SDXL-SIGE: Efficient Image Editing

Elizabeth Rabenold, Jordan Lam, Luke Wagner, Raul Hernandez
{rabenold, jordlam, wagnerl, rhern}@mit.edu
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

During image editing, existing generative models tend to reconstruct entire outputs, including unedited regions. As much of an image is left untouched when an edit is made, these generative models are often unnecessarily computationally intensive. Our methods build on the recent strides of Sparse Incremental Generative Engine (SIGE) to produce efficient, high-quality editing results via off-the-shelf hardware. We propose SDXL-SIGE, an integration of SIGE with Stable Diffusion XL (SDXL), a state of the art diffusion model architecture. By replacing SIGE's stable diffusion model with SDXL, SDXL-SIGE mitigates redundant computation in the diffusion process and maintains image quality. We also extend the codebase to integrate SDXL's companion models: the SDXL Refiner and SDXL-Turbo. SDXL-SIGE is able to successfully incorporate these additional models and achieve similar results to the base SDXL.

1 Motivation

At the frontier of machine learning models, metrics such as memory usage, model size, and model latency are being tackled to further increase model scale and deployability. In the realm of generative AI for images, the use of different generative models is often computationally intensive. Specifically, when editing an image, generative models perform operations over the entire image for each edit. This leads to intense computations which perform a lot of redundant work, since in general, much of an image is left untouched when an edit is made. Furthermore, many image editing applications consist of edit operations that are frequently applied within a short frame of time. With such operations being applied at a high recurrence, total editing time for the aforementioned generative models grows at an astonishing rate, thus deterring many users. The possibility of producing editing results in a short amount of time in both singular and consecutive edits appeals to the goal of real-time rendering.

2 Related Work

2.1 Diffusion Models

Diffusion models are a foundational technique in image synthesis. The main idea behind diffusion is to "denoise" the input, which is a random chunk of data, to reveal the underlying image, often under guidance of additional conditional inputs such as text prompts [1]. A few important diffusion models are discussed below.

Stable Diffusion (SD) [5] models first use encoders to translate the image into a latent space and then proceed to work through the diffusion process on the latent blocks. SD inserts conditions into the

latent space by encoding the conditional blocks and concatenating the conditional latent blocks with the image latent blocks. During the diffusion process, SD relies heavily on cross-attention layers within the UNet architecture to diffuse the noise. SD iteratively denoises the latent blocks, and then uses a decoder to translate the image back into pixel space, successfully generating the final image.

SDXL improves on the SD backbone with three simple, yet effective, extensions [4]. SXDL utilizes a three-times larger UNet, two additional conditioning inputs, and an additional diffusion model, known as the refiner, to further enhance the visual quality of the generated images. The refiner, which uses a technique known as SDEdit [3], does not have to be used in image synthesis, but it has been found to be successful in improving the generated image quality.

SDXL Turbo utilizes Adversarial Diffusion Distillation (ADD) [6] to decrease the number of iterations needed in the diffusion process to one. ADD combines the power of diffusion for image synthesis tasks with the inherent performance of Generative Adversarial Networks (GANs). With the combination of these two techniques, SDXL Turbo achieves vastly superior performance to SDXL while maintaining similar results.

2.2 Sparse Incremental Generative Engine (SIGE)

One approach to minimize the redundancies in diffusion computation is SIGE [2]. In previous diffusion models, computations done on unedited regions contributed to additional computational complexity. Accounting for this, SIGE takes advantage of spatial sparsity when updating an edited image, using the fact that images are often edited incrementally. SIGE works by caching the results from the original image, identifying "active" blocks in which users made edits, and only processing information in the active regions. SIGE spatially processes both convolution and attention layers to achieve better performance. In the original implementation of SIGE, SD 1.4 was used as the underlying generative model.

3 Methods

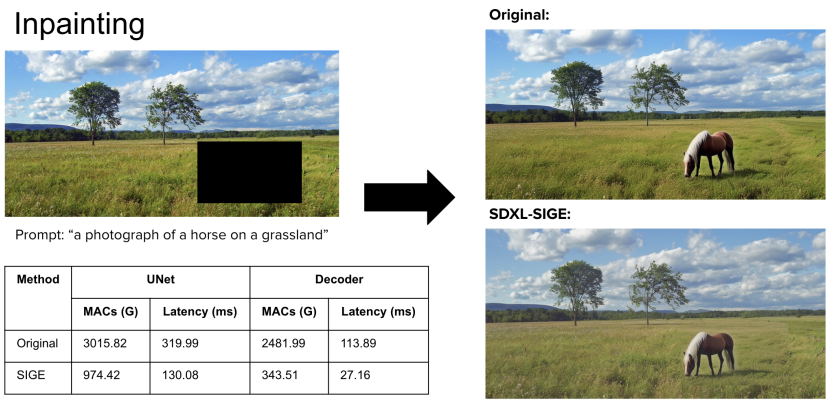
We began by replacing SIGE's standard SD 1.4 model with the implementation of SDXL. In this replacement, we made sure to maintain compatibility between the SDXL model and SIGE's codebase. This also entailed having to replace the corresponding SDXL module with SIGE's wrapped module instead. Once SDXL was fully incorporated into the SIGE pipeline, we used the same testing examples used to benchmark SIGE's standard performance to now benchmark the performance of the SDXL-SIGE pipeline. Additionally, we expanded the codebase to support both the SDXL Refiner and SDXL-Turbo, benchmarking each model on the SIGE examples. Each of these components is discussed further below.

3.1 Integration with SDXL

We first began by exploring SDXL's codebase in Stability-AI's generative model repository [7]. After gaining an understanding of the code and the full model pipeline, we were able to fork the corresponding modules into our own codebase. After transferring the necessary modules, we worked to wrap each component of the model with the necessary function calls to utilize SIGE. As a default, SDXL used linear layers in its spatial transformer blocks. Our initial deployment utilized the linear layers, but, once the integration was complete and our quality checks were passing, the performance gains were minimal as a majority of computation exists in these blocks. Thus, we adapted the transformer modules to be compatible with SIGE and updated the SDXL configuration to utilize convolution in the transformers, leading to the performance gains discussed in section 4.

3.2 Expansion to SDXL Refiner and SDXL-Turbo

After SDXL integration was complete, we then turned towards integrating both the SDXL Refiner and SDXL-Turbo modules. We started by incorporating SDXL-Turbo into SDXL-SIGE's codebase



Method	UNet		Decoder	
	MACs (G)	Latency (ms)	MACs (G)	Latency (ms)
Original	3015.82	319.99	2481.99	113.89
SIGE	974.42	130.08	343.51	27.16

Figure 1: A demonstration of the inpainting task both for the baseline system and our SDXL-SIGE system. For our system, the quality of the inpainting output stays consistent, though with a significant improvement to performance (with respect to MACs and latency) across the board.

since SDXL-Turbo utilizes the underlying SDXL architecture. After updating the diffusion sampling process to mimic SDXL-Turbo's sampling steps, we were able to instantly gain compatibility with SIGE and observed similar results to the base model. Finally, we integrated the SDXL Refiner to SDXL-SIGE by importing the necessary configurations and expanding the SDXL-SIGE pipeline to include the extra steps required to run the refiner. Once the pipeline was complete, we were once again able to see the same performance gains as before with no image quality degradation. Both of these results are further highlighted in section 4.

4 Results

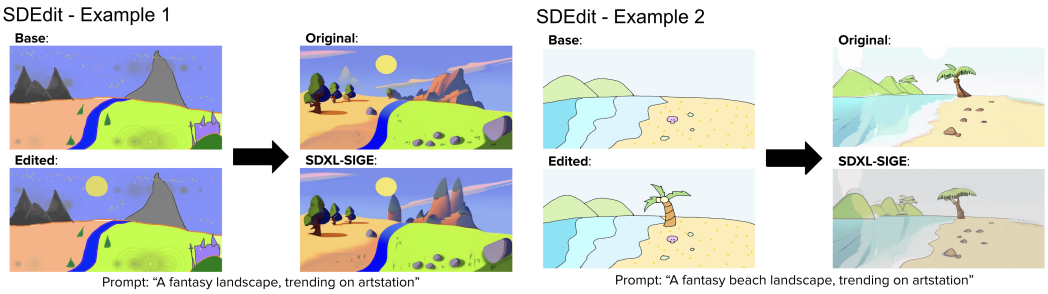
As anticipated, SDXL-SIGE resulted in significant speedups over SDXL while maintaining similar visual appeal. The outputs and results are discussed below.

4.1 Inpainting

Inpainting is a task where, given an image, a mask, and a prompt, the diffusion model fills in the masked area according to the surrounding image and the prompt. This benchmark, illustrated in Figure 1, included an image of a field, a mask in the middle of the field, and the following prompt: *a photograph of a horse on a grassland*. We compared the visual similarity and measured the number of MACs in a single pass through the denoiser model (UNet) and the decoder. As shown in Figure 1, we found no significant differences between SDXL's and SDXL-SIGE's visual outputs. However, we observed a $3.09\times$ decrease in MACs and a $2.46\times$ reduction in latency for the UNet model and a $7.23\times$ decrease in MACs and a $4.19\times$ reduction in latency for the decoder. Thus, SDXL-SIGE is able to vastly improve the performance of SDXL in inpainting tasks while producing visually similar outputs.

4.2 SDEdit

SDEdit is a task where the diffusion model generates an image using an initial inputted image and prompt as guidance. As SIGE works on incrementally edited images, this benchmark, illustrated in Figure 2, included two examples of drawn images with varying edit ratios between the original and edited versions (1.72% in example 1 and 2.78% in example 2). We compared the visual similarity and measured the number of MACs in a single pass through the denoiser model (UNet), the encoder, and the decoder. As shown in Figure 2, we found no significant differences between SDXL's and SDXL-SIGE's visual outputs for both examples. However, for example 1, we observed a $5.09\times$ decrease in MACs and a $2.86\times$ reduction in latency for the UNet model, a $27.65\times$ decrease in MACs and a $10.13\times$ reduction in latency for the encoder, and a $16.09\times$ decrease in MACs and a



Method	Example	UNet		Encoder		Decoder	
		MACs (G)	Latency (ms)	MACs (G)	Latency (ms)	MACs (G)	Latency (ms)
Original	--	3015.82	320.77	1083.27	62.53	2481.99	114.06
SIGE	1	592.03	112.15	39.18	6.17	154.22	19.09
	2	876.87	125.16	76.29	8.32	318.28	26.13

Figure 2: Demonstrations of the SDEdit task both for the baseline system and our SDXL-SIGE system. For our system, the quality of the SDEdit output stays consistent compared to the baseline, though with a significant improvement to performance (with respect to MACs and latency) across the board.

$5.97\times$ reduction in latency for the decoder. We also observed similar improvements in example 2. Once again, SDXL-SIGE is able to vastly improve the performance of SDXL in SDEdit tasks while maintaining the visual quality of the outputs.

4.3 SDXL Refiner

SDXL boasts an additional stage to its pipeline: the refiner. The refiner is another diffusion model used for tuning the output of SDXL to improve its visual appeal. Utilizing SIGE, we were able to observe similar speedups to the refiner stage as with the base model. Specifically, in the denoiser stage of the refiner (UNet) we measured a $3.10\times$ decrease in MACs and a $2.58\times$ reduction in latency for the inpainting task and a $5.87\times$ decrease in MACs and a $3.46\times$ reduction in latency for the first example of the SDEdit task. We achieved similar improvements in the second example for SDEdit. As the encoder and decoder are not affected by the refiner, these stages were not re-profiled. As shown in Figure 3, we were once again able to achieve visually similar results despite the increase in performance.

4.4 SDXL-Turbo

SDXL-Turbo is a fine tuned version of SDXL, resulting in fewer rounds of diffusion. Utilizing SIGE, we were able to observe similar speedups for SDXL-Turbo as we did with the base SDXL. Specifically, in the denoiser stage of the refiner (UNet) we measured a $3.09\times$ decrease in MACs and a $2.47\times$ reduction in latency for the inpainting task and a $5.09\times$ decrease in MACs and a $2.80\times$ reduction in latency for the first example of the SDEdit task. We achieved similar improvements in the second example for SDEdit. As the encoder and decoder are not affected by the refiner, these stages were not re-profiled. Despite the performance gains, we were once again able to achieve visually similar outputs as seen in Figure 4.

5 Future Work

We believe SDXL-SIGE should be incorporated into image editing applications to fully realize its performance potential. In addition to drawing on an image to introduce edits, we see image rotation and scene panning as potential areas of integration with SDXL-SIGE. This application is currently under development.

Refiner:

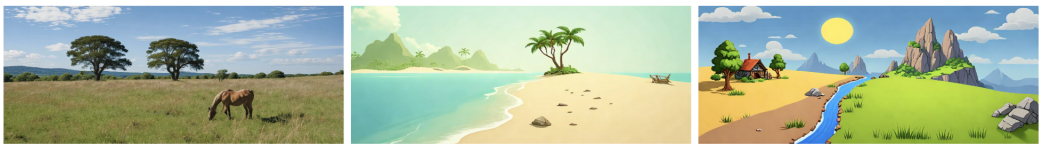
Refiner-SIGE:

Method	UNet	
	MACs (G)	Latency (ms)
Original	3058.80	304.85
SIGE	985.25	118.37

Method	Example	UNet	
		MACs (G)	Latency (ms)
Original	--	3058.80	304.44
SIGE	0	521.47	88.04
	1	777.37	108.01

Figure 3: Demonstrations of the Inpainting and SDEdit task both for the baseline refiner and our SIGE-enabled refiner. The quality of the images remain the same, but we see significant performance gains.

Turbo:



Turbo-SIGE:

Method	UNet	
	MACs (G)	Latency (ms)
Original	3015.82	321.03
SIGE	974.42	130.03

Method	Example	UNet	
		MACs (G)	Latency (ms)
Original	--	3015.82	320.21
SIGE	0	592.03	114.51
	1	876.87	124.76

Figure 4: Demonstrations of the Inpainting and SDEdit task both for the baseline SDXL-Turbo and our SIGE-enabled SDXL-Turbo. The quality of the images remain the same, but we see significant performance gains.

5.1 Image Rotation

In standard image editing suites, an image is rotated against a backdrop or workspace that remains fixed. Empty spaces result from rotating the image without simultaneously rotating the background. Diffusion can be applied to these blank spaces (i.e. image inpainting); thus, SDXL-SIGE can generatively fill in the frame's empty spaces. By observation, significant portions of an image's information remain the same between states of rotations. Thus, SDXL-SIGE can leverage sparsity to achieve efficient editing results across consecutive rotations and superior editing quality.

5.2 Scene Panning

Panning an image will involve creating new canvas space in the direction of the pan, followed by a translation of the image in the opposite direction. In a similar vein as image rotation, redundant information remains across image states, and thus SDXL-SIGE can be utilized to apply diffusion to fill in the blank canvas space, thereby expanding the scene in a new direction.

6 Conclusion

Overall, SDXL-SIGE was able to vastly increase the performance of SDXL on inpainting and SDEdit tasks. Furthermore, our expansions into SDXL's companion models imply SDXL-SIGE's ability to be adaptable to new image generation models released by Stability-AI. Thus, SDXL-SIGE should have lasting impact as newer, better models are released.

7 Contributions

Every team member contributed to each part of the project. Luke and Jordan took the lead on integrating SIGE into the SDXL pipeline and running the benchmarking experiments. Elizabeth and Raul led the expansion and incorporation of the SDXL Refiner and SDXL-Turbo. A lot of work was completed via pair programming and discussions as a group.

8 Codebase

All of our code is available at `https://github.com/lwag212/sdxl-sige/`. Our contributions are contained within the `sdxl` folder.

Acknowledgments

We would like to thank the MIT HAN Lab for their support in this work.

References

[1] *Han Song*. Lecture notes in TinyML and Efficient Deep Learning Computing. November 2024.

[2] *Li Muyang, Lin Ji, Meng Chenlin, Ermon Stefano, Han Song, Zhu Jun-Yan*. Efficient Spatially Sparse Inference for Conditional GANs and Diffusion Models. 2023.

[3] *Meng Chenlin, He Yutong, Song Yang, Song Jiaming, Wu Jiajun, Zhu Jun-Yan, Ermon Stefano*. SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations. 2022.

[4] *Podell Dustin, English Zion, Lacey Kyle, Blattmann Andreas, Dockhorn Tim, Müller Jonas, Penna Joe, Rombach Robin*. SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis. 2023.

[5] *Rombach Robin, Blattmann Andreas, Lorenz Dominik, Esser Patrick, Ommer Björn*. High-Resolution Image Synthesis with Latent Diffusion Models. 2021.

[6] *Sauer Axel, Lorenz Dominik, Blattmann Andreas, Rombach Robin*. Adversarial Diffusion Distillation. 2023.

[7] *Stability-AI* . generative-models. 2024.