Mixture of Efficient Diffusion Experts Through Automatic Interval and Sub-Network Selection

Alireza Ganjdanesh[1⋆], Yan Kang[2], Yuchen Liu[2], Richard Zhang[2], Zhe Lin[2], and Heng Huang[1]

[1] Department of Computer Science, University of Maryland College Park
[2] Adobe Research
{aliganj,heng}@umd.edu, {yankang,yuliu,rizhang,zlin}@adobe.com

Abstract. Diffusion probabilistic models can generate high-quality samples. Yet, their sampling process requires numerous denoising steps, making it slow and computationally intensive. We propose to reduce the sampling cost by pruning a pretrained diffusion model into a mixture of efficient experts. First, we study the similarities between pairs of denoising timesteps, observing a natural clustering, even across different datasets. This suggests that rather than having a single model for all time steps, separate models can serve as "experts" for their respective time intervals. As such, we separately fine-tune the pretrained model on each interval, with elastic dimensions in depth and width, to obtain experts specialized in their corresponding denoising interval. To optimize the resource usage between experts, we introduce our Expert Routing Agent, which learns to select a set of proper network configurations. By doing so, our method can allocate the computing budget between the experts in an end-to-end manner without requiring manual heuristics. Finally, with a selected configuration, we fine-tune our pruned experts to obtain our mixture of efficient experts. We demonstrate the effectiveness of our method, *DiffPruning*, across several datasets, LSUN-Church, LSUN-Beds, FFHQ, and ImageNet, on the Latent Diffusion Model architecture.

Keywords: Efficient Deep Learning · Model Pruning · Diffusion Models

1 Introduction

Diffusion Probabilistic Models (DPMs) [28, 55, 57] have become the de facto models for generative modeling applications like image synthesis [10, 28], image editing [68, 71], super-resolution [18, 53], and video generation [27]. They train a denoising model that learns to generate samples from an input noise in an iterative denoising scheme. DPMs have achieved better mode coverage and training stability [10] than GANs [20] and show higher sample quality than VAEs [33]. Yet, the main drawback of DPMs is their slow and computationally intensive sampling process, making their cloud deployment costly and hindering usage on resource-constrained edge devices.

⋆ Part of this work was done during an internship at Adobe Research.

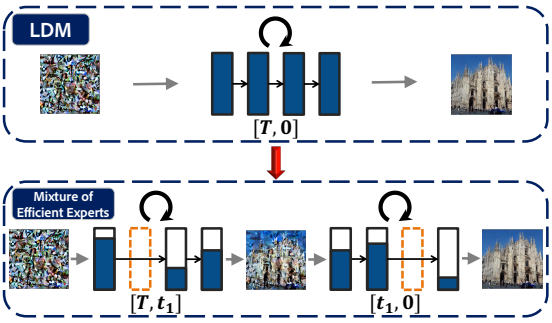

Fig. 1: Overview of DiffPruning. We prune a pre-trained LDM model [50] (top) into a mixture of efficient experts (bottom). Each expert handles an interval, which allows their architectures to be separately specialized by removing layers or channels.

Two important factors contribute to slow sampling in DPMs: the models use 1) a large number of denoising steps and 2) a large number of parameters in each denoising step. Methods to speed up DPMs have primarily focused on reducing the sampling steps, using techniques like faster solvers [2, 39, 41, 69], better noise schedules [47, 56], and distillation [22, 45, 54]. In an orthogonal direction, a group of methods address the second factor and develop more efficient architectures for DPMs. Latent Diffusion Models (LDMs) [50] perform the diffusion process in a latent space with lower dimensions than pixel space, thereby significantly speeding up the sampling process while retaining a competitive performance. Accordingly, LDMs have been deployed in modern generative models like DALL-E 3 [4] and Stable Diffusion [50]. Thus, compressing LDMs is of significant interest. As LDMs do not have redundancies of the pixel-space DPMs by design, pruning them is much more challenging than pruning pixel-space DPMs.

Recently, several works [34, 37, 67] have explored architectural efficiency for LDMs. They divide the denoising path of an LDM into several intervals and use a distinct model for each one. These methods [34, 37, 67] are mainly inspired by studies [1, 64] showing different timesteps have distinct roles in the denoising process, and employing a single denoising model for all timesteps is sub-optimal [1, 19]. Thus, the key design choices here are the clustering scheme of the denoising timesteps and the method for allocating the resource budget between the selected clusters. MEME [34] uses uniform clustering, and TMDA [67] clusters the denoising timesteps by their loss values' similarities. Both MEME [34] and TDMA [67] manually design a distinct U-Net model [51] for each cluster, thereby heuristically allocating the resource budget between the denoising intervals. However, by doing so, these methods need to re-design intervals' models for a new distinct budget, which is a complex, time-consuming, and labor-intensive task. OMS-DPM [37] avoids manual designing intervals' models as it trains a model zoo with different sizes and searches for an optimal mixture of denoising models, given a desired computational budget. Still, training a model zoo

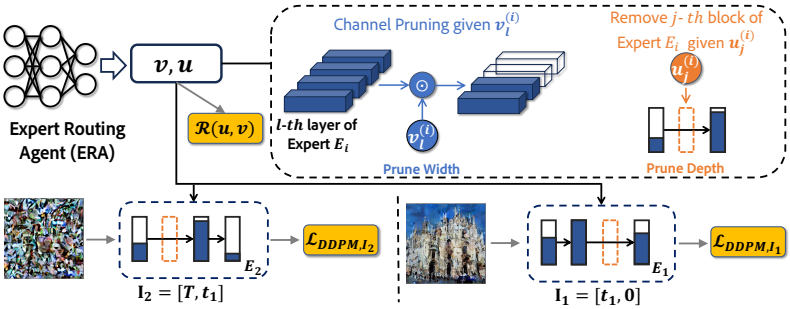

Fig. 2: Our Pruning Scheme: We train our Expert Routing Agent (ERA) to prune the experts into a mixture of efficient experts (Sec. 2.3). The ERA predicts the architecture vectors (v, u) to prune experts' width and depth. Then, we calculate the denoising objectives of selected sub-networks of experts, $\mathcal{L}_{\text{DDPM},\mathcal{I}_i}$, as well as our Resource regularization term, $\mathcal{R}$, that encourages the ERA to provide a mixture of efficient experts with a desired compute budget (MACs). We train ERA's parameters to minimize the objective functions. Thus, it learns to automatically allocate the compute budget (MACs) between experts in an end-to-end manner.

of various LDMs is extremely costly, even for medium-sized datasets, making OMS-DPM expensive to deploy in practice.

In this paper, we propose a novel approach to make LDMs more efficient by pruning a large pretrained LDM into a mixture of efficient experts (Fig. 1) in four steps. First, we find an optimal division of time intervals by studying how aligned pairs of denoising steps are to each other in a pretrained LDM. Interestingly, while different datasets all show natural clustering, the exact time intervals differ slightly between them. Thus, we adapt our clustering depending on the behavior of the dataset rather than using a static approach across datasets as in previous work [34]. Second, we fine-tune the pretrained model with elastic depth and width on each interval so that the sub-networks of the resulting model have a strong performance on that interval. We denote the elastically fine-tuned models as *experts* for the intervals. Our elastic fine-tuning provides an 'implicit' model zoo within each expert for its corresponding interval with fewer training iterations than training multiple models from scratch like OMS-DPM [37]. Third, we develop an Expert Routing Agent (ERA) that learns to select proper network configurations for the experts simultaneously, guided by the sub-networks' denoising objectives and allocated compute resource (*e.g.,* MACs). As we train our ERA in an end-to-end manner, it can automatically allocate computing resources between the experts without the need for complex heuristics [34, 67]. We summarize our contributions as follows:

- We introduce a method for pruning LDMs into a mixture of efficient experts.
- We propose to cluster denoising timesteps of a pretrained LDM into several intervals based on their pairwise alignment scores, showing that the optimal

clustering intervals are distinct for different datasets. We employ a specialized efficient model for each interval.
- We fine-tune the pretrained LDM on selected intervals with elastic dimensions so that resulting expert models will have strong sub-networks to choose from. Thus, we can readily prune the experts for different computational budgets, and the pruned experts can properly recover their performance without long fine-tuning iterations.
- We develop a new pruning scheme in which our expert routing agent learns to select optimal layouts of the experts together in an end-to-end manner, thereby allocating the compute budget between experts automatically.

2 Related Work

Mixture of Experts (MoE) diffusion models. MoE methods cluster denoising timesteps of DPMs into intervals and train a separate *expert* model for each. eDiff-I [1] supports developing MoE for DPMs by showing that different denoising timesteps have separate roles. Yet, how to cluster timesteps is non-trivial. eDiff-I employs a tree-based-branching scheme, sequentially dividing the denoising path into two intervals and initializing a child model by its parent. ERNIE-ViLG [13] and MEME [34] uniformly cluster the denoising timesteps. Yet, these heuristic schemes do not necessarily transfer to other tasks. Alternatively, we propose to cluster denoising timesteps in a data-driven way by measuring the alignment between their training objectives. We observe that the optimal cluster assignments are different for distinct datasets. We note that although NT [19] has explored timesteps' alignment scores *in the course of training*, our paper is the first one to leverage *post-training scores* to cluster the timesteps for MoE DPMs.
Efficient DPMs. Ideas for improving DPMs' efficiency mostly reduce their denoising steps by faster samplers [42, 63, 69], distillation [22, 45, 54], better noise schedules [2, 32, 47, 56, 70, 73], learning denoising steps [60, 61], and caching [43]. We explore an orthogonal direction, compressing DPMs' architectures.

A few ideas have recently addressed compressing DPMs' architectures having two main categories. **Single-model** methods develop a single efficient model for all denoising timesteps. SP [12] approximates weights' importance using the Taylor expansion and removes structures with low scores. Yet, SP's performance has been mainly verified on pixel space DPMs, and its pruned models on datasets like LSUN-Church [66] still have more than 6× MACs than the full-size LDM [50]. MobileDiffusion [72] introduces heuristics to enhance DPMs' efficiency and develops two efficient architectures. Nevertheless, it is highly non-trivial how to generalize the heuristics for different compute budgets. Spectral Diffusion (SD) [64] performs frequency domain distillation from a teacher model into a small LDM. However, the main weakness of single-model methods is that they use the same model for all denoising steps, which is shown to be sub-optimal [1, 19]. **Mixture of expert** methods employ a separate model for different stages of the denoising process. OMS-DPM [37] trains a model zoo with various sizes and searches for a proper model schedule given a desired compute budget. Yet, gathering a model

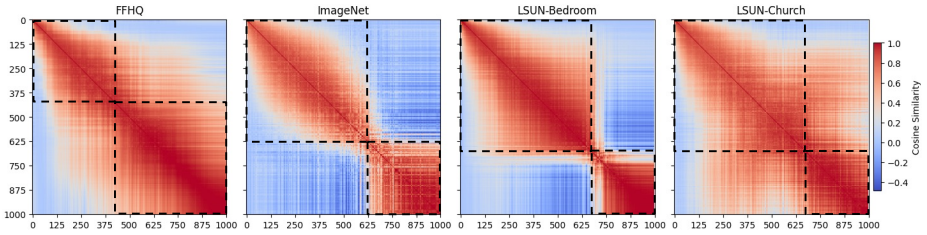

Fig. 3: Our Interval Selection Scheme: We calculate gradients of denoising timesteps' objectives *w.r.t* the pre-trained LDM's parameters and take the cosine similarity value of two timesteps' gradients as their alignment score. The dashed lines show our selected cluster intervals for the experts. One can observe the optimal cluster assignments are different for distinct datasets, and employing a deterministic clustering strategy [1] like uniform clustering [13] for all datasets is sub-optimal.

zoo is very costly on large-scale datasets, making OMS-DPM impractical for them. MEME [34] and TMDA [67] cluster denoising timesteps and design a distinct expert for each. However, they need to manually allocate the compute budget between experts and re-design the experts for a new budget, which makes them cumbersome in practice. We prune an LDM into a mixture of efficient experts. We cluster denoising steps into intervals using their alignment scores. Then, we fine-tune the pre-trained model with elastic dimensions on each interval to obtain our experts. Thus, our method gathers an *implicit* model zoo *within* each expert with much lower training iterations than OMS-DPM. Finally, we prune all experts simultaneously using our expert routing agent to obtain our mixture of efficient experts. By doing so, in contrast with MEME [34] and TMDA [67], our method automatically allocates the compute resource (*e.g.,* MACs) between experts. We refer to supplementary materials for a review of other related works.

3 Method

We introduce a framework to prune an LDM [50] model into a mixture of efficient experts in four steps. First, we cluster denoising timesteps of the model into several intervals based on their objectives' alignment scores. Second, we fine-tune the pre-trained model on the selected intervals with elastic dimensions to obtain our interval experts. Third, we prune the experts together using our expert routing agent in an end-to-end manner (Fig. 2). Finally, we fine-tune the pruned experts to obtain our mixture of efficient experts.

3.1 Background

Given a random variable $\mathbf{x_0} \sim \mathcal{P}$, the goal of DPMs [28, 55] is to model the underlying distribution $\mathcal{P}$ using a training set $\mathcal{D} = \{x_0\}$ of samples. To do so, first, DPMs define a forward process parameterized by t in which they gradually perturb each sample x_0 with Gaussian noise with the variance schedule of β_t:

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

where $t \in [1, T]$. Thus, $q(x_t|x_0)$ has a Gaussian form:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{2}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. The noise schedule β_t is usually selected [28] such that $q(x_T) \to \mathcal{N}(0, I)$. Assuming β_t is small, DPMs approximate the denoising distribution $q(x_{t-1}|x_t)$ by a parameterized Gaussian distribution $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)), \sigma_t^2 I)$, and σ_t^2 is often set to β_t. DPMs implement $\epsilon_\theta(.)$ with a neural network called the denoising model and train it with the variational evidence lower bound (ELBO) objective [28]:

$$\begin{aligned} \mathcal{L}_{\text{DDPM}} &= \mathbb{E}_{t\sim[1,T]}\mathcal{L}_t \\ &= \mathbb{E}_{t\sim[1,T],\epsilon\sim\mathcal{N}(0,I),x_t\sim q(x_t|x_0)}||\epsilon_\theta(x_t, t) - \epsilon||^2 \end{aligned} \tag{3}$$

DPMs generate a new sample by sampling an initial noise from $x_T \sim p(x_T) = \mathcal{N}(0, I)$ and iteratively denoising it using the denoising model by sampling from $p_\theta(x_{t-1}|x_t)$. Thus, the sampling process requires T sequential forward passes to the large denoising model, making it a slow and costly process.

3.2 Notations

Fig. 4 shows the U-Net [51] architecture used in LDM [50] models. The encoder and decoder branches have several *stages* (each row in Fig. 4). Each stage has one or several *layers*. We represent the layers' functions and feature maps with $f_l(.)$ and $\mathcal{F}_l$, respectively, where $l \in [1, L]$, and L is the total number of layers. Each layer consists of one or several *blocks*. For instance, the green-colored *layers* in Fig. 4 are in the third *stage* of its encoder and decoder and consist of a Residual *block* [24] and an Attention *block* [59].

3.3 Clustering Denoising Timesteps into Intervals

We propose to cluster denoising timesteps $\mathcal{T} = [1, T]$ of an LDM into N intervals $\{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_N\}$ such that $\mathcal{T} = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \cup \mathcal{I}_N$. The intuition is that different intervals have separate roles [1]. For example, it has been empirically shown [9] that an LDM first generates the layout of an image in high-noise timesteps and then fills in the details in low-noise ones. Thus, using the same denoising model for all timesteps is sub-optimal. We employ alignment scores of training objectives $\mathcal{L}_t$ (Eq. 3) for denoising timesteps of a pre-trained LDM to cluster them. We estimate the gradient of each $\mathcal{L}_t$ *w.r.t* the denoising model's parameters (θ) using a random batch of samples in the training data and take the cosine similarity between the gradients of $\mathcal{L}_t$ and $\mathcal{L}_s$ as the alignment score of timesteps t and s.

We visualize pairwise alignment scores of denoising time-steps for pre-trained LDMs [50] of different datasets in Fig. 3. We select two distinct clusters ($N = 2$)

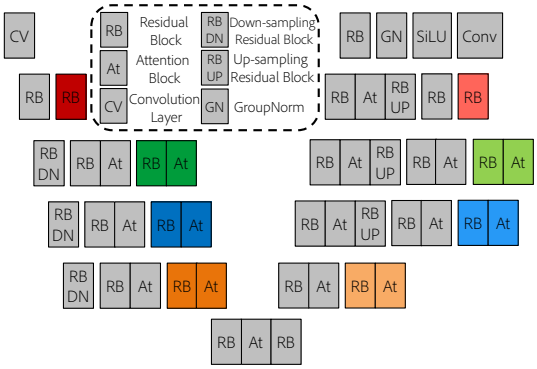

Fig. 4: U-Net architecture of the LDM [50]. We randomly drop/preserve each colored layer in our elastic depth fine-tuning.

for all datasets (shown by dashed lines in Fig. 3) in our experiments. We choose the cut-off point between the clusters to be the one that maximizes the weighted mean of the average scores of the clusters, and we refer to the supplementary for the formulation. We do not use more than two experts in our experiments for computational efficiency. However, our formulation can find cut-off points for more than two clusters, as we elaborate in the supplementary. One can observe that intra-cluster alignment scores are high, and inter-cluster scores are small and even negative for LSUN-Bedroom [66], ImageNet [52], and FFHQ [31]. Accordingly, further training the denoising model on one of the clusters degrades its performance on the other. This observation supports our decision to employ a specialized model for each interval, that using a single model for all intervals is sub-optimal. Further, the optimal cluster assignment is different for distinct datasets. Thus, our clustering method is more robust than deterministic ones [1, 13]. In summary, we select clusters $(\mathcal{I}_1, \mathcal{I}_2)$ to be $([0, 700], [701, 1000])$ for LSUN-Church as well as LSUN-Bedroom, $([0, 400], [401, 1000])$ for FFHQ, and $([0, 625], [626, 1000])$ for ImageNet.

3.4 Fine-tuning with Elastic Dimensions

We fine-tune the pre-trained LDM with elastic dimensions (depth and width) on each denoising interval $\mathcal{I}_i$ $(i \in [1, N])$ after clustering the denoising timesteps. We call the resulting models *experts* and denote them with E_i, corresponding to $\mathcal{I}_i$. Our main inspiration is that by doing so, each sub-network of an expert E_i has a decent performance on the denoising interval $\mathcal{I}_i$, which brings in several key benefits: First, the loss value of each sub-network of E_i will be a proper proxy for its actual performance after fine-tuning on $\mathcal{I}_i$. Second, the pruned experts will be able to recover their performance promptly during fine-tuning without requiring long fine-tuning iterations. Finally, our elastic fine-tuning provides a model zoo *within* the expert E_i for the denoising interval $\mathcal{I}_i$ without extreme computational and memory expensive training of several architectures from scratch like in

OMS-DPM [37]. We first fine-tune the pre-trained model with elastic depth on each interval. Then, we fine-tune the resulting model with elastic width. We do not perform elastic depth and width training together to prevent instabilities.

Fine-tuning with elastic depth. We randomly drop the last layer in each stage of the U-Net's encoder and decoder (colored blocks in Fig. 4) for our elastic depth training. Formally, for each training batch, we randomly select to map the last depth layers $f_j^{(i)}$ in stages of the expert E_i independently with a probability p to the identity function:

$$\hat{f}_j^{(i)} = f_j^{(i)} \mathbf{1}_{\{s_j^{(i)}=0\}} + I \mathbf{1}_{\{s_j^{(i)}=1\}}, \quad s_j^{(i)} \sim \text{Bernoulli}(p) \tag{4}$$

We train selected sub-network's parameters with the interval denoising objective:

$$\mathcal{L}_{\text{DDPM},\mathcal{I}_i} = \mathbb{E}_{t\sim\mathcal{I}_i} \mathcal{L}_t \tag{5}$$

where $\mathcal{L}_t$ has the same formulation as Eq. 3.

Fine-tuning with elastic width. After fine-tuning the pre-trained model on each interval $\mathcal{I}_i$ with elastic depth, we fine-tune the resulting experts with elastic width. For each ResBlock in the U-Net, we sort the channels of its convolution layers based on an estimate of their importance (determined by their L_1 norm [5,35]). Then, for each training batch, we randomly remove some ratio of the least important channels of each convolution layer. Similarly, for the attention layers [59], we sort the attention heads based on the L_1 norm of their projection weights, and we randomly drop some of the least important heads during our elastic width fine-tuning. Finally, we update the selected sub-network's weights using $\mathcal{L}_{\text{DDPM},\mathcal{I}_i}$ (Eq. 5). We refer to supplementary for more details.

3.5 Expert Routing Agent

We develop our Expert Routing Agent (ERA) to prune the elastically fine-tuned experts E_i ($i \in [1, N]$) into a mixture of efficient experts. We denote our ERA as a function $h_{\text{ERA}}(.;\beta)$ parameterized by β, predicting architecture vectors (u, v):

$$u, v = h_{\text{ERA}}(z;\beta) \tag{6}$$

z is a constant, randomly initialized input. Vectors $u = [u^{(i)}]_{i=1}^N$ determine pruning depth layers $f_j^{(i)}$ (Eq. 4). Similarly, vectors $v = [v^{(i)}]_{i=1}^N$ determine widths of blocks of N experts. Together, (u, v) select sub-networks e_i from experts E_i.

Given a total constraint on the computation budget (*e.g.*, MACs, latency, etc.), denoted as T_d, we optimize the ERA model's parameters β to predict architecture vectors (u, v) for an efficient and high-performing set of experts' architectures. Next, we describe how we parameterize and apply (u, v). We show the formulation to determine the compute budget of selected architectures and the final optimization procedure for the ERA in Eq. 15.

Pruning Width: Although one can prune widths of blocks in an expert E_i using a binary vector $v^{(i)}$, keeping the j^{th} channel when $v_j^{(i)}$ is 1 and vice versa, such an operation is not differentiable, making the optimization of parameters β of the ERA challenging. Thus, we introduce soft vectors $\mathbf{v}^{(i)}$, relax them to have continuous values, and use them for width pruning. We calculate them as follows:

$$\mathbf{v}^{(i)} = \text{sigmoid}(\frac{v^{(i)} + n}{\tau}) \tag{7}$$

$n \sim \text{Gumbel}(0, 1)$ is a noise from the Gumbel distribution [21]. Parameter τ is the temperature that when set appropriately, brings elements of $\mathbf{v}^{(i)}$ close to 0 or 1. The calculation from $v^{(i)}$ to $\mathbf{v}^{(i)}$ is called the Gumbel-Sigmoid trick [30, 44]. It is a differentiable estimation of sampling from a Bernoulli distribution with the Bernoulli parameter of $\texttt{sigmoid}(v^{(i)})$. We apply vectors $\mathbf{v}^{(i)} = [\mathbf{v}_l^{(i)}]_{l=1}^{L}$ to prune the width of blocks in all of the layers $f_l^{(i)}$ for the expert E_i:

$$\hat{f}_l^{(i)} = f_l^{(i)}(.; \mathbf{v}_l^{(i)}) \tag{8}$$

Here, we apply (multiply) the width vector $\mathbf{v}_l^{(i)}$ to feature maps of the first convolution layer in the ResBlocks and inputs of the attention operation in the attention blocks in the layer $f_l^{(i)}$. The granularity of our width pruning is similar to our elastic width fine-tuning, *i.e.,* we prune channels of convolution layers of ResBlocks and heads of the attention layers.

Pruning depth. Similarly, we employ relaxed continuous vectors $\mathbf{u}^{(i)} = [\mathbf{u}_j^{(i)}]$ for pruning the depth layers $f_j^{(i)}$ (Eq. 4) of the expert E_i:

$$\mathbf{u}^{(i)} = \text{sigmoid}(\frac{u^{(i)} + n}{\tau}) \tag{9}$$

As there are skip connections in the U-Net, we apply the depth architecture vectors for the encoder and decoder branches differently.

Encoder depth pruning. We use the following formulation to apply the vector $\mathbf{u}^{(i)}$ for pruning depth layers in the encoder of the expert E_i:

$$\widehat{\mathcal{F}}_j^{(i)} = \mathbf{u}_j^{(i)} f_j^{(i)}(\mathcal{F}_{j-1}^{(i)}) + (1 - \mathbf{u}_j^{(i)})\mathcal{F}_{j-1}^{(i)} \tag{10}$$

In other words, we interpolate between the feature map of the previous layer, $\mathcal{F}_{j-1}^{(i)}$, and the result of applying the current layer to it, $f_j^{(i)}(\mathcal{F}_{j-1}^{(i)})$. The $\mathbf{u}_j^{(i)}$ values close to 1 simulate preserving the layer, and 0 simulate removing the layer.

Decoder depth pruning. The input for the layer $f_j^{(i)}$ in the decoder of the expert E_i is the concatenation of feature maps $\mathcal{F}_{j-1}^{(i)}$ of its previous layer and the skip connection feature maps $\mathcal{F}_{j,skip}^{(i)}$. Thus, we apply the vector $\mathbf{u}_j^{(i)}$ to it as:

$$\widehat{\mathcal{F}}_j^{(i)} = \mathbf{u}_j^{(i)} f_j^{(i)}(\mathcal{F}_{j-1}^{(i)} \parallel \mathcal{F}_{j,skip}^{(i)}) + (1 - \mathbf{u}_j^{(i)})\mathcal{F}_{j-1}^{(i)} \tag{11}$$

where $\parallel$ denotes concatenation. Similar to Eq. 10, we interpolate between applying or removing the layer in Eq. 11.

3.6 Pruning the Mixture of Experts

We train our Expert Routing Agent to select competent sub-networks of elastically trained experts given a desired total compute budget. We measure the compute budget of our models with MACs, following [12,34]. Given an architecture width vector $\mathbf{v}_l^{(i)}$, the MACs of the layer $f_l^{(i)}(.)$ after applying $\mathbf{v}_l^{(i)}$ will be:

$$\widehat{T}_l^{(i)} = \mathbf{1}^T \times \lfloor \mathbf{v}_l^{(i)} \rceil \times T_l^{(i)} \tag{12}$$

where $\mathbf{1}$ denotes a vector of all ones. $\lfloor \cdot \rceil$ is the function that rounds to the nearest integer, and $T_l^{(i)}$ is the MACs of the layer $f_l^{(i)}(.)$. Similarly, the MACs for the layers $f_j^{(i)}(.)$ that we use for depth pruning (Eq. 4) after applying $\mathbf{u}_j^{(i)}$ will be:

$$\widehat{T}_j^{(i)} = \lfloor \mathbf{u}_j^{(i)} \rceil \times \mathbf{1}^T \times \lfloor \mathbf{v}_j^{(i)} \rceil \times T_j^{(i)} \tag{13}$$

After applying architecture vectors of each expert E_i, we calculate the total MACs of our mixture of experts as:

$$\widehat{T}(u, v) = \sum_{i=1}^{N} \frac{|\mathcal{I}_i|}{\sum_{k=1}^{N} |\mathcal{I}_k|} \widehat{T}^{(i)}(u^{(i)}, v^{(i)}) \tag{14}$$

where $\widehat{T}^{(i)}(u^{(i)}, v^{(i)})$ is the MACs of the expert E_i after applying its architecture width and depth vectors. In Eq. 14, we assume that the denoising schedule is *linear* such that the number of denoising steps that the expert E_i will contribute to the denoising process is proportional to the size of its interval $|\mathcal{I}_i|$. One can alter Eq. 14 for other denoising schedules like quadratic [56], but we focus on the linear schedule as it has been widely adopted in the literature [12,34,50,56].

Given a desired MACs budget T_d, we train our ERA with the following objective to encourage it to select sub-networks of experts such that each of them has a high performance and their mixture has a total MACs close to T_d:

$$\min_{\beta} \mathcal{J}(T_d) = [\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{DDPM},\mathcal{I}_i}(E_i(u^{(i)}, v^{(i)}))] + \mathcal{R}(\widehat{T}(u, v), T_d) \tag{15}$$

$\mathcal{L}_{\text{DDPM},\mathcal{I}_i}(E_i(u^{(i)}, v^{(i)}))$ is the interval denoising objective (Eq. 5) of the sub-network of E_i chosen by $(u^{(i)}, v^{(i)})$. $\widehat{T}(u, v)$ is the total MACs of the mixture of experts (Eq. 14) determined by the architecture vectors (u, v) that are functions of the ERA's parameters β. $\mathcal{R}(\cdot)$ is the MACs regularization term that we implement it as $\mathcal{R}(x, y) = \log(\max(x, y)/\min(x, y))$. Now, as the round function $\lfloor \cdot \rceil$ used in Eqs. (12, 13) is not differentiable, we use the Straight Through Estimator (STE) [3] to calculate the gradients of $\mathcal{R}$ *w.r.t* the parameters β of our ERA. We implement our ERA model with a GRU [8] layer followed by dense layers. We found in our experiments that a lightweight ($\sim 0.5M$ parameters) ERA model suffices to obtain a performant mixture of efficient experts. We show our pruning scheme in Fig. 2 and refer to supplementary for more details of our pruning algorithm as well as the ERA's architecture.

Fine-tuning pruned models. After our pruning stage, we use architecture vectors predicted by the ERA to prune experts. Then, we fine-tune the experts with the same settings as the original LDM model [50].

4 Experiments

We experiment on the LSUN-Church [66], LSUN-Bedroom [66], FFHQ [31], and class-conditional ImageNet [52] to verify our method's effectiveness. We apply our method to the LDM [50] that implements their denoising model with a U-Net [51] architecture. For all datasets, we prune the LDM model with three MACs budgets of 70%, 50%, and 30%. In all experiments, we mainly follow the same hyper-parameter settings as LDM [50], and we refer to supplementary for more details of our experimental setup. We denote our method as DiffPruning in the rest of the experiments section. We mainly compare our method with a few recent baselines on architectural efficiency of pixel-space [12] and latent space [34, 37] DPMs. We do not benchmark with Spectral Diffusion (SD) [64] because it uses a package[3] that does not count MACs of the attention operation `QKVAttention` implemented in the LDM repository[4]. Hence, the MACs of models reported by SD [64] are not accurate. For instance, SD reports the LDM [50] for LSUN-Church has 18.7G MACs, but it actually has 20.96G (Tab. 1c).

4.1 Comparison Results

We summarize comparison results in Tab. 1 and refer to supplementary for FID *vs.* MACs as well as FID *vs.* Throughput curves of our method and baselines.
LSUN-Bedroom. Tab. 1a presents the results on LSUN-Bedroom. First, we can observe that the LDM [50] can achieve better sample quality (lower FID) while having significantly lower MACs (higher sampling speed) than the pixel-space DPM, DDPM [28]. Although SP [12] prunes more than 44% MACs of the DDPM [28], its pruned model still has 36% more MACs than LDM while drastically degrading the sample quality to 18.6 FID. These results demonstrate that pixel-space DPMs have much more redundancies than LDMs, and pruning LDMs is significantly more challenging. Second, our pruned models can achieve higher throughput speed-up ratio than their pruning ratio while having competitive FID scores. DiffPruning 70%/50%/30% models reduce MACs by 30%/50%/70%, but can secure 43%/87%/135% sampling speed up compared to LDM [50]. Notably, DiffPruning 70% (50%) has 5.90 (6.73) FID score which is fairly close to the original LDM (4.39) while substantially better than the pruned model by SP [12] (18.6). Finally, although not directly comparable, our pruned models require less training iterations than SP, and we refer to supplementary for details.
LSUN-Church. The architecture MACs of the LDM [50] for LSUN-Church is smaller than LDM models for other datasets as its encoder reduces the spatial

[3] https://github.com/sovrasov/flops-counter.pytorch
[4] https://github.com/CompVis/latent-diffusion

Table 1: Comparison results of DiffPruning *vs.* baselines. Throughput values are calculated using an NVIDIA A100 GPU. †: the values are average of our two efficient experts. ∗: calculated by sampling from provided checkpoints. ‡: speed-ups relative to the LDM model. The shadowed values are inaccurate, and we refer to supplementary for a detailed discussion.

LSUN-Bedroom (256 × 256)

Model	Complexity		Performance	
	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
DDPM [28]	113.7M	248.7G	0.74	6.62
SP [12]	63.2M	138.8G	-	18.6
LDM [50]	274.06M	101.32G	2.01	4.39*
DiffPruning (70%)	162.06M†	70.84G	**3.11** (×**1.55**)‡	5.90
DiffPruning (50%)	100.87M†	50.69G	**3.75** (×**1.87**)‡	6.73
DiffPruning (30%)	48.43M†	31.11G	**4.73** (×**2.35**)‡	9.22

(a)

FFHQ (256 × 256)

Model	Complexity		Performance	
	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
LDM [50]	274.06M	101.32G	1.01	9.53*
DiffPruning (70%)	194.79M†	71.05G	**1.35** (×**1.33**)‡	9.80
DiffPruning (50%)	134.67M†	51.87G	**1.83** (×**1.81**)‡	9.90
DiffPruning (30%)	63.07M†	30.68G	**2.90** (×**2.87**)‡	10.66

(b)

LSUN-Church (256 × 256)

Model	Complexity			Performance	
	Sampler	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
LDM [50]	DDIM-100	294.7M	20.96G	5.19	5.21*
LDM [50]	DDIM-200	294.7M	20.96G	2.60	5.11*
DDPM [56]		113.7M	248.7G	0.74	10.58
SP [12]	DDIM-100	63.2M	138.8G	-	13.9
DiffPruning (70%)		188.09M†	14.64G	**5.73** (×**1.11**)‡	9.39
OMS-DPM [37]	Searched	-	-	2.56	11.10
DiffPruning (50%)	DDIM-200	112.6M†	10.48G	3.15	10.22
DiffPruning (50%)	DDIM-100	112.6M†	10.48G	**6.28** (×**1.21**)‡	10.89
OMS-DPM [37]	Searched	-	-	6.4	13.7
DiffPruning (30%)	DDIM-100	36.9M†	6.35G	**6.87** (×**1.32**)‡	11.39

(c)

Class-Conditional ImageNet (256 × 256)

Model	Complexity		Performance	
	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
ADM [10]	607.9M	1186.4G	0.07	4.59
LDM [50]	400.82M	108.78G	0.32	3.60
DiffPruning (70%)	250.79M†	76.24G	**0.43** (×**1.34**)‡	8.03
LDM 50% Scratch [12]	189.43M	52.71G	-	51.45
Taylor [12]	189.43M	52.71G	-	11.18
SP [12]	189.43M	52.71G	-	9.16
DiffPruning (50%)	161.06M†	54.32G	**0.56** (×**1.75**)‡	8.45
DiffPruning (30%)	79.82M†	32.71G	**0.92** (×**2.87**)‡	13.18

(d)

dimension by 8 *vs.* 4 for others. Thus, pruning the LDM for LSUN-Church is more challenging than other ones, and Tab. 1c summarizes the results. First, we can observe that DiffPruning 70% achieves drastically better FID than the DDPM model pruned by SP [12] while having almost 9.5× fewer MACs. Remarkably, DiffPruning 70% achieves better FID than the full DDPM model, illustrating that LDMs have better computation-performance frontier than pixel-space DPMs. Second, DiffPruning 50% and 30% models can achieve both higher throughput and better FID while requiring more than 7× less training iterations than OMS-DPM [37] (details in supplementary). DiffPruning 50% with the 100-step DDIM [56] sampler has 2.45× higher throughput (6.28 *vs.* 2.56) than OMS-DPM with a lower FID. Also, DiffPruning 50% with 200 steps DDIM sampler still has a higher throughput and better FID than OMS-DPM. Notably, DiffPruning 50% with 200 steps DDIM sampler can outperform the full DDPM model (10.22 *vs.* 10.58 FID) while having 4.25× faster sampling throughput. Finally, DiffPruning 30% has higher throughput (6.87 *vs.* 6.4 FID) while outperforming OMS-DPM's model by 2.31 FID. In summary, the comparison results with OMS-DPM demonstrate the benefit of our elastic fine-tuning for our experts that enables our method to gather a model zoo without requiring training several models from scratch, thereby obtaining a higher-performing mixture of efficient experts with much lower training iterations than OMS-DPM.

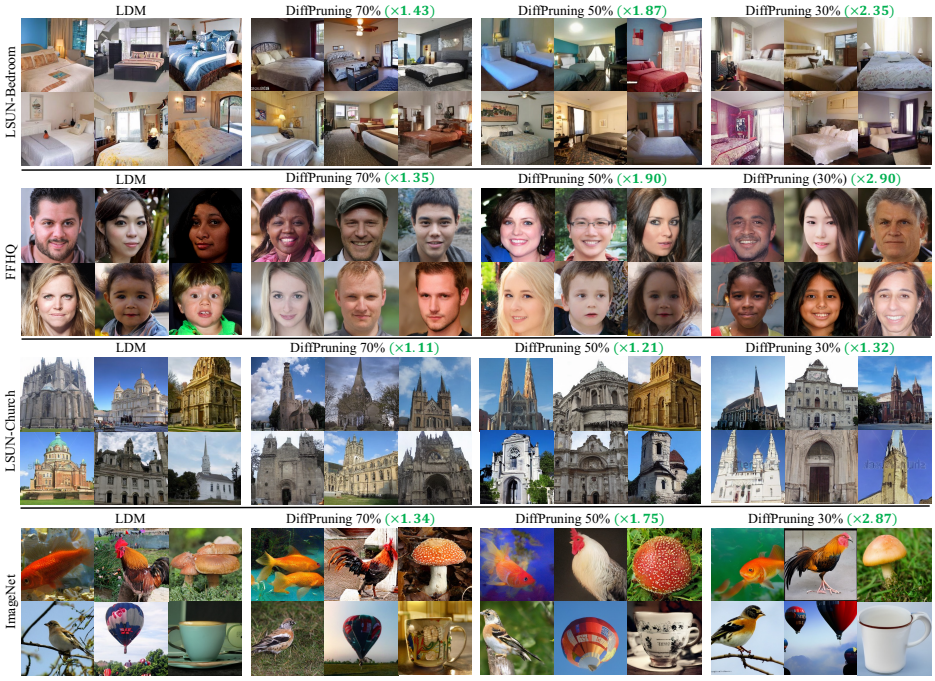

Fig. 5: Samples from the LDM [50] model and our pruned mixture of experts for different MACs budgets. The green numbers show the relative sampling throughput speed-up of our pruned models compared to LDM on an NVIDIA A100 GPU.

FFHQ. Tab. 1b shows the results on FFHQ. DiffPruning 70%/50% models can achieve close FID scores to LDM [50] while enjoying 33%/81% throughput speed-ups. In the extreme case of 30% MACs budget, DiffPruning secures 2.87× speed-up while having a 1.13 worse FID score than LDM, which shows that it can successfully prune the LDM for small-scale datasets like FFHQ as well.
Class-Conditional ImageNet. Tab. 1d summarizes results for ImageNet. DiffPruning 50% model can achieve 0.71 better FID score than SP [12]. The reported MACs values by SP are not directly comparable to ours, and we elaborate on the reason in supplementary. In addition, DiffPruning 70%/30% obtain 1.34×/2.87× increase in throughput compared to LDM. Thus, DiffPruning can effectively prune conditional LDMs as well. In summary, our experimental results demonstrate that our method can effectively prune both unconditional and conditional LDM models for datasets with various scales. We provide samples generated by the original LDM and our pruned mixture of experts with different budgets in Fig. 5.

4.2 Ablation Study

We conduct an ablation study to explore the contribution of each component of our method to its final performance. We implement a Baseline that uses naive

parameterizations $\mathbf{v} = \text{sigmoid}(-(\beta_v + n)/\tau)$ (Eq. 7) and $\mathbf{u} = \text{sigmoid}(-(\beta_u + n)/\tau)$ (Eq. 9) for pruning a single model. Then, we add the mixture of experts, our ERA model, elastic depth fine-tuning, and elastic width fine-tuning one at a time for pruning the model. Tab. 2 summarizes the results. First, we can observe that employing the mixture of experts improves both the sample quality FID score (which is aligned with the prior works [1, 13]) and the inference throughput of the pruned model. This result quantitatively justifies our design choice for clustering the denoising timesteps into intervals and using a specialized model for each of them. Employing our ERA model yields a faster model than the naive parameterization. The reason may be that the naive parameterization cannot properly model the complex interactions between experts and between different layers within an expert. Noticeably, employing each component of our method improves *both dimensions* of sampling throughput and sample quality such that our method can obtain a 1.28× faster model (throughput 3.75 *vs.* 2.92) with 3.59 better FID than the naive Baseline. In summary, our ablation experiments verify our design choices for DiffPruning.

Table 2: Ablation results of our proposed method for pruning the LDM model [50] for LSUN-Bedroom to 50% MACs budget.

Model	Sampler	MACs	Throughput(↑) (Sample/Sec)	FID (↓)
Baseline	DDIM-100	50.69G	2.92	10.32
+ Mixture of Experts			3.05	9.65
+ Expert Routing Agent			3.25	8.53
+ Elastic Depth			3.61	8.03
+ Elastic Width (Ours)			3.75	6.73
LDM [50]		101.32G	2.01	4.39

5 Conclusions

We introduce a novel approach for pruning an LDM model into a mixture of efficient experts in which each expert performs the denoising task on an interval of the denoising path. We employ the model's denoising timesteps' alignment scores to cluster them into several intervals and empirically show that the optimal cluster assignments are different for distinct datasets. Thus, using static clustering schemes is sub-optimal. We propose to fine-tune the pre-trained LDM on each cluster interval with elastic dimensions to obtain our interval experts. By doing so, each expert contains an implicit model zoo within itself for its corresponding interval. Finally, we develop a new pruning scheme in which our Expert Routing Agent (ERA) learns to prune the elastically trained experts together in an end-to-end manner. Thus, our ERA automatically allocates the compute budget between experts. Our experimental results validate our method's effectiveness, and our ablation studies show that our design choices improve both dimensions of the pruned model's throughput and its sample quality.

Acknowledgments

Alireza Ganjdanesh and Heng Huang were partially supported by NSF IIS 2347592, 2347604, 2348159, 2348169, DBI 2405416, CCF 2348306, CNS 2347617.

References

1. Balaji, Y., Nah, S., Huang, X., Vahdat, A., Song, J., Kreis, K., Aittala, M., Aila, T., Laine, S., Catanzaro, B., et al.: ediffi: Text-to-image diffusion models with an ensemble of expert denoisers. arXiv preprint arXiv:2211.01324 (2022) 2, 4, 5, 6, 7, 14, 30, 31
2. Bao, F., Li, C., Zhu, J., Zhang, B.: Analytic-DPM: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=0xiJLKH-ufZ 2, 4, 30
3. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013) 10
4. Betker, J., Goh, G., Jing, L., Brooks, T., Wang, J., Li, L., Ouyang, L., Zhuang, J., Lee, J., Guo, Y., et al.: Improving image generation with better captions. Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf **2**(3), 8 (2023) 2
5. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=HylxE1HKwS 8, 23
6. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=HylxE1HKwS 31
7. Cheng, H., Zhang, M., Shi, J.Q.: A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. arXiv preprint arXiv:2308.06767 (2023) 31
8. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. ACL (2014). https://doi.org/10.3115/v1/d14-1179, https://doi.org/10.3115/v1/d14-1179 10, 24
9. Choi, J., Lee, J., Shin, C., Kim, S., Kim, H., Yoon, S.: Perception prioritized training of diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11472–11481 (2022) 6
10. Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis. Advances in neural information processing systems **34**, 8780–8794 (2021) 1, 12
11. Esser, P., Rombach, R., Ommer, B.: Taming transformers for high-resolution image synthesis. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 12873–12883 (2021) 31
12. Fang, G., Ma, X., Wang, X.: Structural pruning for diffusion models. In: Advances in Neural Information Processing Systems (2023) 4, 10, 11, 12, 13, 21, 28, 29, 30, 31

13. Feng, Z., Zhang, Z., Yu, X., Fang, Y., Li, L., Chen, X., Lu, Y., Liu, J., Yin, W., Feng, S., et al.: Ernie-vilg 2.0: Improving text-to-image diffusion model with knowledge-enhanced mixture-of-denoising-experts. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10135–10145 (2023) 4, 5, 7, 14, 30
14. Ganjdanesh, A., Gao, S., Alipanah, H., Huang, H.: Compressing image-to-image translation gans using local density structures on their learned manifold. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38, pp. 12118–12126 (2024) 31
15. Ganjdanesh, A., Gao, S., Huang, H.: Interpretations steered network pruning via amortized inferred saliency maps. In: European Conference on Computer Vision. pp. 278–296. Springer (2022) 31
16. Ganjdanesh, A., Gao, S., Huang, H.: Effconv: efficient learning of kernel sizes for convolution layers of cnns. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 37, pp. 7604–7612 (2023) 31
17. Ganjdanesh, A., Gao, S., Huang, H.: Jointly training and pruning cnns via learnable agent guidance and alignment. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16058–16069 (2024) 31
18. Gao, S., Liu, X., Zeng, B., Xu, S., Li, Y., Luo, X., Liu, J., Zhen, X., Zhang, B.: Implicit diffusion models for continuous super-resolution. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10021–10030 (2023) 1
19. Go, H., Kim, J., Lee, Y., Lee, S., Oh, S., Moon, H., Choi, S.: Addressing negative transfer in diffusion models. In: Thirty-seventh Conference on Neural Information Processing Systems (2023), https://openreview.net/forum?id=3G2ec833mW 2, 4, 30, 31
20. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in neural information processing systems **27** (2014) 1
21. Gumbel, E.J.: Statistical theory of extreme values and some practical applications: a series of lectures, vol. 33. US Government Printing Office (1954) 9
22. Habibian, A., Ghodrati, A., Fathima, N., Sautiere, G., Garrepalli, R., Porikli, F., Petersen, J.: Clockwork diffusion: Efficient generation with model-step distillation. arXiv preprint arXiv:2312.08128 (2023) 2, 4, 30
23. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. Advances in neural information processing systems **28** (2015) 31
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 6, 23
25. He, Y., Xiao, L.: Structured pruning for deep convolutional neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence pp. 1–20 (2023). https://doi.org/10.1109/TPAMI.2023.3334614 31
26. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the European conference on computer vision (ECCV). pp. 784–800 (2018) 31
27. Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A., Kingma, D.P., Poole, B., Norouzi, M., Fleet, D.J., et al.: Imagen video: High definition video generation with diffusion models. arXiv preprint arXiv:2210.02303 (2022) 1
28. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Advances in neural information processing systems **33**, 6840–6851 (2020) 1, 5, 6, 11, 12, 21, 29

29. Hou, L., Huang, Z., Shang, L., Jiang, X., Chen, X., Liu, Q.: Dynabert: Dynamic bert with adaptive width and depth. Advances in Neural Information Processing Systems **33**, 9782–9793 (2020) 31
30. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=rkE3y85ee 9, 25, 28
31. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 4401–4410 (2019) 7, 11, 22
32. Kingma, D., Salimans, T., Poole, B., Ho, J.: Variational diffusion models. Advances in neural information processing systems **34**, 21696–21707 (2021) 4, 30, 31
33. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014), http://arxiv.org/abs/1312.6114 1
34. Lee, Y., Kim, J.Y., Go, H., Jeong, M., Oh, S., Choi, S.: Multi-architecture multi-expert diffusion models. arXiv preprint arXiv:2306.04990 (2023) 2, 3, 4, 5, 10, 11, 30, 31
35. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=rJqFGTslg 8, 23
36. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=rJqFGTslg 31
37. Liu, E., Ning, X., Lin, Z., Yang, H., Wang, Y.: Oms-dpm: Optimizing the model schedule for diffusion probabilistic models. In: International Conference on Machine Learning. pp. 21915–21936. PMLR (2023) 2, 3, 4, 8, 11, 12, 21, 29, 31
38. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=S1eYHoC5FX 31
39. Liu, L., Ren, Y., Lin, Z., Zhao, Z.: Pseudo numerical methods for diffusion models on manifolds. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=PlKWVd2yBkY 2
40. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=Bkg6RiCqY7 28
41. Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., 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) 2
42. Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., Zhu, J.: DPM-solver++: Fast solver for guided sampling of diffusion probabilistic models (2023), https://openreview.net/forum?id=4vGwQqviud5 4, 30
43. Ma, X., Fang, G., Wang, X.: Deepcache: Accelerating diffusion models for free. arXiv preprint arXiv:2312.00858 (2023) 4, 30
44. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=S1jE5L5gl 9, 25
45. Meng, C., Rombach, R., Gao, R., Kingma, D., Ermon, S., Ho, J., Salimans, T.: On distillation of guided diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14297–14306 (2023) 2, 4, 30

46. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019) 31
47. Nichol, A.Q., Dhariwal, P.: Improved denoising diffusion probabilistic models. In: International Conference on Machine Learning. pp. 8162–8171. PMLR (2021) 2, 4, 30
48. Pan, Z., Zhuang, B., Huang, D.A., Nie, W., Yu, Z., Xiao, C., Cai, J., Anandkumar, A.: T-stitch: Accelerating sampling in pre-trained diffusion models with trajectory stitching. arXiv preprint arXiv:2402.14167 (2024) 31
49. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019) 30
50. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., 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) 2, 4, 5, 6, 7, 10, 11, 12, 13, 14, 20, 21, 26, 27, 28, 29, 30, 31
51. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. pp. 234–241. Springer (2015) 2, 6, 11
52. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y 7, 11, 22
53. Saharia, C., Ho, J., Chan, W., Salimans, T., Fleet, D.J., Norouzi, M.: Image super-resolution via iterative refinement. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(4), 4713–4726 (2022) 1
54. Salimans, T., Ho, J.: Progressive distillation for fast sampling of diffusion models. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=TIdIXIpzhoI 2, 4, 30
55. Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: International conference on machine learning. pp. 2256–2265. PMLR (2015) 1, 5
56. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=St1giarCHLP 2, 4, 10, 12, 21, 27, 29, 30
57. Song, Y., Ermon, S.: Generative modeling by estimating gradients of the data distribution. Advances in neural information processing systems **32** (2019) 1
58. Vahdat, A., Kreis, K., Kautz, J.: Score-based generative modeling in latent space. Advances in Neural Information Processing Systems **34**, 11287–11302 (2021) 30
59. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017) 6, 8, 23
60. Watson, D., Chan, W., Ho, J., Norouzi, M.: Learning fast samplers for diffusion models by differentiating through sample quality. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=VFBjuF8HEp 4, 30

61. Watson, D., Ho, J., Norouzi, M., Chan, W.: Learning to efficiently sample from diffusion probabilistic models (2022), https://openreview.net/forum?id=LOz0xDpw4Y 4, 30
62. White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., Hutter, F.: Neural architecture search: Insights from 1000 papers. arXiv preprint arXiv:2301.08727 (2023) 31
63. Xu, Y., Deng, M., Cheng, X., Tian, Y., Liu, Z., Jaakkola, T.: Restart sampling for improving generative processes. arXiv preprint arXiv:2306.14878 (2023) 4, 30
64. Yang, X., Zhou, D., Feng, J., Wang, X.: Diffusion probabilistic model made slim. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 22552–22562 (2023) 2, 4, 11, 30, 31
65. Yao, L., Pi, R., Xu, H., Zhang, W., Li, Z., Zhang, T.: Joint-detnas: Upgrade your detector with nas, pruning and dynamic distillation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10175–10184 (2021) 31
66. Yu, F., Zhang, Y., Song, S., Seff, A., Xiao, J.: Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv preprint arXiv:1506.03365 (2015) 4, 7, 11, 22, 31
67. Zhang, H., Lu, Y., Alkhouri, I., Ravishankar, S., Song, D., Qu, Q.: Improving efficiency of diffusion models via multi-stage framework and tailored multi-decoder architectures. arXiv preprint arXiv:2312.09181 (2023) 2, 3, 5, 31
68. Zhang, L., Agrawala, M.: Adding conditional control to text-to-image diffusion models. arXiv preprint arXiv:2302.05543 (2023) 1
69. Zhang, Q., Chen, Y.: Fast sampling of diffusion models with exponential integrator. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=Loek7hfb46P 2, 4, 30
70. Zhang, Q., Tao, M., Chen, Y.: gDDIM: Generalized denoising diffusion implicit models. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=1hKE9qjvz- 4, 30
71. Zhao, S., Chen, D., Chen, Y.C., Bao, J., Hao, S., Yuan, L., Wong, K.Y.K.: Uni-controlnet: All-in-one control to text-to-image diffusion models. arXiv preprint arXiv:2305.16322 (2023) 1
72. Zhao, Y., Xu, Y., Xiao, Z., Hou, T.: Mobilediffusion: Subsecond text-to-image generation on mobile devices. arXiv preprint arXiv:2311.16567 (2023) 4, 31
73. Zheng, H., He, P., Chen, W., Zhou, M.: Truncated diffusion probabilistic models and diffusion-based adversarial auto-encoders. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=HDxgaKk956l 4, 30
74. Zoph, B., Le, Q.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=r1Ue8Hcxg 31

Supplementary Materials for Mixture of Efficient Diffusion Experts Through Automatic Interval and Sub-Network Selection

Alireza Ganjdanesh[1⋆], Yan Kang[2], Yuchen Liu[2], Richard Zhang[2], Zhe Lin[2], and Heng Huang[1]

[1] Department of Computer Science, University of Maryland College Park
[2] Adobe Research
{aliganj,heng}@umd.edu, {yankang,yuliu,rizhang,zlin}@adobe.com

We elaborate on more details about our method, experimental setup, experimental results, and related work in the supplementary materials. We follow the same notations introduced in the paper.

1 Experimental Results

We present the FID *vs.* MACs and FID *vs.* Throughput of our method and baselines in Fig. 1.

2 Details of Our Method

2.1 Clustering Denoising Time-Steps into Intervals

In this section, we provide details of our method to cluster denoising time-steps of an LDM [50] into intervals. As mentioned in Sec. (3.3) in the paper, we use two experts in our experiments for computational efficiency. Thus, we explain our approach for clustering the denoising path into two intervals, but one may extend it to a higher number of intervals as well.

Given denoising time-steps $\mathcal{T} = [1, T]$, we divide it into two intervals $\mathcal{I}_1 = [T, t_1]$ and $\mathcal{I}_2 = [t_1, 1]$. Now, the main question is how to determine the cut-off time-step t_1. We propose to leverage alignment scores of time-steps to find the optimal t_1. We take the cosine similarity between the gradients of training objectives $\mathcal{L}_t$ and $\mathcal{L}_s$ as the alignment score of the time-steps t and s and denote it with $a_{t,s}$. We propose to select the cut-off point that maximizes the weighted average of the mean of alignment scores in clusters:

$$\max_{t_1} \mathcal{J}(t_1) = \sum_{i=1}^{2} [w_i (\sum_{j \in \mathcal{I}_i} \sum_{k \in \mathcal{I}_i} \frac{a_{j,k}}{|\mathcal{I}_i|^2})] \tag{1}$$

$$w_i = \frac{|\mathcal{I}_i|}{T} \tag{2}$$

⋆ Part of this work was done during an internship at Adobe Research.

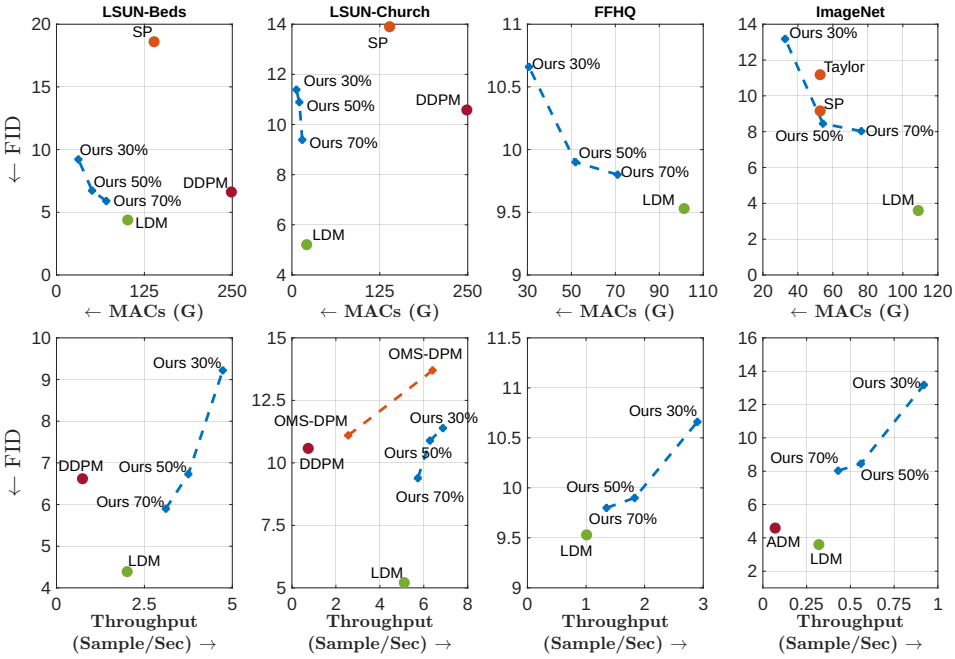

Fig. 1: Comparison Results of our method *vs.* baselines, SP [12], OMS-DPM [37], DDPM [28], and LDM [50]. **First Row:** FID *vs.* MACs curves. **Second Row:** FID *vs.* Throughput curves. We calculate the Throughput values with an NVIDIA A100 GPU. Higher Throughput and Lower FID and MACs indicate a better performance.

where $|\mathcal{I}_i|$ is the number of time-steps in $\mathcal{I}_i$ and T is the total number of denoising time-steps that is usually set to 1000 in practice [28,50,56]. The Obj 1 encourages to choose t_1 such that the average of alignment scores be high in each cluster while the weights w_i adjust the contribution of each cluster to the objective based on their size. Our weighting scheme prevents degenerate solutions such as choosing a single time-step as a separate cluster. Figs (2-5) show the $\mathcal{J}(t_1)$ functions for different datasets. We choose the cut-off values 400, 625, 700, and 700 for the FFHQ, ImageNet, LSUN-Bedroom, and LSUN-Church models to maximize their $\mathcal{J}(t_1)$ values. These values result in the $(\mathcal{I}_1, \mathcal{I}_2)$ clusters that we chose in Sec. (3.3) and Fig. (3) in the paper. We believe that one can readily extend our method to the cases with a higher number of experts by optimizing for the cut-off points that maximize similar objectives to $\mathcal{J}$. Specifically, if one decides to use C+1 experts (clusters), they should find C cut-off points $t_1<t_2<\cdots<t_C$ to optimize the following objective:

$$\max_{t_1,t_2,\cdots t_C} \mathcal{J} = \sum_{i=1}^{C+1} [w_i (\sum_{j\in\mathcal{I}_i} \sum_{k\in\mathcal{I}_i} \frac{a_{j,k}}{|\mathcal{I}_i|^2})] \tag{3}$$

with the same definitions as Eqs. (1, 2).

Finally, we note that we use 1024 random images to estimate the gradient of each time-step for the models for LSUN-Church [66], LSUN-Bedroom [66], and FFHQ [31]. We employ 16384 samples to do so on ImageNet [52].

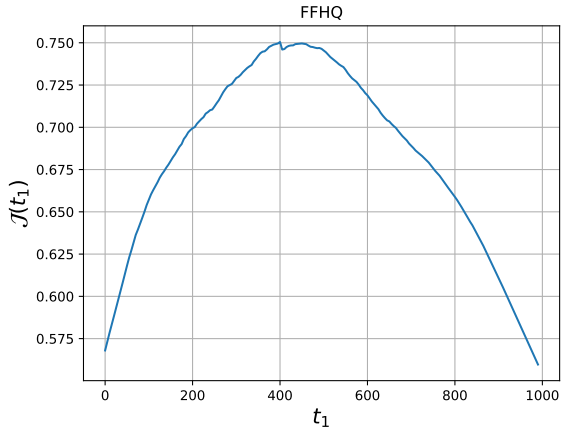

Fig. 2: Weighted average $\mathcal{J}(t_1)$ (Eq. 1) of the mean of alignment scores in two clusters for the LDM trained on FFHQ.

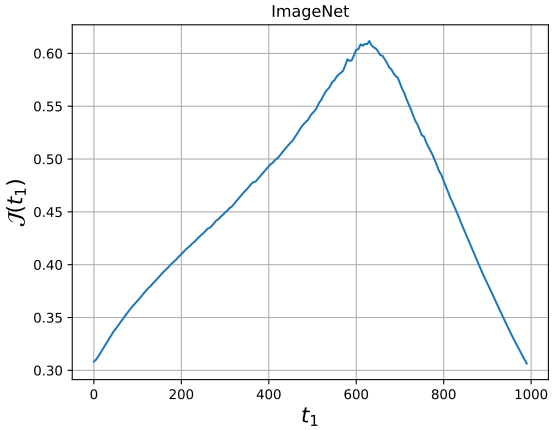

Fig. 3: Weighted average $\mathcal{J}(t_1)$ (Eq. 1) of the mean of alignment scores in two clusters for the LDM trained on ImageNet.

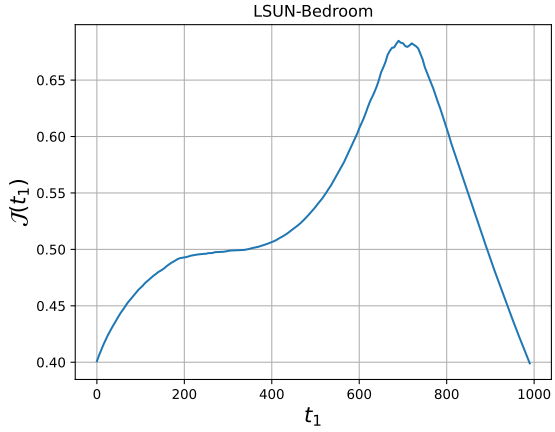

Fig. 4: Weighted average $\mathcal{J}(t_1)$ (Eq. 1) of the mean of alignment scores in two clusters for the LDM trained on LSUN-Beds.

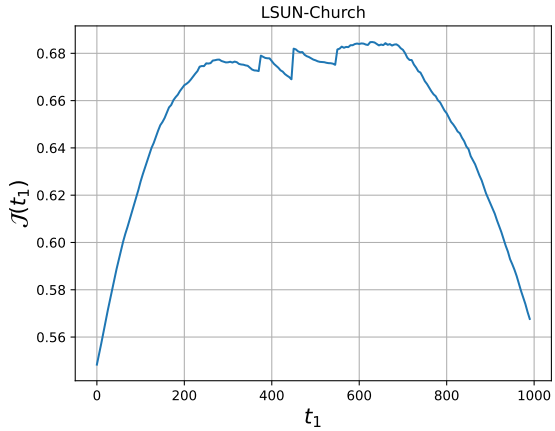

Fig. 5: Weighted average $\mathcal{J}(t_1)$ (Eq. 1) of the mean of alignment scores in two clusters for the LDM trained on LSUN-Church.

2.2 Fine-tuning with Elastic Width

We mentioned in Sec. (3.4) in the paper that we fine-tune the experts with elastic width after training them with elastic depth. Here, width means the channels of the convolution layers in the Residual Blocks [24] and heads of the attention layers [59] in the attention blocks of the U-Net. Before starting the elastic width fine-tuning, we sort the channels in the convolution layers in ResBlocks based on their importance, determined by their L_1 norm [5, 35]. Similarly, we sort the attention heads based on the L_1 norm of their projection weights. Then, in each batch of elastic width fine-tuning, We independently sample a random ratio r

($r \sim \mathcal{U}[0,\ 1)$) for each convolution layer with W channels (attention layer with W heads) and drop the $\lfloor Wr \rfloor$ least important channels (attention heads) of the layer. We illustrate our elastic width channel selection for a convolution layer with 4 channels in Fig. 6. The channels are sorted based on their L_1 norm (shown by their color intensity). The values $o_{1:4}$ represent different possible channel dropping cases for our elastic width training of the convolution layers. We similarly drop a ratio of least important attention heads.

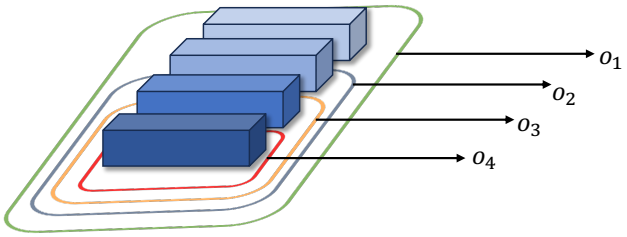

Fig. 6: Illustration of our Elastic Width training. We sort the convolution channels (attention heads) based on their importance (L_1 norm) before starting elastic width training. We drop a random ratio of the least important channels (heads) for convolution layers (attention layers) for each batch of training. The values $o_{1:4}$ represent different possible dropping ratios for a convolution layer with 4 channels.

2.3 Expert Routing Agent

We use a Gated Recurrent Unit (GRU) [8] and dense layers to implement our Expert Routing Agent (ERA). As mentioned in Sec. (3.5) in the main text, our ERA predicts architecture vectors ($u^{(i)},\ v^{(i)}$) that determine (depth, width) pruning for the expert E_i. We assume that each expert has D depth pruning layers and L layers for width pruning. We show the detailed architecture in Tab. 1. We randomly initialize the inputs $z^{(i)}$ and keep them fixed during our pruning process. The values $C_k^{(i)}$ when $k \in [1, \cdots, L]$ are equal to the widths of the layers. In addition, $C_{L+1}^{(i)} = D$ as we use the outputs of the $Dense_{L+1}$ layer to calculate the depth architecture vectors $u^{(i)}$.

Formulation of Architecture Vectors In this section, we describe our approach to calculate the architecture vectors $(u^{(i)}, v^{(i)})$ from the output vectors $o^{(i)}$ (Tab. 1) of the ERA.

Width Architecture Vectors: We design our width pruning method while considering our elastic width fine-tuning scheme. As mentioned in Sec. 2.2 and Fig. 6, we drop a random ratio of the least important convolution channels (attention heads) when training the convolution layers (attention layers) in the elastic width manner. We call each convolution channel and attention head a

Table 1: The architecture of our Expert Routing Agent. We calculate width architecture vectors $v^{(i)}$ from the outputs $o_k^{(i)}$ $(k \in [1, L])$. We compute the depth architecture vector $u^{(i)}$ from $o_{L+1}^{(i)}$. We refer to Sec. 2.3 for detailed formulations.

Inputs $z^{(i)} = [z_k^{(i)}]$, $(k = 1, \cdots, L+1)$, $(i = 1, \cdots, N)$
GRU(128, 256), WeightNorm, ReLU $\text{Dense}_k(256, C_k^{(i)})$, WeightNorm, $(k = 1, \cdots, L+1)$
Outputs $o_k^{(i)}$, $(k = 1, \cdots, L+1)$

'width unit.' Due to our dropping scheme, the weights of more important width units get trained more than the lower-importance ones in a layer and are robust to removing the lower-importance units.

Accordingly, we embed such a prior into the calculation of our width pruning architecture vectors $v^{(i)} = [v_l^{(i)}]_{l=1}^{L}$. Let's assume that the l-th layer of the expert E_i has W width units $[c_w]_{w=1}^{W}$ that are sorted based on their orders, namely c_1 is the most important unit, and c_W is the least important one. As mentioned in Sec. (3.5.1) of the paper, the calculation from $v^{(i)}$ to $\mathbf{v}^{(i)}$ is called the Gumbel-Sigmoid trick [30,44], which is a differentiable estimation of sampling from a Bernoulli distribution with the Bernoulli parameter of $\texttt{sigmoid}(v^{(i)})$. We calculate the vector $v_l^{(i)} = [v_{l,w}^{(i)}]_{w=1}^{W}$ from the output vector $o_l^{(i)}$ (Tab. 1) such that the Bernoulli parameters $\texttt{sigmoid}(v_{l,w}^{(i)})$ follow the importance order for the width units c_w. By doing so, the probability of preserving the more important width units is higher than low-importance ones. Specifically, we calculate $v_l^{(i)}$ as follows:

$$y_l^{(i)} = \text{Softmax}(o_l^{(i)}) \tag{4}$$

$$p_l^{(i)} = \text{cumsum}(y_l^{(i)}) \tag{5}$$

$$v_l^{(i)} = \text{inverse-sigmoid}(p_l^{(i)} - \epsilon) \tag{6}$$

In other words, first, we calculate the Softmax of the output logits vector $o_l^{(i)}$. Then, we take the cumulative summation of the elements of $y_l^{(i)}$ as $p_l^{(i)}$ such that $p_{l,e}^{(i)} = \sum_{w=e+1}^{W} y_{l,w}^{(i)}$. Thus, $p_{l,1}^{(i)} > p_{l,2}^{(i)} > \cdots > p_{l,W}^{(i)}$. Finally, we calculate the inverse sigmoid function for the elements of the probability vector $p_l^{(i)}$ to obtain the vector $v_l^{(i)}$ (the small constant ϵ ensures numerical stability of the inverse sigmoid function). Doing so ensures that the ERA will preserve the more important width units with a higher probability than the low-importance ones.

Depth Architecture Vectors: We calculate the depth architecture vectors $u^{(i)}$ similar to the scheme for $v_l^{(i)}$:

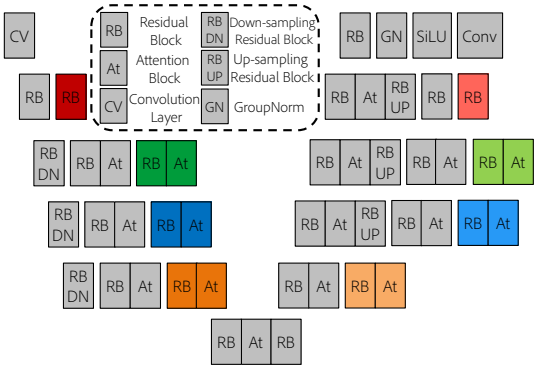

Fig. 7: U-Net architecture of the LDM [50].

$$y_{L+1}^{(i)} = \text{Softmax}(o_{L+1}^{(i)}) \tag{7}$$

$$p_{L+1}^{(i)} = \text{cumsum}(y_{L+1}^{(i)}) \tag{8}$$

$$u^{(i)} = \text{inverse-sigmoid}(p_{L+1}^{(i)} - \epsilon) \tag{9}$$

For the depth layers, we empirically found that removing the shallower depth layers results in a more severe increase in training loss values as well as degradation in sample quality. Similarly we observed that for the depth blocks in the same stage of the U-Net, the depth block in the decoder branch is more crucial to the model's performance than the one in the encoder branch. Thus, we rank the depth blocks based on 1) their stage and 2) the branch of the U-Net that the block belongs to. For instance, we rank the depth pruning blocks in Fig 7 as: $r =$ [decoder red block, encoder red block, decoder green block, encoder green block, decoder blue block, encoder blue block, decoder orange block, encoder orange block]. We then apply the elements of the soft relaxed depth pruning architecture vector $\mathbf{u}^{(i)}$ calculated from $u^{(i)}$ (Eq. 9 in the main text) with the same order as the ranking r to the depth blocks. By doing so, the probabilities that our method preserve the depth pruning blocks will have the same order as the ranking r.

Table 2: Hyperparameters of fine-tuning our models with elastic dimensions.

		LSUN-Bedroom	LSUN-Church	FFHQ	ImageNet
Elastic Depth Fine-tuning	Batch Size×Num GPU	32×8	32×8	32×8	32×8
	learning rate	9.6e-5	5e-5	8.4e-5	8e-5
	Iterations	200k	50k	30k	40k
Elastic Width Fine-tuning	Batch Size×Num GPU	32×8	32×8	32×8	32×8
	learning rate	9.6e-5	5e-5	8.4e-5	8e-5
	Iterations	130k	50k	30k	40k

Algorithm 1: Our Pruning Algorithm

Input: Training dataset $\mathcal{D} = \{(x_m, c_m)\}_{m=1}^{D}$ of images x_i and possible conditional inputs c_i; ERA model $h_{\text{ERA}}(z;\beta)$; Elastically fine-tuned experts E_i; pruning iterations G; Total MACs budget T_d
Output: Trained Expert Routing Agent.
for $e := 1$ *to* G **do**
1. Sample a mini-batch $(\mathbf{x}, \mathbf{c})$ from $\mathcal{D}$.
2. Calculate architecture vectors $(u^{(i)}, v^{(i)})$ using the ERA model $h_{\text{ERA}}(z;\beta)$ and Eqs. 6, 9.
3. Compute soft pruning vectors $(\mathbf{u}^{(i)}, \mathbf{v}^{(i)})$ using Eqs 7, 9.
4. Apply the soft pruning vectors $(\mathbf{u}^{(i)}, \mathbf{v}^{(i)})$ to the experts using Eqs. 8, 10, 11.
5. Calculate the interval denoising objectives $\mathcal{L}_{\text{DDPM},\mathcal{I}_i}(E_i(u^{(i)}, v^{(i)}))$ for the experts using the samples $(\mathbf{x}, \mathbf{c})$ in the mini-batch.
6. Compute the MACs regularization term $\mathcal{R}(\widehat{T}(u, v), T_d)$.
7. Compute the training objective $\mathcal{J}(T_d)$, backpropagate the gradients *w.r.t* the ERA parameters β and update them.

end
Return: Trained ERA model.

2.4 Pruning Algorithm

We present our pruning algorithm to train our Expert Routing Agent to select proper sub-networks of the experts in Alg.1.

3 Experiments

We provide more details about our experimental setup as well as experimental results in this section.

3.1 Setup

We implement our method upon the LDM codebase[3] and mainly follow the hyperparameter settings of the LDM [50]. We refer to Tables (12, 13) of LDM [4] for hyperparameters of the architecture of the pretrained models that we use in our experiments. We use the DDIM sampler [56] for sampling from our pruned models. We set the number of sampling steps to 100 for the LSUN-Bedroom, 200 for the FFHQ, and 250 for the ImageNet experiments. We conduct all of our experiments on a server with 8 NVIDIA A100 GPUs. We calculate the inference throughput value for each model by sampling a batch of 64 samples from it 100 times and averaging the throughput values. We provide more details of our experimental setup for each stage of our method in the following.

[3] https://github.com/CompVis/latent-diffusion
[4] https://arxiv.org/pdf/2112.10752.pdf

Fine-tuning with Elastic Dimensions Tab 2 summarizes the hyperparameters that we use to fine-tune our experts with elastic depth and width on their intervals. We adopt the learning rate values from the settings used to train the pre-trained checkpoints in the LDM [50] paper.

Pruning and Fine-tuning We provide the hyperparameters for the pruning and fine-tuning stages of our method in Tab. 3.

For the pruning stage of our method on all datasets, we use the AdamW optimizer [40] with a learning rate of 0.001, weight decay of 0.01, and beta parameters $(\beta_1, \beta_2) = (0.9, 0.999)$ to train the ERA model's parameters. We also set the temperature parameter τ for the Gumbell-Sigmoid [30] estimations to 0.4 for all experiments.

Ablation Experiments We prune all the baselines in the ablation experiments for 60k iterations. Then, we match their fine-tuning iterations with the summation of the iterations of our elastic depth fine-tuning, elastic width fine-tuning, and fine-tuning the mixture of efficient experts for the 50% MACs budget (550k iterations) for a fair comparison.

Table 3: Hyperparameters for the pruning and fine-tuning stages of our method for different MACs pruning ratios (30%, 50%, and 70%).

		Pruning			Fine-tuning		
Dataset	Parameters	30%	50%	70%	30%	50%	70%
LSUN-Bedroom	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	9.6e-5	9.6e-5	9.6e-5
	Iterations	70k	60k	50k	270k	220k	195k
LSUN-Church	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	5e-5	5e-5	5e-5
	Iterations	70k	60k	50k	165k	180k	90k
FFHQ	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	8.4e-5	8.4e-5	8.4e-5
	Iterations	40k	30k	20k	90k	85k	100k
ImageNet	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	8e-5	8e-5	8e-5
	Iterations	50k	40k	30k	205k	130k	135k

3.2 Comparison of Training Iterations

We provide a comparison of the number of training iterations for different methods to obtain a pruned model on LSUN-Bedroom and LSUN-Church in Tabs. 4, 5 respectively. For example, our method's total number of iterations is the summation of iterations for pre-training, elastic depth fine-tuning, elastic width fine-tuning, pruning, and fine-tuning the mixture of experts.

Although not directly comparable to SP [12] as it prunes a pixel-space DPM, our method can obtain a pruned model with significantly better quality with less

iterations than SP. This shows that LDMs have much fewer redundancies than pixel-space DPMs. Thus, they can converge faster and pruning them is more challenging.

On LSUN-Church, On the one hand, DiffPruning (70%) converges with only 0.74M iterations while pixel-space pruned model by SP [12] requires 4.9M iterations to converge with a 4.51 worse FID score. On the other hand, the comparison results with OMS-DPM [37] clearly demonstrate the value of our elastic fine-tuning. DiffPruning 50% and 30% require more than 7× less training iterations than OMS-DPM to obtain a performant mixture of efficient experts. The reason is that our elastic fine-tuning scheme provides an implicit model zoo within the experts for each interval without requiring to train multiple models from scratch to obtain a model zoo as done in OMS-DPM [37].

Table 4: Comparison of the number of training iterations for different methods on LSUN-Bedroom. The "Method's Iterations" column denotes the number of all the training iterations that the pruning method performs to obtain its final efficient model.

LSUN-Bedroom (256 × 256)						
	Complexity				Performance	
Model	Pre-training Iterations	Method's Iterations	Total Iterations	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
DDPM [28]	2.4M	-	2.4M	248.7G	0.74	6.62
SP [12]	2.4M	0.2M	2.6M	138.8G	-	18.6
LDM [50]	1.9M	-	1.9M	101.32G	2.01	4.39
DiffPruning (70%)	1.9M	0.575M	2.475M	70.84G	3.11	5.90
DiffPruning (50%)	1.9M	0.61M	2.51M	50.69G	3.75	6.73
DiffPruning (30%)	1.9M	0.67M	2.57M	31.11G	4.73	9.22

Table 5: Comparison of the number of training iterations for different methods on LSUN-Church. The "Method's Iterations" column denotes the number of all the training iterations that the pruning method performs to obtain its final efficient model.

LSUN-Church (256 × 256)						
	Complexity				Performance	
Model	Pre-training Iterations	Method's Iterations	Total Iterations	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
LDM [50]	0.5M	-	0.5M	20.96	5.19	5.21
DDPM [28, 56]	4.4M	-	4.4M	248.7G	0.74	10.58
SP [12]	4.4M	0.5M	4.9M	138.8G	-	13.9
DiffPruning (70%)	0.5M	0.24M	0.74M	14.64G	5.73	9.39
OMS-DPM [37]	0	>6M	>6M	-	2.56	11.10
DiffPruning (50%)	0.5M	0.34M	0.84M	10.48G	6.28	10.89
OMS-DPM [37]	0	>6M	>6M	-	6.4	13.7
DiffPruning (30%)	0.5M	0.335M	0.835M	6.35G	6.87	11.39

3.3 Errors in MACs Calculation

We mentioned in the caption of Tab. (1) as well as Sec. (4.1) of the paper that the MACs values reported by SP [12] for the LDM [50] models for the ImageNet experiments are inaccurate. We describe the reason in the following. SP [12] adopts the 'flops-counter.pytorch' package[5] to measure models' MACs. This package defines a hook for each of the standard PyTorch [49] layers like `nn.Conv2d` and keeps a mapping dictionary between standard PyTorch layers and their hooks. The package calculates the MACs of the model by performing the forward pass of the model with a random input and counting the layers' MACs using the defined hooks. Now, SP [12] implements the Attention layer in the U-Net architecture of LDM manually, and the defined Attention module is not an element of the mapping dictionary for the MACs calculation hooks. Thus, the package does not count the number of MACs for the scaled dot product attention operation as it is not a native PyTorch layer. For instance, SP reports that (Tab. (3) in SP [12]) the LDM model for ImageNet has 99.8G MACs. However, we manually implemented counting the MACs for the attention layers and found that the model actually has 108.78G MACs.

We found a similar problem in the numbers reported by SD [64] . For instance, SD reports that the LDM for LSUN-Church has 18.7G MACs. We could reproduce the same number when directly using the 'flops-counter.pytorch' package. Yet, we found that the model actually has 20.96G MACs after adding the attention layers' MACs.

4 Related Work

Mixture of Experts (MoE) Diffusion Models: MoE methods cluster denoising time-steps of DPMs into intervals and train a separate *expert* model for each. eDiff-I [1] supports developing MoE for DPMs by showing that different denoising time-steps have distinct roles. Yet, how to cluster time-steps is non-trivial. eDiff-I employs a complex tree-based-branching scheme to divide the denoising path into two intervals sequentially and initializes a child model by its parent. ERNIE-ViLG [13] and MEME [34] uniformly cluster the denoising time-steps. Yet, these heuristic schemes do not necessarily transfer to other tasks. Different from these methods, we propose to cluster the denoising time-steps by measuring the alignment between their training objectives. We emphasize that although a recent work [19] has explored the time-steps' alignment scores *in the course of training*, our paper is the first one to leverage them to cluster the time-steps for MoE DPMs.

Efficient DPMs. The majority of ideas for improving DPMs' efficiency reduce their denoising steps by faster samplers [42, 63, 69], distillation [22, 45, 54], better noise schedules [2, 32, 47, 56, 70, 73], learning denoising timesteps to use [60, 61], and caching [43]. We explore an orthogonal direction, compressing the architecture of DPMs. LSGM [58] and LDM [50] perform the diffusion process in a lower

[5] https://github.com/sovrasov/flops-counter.pytorch

dimensional latent space of an encoder-decoder pair [11, 32], thereby enjoying significantly faster sampling than pixel-space DPMs.

A few ideas have recently addressed compressing DPMs' architectures having two main categories. **Single-model** methods develop a single efficient model for all denoising timesteps. Structural Pruning (SP) [12] approximates weights' importance using the Taylor expansion and removes structures with low scores. Yet, SP's performance has been mainly verified on pixel space DPMs, and its pruned models on datasets like LSUN-Church [66] still have more than 6× MACs than the full-size LDM [50]. MobileDiffusion [72] introduces heuristics to enhance DPMs' efficiency and develops two efficient architectures. Nevertheless, it is highly non-trivial how to generalize the heuristics for different compute budgets. Spectral Diffusion (SD) [64] introduces a wavelet gating operator and performs frequency domain distillation from a teacher model into a small LDM. However, the main weakness of single-model methods is that they use the same model for all denoising steps, which is shown to be sub-optimal [1, 19]. **Mixture of expert** methods employ a separate model for different stages of the denoising process. OMS-DPM [37] trains a model zoo with various sizes and searches for a proper model schedule given a desired compute budget. Yet, gathering a model zoo is very costly on large-scale datasets, making OMS-DPM impractical for them. T-Stich [48] stitches several models with different sizes, each performing a part of the denoising process. But, similar to OMS-DPM [37], it requires several pretrained models of various sizes, making it costly for practical scenarios. MEME [34] and TMDA [67] cluster denoising timesteps and design a distinct expert for each. However, they need to manually allocate the compute budget between experts and re-design the experts for a new budget, which makes them cumbersome in practice. We propose to prune an LDM into a mixture of efficient experts. We cluster denoising timesteps into intervals using their alignment scores. Then, we fine-tune the pre-trained model with elastic dimensions on each interval to obtain our experts. Thus, our method gathers an *implicit* model zoo *within* each expert with much lower training iterations than OMS-DPM. Finally, we prune all experts simultaneously using our expert routing agent to obtain our mixture of efficient experts. By doing so, in contrast with MEME [34] and TMDA [67], our method automatically allocates the compute resource (*e.g.,* MACs) between experts.

Model pruning and architecture search. Our work is also related to model pruning [14,15,17,23,26,36,46] and Neural Architecture Search (NAS) [6,16,29,38, 65, 74] methods that prune the pretrained models and design novel architectures given a set of computational constraints. These ideas mainly focus on developing new architectures for image classification tasks, while we aim to design a novel pruning method for latent diffusion models [50]. We refer to recent surveys [7, 25, 62] for a detailed reviewing of pruning and NAS methods.