A highlight of our research is the novel application of Low-Rank Adaptation (LoRA) [11] in compressing diffusion models. LoRA allows us to reduce the number of trainable parameters significantly by decomposing weight matrices into lower-dimensional representations. This technique complements existing methods and provides an additional boost in compression efficiency with minimal information loss.

By evaluating different combinations of these techniques, we develop the Lite-Diffusion (L-Diff) model tailored for edge devices. Using the CIFAR-10 dataset for training and testing, we apply standard data augmentation and pre-processing to ensure robustness. Our experiments show that L-Diff achieves substantial reductions in model size and computational load, enabling efficient and quick inferencing on edge platforms.

We assess the performance of our compressed models using Fréchet Inception Distance (FID) [8] for quantitative analysis. FID measures the realism and diversity of the generated images. The results demonstrate that our compression methods maintain high-quality image generation despite the reduced computational footprint.

The successful creation of L-Diff (Figure 1) paves the way for the practical deployment of diffusion models on edge devices. This advancement not only broadens the accessibility of generative models in real-world applications but also opens up new possibilities for innovation in areas like mobile computing, autonomous systems, and personalized user experiences.

2. Related Work

Recent advancements in diffusion models, such as Denoising Diffusion Probabilistic Models (DDPMs) [10], have achieved state-of-the-art results in generative tasks like image creation. However, these models are computationally and memory expensive, requiring multiple denoising steps and complex backbones, which limit their deployment in resource-constrained and edge devices. To address these

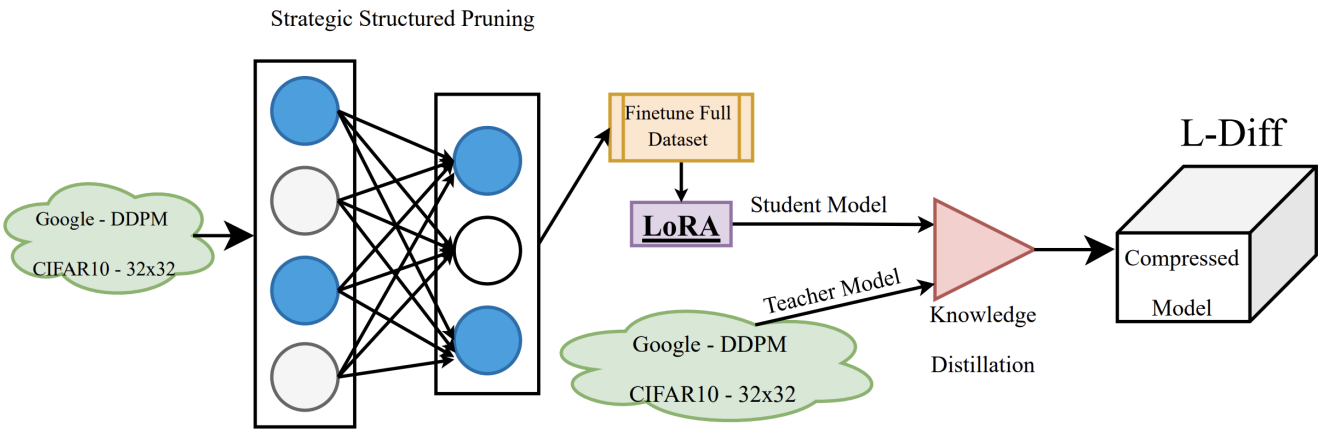


Figure 1. LiteDiffusion Pipeline. 1. Strategic structured pruning. 2. Finetune with Full CIFAR10. 3. Apply LoRA. 4. Use the LoRA Output weights as student model for knowledge distillation. 5. Knowledge Distillation with Student as the output weights after LoRA step, and teacher model as the Base DDPM CIFAR10 32 model [19]. 6. L-Diff compressed model.

issues, researchers have explored pruning and quantization techniques to create compact diffusion models that reduce the model size and computation while also generating high quality images.

2.1. DDPM models

In DDPMs, the denoising neural network is often implemented using a U-Net architecture due to its ability to capture both local and global image features effectively. The U-Net consists of a symmetric encoder-decoder structure with skip connections that merge feature maps from corresponding layers, enabling the preservation of fine-grained details while reconstructing high-level representations. During the reverse diffusion process, the U-Net predicts the noise to be subtracted at each timestep, conditioned on the noisy input image and a timestep embedding. This design allows the model to efficiently model complex data distributions across multiple scales, resulting in the generation of high-quality images. The high level architecture and concept of UNet DDPM is showcased in figure [2].

2.2. Pruning Techniques

Pruning is the process of removing unimportant parameters or layers from a neural network to reduce computational and memory overhead. In the context of diffusion models, structural pruning typically removes entire channels, layers, or timesteps instead of individual weights, targeting larger, redundant components. Recent work by *Diff-Pruning* [3] applies Taylor expansion to estimate the importance of parameters by analyzing their gradients across timesteps. This method takes advantage of the fact that earlier timesteps in the diffusion process contribute significantly to the overall structure of generated images, while later timesteps fine-tune details. As a result, Diff-Pruning

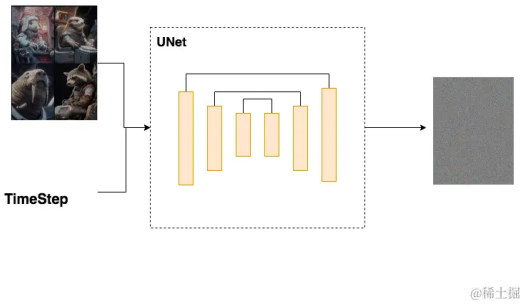


Figure 2. UNet Structure DDPM. Image is downsampled and then upsampled to create a noisy image in the diffusion process. Image is recreated using the reverse diffusion process. [18]

focuses on pruning the later timesteps with minimal impact on quality, achieving up to a 20% reduction in computational cost. The study compares several pruning strategies, including random pruning, magnitude pruning, and Taylor-based pruning, on the CIFAR-10 dataset. Random pruning, which removes parameters in a non-deterministic fashion, achieves FID 5.62 but lacks the consistency needed for generative tasks. Magnitude pruning, which removes parameters with smaller weights deemed less important, performs slightly better with FID 5.48. Taylor pruning achieves the best results with FID 5.49, indicating its effectiveness in preserving essential features during pruning.

Another approach to pruning diffusion models is the use of step-aware, slimmable networks, which dynamically adjust the model's capacity at each timestep [21]. Yang, et al. achieved results comparable to Diff-pruning paper by Fang, et al. in terms of FID with a superior computational with 3.10 GMACs against 3.12 GMACs of Diff-

Pruning on CelebA. In this method, the model activates only the necessary sub-networks at each stage of the denoising process, using larger configurations for the initial steps that define global structure and smaller configurations for later steps that refine details. This fine-grained control over model size ensures that only the required computational resources are used at each timestep, significantly optimizing inference efficiency without sacrificing generation quality. By leveraging a slimmable architecture, step-aware pruning achieves flexible, efficient pruning that adapts to the diffusion model's progressive generation process, further reducing overhead.

2.3. Quantization Techniques

Quantization reduces the precision of model parameters and activation values, usually from 32-bit floating-point to lower precision values such as 8-bit or 4-bit. Post-training quantization (PTQ) converts pre-trained models directly into a lower precision format without requiring additional training. However, diffusion models present difficulties for quantization. Quantization errors can accumulate over the multiple iterative denoising steps, which increases the risk of performance degradation [16]. Methods such as *Q-DM* [15] leverage mixed-precision quantization, ensuring higher precision is used where needed, reducing errors without large performance losses. PTQ4DM [20] addresses the unique challenges of quantizing diffusion models by employing a normally distributed timestep calibration (NDTC) technique. NDTC carefully selects representative samples from various timesteps to capture the distribution of activations, which helps minimize accumulated quantization errors across denoising steps. This calibration process allows PTQ4DM to effectively maintain model quality even at lower bit precision.

2.4. Knowledge Distillation

Knowledge Distillation (KD) [9] is a model compression technique that transfers knowledge from a large, complex "teacher" model to a smaller, more efficient "student" model. In KD, the student model is trained not only on the original dataset but also on the soft targets generated by the teacher model. These soft targets are the teacher's output probabilities, which contain rich information about the inter-class relationships learned during training. By minimizing the divergence between the student's and teacher's output distributions, the student model learns to replicate the teacher's performance. This approach enables the student to capture the intricate patterns and generalization capabilities of the teacher while significantly reducing model size and computational complexity. Consequently, KD facilitates the deployment of high-performing models in resource-constrained environments without substantial loss in accuracy.

2.5. LoRA: Low Rank Adaptation

Low-Rank Adaptation (LoRA) [11] operates by decomposing the weight matrices of neural networks into low-rank representations, effectively reducing the number of parameters needed for training. Instead of updating the full weight matrices, LoRA introduces low-rank matrices that capture essential features of the data. By approximating the original weights with the product of smaller matrices, LoRA significantly lowers computational and memory requirements. This technique enables efficient fine-tuning of large models, as only the low-rank adaptation matrices are trained while the original weights remain frozen, thus maintaining performance with reduced resource consumption. TimeLoRA [2], introduced by Ryu et al., is an innovative adaptation of Low-Rank Adaptation (LoRA) tailored specifically for Denoising Diffusion Probabilistic Models (DDPMs). It addresses the inefficiencies of traditional time embeddings in diffusion models by replacing them with low-rank adaptation matrices. Instead of relying on large, static time embeddings, TimeLoRA dynamically learns compact representations that are optimized for the denoising process. This significantly reduces the number of trainable parameters while maintaining or even enhancing the model's performance.

3. Baselines and Approach

Our aim is to explore different pruning, quantization methods in combination with knowledge distillation and low-rank approximation, and compare it with selected compact models as well as the full-sized models. Several baseline approaches exist for optimizing diffusion models through pruning and quantization. We compare our models with the baseline to be the full google's DDPM model trained on CIFAR-10 from huggingface. We also compare our model with the following baseline models as well:

- **Diff-Pruning**: Uses Taylor expansion to identify and remove less important parameters across multiple timesteps. They finetune the Diff-Pruned DDPM for 100K train steps achieving a 42% reduction in model size with a 32% increase in FID-score [3].
- **PTQ4DM**: Introduces post-training quantization (PTQ) to convert diffusion models into 8-bit versions. On CIFAR-10 with 4000 DDPM steps, the 8-bit quantized model achieves an IS of 9.55 and FID of 7.10, slightly outperforming the full-precision model's IS of 9.28 and FID of 7.14. For fewer steps (250 DDIM steps), the quantized model performs close to the full-precision version with a minor increase in FID, showcasing its efficiency while maintaining high-quality image generation [20].

****Note: Although we include Diff-Pruning and PTQ4DM as our baseline comparison, the computational power for extensive training was not matched while conducting the experiments included in this paper.**

Our results would not be comparable to Diff-Pruning because of the large amount of trainsteps and inference steps using the best available hardware, 100K specifically for 10000 evaluation images. Further, our results would not be comparable with PTQ4DM because of the more complex UNet2D structure with 3 Resnet blocks (google/ddpm-cifar10-32 [19] contains 2 Resnet blocks) and large amount of training time steps and inference time steps they employ on the best available hardware, 4000 specifically for 10000 evaluation images.
Hence, we generate 2500 images from the available google DDPM Base CIFAR10 and it would be more valid to compare performance of our experimental models. Additionally, with the resources available at our disposal, we generate 2500 images to calculate FID scores which would further improve with more generated images.
Furthermore, in order to stay consistent with google's DDPM-CIFAR10-32, we keep the hyperparameters in DDPM configuration constant throughout our experimental models. Specifically, we generate images and train our models with 1000 inference time steps and training time steps, respectively. For finetuning, training steps are 19.5K

4. Methods

In this section we describe how we used each of pruning, quantization, knowledge distillation, and LoRA to make our model compact as well as achieve quick finetuning/inference times.

4.1. Pruning

We experiment with different well-known pruning techniques to evaluate their effectiveness on the reduction of model size and increasing efficiency while maintaining performance. We focus on post-training (testing-time) pruning to understand the pros and cons of the different pruning techniques to ensure the most applicable and effective technique is used for L-Diff. For this section refer to table [1].

L1 unstructured pruning [7] removes weights based on their L1 norm, setting a fraction of the lowest-magnitude weights to zero across the network without altering the structure. This method showed the best results, as it preserves all the weights except for pruning% lowest weights:

$$w_{i,j} = 0 \quad \text{for} \quad |w_{i,j}| \leq \text{threshold} \tag{1}$$

Although this method showed the most promising FID scores, it failed to reduce the model size since it only changes the weights lower than the threshold to 0 rather than entirely deleting neurons. Consequently, computation complexity and physical size of the model remains the same.

Structured pruning [14] removes a portion of weights in a layer-by-layer, structured manner (e.g., entire filters), resulting in a direct reduction of model size. Currently, we remove the first 30% of the weights of every layer without any regard to importance. To improve performance, we plan to sort and prune the least important 30% in each layer:

$$w_i = 0 \quad \text{for} \quad w_i \in \text{sorted weights}_{\text{layer}} \tag{2}$$

Disregarding the pre-finetune FID score of this method, we see that this method effectively reduces the total parameters and as a result the model size. Consequently, it would reduce the computational complexity per inference.

Random unstructured pruning [4] removes weights randomly across the network, leading to the worst performance due to the potential loss of important weights. Additionally, this method shares the same drawbacks with respect to the size as the unstructured pruning method.

Global unstructured pruning [5] zeroes out a fraction of the lowest-magnitude weights across the entire network, enhancing sparsity while maintaining the model's architecture:

$$w_{i,j} = 0 \quad \text{for} \quad |w_{i,j}| \leq \text{global threshold} \tag{3}$$

As shown in table [1], this method also shares the same drawbacks as Random Unstructured pruning and Unstructured pruning

For L-Diff, we leverage structured pruning for its significant reduction in model size. To counteract the worsening performance of this method, we use fine-tuning to ensure the resulting pruned model weights are adjusted to the CIFAR10 distribution to offset the lost feature weights. Loss function for finetuning the pruned model is mean squared error loss [1]. We use Adam optimizer [13] with cosine annealing learning rate with warm restarts [17] to improve the rate of convergence.

Employing structured pruning strategically for the modern DDPM-CIFAR10-32 is important since it is a highly time-dependent neural network. We prune the attention and convolution layers and preserve the time-embedding layers. We hypothesize that this would prove to be advantageous in preserving necessary information for the sequential denoising process of DDPMs.

4.2. Quantization

We employed dynamic quantization to reduce the model's size and improve inference efficiency. Dynamic quantization modifies the data representation of specific layers such as nn.Conv2d and nn.Linear by converting their weights and activations to lower-precision types (in this case, from float32 to int8). This process occurs dynamically during inference, enabling faster computation without requiring retraining of the model. The transformation is handled by PyTorch's torch.quantization.quantize_dynamic function, which selectively targets layers based on their computational cost and impact on model performance.

Pruning Method	Pre-finetune FID	Model Size	Total Parameters	Sparsity	Pruned Amount
Base DDPM	31.24	143.11 MBs	35.7463 M	0%	30%
Unstructured	53.53	143.11 MBs	35.7463 M	30%	30%
Structured	287.87	77.91 MBs	20.3793 M	0%	30%
Random Unstructured	357.51	143.11 MBs	35.7463 M	30%	30%
Global Unstructured	42.34	143.1 MBs	35.7463 M	26.21%	30%

Table 1. Pruning Method Comparison. Test well-known pruning techniques for Level-1 testing to compare Model size, total parameters reduction and information loss with Pre-finetune FID.

The key advantage of dynamic quantization is that it retains the original precision of the model weights when they are stored but converts them to lower precision during execution. This makes it a lightweight and efficient approach for models where performance is sensitive to accuracy degradation, as it avoids introducing significant approximation errors.

Model	FID	Train timesteps
PTQ4DM **	7.14	4000
Base DDPM Model	31.136	1000
Base DDPM + qint8	56.88	1000

Table 2. Quantization types exploration. This table describes the results of applying quantization to base DDPM Model and its performance as compared with the Base DDPM Model. ** Refer to note in Section 3.

We successfully quantize Base DDPM model by converting weights from float32 to int8. According to table 2, we see that the converting float32 weights to int8 weights, the performance worsens. Finetuning a quantized model with float16 and float8 weights is not compatible with the way GPUs perform operations.

4.3. Knowledge Distillation

Model	FID	Model Size	GMACs
Base DDPM Model	31.136	143.16 MBs	6.0641
Base DDPM + KD (s=2)	63.84	23.654 MBs	0.5116
Base DDPM + KD (s=4)	103.633	6.091 MBs	0.1526

Table 3. Fraction of channel reduction from Teacher Model (Base DDPM Model) to distill knowledge in the Student Model (Base DDPM + KD). 's' is the channel division factor. FID lower is better. GMACs lower means less operations for inference and better computational complexity.

The "teacher" model in this KD setup is the pretrained UNet model used within the DDPM pipeline, which is trained to predict noise in the diffusion process. To achieve a smaller and more efficient "student" model, we employed size reduction through scaling factors. These factors determine the number of channels and other architectural dimensions in the student model, resulting in a compact design that requires fewer parameters and computational resources. For instance, a scaling factor of 4 was used, meaning the number of channels and certain layers were proportionally reduced to one-fourth of the original size.

The KD process was guided by a loss function combining two components:

Base Loss: A mean squared error (MSE) loss between the true noise added during diffusion and the noise predicted by the student model. Distillation Loss: An MSE loss between the teacher model's predictions and the student model's predictions, ensuring the student model closely approximates the teacher's behavior. These losses were weighted using a hyperparameter kd_frac (set in the range 0.6-0.8 in this case), which determines the balance between learning from the true labels and mimicking the teacher's predictions.

From 3, we see that upon applying KD, with increasing factors of channel reduction, the model becomes significantly worse off in terms of performance measured by FID, however, the model also significantly reduces in size. This means that KD is unable to transfer enough knowledge due to overly simple architecture of the halved and quartered models.

4.4. LoRA

Model	FID	Model Size	GMACs
Base DDPM Model	31.136	143.16 MBs	6.0641
Base DDPM + LoRA (r=2)	226.65	30.99 MBs	0.0067
Base DDPM + LoRA (r=4)	246.75	15.525 MBs	0.0067
Base DDPM + LoRA (r=6)	252.22	10.21 MBs	0.0067
Base DDPM + LoRA (r=8)	328.64	7.791 MBs	0.0067

Table 4. LoRA applied to the base DDPM Unet Model at different values of rank proportion factor r.

LoRA was utilized to efficiently fine-tune the UNet model for generating CIFAR10 images while significantly reducing the number of trainable parameters. LoRA mod-

ifies a pre-trained model by adding low-rank trainable weights, allowing fine-tuning with minimal computational cost and memory overhead.

The core idea behind LoRA is to decompose weight updates into low-rank matrices during fine-tuning. In this setup:

- LoRA Layers were introduced for both fully connected (nn.Linear) and convolutional (nn.Conv2d) layers in the UNet architecture.
- Two low-rank matrices, A and B, were used to approximate the weight updates during training. The rank of the weights matrix for each layer was determined using a rank proportion factor r. (Eg. with r=2, we approximated the weights matrix of each layer using the original full rank divided by 2; the rank of each layer is half of its original rank). We tested with various values of r ranging from 2 to 10. We observed that for higher values of r (lower rank), the FID scores progressively worsened.
- A scaling factor alpha (set to 1.0) was applied to scale the weight updates, ensuring numerical stability during training.

The original model weights were frozen, and only the LoRA parameters (A and B) were updated during training and used for image generation. This approach enabled efficient adaptation of the pre-trained model without the need to retrain or store the full set of weights, making it highly suitable for resource-constrained environments.

From 4, we can see that as the rank division factor (r) increases, the performance is significantly worse off. The initial application of LoRA majorly increases the FID of the images. This means that the LoRA fails to preserve important temporal information necessary for the sequential denoising process.

5. Results

Table 1 summarizes the impact of different pruning methods on FID, model size, and sparsity. We selected strategic structured pruning as our chosen method of model compression. Refer to section 4.1 for more information on strategic structured pruning for Base DDPM.

From table 5, we observe that our strategic structured pruning technique successfully creates pruned models without much loss in information. The results in the table demonstrate a clear pattern where increasing the structured pruning rates leads to significant reductions in model size, total parameters, and computational complexity (as indicated by GMACs), while only causing a modest increase in the Post-finetune FID scores. Specifically, pruning up to 70% of the neurons results in the model size shrinking from 143.1 MB to 28.80 MB and total parameters decreasing from 35.75 million to 7.51 million compared with the baseline. From 10% pruned model to 70% pruned model, the FID ranges from 37.44 to 38.15. The FID scores do showcase an increasing pattern as prune% increases, however, the range of min-max difference is very low while significantly reducing model sizze from 115.07 MBs (10%) to 28.80 MBs (70%). Overall, despite this substantial compression, the Post-finetune FID score increase only from 31.24 to a maximum of 38.15 occuring at 60% pruned model, indicating that the quality of image generation remains relatively high. This pattern highlights that strategic structured pruning effectively removes redundant or less important neurons, achieving a balance between efficiency and performance. The pruned models maintains a competitive FID score even at high pruning rates, showcasing that significant size reductions can be attained without substantially compromising the generative capabilities of the DDPM.

Table 6 presents the various combinations of compression techniques we explored. Analyzing the results reveals clear patterns that emerge with the addition of each technique. In our knowledge distillation experiments (see Table 4.3), reducing the number of channels significantly decreased both the GMACs and the model size. However, this reduction came with substantial performance loss as the channel reduction factor increased. Applying quantization to the knowledge-distilled models with scaling factors s=2 and s=4 led to further degradation in performance without notable reductions in model size or GMACs. Consequently, we opted to use s=1 for the remainder of our experiments.

We also applied quantization to the previously mentioned pruned and fine-tuned models. The FID scores for these quantized models were significantly worse than those of the original models before quantization, suggesting a considerable amount of information loss. Additionally, as shown in Table 5, quantization of pruned models did not significantly reduce the model size, and the GMACs remained unchanged. This is because quantization changes the representation of weights from 32-bit floating-point numbers to lower bit-width numbers without altering the overall architecture or the number of operations.

To further improve the performance of the compressed models, we employed knowledge distillation using the pruned model as the student and the base DDPM model as the teacher, denoted as KD(Teacher, Prune(Base, pruning ratio, s=1)). As we increased the pruning ratio, we observed a gradual decline in performance, as indicated by the FID scores. The optimal setting for this combination was achieved with a 50% pruned base DDPM model, resulting in an FID of 46.61, a model size of 60.8 MB, and GMACs of 2.357. In this configuration, the model preserved most of the crucial information while achieving significant reductions in both model size and computational complexity.

We applied quantization to the knowledge-distilled pruned models using the same approach KD(Teacher, Prune(Base, pruning ratio, s=1)). This procedure exhibited trends similar to those observed in the previously quantized

Pruning Method	Pruned Amount	Post-finetune FID	Model Size	Total Parameters	GMACs	Trainsteps
Diff-Prune	**44.5%***	**5.29**	**Not Reported**	**19.8 M**	**3.4**	**100K**
Base DDPM	0%	31.24	143.1 MBs	35.7463 M	6.0641	19.5K
Structured	10%	37.44	115.07 MBs	30.1202 M	4.8772	19.5K
Structured	20%	37.90	97.30 MBs	25.4615 M	4.2880	19.5K
Structured	30%	38.00	77.91 MBs	20.3793 M	3.1995	19.5K
Structured	40%	37.60	62.00 MBs	16.2101 M	2.6886	19.5K
Structured	50%	37.73	59.94 MBs	15.6703 M	2.6574	19.5K
Structured	60%	38.15	42.66 MBs	11.1403 M	1.6707	19.5K
Structured	70%	38.00	28.80 MBs	7.5075 M	1.2449	19.5K

Table 5. Structured pruning results with magnitude importance for increasing pruning rates: Pruned amount is the total percentage of neurons in the UNET DDPM that are deactivated/removed. Post-finetune FID is to measure the quality of the image generation. Lower post-finetune FID is better. Model size reports the size reduction post pruning. Total parameters reports the unpruned parameters in the network. GMACs is Giga Multiply-Add Operations per Second. Lower GMACs means less computations required to inference. Trainsteps is the total gradient update steps. Higher Trainsteps means convergence to a more stable value.

Model	FID	Model Size	GMACs
Base DDPM Model (BM)	**31.136**	143.16 MBs	6.0641
Quant(KD(BM, s=2), int8)	102.884	23.05 MBs	0.5036
Quant(KD(BM, s=4), int8)	145.035	5.85 MBs	0.150
Quant(Prune(BM, p=0.1), int8)	88.739	109.35 MBs	4.8772
Quant(Prune(BM, p=0.2), int8)	86.537	93.97 MBs	4.2880
Quant(Prune(BM, p=0.3), int8)	89.81	73.93 MBs	3.1995
Quant(Prune(BM, p=0.4), int8)	98.631	58.82 MBs	2.6886
Quant(Prune(BM, p=0.5), int8)	94.34	57.73 MBs	2.6574
Quant(Prune(BM, p=0.6), int8)	139.53	40.95 MBs	1.6707
KD(BM, Prune(BM, p=0.1), s=1)	41.150	120.60 MBs	4.6772
KD(BM, Prune(BM, p=0.2), s=1)	43.4044	81.64 MBs	3.1995
KD(BM, Prune(BM, p=0.3), s=1)	43.33	76.31 MBs	2.8952
KD(BM, Prune(BM, p=0.4), s=1)	57.30	64.96 MBs	2.2478
KD(BM, Prune(BM, p=0.5), s=1)	46.61	60.80 MBs	2.1346
KD(BM, Prune(BM, p=0.6), s=1)	81.945	44.68 MBs	1.2936
Quant(KD(BM, Prune(BM, p=0.1), s=1), int8)	80.739	109.32 MBs	4.6772
Quant(KD(BM, Prune(BM, p=0.2), s=1), int8)	79.537	81.93 MBs	3.1995
Quant(KD(BM, Prune(BM, p=0.3), s=1), int8)	82.81	73.93 MBs	2.8952
Quant(KD(BM, Prune(BM, p=0.4), s=1), int8)	98.631	58.83 MBs	2.2478
Quant(KD(BM, Prune(BM, p=0.5), s=1), int8)	83.34	57.73 MBs	2.1346
Quant(KD(BM, Prune(BM, p=0.6), s=1), int8)	167.53	40.95 MBs	1.2936
KD(BM, LoRA(Prune(BM, p=0.1), s=1)), r=2)	321.17	25.71 MBs	0.0059
KD(BM, LoRA(Prune(BM, p=0.2), s=1)), r=2)	354.21	17.24 MBs	0.0047
KD(BM, LoRA(Prune(BM, p=0.3), s=1)), r=2)	326.43	17.14 MBs	0.0047
KD(BM, LoRA(Prune(BM, p=0.4), s=1)), r=2)	355.25	13.73 MBs	0.0045
KD(BM, LoRA(Prune(BM, p=0.5), s=1)), r=2)	352.098	12.853 MBs	0.0045
KD(BM, LoRA(Prune(BM, p=0.6), s=1)), r=2)	435.21	12.23 MBs	0.0042

Table 6. Combinations of model compression techniques explored to select best pipeline. Model column describes sequential utilization of the mentioned techniques. Knowledge Distillation = KD(Teacher model, student model[optional], s=Factor of channel reduction). Structured prune = Prune(Model, p=pruning ratio). Quantization = Quant(Model, conversion type). Low Rank Adaptation = LoRA(Model, r=rank proportion factor). For FID, Model Size and GMACs preferred trends refer to previous tables or Section 1.

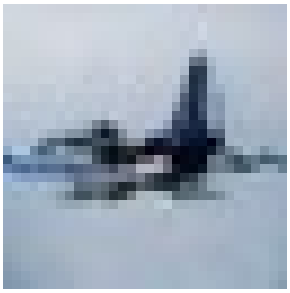
models, with a significant increase in FID scores and minimal reductions in model size. The GMACs remained constant, indicating that quantization alone did not effectively enhance the compression of these models.

Finally, when we applied Low-Rank Adaptation (LoRA) to the models, we observed substantial reductions in computational complexity and model sizes. However, this reduction was accompanied by a disproportionately large increase in FID scores, signifying a severe deterioration in image generation quality. The addition of LoRA did not improve performance in the context of compression; instead, it resulted in unacceptable performance loss. Notably, the GMACs and model sizes achieved with LoRA were significantly smaller than those obtained with the knowledge-distilled pruned models KD(Base, Prune(Base, p), s=1). This indicates that while LoRA is highly effective in compressing the model to a greater extent, it compromises critical information necessary for maintaining performance.

These findings highlight that while individual compression techniques like pruning and quantization can reduce model size and computational load, their combination does not always lead to proportional performance improvements. Strategic structured pruning proved effective in significantly reducing model sizes without substantial compromise in performance. However, quantization, especially when applied to already compressed models, may introduce considerable information loss without providing additional benefits in terms of computational efficiency. Similarly, although LoRA can achieve greater compression, it may do so at the expense of model fidelity, underscoring the need for a balanced approach when selecting compression techniques for diffusion models.

6. Conclusion

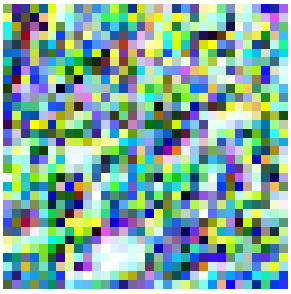
In this work, we explored various model compression techniques to optimize Denoising Diffusion Probabilistic Models (DDPMs) for image generation on the CIFAR-10 dataset, with the aim of deploying these models on



(a) Image generated with Quant(Prune(BM, p=0.5), int8) model

(b) Image generated with KD(BM, Prune(BM, p=0.5), s=1) model

(c) Image generated with Quant(KD(BM, Prune(BM, p=0.5), s=1), int8) model



(d) Image generated with KD(BM, LoRA(Prune(BM, p=0.5), s=1)), r=2)

Figure 3. Examples of the generated images by the bolded model names in Table 6. These models showcase reasonable preservation of Base DDPM CIDAR10-32 model as seen from FID after pruning 50% of the neurons in convolutions and attention layers and 0% prune in the time embedding layers of the DDPM.

resource-constrained edge devices. We investigated different pruning methods—including L1 unstructured pruning, Ln structured pruning, and global unstructured pruning—to balance the trade-off between model size reduction and performance retention.

Our findings indicate that strategic structured pruning, particularly Ln structured pruning, is the most effective method for compressing diffusion models without substantial loss in image generation quality. By selectively pruning less critical components such as attention weights and convolutional layers while preserving essential layers like time embeddings, we achieved significant reductions in model size and computational complexity. The time embedding layers were identified as crucial for maintaining the temporal coherence necessary for the sequential denoising process inherent in DDPMs. Preserving these layers ensures that the model's foundational ability to guide the denoising process at each timestep remains intact.

Moreover, combining structured pruning with fine-tuning and knowledge distillation further enhanced the compressed model's performance. Fine-tuning allowed the pruned model to adjust to the loss of certain parameters, mitigating potential degradation in image quality. Knowledge distillation, using the original uncompressed model as the teacher, helped the pruned student model learn to approximate the teacher's performance more closely. This combination proved superior to other compression techniques we evaluated, such as quantization and the application of Low-Rank Adaptation (LoRA), which either did not yield significant improvements or resulted in unacceptable performance loss.

The implications of our work for model compression are threefold:

- **Targeted Pruning Strategies:** By focusing on preserving layers that encode essential information—specifically, the time embedding layers—we demonstrated that selective pruning can lead to more effective compression without compromising the model's integrity. This approach allows for a substantial reduction in model size and computational requirements while maintaining high performance in image generation tasks.
- **Designing for Edge Deployment:** The successful application of strategic structured pruning makes it feasible to deploy DDPMs on resource-constrained devices such as smartphones and other edge platforms. This enables real-time image generation applications, providing offline capabilities that preserve user privacy and reduce reliance on cloud computing resources.
- **Future Research Directions:** Our study highlights the importance of understanding the roles of different layers within diffusion models. Further exploration of which layers can be pruned with minimal impact on performance can inform the development of more efficient architectures and advanced compression techniques. Future work will focus on fine-tuning these methods to enhance efficiency further while preserving image generation quality, as well as integrating quantization techniques where appropriate. Last but not the least, LoRA is a very powerful method in neural network compression because of its efficiency in representing massive weights in very small dimensions and hence, space and computation complexity. This further encourages us to explore the reason of losing important information and how to counteract it.

In summary, the strategic structured pruning method, complemented by fine-tuning and knowledge distillation, emerged as the most effective approach for compressing diffusion models in our experiments. This method strikes an optimal balance between reducing model complexity and retaining high-quality image generation, making advanced generative models more accessible for practical, real-world applications on edge devices. Our work lays the groundwork for future advancements in model compression techniques, with the potential to broaden the deployment of sophisticated machine learning models across a wide range of resource-constrained environments.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 4

[2] Joo Young Choi, Jaesung Park, Inkyu Park, Jaewoong Cho, Albert No, and Ernest K. Ryu. Lora can replace time and class embeddings in diffusion probabilistic models, 2023. 3

[3] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models, 2023. 2, 3

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 4

[5] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019. 4

[6] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016. 1

[7] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 1135–1143, 2015. 4

[8] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30, pages 6626–6637, 2017. 1

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 3

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. 1

[11] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 1, 3

[12] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 1

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[14] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. 4

[15] Yanjing Li, Sheng Xu, Xianbin Cao, Xiao Sun, and Baochang Zhang. Q-DM: An efficient low-bit quantized diffusion model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 3

[16] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction, 2021. 3

[17] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. arXiv preprint arXiv:1608.03983. 4

[18] Author's Name. Cv model series: Proliferation model cornerstone ddpm (source code interpretation and practice), 2024. 2

[19] Google Research. Pre-trained ddpm model for cifar-10. https://huggingface.co/google/ddpm-cifar10-32, 2020. Accessed: November 13, 2023. 2, 4

[20] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models, 2023. 3

[21] Shuai Yang, Yukang Chen, Luozhou Wang, Shu Liu, and Yingcong Chen. Denoising diffusion step-aware models, 2024. 2