Squeezing Large-Scale Diffusion Models for Mobile

Jiwoong Choi [1] **Minkyu Kim** [1] **Daehyun Ahn** [1] **Taesu Kim** [1] **Yulhwa Kim** [2] **Dongwon Jo** [2] **Hyesung Jeon** [2] **Jae-Joon Kim** [2] **Hyungjun Kim** [1]

Abstract

The emergence of diffusion models has greatly broadened the scope of high-fidelity image synthesis, resulting in notable advancements in both practical implementation and academic research. With the active adoption of the model in various real-world applications, the need for on-device deployment has grown considerably. However, deploying large diffusion models such as Stable Diffusion with more than one billion parameters to mobile devices poses distinctive challenges due to the limited computational and memory resources, which may vary according to the device. In this paper, we present the challenges and solutions for deploying Stable Diffusion on mobile devices with TensorFlow Lite framework, which supports both iOS and Android devices. The resulting Mobile Stable Diffusion achieves the inference latency of smaller than 7 seconds for a 512 $\times$ 512 image generation on Android devices with mobile GPUs.

1. Introduction

Recently, diffusion models have gained significant interest by achieving impressive performance in image synthesis and related tasks. Since the public release of Stable Diffusion (Rombach et al., 2022), one of the foundation models in diffusion models, there has been a surge of interest in exploring the potential of the diffusion models in various fields including image synthesis (Ho et al., 2020; Song et al., 2021; Rombach et al., 2022; Ho & Salimans, 2022; Saharia et al., 2022), super-resolution (Li et al., 2022; Sahak et al., 2023; Gao et al., 2023), inpainting (Lugmayr et al., 2022; Nichol et al., 2022; Avrahami et al., 2022; Gao et al., 2023), and many other applications (Luo et al., 2023; Blattmann et al., 2023; Yang et al., 2023; Liu et al., 2023).

[1]SqueezeBits Inc., Seoul, South Korea [2]Seoul National University, Seoul, South Korea. Correspondence to: Hyungjun Kim <hyungjun.kim@squeezebits.com>.

Workshop on Challenges in Deployable Generative AI at International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA. 2023.

Deploying large diffusion models on mobile devices offers significant advantages such as reduced server costs and improved user privacy, but it presents unique challenges. These challenges arise from the large number of parameters, typically exceeding one billion, which necessitates compressing the model for deployment on mobile devices. Moverover, ensuring that the computation latency remains within an acceptable range is also a crucial consideration.

In this paper, we introduce the implementation of Mobile Stable Diffusion based on the Stable Diffusion v2.1, achieving the lowest inference latency on GPU-powered Android devices, to the best of our knowledge ($\sim$7 seconds on Samsung Galaxy S23 to generate a 512 $\times$ 512 image).

2. Background

Diffusion models utilize the reverse diffusion process to generate images from noise. These models have been recognized for their ability to address significant challenges in the field of image synthesis. Specifically, they mitigate problems such as mode-collapse, training instability, and quality degradation that are commonly encountered in previous approaches such as Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs). Ho et al. (2020) initially showcased the capability of diffusion models in generating high-quality images, although they came with high computational costs. Subsequent works (Song et al., 2021; Rombach et al., 2022) have focused on reducing the computational cost of diffusion models. Song et al. (2021) introduced a method to decrease the number of denoising steps based on the non-Markovian diffusion process. On the other hand, Rombach et al. (2022) proposed to improve efficiency of diffusion models by applying denoising steps on latent space.

The advancement in improving efficiency in diffusion models contributed to the development of Stable Diffusion, a latent diffusion model for high-resolution image generation. Stable Diffusion has demonstrated impressive capabilities in both text-to-image and image-to-image synthesis tasks. The model combines three modules to implement text-to-image synthesis; a Contrastive Language–Image Pre-training (CLIP) module that generates guidance from a given text prompt (text encoder), a U-Net module that conducts

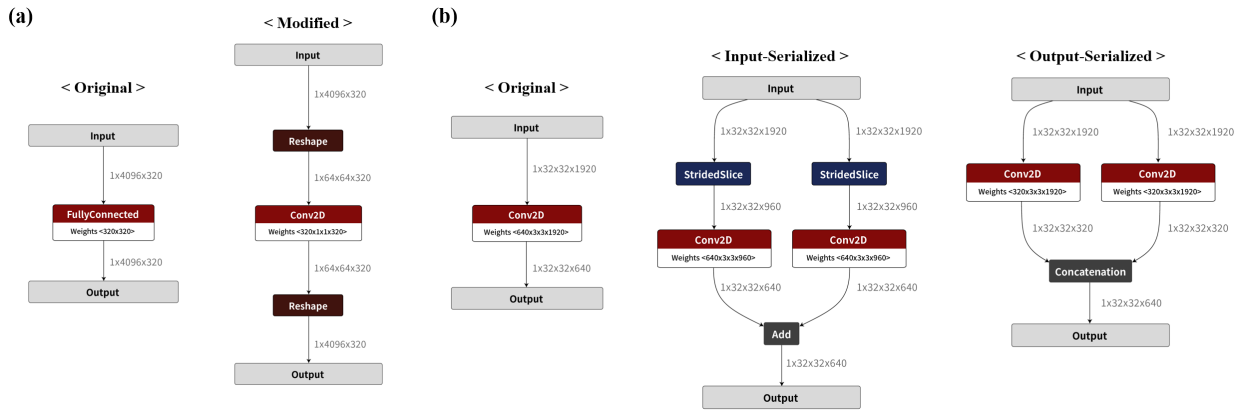

Figure 1. (a) Converting a fully-connected layer into a Conv2D layer. (b) Input- and Output-Serialization of a large Conv2D layer.

the reverse diffusion process (denoising network), and a Decoder module from a VAE model that generates an image from the output latent tensor (image decoder).

There is a growing demand for on-device image synthesis using the diffusion models, with a focus on enhancing the models in terms of latency, scalability, and user privacy. Orhon et al. (2022) introduced the official support for on-device computations of Stable Diffusion on iOS mobile devices. On Android devices, Hou & Asghar (2023) recently announced the first mobile deployment of Stable Diffusion based on the Hexagon processor of the latest Snapdragon 8 Gen 2 platform. Chen et al. (2023) has also demonstrated a faster implementation of Stable Diffusion using mobile GPUs based on private OpenCL kernels. While prior works have demonstrated the feasibility of deploying Stable Diffusion on-device, these works commonly relied on custom-built kernels for acceleration. Particularly in the case of Android devices, Hou & Asghar (2023) relied on the Hexagon processor and the dedicated SDK. Additionally, Chen et al. (2023) reported extensive use of private OpenCL-based kernels, pursuing additional performance gain with optimized memory access and faster computation.

3. Challenges and Proposed Solutions

We have chosen Google's TensorFlow Lite (TFLite) runtime (Google, 2017) as our deployment framework, rather than constructing custom-built kernels. Opting for TFLite offers two significant benefits over building custom kernels. First, the publicly accessibility of TFLite is likely to stimulate further adoption of on-device Stable Diffusion models in real-world applications. Moreover, the versatility of TFLite facilitates the rapid deployment of various diffusion models on different mobile devices using the same optimization techniques. In this section, we introduce several technical challenges we encountered while deploying the Stable Diffusion model using TFLite on a mobile GPU and propose solutions for them.

3.1. Complete Mobile GPU Delegation

TFLite enables the use of the mobile GPU via a hardware driver called GPU delegate. It selectively runs supported operators in a computation graph on the GPU, leaving the unsupported operators to run on the CPU. However, such selective execution often leads to sub-optimal performance due to the expensive communication between CPU and GPU. Therefore, complete delegation is necessary for achieving optimal performance.

While the TFLite GPU delegate provides the acceleration for the most operators involved in Stable Diffusion, it fails to delegate even officially supported operators when the input activation size is large. To address the incomplete GPU delegation, we propose three methods involving modifications in the computation graph of the model.

Converting $FullyConnected$ to $Conv2D$

In spatial transformer blocks of the denoising U-Net network, there exist several fully-connected layers with large input activations (e.g., $1 \times 4096 \times 320$). Since the large fully-connected layers failed to be delegated, we convert them to equivalent convolution layers as shown in Fig. 1. Note that the depicted $FullyConnected$ layer and the $Reshape$-$Conv2D$-$Reshape$ layers result the same output and show almost the same latency when benchmarked on the GPU. Hence, converting all $FullyConnected$ operators into equivalent $Conv2D$ operators is preferable to prevent the GPU delegation failure.

Serializing $Conv2D$ with large activations

Although converting fully-connected layers to equivalent convolution layers enables delegation of layers with large input activations, we observed that one 3×3 convolution layer in the denoising network failed to be delegated with OpenCL backend due to its large input and output activation sizes: $1 \times 32 \times 32 \times 1920$ and $1 \times 32 \times 32 \times 640$, respectively.

Figure 2. Images generated with the same textual description and initial latent with 20 iterations. From left to right: baseline, after applying input serialization for Conv2D, numerically stable GELU approximation on Macbook M1 Pro.

Serializing the $Conv2D$ operator can solve this problem by reducing the activation sizes, but at the cost of multiple kernel call overhead. Therefore, the minimal serialization factor should be chosen to avoid excessive overhead.

The serialization can be applied along the input or output channel dimension as shown in Fig. 1. We find that the minimal serialization factor that enables complete delegation is 2 with the latency of 15.5 ms for the input dimension, and 8 with the latency of 40.9 ms for the output dimension by trying possible serialization factors in increasing order along each dimension. Thus, we chose the input serialization for its lower latency.

As the input serialization is a simple reordering of the computation sequence, the output should be very similar to that of the original graph. We qualitatively examined the generated images before and after applying the serialization. The difference between the images was subtle, as shown in the first two images in Fig. 2.

Broadcast-free Group Normalization

Group normalization is not represented as a single operator in the TFLite but as a computation graph consisting of basic operators such as $Mean$, $Square$, $Rsqrt$, and $BroadcastTo$. However, $BroadcastTo$ is not supported by the TFLite GPU delegate, which makes it necessary to modify the implementation of the group normalization layer.

We notice that the TFLite converter does not create an explicit $BroadcastTo$ operator when the activations are 4-dimensional or lower tensors. Hence, we reformat the group normalization layer so that the dimensions of the activation tensors are at most 4. Please refer to Fig. 7 in Appendix for the modified group normalization graph.

3.2. Numerically Stable Approximation of GELU

The images generated on different hardwares are noticeably different even if identical textual description and initial latent have been used as inputs (Fig. 3).

Stable Diffusion adopts float16 as the default data type for faster operations, which generally works well on server GPUs without causing any issues. However, it is important

Figure 3. The images generated by different hardwares with the same initial latent and textual description with 20 iterations. [left: Galaxy S23 Ultra, right: Apple M1 Pro]

to note that on certain mobile devices, the use of float16 can lead to floating-point exceptions. We identify that the numerical instability is caused by the approximated $GELU$ operator in its cubic polynomial term.

$$GELU(x) \approx 0.5x(1 + \tau(x))$$

$$\text{where } \tau(x) := tanh\left(\sqrt{\tfrac{2}{\pi}}(x + 0.044715x^3)\right)$$

Instead of this well-known approximation, we use the following more numerically stable approximation:

$$GELU(x) \approx 0.5x\big(1 + \tau\big(\gamma_M(x)\big)\big)$$

where

$$\gamma_M(x) := \begin{cases} x, \text{ if } |x| \leq M \\ M, \text{ otherwise} \end{cases}$$

is a clipping function. We use an empirical value $M = 10$, which suppresses the floating-point exceptions and maintains the image quality as shown in Fig. 2.

3.3. Pipelined Execution

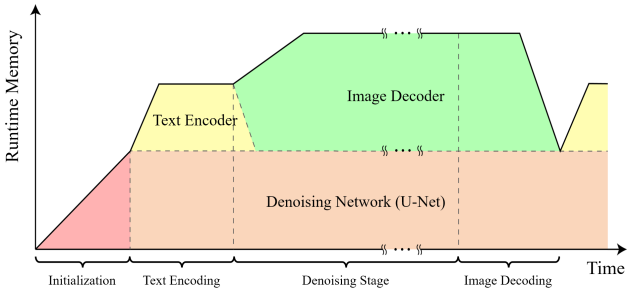

Figure 4. A qualitative illustration of the memory occupation of each component of Stable Diffusion during the pipelined execution. The orange (resp. yellow, green) area represents the memory occupied by the denoising network (resp. text encoder, image decoder).

Due to the limited memory available on the mobile devices, it is often not practical to load all three components of Stable Diffusion on the memory simultaneously.

We propose a pipelined execution strategy for devices with small processor memory. While the denoising network is

Table 1. Comparison with different Stable Diffusion on Mobile. (image resolution: 512 × 512)

	MODEL	LATENCY	DEVICE	HARDWARE	ENGINE
HOU & ASGHAR (2023)	SD V1.5	∼ 15S	(GALAXY S23)	HEXAGON PROC.	QUALCOMM AI ENGINE
CHEN ET AL. (2023)	SD V1.4	∼ 12S	GALAXY S23 ULTRA	MOBILE GPU	CUSTOM KERNEL
OURS	SD V2.1	∼ 7S	GALAXY S23	MOBILE GPU	TFLITE

retained on the memory throughout the entire execution, the text encoder and the image decoder are loaded interchangeably via a child thread running parallel with the main thread, as described in Fig. 4.

3.4. Model Compression

We apply quantization and pruning techniques to the pretrained model to reduce the overall memory consumption. Since mobile GPU does not support integer matrix multiplications, float16 is applied for the activations. However, we quantize weights into 8-bit precision to reduce the model size; thus, weights are casted from 8-bit integers to 16-bit floating points before being involved in the computation. We further apply structured pruning on huge convolution layers to minimize memory requirements.

Since it is not straightforward to measure the performance degradation caused by the quantization and pruning quantitatively, we used block-wise reconstruction error (Li et al., 2021; Wei et al., 2022) as an indirect metric and the quality of generated images as a qualitative measure. Fig. 5 shows the output images of the baseline, quantized, and quantized and pruned model, respectively. Although each image shows differences in details, they are less prominent than in Fig. 3.

Figure 5. From left to right: baseline, after applying 8-bit weight quantization, and pruning.

4. Experiment

In this work, we use Stable Diffusion v2.1 as a baseline model and optimize it for mobile deployment. We choose Samsung Galaxy S23 device to measure end-to-end benchmark latency. The device has Snapdragon 8 Gen 2 processor which includes Adreno 740 GPU. In addition to the quantization and pruning, we apply knowledge distillation to reduce the number of inference steps following Salimans & Ho (2022) and Meng et al. (2023).

Table 1 shows the end-to-end latency of our model and the comparison with previous approaches to deploy Stable Diffusion on mobile. For a fair comparison with previous works, we measure end-to-end latency for text encoding, 20 effective denoising steps and image decoding. The proposed approach can successfully generate a 512x512 image from a given text prompt within 7 seconds as shown in Fig. 6. In addition, while previous approaches use dedicated or custom engine to deploy Stable Diffusion on mobile, our approach enables using off-the-shelf TFLite engine without any custom modification.

Figure 6. Example images generated by our method on a mobile device.

5. Conclusion

In this paper, we have discussed a series of optimization techniques that, in combination, enable the fastest on-device image synthesis using the Stable Diffusion. These solutions can be extended to the deployment of other diffusion models, thereby facilitating the implementation of these models on various mobile devices, while leveraging the computation capability of TFLite. We believe that the optimized deployment to a common and accessible inference framework will enrich the ecosystem of real-world mobile applications built upon diffusion models.

References

Avrahami, O., Lischinski, D., and Fried, O. Blended diffusion for text-driven editing of natural images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18208–18218, 2022.

Blattmann, A., Rombach, R., Ling, H., Dockhorn, T., Kim, S. W., Fidler, S., and Kreis, K. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22563–22575, June 2023.

Chen, Y.-H., Sarokin, R., Lee, J., Tang, J., Chang, C.-L., Kulik, A., and Grundmann, M. Speed is all you need: On-device acceleration of large diffusion models via gpu-aware optimizations. *arXiv preprint arXiv:2304.11267*, 2023.

Gao, S., Liu, X., Zeng, B., Xu, S., Li, Y., Luo, X., Liu, J., Zhen, X., and 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.

Google. Tensorflow lite: Machine learning for mobile and edge devices. https://www.tensorflow.org/lite, 2017.

Ho, J. and Salimans, T. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models, 2020.

Hou, J. and Asghar, Z. World's first on-device demonstration of stable diffusion on an android phone. 2023.

Li, H., Yang, Y., Chang, M., Chen, S., Feng, H., Xu, Z., Li, Q., and Chen, Y. Srdiff: Single image super-resolution with diffusion probabilistic models. *Neurocomputing*, 479:47–59, 2022.

Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., Yu, F., Wang, W., and Gu, S. Brecq: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2021.

Liu, H., Chen, Z., Yuan, Y., Mei, X., Liu, X., Mandic, D., Wang, W., and Plumbley, M. D. Audioldm: Text-to-audio generation with latent diffusion models. *arXiv preprint arXiv:2301.12503*, 2023.

Lugmayr, A., Danelljan, M., Romero, A., Yu, F., Timofte, R., and Van Gool, L. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11461–11471, 2022.

Luo, Z., Chen, D., Zhang, Y., Huang, Y., Wang, L., Shen, Y., Zhao, D., Zhou, J., and Tan, T. Videofusion: Decomposed diffusion models for high-quality video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10209–10218, June 2023.

Meng, C., Rombach, R., Gao, R., Kingma, D., Ermon, S., Ho, J., and 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.

Nichol, A. Q., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., Mcgrew, B., Sutskever, I., and Chen, M. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. In *International Conference on Machine Learning*, pp. 16784–16804. PMLR, 2022.

Orhon, A., Siracusa, M., and Wadhwa, A. Stable diffusion with core ml on apple silicon, 2022.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Sahak, H., Watson, D., Saharia, C., and Fleet, D. Denoising diffusion probabilistic models for robust image super-resolution in the wild. *arXiv preprint arXiv:2302.07864*, 2023.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35: 36479–36494, 2022.

Salimans, T. and Ho, J. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. *International Conference on Learning Representations ICLR*, 2021.

Wei, X., Gong, R., Li, Y., Liu, X., and Yu, F. Qdrop: Randomly dropping quantization for extremely low-bit post-training quantization. In *International Conference on Learning Representations*, 2022.

Yang, D., Yu, J., Wang, H., Wang, W., Weng, C., Zou, Y., and Yu, D. Diffsound: Discrete diffusion model for text-to-sound generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.

A. Visualization of computational graphs

We provide visualization of computational graphs proposed in the main text. Fig. 7 shows the computational graph of original group normalization layer in TFLite format and that of the reimplemented group normalization layer. All of the $BroadcastTo$ operations and 5-dimension activations are removed in the reimplemented version.

In Fig. 8 , the computational graph of the modified version of GELU is depicted. Note that the additional operations ($Minimum$ and $Maximum$) are added in the beginning of the graph.

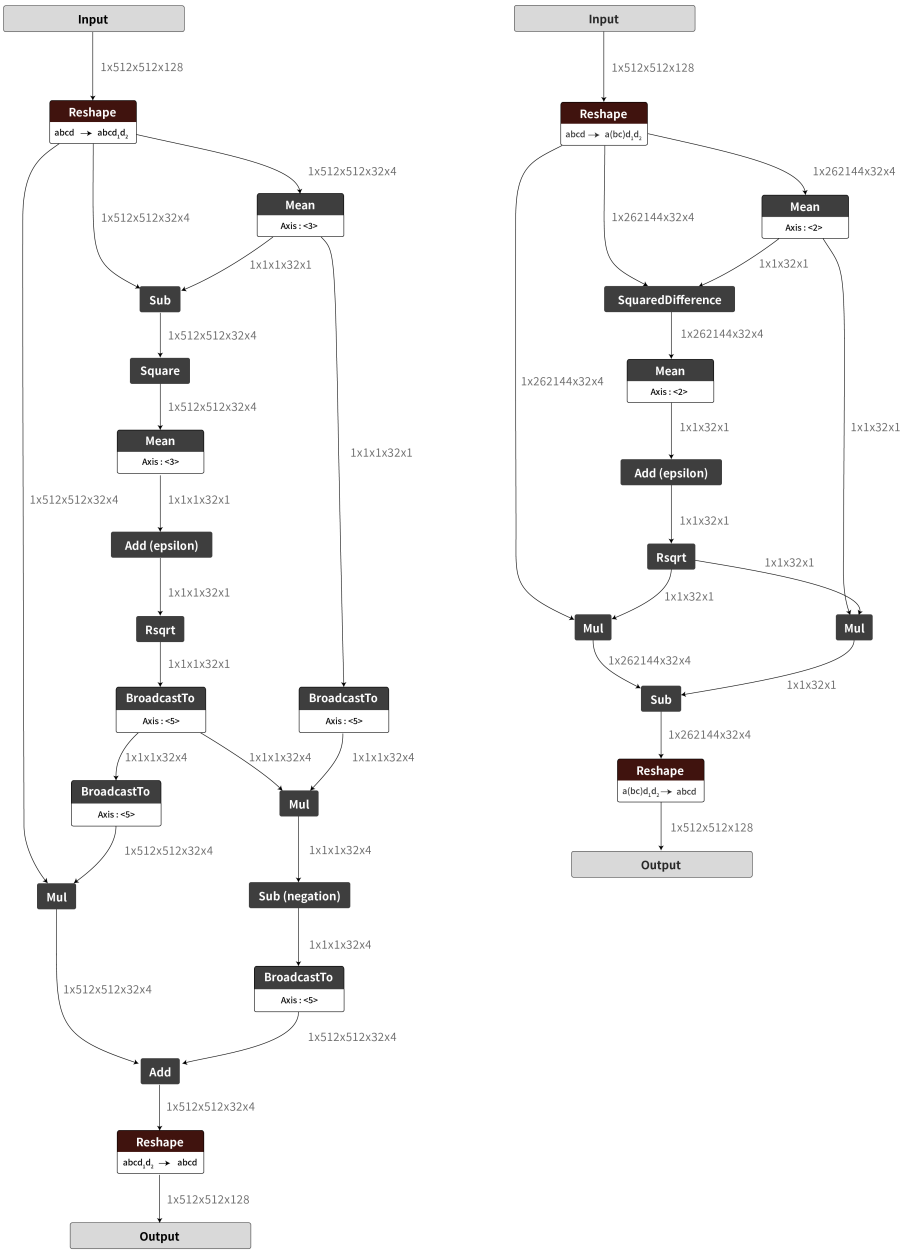

Figure 7. Left: the original group normalization; Right: reimplemented group normalization without any $BroadcastTo$ operator

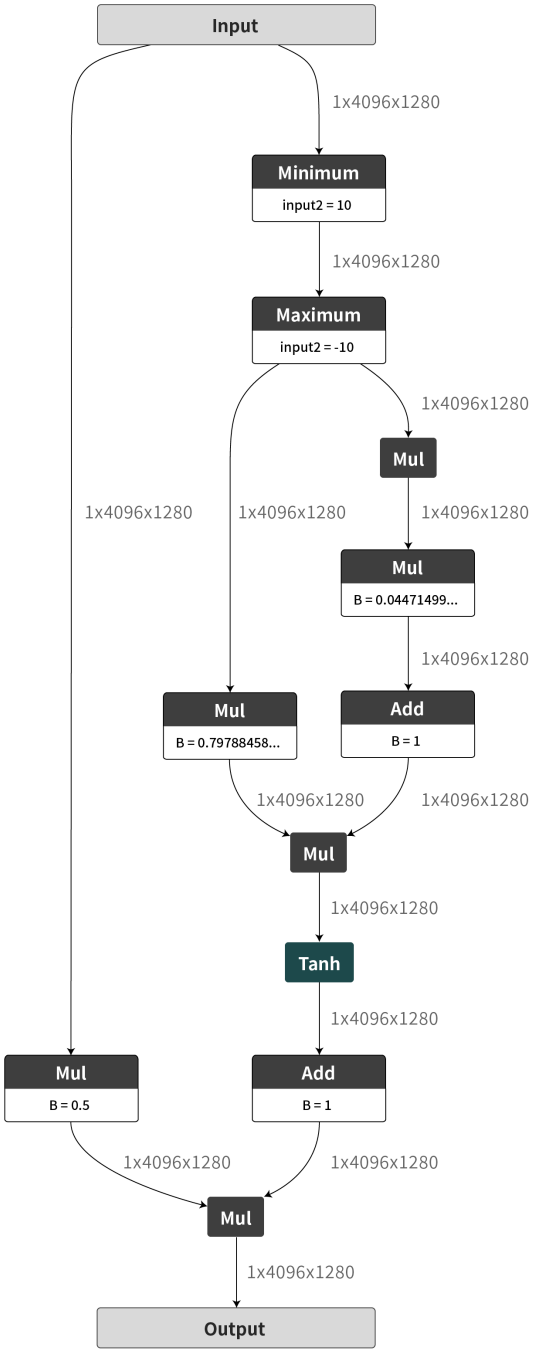

Figure 8. The numerically stable approximation of GELU