SMALL TRANSFORMERS, BIG RESULTS: EFFICIENT DIFFUSION WITH PARAMETER SHARING

Mohamed Osman
Department of Computer Science
Virginia Commonwealth University
Richmond, VA 23284, USA
osmanmw@vcu.edu

Daniel Z. Kaplan
realiz.ai
Edison, NJ, 08817
daniel.z.kaplan@gmail.com

ABSTRACT

The interplay between model depth, computational complexity, and parameter count remains an intricate aspect of neural network design. We propose a novel block sharing mechanism for denoising diffusion generative models, enabling us to maintain or even improve model quality while reducing parameter count. Our approach leverages the architectural homogeneity of Vision Transformers and demonstrates enhanced performance with less computational overhead on various datasets. We provide our code and pre-trained models to facilitate further research.

1 INTRODUCTION

Scaling up neural networks is a standard approach to improve performance across various tasks (Alabdulmohsin et al., 2023). However, this scalability often comes at the cost of increased computational resources, limiting the deployment on resource-constrained devices. Furthermore, the communication overhead in distributed training environments can significantly offset the benefits of scaling (Rajbhandari et al., 2020).

Denoising diffusion probabilistic models (DDPMs) (Ho et al., 2020) and score-based generative models (Song et al., 2020) have become prominent for generating high-fidelity images, detailed further in Appendix A. Leveraging these concepts, we enhance Vision Transformers with block sharing, an approach that exploits the repetitive nature of transformer blocks to simulate increased depth without a corresponding rise in parameter count. This technique strikes a balance between model size, computational efficiency, and performance. In line with this, parameter-efficient strategies like ALBERT (Lan et al., 2020), (Pires et al., 2023; Lin et al., 2022), have similarly aimed to boost performance without substantially increasing model dimensions or computational demands.

2 METHODS

2.1 MODEL ARCHITECTURE

Our models are based on Vision Transformers (Dosovitskiy et al., 2020; Vaswani et al., 2017), with a specific focus on image generation, utilizing the k-diffusion repository (Crowson et al., 2023). They utilize a patch-based approach to deconstruct images into sequences processed by transformer blocks, each consisting of a multi-head self-attention mechanism and a position-wise feed-forward network, and finally an unpatching operation to produce the image.

The model also supports conditional generation through an external mapping network, which produces vectors to describe several conditioning factors including class, diffusion timestep, as well as data augmentations. The models are conditioned using AdaRMSNorm (Zhang & Sennrich, 2019).

2.2 PARAMETER SHARING SCHEME

The parameter sharing scheme involves the cyclic repetition of a subset of transformer blocks. The initial conditioning is extended with information of how many times we have already seen the current block. We experiment with both interleaved (e.g., 1212...) and non-interleaved (e.g., 111...222...) configurations. This approach permits additional computational expenditure during training without

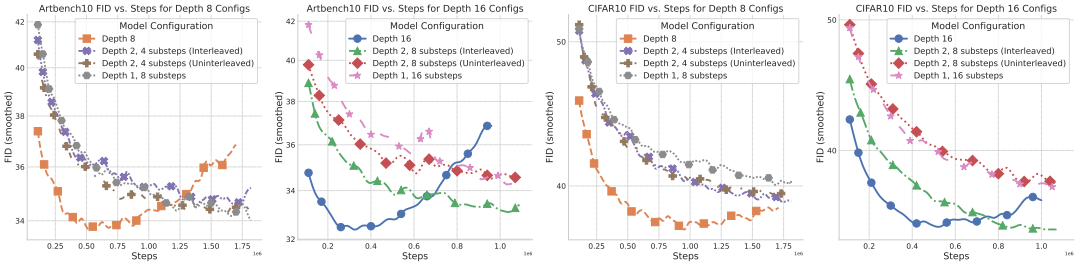

Figure 1: FID scores over training steps for different model configurations on the Artbench10 and CIFAR datasets. From left to right: Artbench10 - Depth 8, Artbench10 - Depth 16, CIFAR - Depth 8, CIFAR - Depth 16.

a corresponding increase in parameters, which we empirically find beneficial for generation fidelity. See Appendix B for pseudocode explaining interleaved vs non-interleaved models.

3 Experiments and Results

3.1 Training Configuration

Our models utilize a diffusion process following the framework established by (Karras et al., 2022), with simulated depth and two block evaluation strategies. Optimization is performed via AdamW with a constant learning rate and an exponential moving average for weights to ensure training stability. Detailed training parameters are available in Appendix C and within our code release.[1]

3.2 Quantitative Results

We present our results in Figure 1 and Table 1. We use Frechet Inception Distance (FID)(Heusel et al., 2017) and Kernel Inception Distance (KID) (Betzalel et al., 2022) to assess the quality of our generated images. Baseline models demonstrate the strongest overall performance, and within block sharing configurations, interleaved consistently outperforms non-interleaved. This suggests that interleaved blocks might learn to perform mutually beneficial transformations. Notably, in simulating a depth of 16, our interleaved model surpasses the baseline model on the CIFAR10 dataset. The greater intra-class variance inherent in Artbench10, as opposed to CIFAR10, contributes to the observed diversity in image quality between models.

Table 1: Best Checkpoint Results by Configuration for CIFAR-10 and Artbench-10. Steps are multiplied by 1e-6 and KID by 1e4. Results presented as Steps/FID/KID. For FID and KID, lower is better.

Dataset	D16	D1S16	D1S8	D2S4UI	D2S4I	D2S8UI	D2S8I	D8
CIFAR10	0.52/34.52/54.39	1.09/37.03/75.98	1.70/39.32/82.09	1.67/38.79/82.78	1.68/38.22/79.93	0.94/37.21/71.14	1.06/34.50/52.91	0.96/36.51/67.98
Artbench10	0.33/31.99/33.72	1.02/33.52/41.59	1.58/33.75/43.06	1.63/33.86/47.26	1.74/34.06/42.69	1.07/34.07/46.83	1.05/32.53/37.66	0.58/33.16/38.70

4 Discussion and Conclusions

Block sharing models exhibit higher training throughput but require more training steps to reach quality parity with baselines. It is worth noting that the hyperparameters tested are optimized for the baselines, further tuning could improve training speed for block-sharing.

Despite these computational demands, block sharing proves to be an efficient strategy in terms of memory and parameter usage, closely matching models 8 times larger parameter-wise.

Much future work remains in exploring block sharing models' training dynamics, especially in complex scenarios. This exploration is crucial to maximize block sharing's potential for large-scale, efficient model architectures, offering promising avenues in resource-efficient machine learning.

[1] `https://github.com/spaghettiSystems/tiny_diffusion`

URM STATEMENT

The authors acknowledge that at least one key author of this work meets the URM criteria of ICLR 2024 Tiny Papers Track.

5 ACKNOWLEDGEMENTS

High Performance Computing resources provided by the High Performance Research Computing (HPRC) core facility at Virginia Commonwealth University (https://hprc.vcu.edu) were used for conducting the research reported in this work. Special thanks to Professor Alberto Cano for providing these resources on short notice.

REFERENCES

Ibrahim Alabdulmohsin, Xiaohua Zhai, Alexander Kolesnikov, and Lucas Beyer. Getting vit in shape: Scaling laws for compute-optimal model design. *arXiv preprint arXiv:2305.13035*, 2023.

Eyal Betzalel, Coby Penso, Aviv Navon, and Ethan Fetaya. A study on the evaluation of generative models. *arXiv preprint arXiv:2206.10935*, 2022.

Katherine Crowson, Romain Beaumont, Tanishq Abraham, Jonathan Whitaker, and storyicon. crowsonkb/k-diffusion: v0.1.1.post1, December 2023. URL `https://doi.org/10.5281/zenodo.10284390`. `https://github.com/crowsonkb/k-diffusion`.

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

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. *Advances in neural information processing systems*, 30, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 35:26565–26577, 2022.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations, 2020.

Chien-Yu Lin, Anish Prabhu, Thomas Merth, Sachin Mehta, Anurag Ranjan, Maxwell Horton, and Mohammad Rastegari. Spin: An empirical evaluation on sharing parameters of isotropic networks, 2022.

Telmo Pessoa Pires, António V. Lopes, Yannick Assogba, and Hendra Setiawan. One wide feedforward is all you need, 2023.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

A BACKGROUND ON DDPM AND SCORE SDE/ODES

Denoising Diffusion Probabilistic Models (DDPMs): DDPMs are generative models that transform a noise distribution into complex data representations following a Markov chain. This process can be mathematically expressed as $x_{t+1} = \sqrt{1-\beta_{t+1}}x_t + \sqrt{\beta_{t+1}}\epsilon_{t+1}$, where β_{t+1} is the noise level at step $t+1$, and ϵ_{t+1} is Gaussian noise.

Score-Based Stochastic Differential Equations (SDEs): These models generalize DDPMs using SDEs, which describe the continuous-time evolution of a data sample x_t as $dx_t = f(x_t, t)dt + g(t)dB_t$. Here, $f(x_t, t)$ is the deterministic component, $g(t)dB_t$ represents stochastic diffusion, and B_t is Brownian motion.

Neural Network Implementation: Neural networks are trained to estimate the reverse of the diffusion process in these models. They approximate the score function $\nabla_x \log p_t(x)$, guiding the sample from noise to data distribution.

B INTERLEAVED VS NOT INTERLEAVED CODE

We present below Python pseudo-code showing how interleaved and non-interleaved executions are performed. They become identical when the value of 'substeps' is one.

```
if interleaved:
    for i in range(substeps):
        for block in self.blocks:
            x = block(x, pos, attn_mask, cond*(i+1))
else:
    for block in self.blocks:
        for i in range(substeps):
            x = block(x, pos, attn_mask, cond*(i+1))
```

Listing 1: Algorithm for Interleaved and Non-Interleaved Execution

C TRAINING CONFIGURATION DETAILS

Parameter	**Value**
Model Type	Image Transformer v1
Input Channels	3
Input Size	32×32
Patch Size	4×4
Width	1024
Depth	2
Loss Configuration	Karras
Loss Weighting	Soft-min-SNR
Noise Schedule Length	50
Dropout Rate	0.05
Augmentation Probability	0.12
σ_{data}	0.5
σ_{min}	1×10^{-2}
σ_{max}	80
Sigma Sample Density Type	Cosine-interpolated
Dataset Type	CIFAR-10
Dataset Location	Data directory
Number of Classes	10
Conditional Dropout Rate	0.1
Optimizer Type	AdamW
Learning Rate (LR)	5×10^{-4}
LR Betas	[0.9, 0.95]
LR Epsilon	1×10^{-8}
Weight Decay	1×10^{-4}
LR Schedule Type	Constant
LR Schedule Warmup	0.0
EMA Schedule Type	Inverse
EMA Schedule Power	0.6667
EMA Schedule Max Value	0.9999

Table 2: Detailed configuration parameters for model training.