Compressing and Accelerating Stable Diffusion

Geoffrey Liu
Harvard University
geoffreyliu@fas.harvard.edu

Aayush Karan
Harvard University
akaran@college.harvard.edu

Alex Kashi
Harvard University
alexkashi@fas.harvard.edu

***Abstract*—Since the release of DALLE-2 and MidJourney, large generative image models have become popular among both technical and non-technical users for creative purposes. Recently, one such model, Stable Diffusion, was open-sourced – however, access to the models is limited to those who have high-end graphics cards or pay a fee for each image to be generated on a server. In this paper, we explore methods for compressing and accelerating Stable Diffusion, resulting in a final compressed model with 80% memory size reduction and a generation speed that is ~ 4x faster, while maintaining text-to-image quality.**

I. Introduction

Diffusion models (DMs) use diffusion processes to decompose image generation into sequential applications of denoising autoencoders. Different variations on the DM architecture – Imagen (Google) , DALL-E 2 (OpenAI), MidJourney, and Stable Diffusion – have recently achieved state-of-the-art synthesis on image data and have become some of the most popular architectures for image-to-image (img2img) and text-to-image (txt2img), among other types of synthesis such as image outpainting and infilling [1]–[4]. Various attempts at making these models accessible to a broader audience include various API's allowing for straightforward generation on a small scale.

Notably, Stable Diffusion – a text-to-image generator – has since been open-sourced, allowing users to recreate such groundbreaking results with access to internal architecture and fully trained weights. However, this capacity is bounded largely by memory requirements, with weight size ranging from 4.27 GB - 7.7 GB (depending on model version), meaning Stable Diffusion is locally inaccessible to most devices. In fact, non-technical users must pay per image for running the model externally to their hardware, with generation time on, e.g. Colab, requiring 30-60s to generate a single prompt-to-image query.

To remedy such barriers to broader access, we present methods to compress memory size and accelerate the generation of diffusion models, demonstrating their efficacy on Stable Diffusion. In particular, we explore zero-shot post-training quantization. Crucially, the patchwork of distinct composite architectures in Stable Diffusion motivates a mixed-precision quantization. To leverage this insight, we explore model sensitivities to such techniques on a subunit-by-subunit basis. To augment generation speed, we supplement the Stable Diffusion source code with Flash Attention [6]. Finally, we present a compressed and accelerated version of Stable Diffusion that maintains comparable image quality with a weight size of 0.75 GB and an image generation time of 2.05s (using an RTX3090) – demonstrating over an 80% reduction in storage and ~ 4x reduction in generation time. Such developments promise broad commercial accessibility to the powerful resources of diffusion models.

II. Background

A. Diffusion Models

Diffusion models have recently overtaken the long-time dominance of generative adversarial networks to become the new state-of-the-art family of deep generative models for image generation. These models are a family of probabilistic generative models that progressively destroy data by injecting noise, subsequently learning to reverse this noising process to generate images [7]. During training, the model learns via a forward diffusion process, where Gaussian noise is added iteratively to the image vector. As the number of steps increase, the original image becomes indistinguishable from Gaussian noise. During inference time, another architecture – typically a denoising auto-encoder – learns to reverse this process by subtracting noise given a time step to reconstruct the original image. In diffusion models such as DALLE-2 and Imagen, the diffusion process takes place in image-space, leading to a large inference cost due to high image dimensionalty.

B. Latent Diffusion (LDM)

Rather than train the diffusion process on pixel-space, the authors in [8] propose to first train a variational auto-encoder to compress images into a latent space, next training a diffusion process on this space to generate a 'latent-space' image. The encoding is then passed through a variational autoencoder-decoder architecture to obtain an image. This allows a drastic training and inference speedup, since the diffusion process takes place on the 64x64 latent space, rather than a 512x512.

Figure 1 shows the exact architecture of the LDM that we consider in this paper. The diffusion process takes place in a 64x64 latent space, the decoder then decodes the output latent vector into a 512x512 image. The text-to-image generation in Stable Diffusion can be broken down into three distinct stages: a text encoder, the de-noising text-conditioned U-Net, and decoder of the variational auto-encoder.

a) Text/Image encoder: The text/image encoder is a frozen pretrained CLIP ViT-L/14 text encoder [5]. CLIP (Contrastive Language Image Pretraining) consists of an image encoder and text encoder. The model is provided by OpenAI, trained with a contrastive loss to map text inputs and image inputs onto the same space using a novel dataset of 400 million image-text pairs. During LDM inference, this CLIP encoder embeds the text prompt into a 77x768 dimension text-vector. Since CLIP maps image and text inputs into the same space, it is easy to apply our methods for img2img generation, although in this paper we only consider txt2img generation.

b) Denoising text-conditioned UNet: The denoising text-conditioned UNet is the backbone of the diffusion model, and it is the only module in the LDM architecture used in the diffusion process. The UNet takes in three main inputs:

- 64x64 time-step embedding which tracks the current time-step in the diffusion process
- 64x64x3 'latent' image vector, which is decoded into a 512x512 image by the decoder.
- 77x768 text embedding from the CLIP model, which is used to condition the UNet

The LDM UNet has the same architecture as the traditional UNet [9] with two minor changes, the first change is the addition of a time-step embedding which keeps track of the current iteration of the diffusion process, the second change is that before each downsample and upsampling layer in the UNet, there is a cross-attention layer between the latent image input and the text embedding.

c) Varational autoencoder decoder: The final output of the diffusion process is passed into the Variational Autoencoder Decoder, which decodes the 64x64x3 latent image vector to a 512x512x3 image. The variational autoencoder is pretrained on a 2.3B subset of the LAION-5B data set (a dataset that consists of 5.85 billion image-text pairs), filtered for high quality images. The encoder of the variational autoencoder is only required during training of the UNet and not used during txt2img or img2img inference.

III. Problem Statement

With this diverse composite architecture of Stable Diffusion, we are motivated to specifically tailor the tools of quantization to optimize model compression while maintaining text-to-image generation quality. Our project goal is to achieve such a compressed diffusion model to under 1 GB with at least a 2 – 5x speed up in inference. To this end, we partition the diffusion architecture into well-defined subunits, or blocks, calculating relative L2 error at the output as a proxy for local quantization tolerance.

The insights gleaned from this sensitivity exploration informs contender compressed models by mixed-precision quantization, which we test against well-established metrics for image quality: Inception Score (IS), Frechet Inception Distance (FID) and the Kernel Inception Distance (KID) [4], [10], [11]. The FID score is a distance metric between distributions of images, typically between a ground truth set with associated text captions and a set generated from such captions, and is

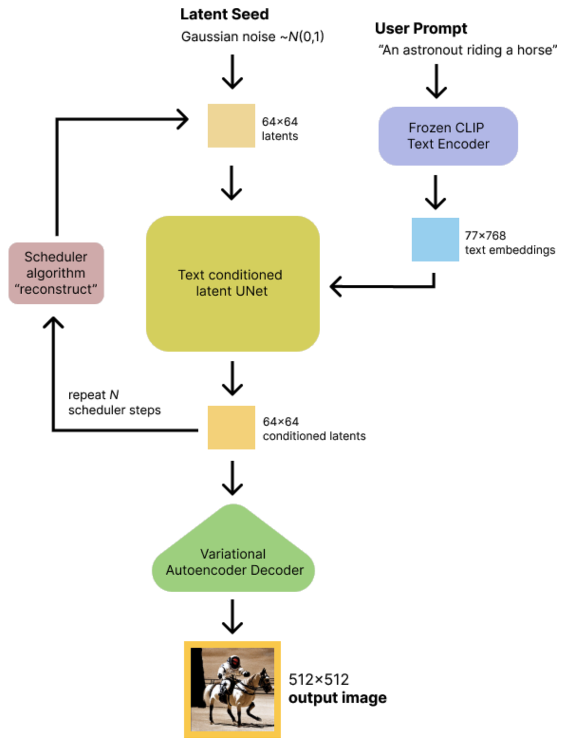

Fig. 1: Stable Diffusion Model Architecture during model inference.

most prominently used in diffusion model benchmarking. The IS and KID capture similar sentiments of distribution distance, and we refer the reader to the citations for further explanation. Our final model demonstrates one strong example of sensitivity-informed mixed-precision quantization, for which we also benchmark according to these metrics, but relative to the performance of the original Stable Diffusion model.

IV. Methods

A. Quantization

Model weights are typically stored in a 32-bit floating point format. This level of precision is not strictly necessary, motivating conversion methods to n-bit floating point numbers. At base level, given numbers in the interval $[0, 1]$, a uniform quantization acts as $\mathcal{Q}_{\text{n-bit}} : i \to \frac{\text{round}(i\cdot(2^n-1))}{2^n-1}$, rounding i to the nearest n-bit wide interval. We examine the efficacy of the following extensions of this quantization.

a) Uniform Quantization: This applies an n-bit quantizer to an arbitrary tensor of weights w by the following formula:

$$w_q = 2\text{max}(|w|)\left(\mathcal{Q}_{\text{n-bit}}\left(\frac{w}{2\cdot\text{max}(|w|)}+\frac{1}{2}\right)-\frac{1}{2}\right),$$

scaling w down to $[0, 1]$, quantizing, and rescaling.

b) Stochastic Quantization: This adds stochastic noise to uniform quantization by adding $N(n)$ to the quantizer input, with $N(n) = \frac{\sigma}{2^n-1}$ and $\sigma \sim \text{Unif}(-0.5, 0.5)$.

c) Piecewise Linear Quantization: Weight distributions typically conform to a center-peaked symmetric shape, such as a Gaussian or Laplacian. To take advantage of this structure, instead of quantizing by taking the entire weight range into account, splitting the weights into a tail and center region and quantizing separately gives less quantization error, as rounding to the nearest n-bit number is more precise in smaller ranges. To this end, let σ be the standard deviation of w. Then if $x = \frac{\max(|w|)}{\sigma}$, we can obtain the approximately optimal breakpoint $b = \sigma \cdot \log(m \cdot x + b)$ as calculated by [13], where m and b are constants. Then we perform stochastic quantization on the intervals $[-\max(|w|), -b]$, $[-b, b]$, and $[b, \max(|w|)]$ separately. In experimentation and consultation with [13], we set $m = 0.8030$ and $b = -0.3167$.

The variable quantization grid (6) suggests that piecewise linear quantization (PWLQ) offers the best performance guarantees per a given model size, so this is the quantization method we employ for the remainder of the paper.

B. Mixed-Precision Quantization Search

Uniformly quantizing a model to ultra low-precision can cause significant degradation in the quality of generated images. This issue can be addressed with mixed-precision quantization, where each layer in the model is quantized with a potentially distinct bit precision. One challenge here is selecting the optimal precision per layer, which becomes exponentially harder as the number of layers increases. Moreover, since the inference for LDM's is a combination of multiple neural architectures, the search becomes again harder since there are different inputs and outputs at each stage. In our paper, we calculate the sensitivity of each layer to different levels of quantization to understand which layers can tolerate lower bit precision, and which layers require relatively high bit precision.

Previous work in mixed-precision quantization focuses on relatively simple architectures, mostly working on ResNet50 and similar image-classification models [12]. These models have a quick-to-calculate and well-defined objective measure, classification accuracy. We found that these methods were impossible for our model since both inference and scoring of model outputs were very costly.

a) Quantization Sensitivity: To identify which blocks – composed of 5-10 consecutive layers [1] – are most sensitive to quantization, we sectioned the model into three components, the text encoder, the U-net, and the decoder, where each component contains several blocks. We subsequently quantized each block to 4 and 8 bits using piecewise linear quantization while leaving every other block at 32 bits. We then calculated the relative L2 error at the output at each of the three components as.

$$L = 100\% \sqrt{\frac{\sum_{i=1}^{n} (v^i_{\text{n-bit}} - v^i_{\text{32-bit}})^2}{\sum_{i=1}^{n} (v^i_{\text{32-bit}})^2}}$$

The loss is then averaged across 8 validation prompts. From this information, we can identify blocks which outputs are highly sensitive quantization. Highly sensitive blocks should be reverted to higher precision, while blocks more amenable to quantization should be further reduced.

b) Flash Attention: Within the diffusion process, there are many transformer blocks, which allow for the conditioning of the diffusion process to the input text. These transformer blocks are slow and memory-hungry because the complexity of self-attention and cross-attention are quadratic with respect to the size of the latent space. To address this issue, we replace the standard attention calculation in the model (quadratic with respect to latent size) with the Flash Attention, proposed by [6]. Flash Attention is an IO-aware exact attention algorithm that uses tiling to reduce the number of reads and writes between GPU high bandwidth memory (HBM) and GPU on-chip SRAM. To implement Flash Attention, we utilize [6], who implements the algorithm for CUDA devices with a PyTorch API.

V. Results

a) Mixed-Precision Sensitivity: By performing the block-wise quantization sensitivity analysis as detailed above, the results detailed in Fig 5. The U-net, specifically the center of the U-net is most amenable to quantization only resulting in a 0.1% change to the output of the decoder when quantizing to 4-bits. Another critical observation is that the decoder is highly sensitive to quantization resulting in over 100% relative error at the output when quantizing the last layer to 4-bits. However, quantizing the first layer of the decoder to 4-bits from 8-bits makes almost no difference. The text-encoder's quantization error remains constant at about 10% error when quantized to 4-bits. The results from this figure will go on to inform our mixed precision quantization strategies for the rest of the report.

b) Generative Quality: We now observe some tradeoffs between chosen mixed-precision quantizations, where the U-Net is quantized one degree (i.e. 1 byte) further than the rest of the model, as informed by the prior paragraph.

To measure the generative quality after quantization, we use the Inception Score (IS), Frechet Inception Distance (FID) and the Kernel Inception Distance (KID). To calculate these scores, we randomly select 4096 image-text pairs from MS-COCO 2014. As each image in the dataset is paired with 5 corresponding text captions, we randomly select one text caption to represent the image. We then use our model to generate images using these text prompts and score these generated images against the real images. The MS-COCO images were then resized to 256x256 by first resizing the short edge to 256 while keeping the aspect ratio, then center-cropping the resized image to obtain a 256x256 image. A comparison of these scores by bit precision can be found in Table I. We can see that relative UNet quantization outperforms full model quantization on most metrics by roughly keeping size intact, but full 4-bit quantization of the UNet cascades to many errors.

c) Relative Error: Although measuring the absolute image quality with respect to real images from MS-COCO allows us to quantitatively assess the generative quality of the model after quantization, these types of metrics are not suitable for

[1]Specific block composition can be found in our source code prune_names2.py

TABLE I: Image quality vs bit precision

Note: The higher score of stochastic and deterministic uniform quantization are presented in the 'Full model quantization' and 'UNet quantization', while the Final Model uses PWLQ.

Models		Image Quality			
		IS	FID	KID	Size (GB)
Full model quantization	32bit	33.63	23.92	0.0058	4.1
	16bit	34.95	25.09	0.0029	1.92
	8bit	32.69	26.64	0.0042	0.96
UNet quantization	16bit-UNet	33.85	24.02	0.0058	2.24
	8bit-UNet	32.86	26.21	0.0037	1.12
	4bit-UNet	1.382	125.27	0.3930	0.56
Final Model Quantization	Our model	32.06	25.51	0.0035	0.76

comparing the similarity of the outputs of the quantized model and the original model. Therefore, calculating the output quantization error, i.e. the difference in output between the model after quantization and the original 32-bit model, we consider two metrics: the relative FID score (rFID), relative KID score (rKID) and the mean-squared error between the latent vectors of the quantized model and 32-bit model (MSE). The rFID score is the FID score between the set of images generated by the quantized model and the 32-bit model using 4096 MS-COCO prompts. The MSE is simply the mean-squared error between the final latent vectors (output of the UNet) of the quantized and 32-bit model. Because the auto-encoder decoder provides a continuous mapping between latent and pixel space, it is valid to assume that small differences in the latent space vectors correspond to small differences in the generated image. We are able to see a clear downward trend in all these scores, and when we quantized too far (4bit-UNet), there is a sharp degradation in our metrics.

TABLE II: Relative image quality vs bit precision

Note: the models presented here are the same as in Table I

Models		Latent quality		
		rFID	rKID	rMSE
Full model Quantization	32bit	0.00	0.00	0.00
	16bit	15.53	0.0005	1.447
	8bit	17.67	0.0012	1.483
UNet Quantization	16bit-UNet	15.82	0.0005	1.401
	8bit-UNet	16.61	0.0007	1.449
	4bit-UNet	361.50	0.3767	93.58
Final Model Quantization	Our model	16.71	0.0008	1.423

d) Final Model: For the final model, we selectively quantize all input layers and middle blocks in the UNet to 4-bit using PWLQ (from 3 this portion has the least deviation from 8-bit quantization), and all the convolution layers to 4-bit using PWLQ, with the rest of the model weights are quantized to 8-bit using PWLQ. This mixed-precision quantization scheme is chosen by inferring the sensitivity of layers to quantization. Our final model performs similarly to a full 8bit PLWQ quantized model, with 0.2GB weight saving due to mixed-precision quantization. It also outperforms full 8bit deterministic and stochastic quantization.

e) Compressed: After applying our quantization methods, our final model achieves a weight size of 0.762GB which is a ∼5.5 times decrease in the original weight size while remaining true to the original full-bit model outputs. We achieved this by applying 4-bit PWLQ on the UNet input and middle blocks, 4-bit PWLQ to convolutional layers in the model, and 8-bit PWLQ quantization for all other layers.

f) Speedup in inference: From both quantization of weights and using Flash Attention algorithm within the transformer blocks, we achieve a lower bound inference speedup of ∼4x from 7.6 seconds to 2.05 seconds on the hardware we tested on (RTX3090), compared to full 32-bit inference without Flash Attention. This is a lower bound speed-up since our hardware is optimized only for 16-bit (half-precision) multiplications and accumulation calculations. Using specialized hardware (and software) that can take full advantage of 8-bit and 4-bit to compute units will further increase the inference speedup.

VI. Conclusion

A. Summary

In this project, we conducted a subunit-by-subunit test of quantization sensitivity as proxied by the relative L2 error at the block output, informing which subarchitectures of the multifoliate Stable Diffusion model were most amenable to low precision quantization. We found that the middle layers of the U-Net are least sensitive to quantization, while the decoder is most sensitive. These varying tolerances informed a final model that quantized all middle to early layers of the U-Net as well as all convolutional layers to 4-bit, while quantizing the remainder of the model to 8-bit. Our final model achieves an 80% memory size reduction and $\sim 4x$ generation speedup, while maintaining metric scores roughly equivalent to far more precise quantizations.

B. Future research

Since we began the project, further improvements to Stable Diffusion have been released. These improvements include better optimizations during training (Stable Diffusion v2), the release of a paper by Apple engineers porting the Stable Diffusion model natively on Swift allowing inference on Apple Neural Engine [2], as well as a distilled student model that can generate similar-quality images using $\sim 1-4$ diffusion steps (while the current model requires 50 steps) which increase inference speed by 10x.

Given this, some future areas of research could explore:

- Hardware level optimizations in porting our model weights onto iOS devices, such as adding Flash Attention to their version of the model.
- Applying our quantization methods to the distilled student model, allowing a considerable inference speed-up.

[2]https://machinelearning.apple.com/research/stable-diffusion-coreml-apple-silicon

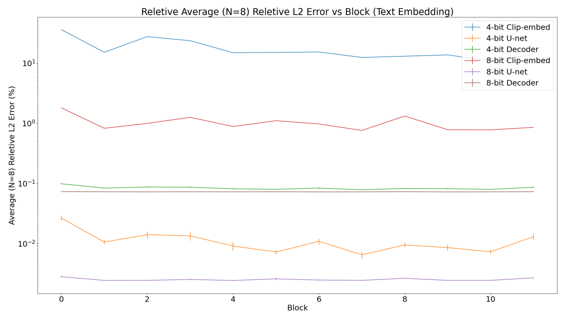

Fig. 2: Blockwise relative L2 error, for text encoder compared with 32-bit reference, averaged over 8 samples

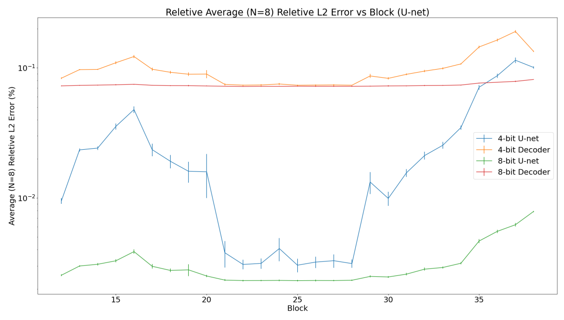

Fig. 3: Blockwise relative L2 error, for U-net compared with 32-bit reference, averaged over 8 samples

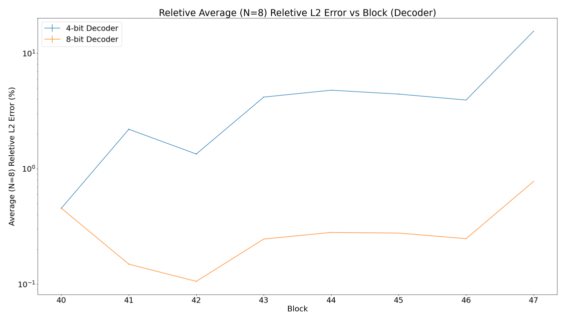

Fig. 4: Blockwise relative L2 error, for decoder compared with 32-bit reference, averaged over 8 samples

Fig. 5: Blockwise Error

References

[1] Saharia, Chitwan, et al. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding." arXiv preprint arXiv:2205.11487 (2022).

[2] Ramesh, Aditya, et al. "Hierarchical text-conditional image generation with clip latents." arXiv preprint arXiv:2204.06125 (2022).

[3] Rombach, Robin, et al. "High-resolution image synthesis with latent diffusion models." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022.

[4] Salimans, Tim; Goodfellow, Ian; Zaremba, Wojciech; Cheung, Vicki; Radford, Alec; Chen, Xi; Chen, Xi (2016). "Improved Techniques for Training GANs". Advances in Neural Information Processing Systems. Curran Associates, Inc. 29.

[5] Radford, Alec, et al. "Learning transferable visual models from natural language supervision." International Conference on Machine Learning. PMLR, 2021.

[6] Dao, Tri, et al. "Flash Attention: Fast and Memory-Efficient Exact Attention with IO-Awareness." arXiv preprint arXiv:2205.14135 (2022).

[7] Yang, Ling, et al. "Diffusion models: A comprehensive survey of methods and applications." arXiv preprint arXiv:2209.00796 (2022).

[8] Rombach, Robin, et al. "High-resolution image synthesis with latent diffusion models." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022.

[9] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

[10] Heusel, Martin; Ramsauer, Hubert; Unterthiner, Thomas; Nessler, Bernhard; Hochreiter, Sepp (2017). "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". Advances in Neural Information Processing Systems. 30.

[11] Binkowski, Mikolaj, Danica J. Sutherland, Michal Arbel and Arthur Gretton. "Demystifying MMD GANs." ArXiv abs/1801.01401 (2018).

[12] Gholami, Amir, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney and Kurt Keutzer. "A Survey of Quantization Methods for Efficient Neural Network Inference." ArXiv abs/2103.13630 (2021).

[13] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H. Hassoun. 2020. Post-training Piecewise Linear Quantization for Deep Neural Networks. In Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II. Springer-Verlag, Berlin, Heidelberg, 69–86. https://doi.org/10.1007/978-3-030-58536-5$_5$

Appendix A
Group contribution statement

A. *Geoffrey*

- Implement Flash Attention
- Write batch-image generation script
- Benchmark speed and memory requirements
- Quantization (Stochastic & Uniform)
- Mixed-precision Quantization experiments
- Metrics (FID, IS, KID, rFID, rKID, rMSE)
- Calculating image quality scores and quantization errors on MS-COCO dataset.
- Experimental design and going through the LDM codebase to understand how SD works.
- Final project writeup

B. *Aayush*

- Stochastic and Uniform Quantization and Comparison Analysis
- Piecewise Linear Quantization
- Unstructured Pruning
- Optimal Model Search
- Final project writeup

C. *Alex*

- Project Proposal
- Per-layer quantization error-analysis
- Structured Pruning

B Comparison of Quantization Methods

A comparison of uniform, stochastic uniform, and piecewise linear quantization can be found in 6.

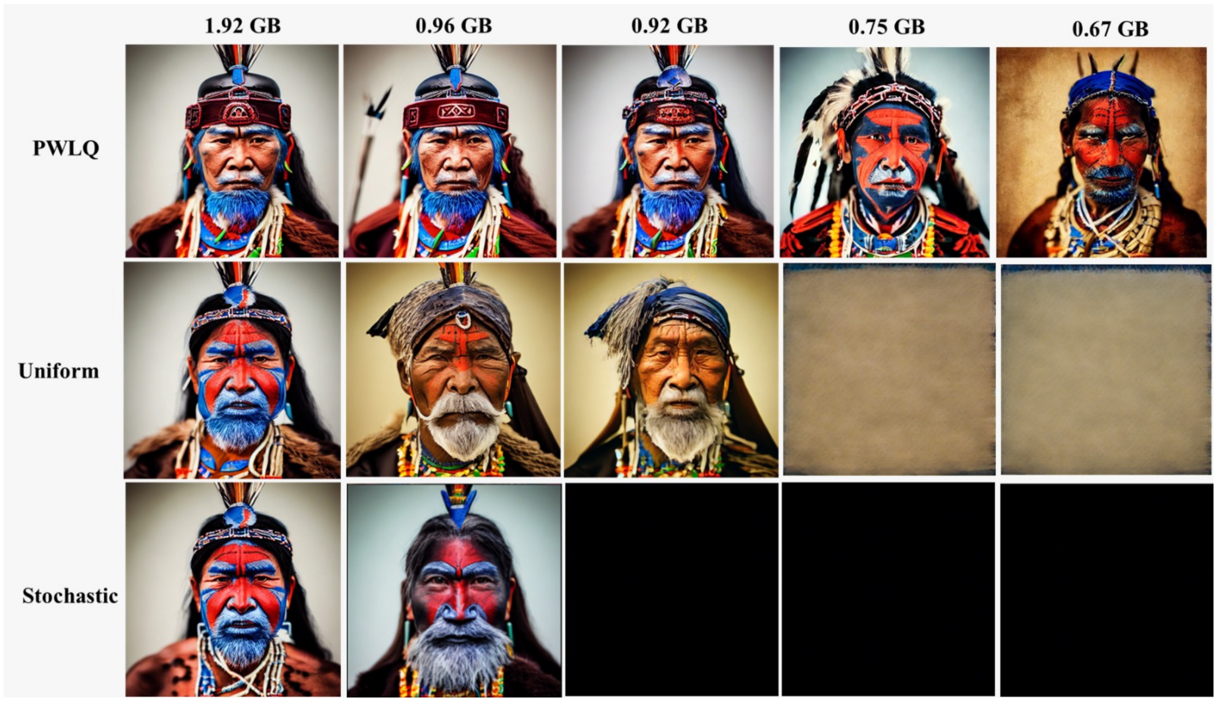

Fig. 6: Comparison of Quantization Methods. Example generated figures are given for piecewise linear (PLWQ), uniform, and stochastic quantization per given model sizes, labeled at the top of columns. The first two model sizes are given by full 16-bit and full 8-bit quantization. The third is 4-bit for convolutional layers and 8-bit elsewehere; the fourth is 4-bit convolutional and transformer layers and 8-bit elsewhere, and the final size is 4-bit convolutional, transformer, and middle U-Net layers, with 8-bit elsewhere. We stop before full 4-bit quantization as that outputs null images for all methods.

C Final Model Image Generations

Here (7) we display the capacity of our model to generate images from text competitive with the original stable diffusion model.

D Further Future Work

The current implementation of Stable Diffusion on iOS devices requires lazily loading the Text Encoder, UNet and Decoder and aggressively unloading each module after each step. This is due to the limited memory on these devices and impacts inference speed considerably. Using our reduced model, as well as Flash Attention, it would be possible to run inference without loading and unloading each model component.

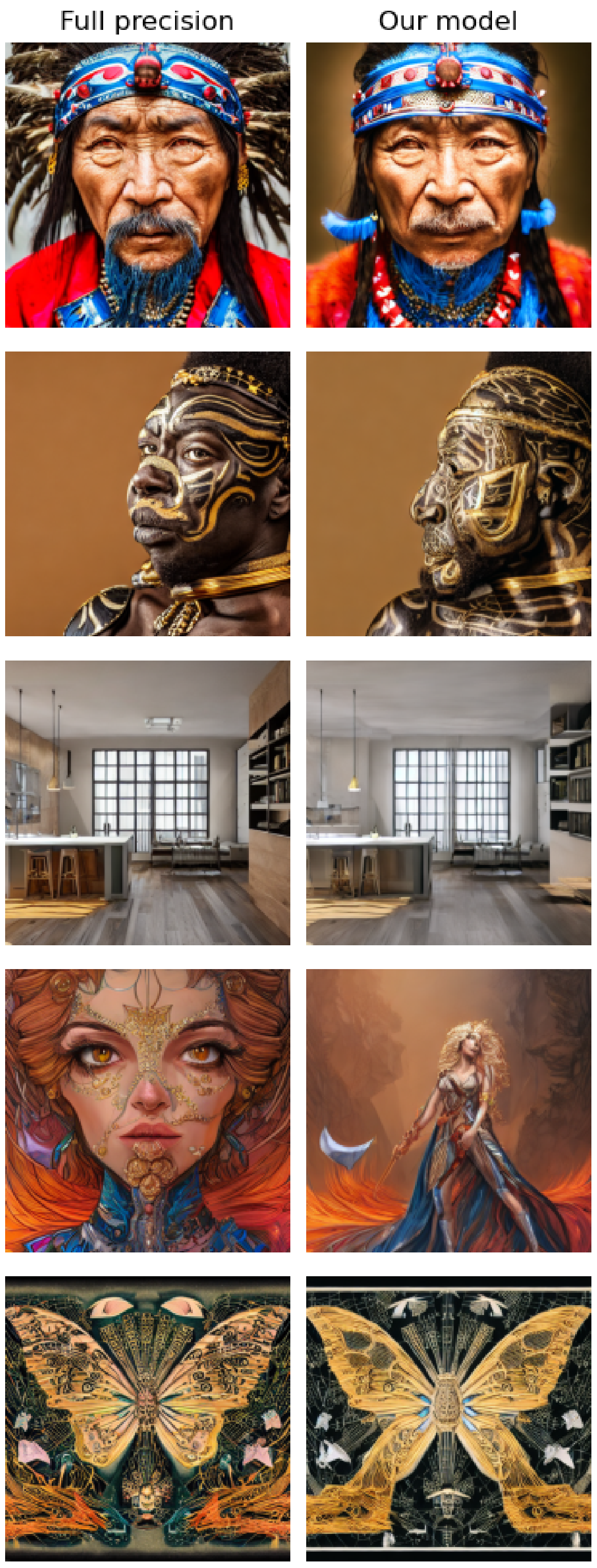

Fig. 7: Comparison of 32-bit output and our mixed 4bit and 8bit precision model on some sample prompts. Our model consists of 4bit convolutional layers using PLWQ, 4bit UNet Input Block and Middle Block layers using PLWQ and 8bit PLWQ quantization for all other layers.