Replication of Token Merging for Fast Stable Diffusion

Henri Balster

A thesis presented for the bachelor's
degree of Computer Science

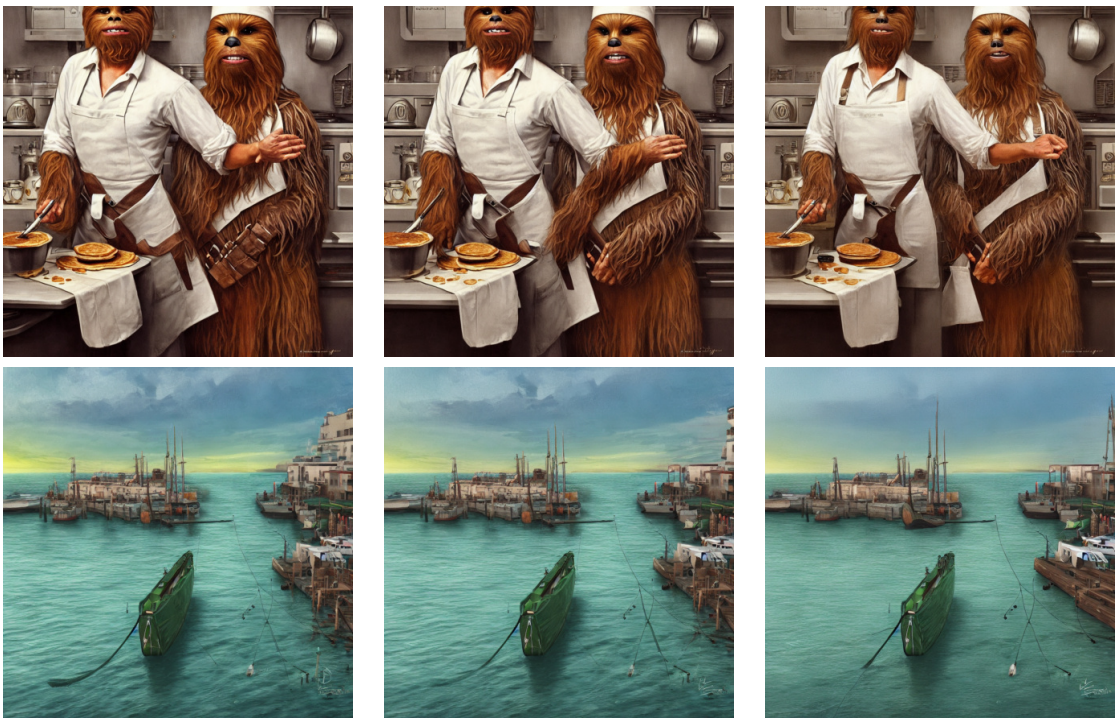
Chair for Dialog Systems and Machine Learning
Dr. Konrad Völkel
Heinrich-Heine-University Düsseldorf

September 2023

Contents

Abstract

Image generation models, such as Stable Diffusion, are popular tools for creating any desired image from a line of text. The inclusion of new transformer-based technologies into these models has improved the quality of their images, though their operating time can be very slow. [Bolya and Hoffman, 2023] have introduced Token Merging (ToMe) for Stable Diffusion to improve computational time by reducing the number of tokens evaluated by the model. The goal of this thesis was to replicate their results and investigate whether performance can be improved by using different configurations for ToMe. Our results show that ToMe's default configuration maintains great image quality (most of the time), while accelerating image generation by up to $2.2\times$, when reducing the number of tokens by up to 60%. Beyond that, we identified different configurations for ToMe that improve the performance compared to the default setup, by using different partitioning methods and expanding the scope of token merging in the transformer. This work further solidifies the viability of token merging in diffusion models and incentivises further research into improving the efficiency of transformer based models. Code and full-scale images are available at https://github.com/HNR1/ba-code



$r = 0\%$ 20% 50%

Table 1: 768×768 images created with the default configuration of ToMe

1 Introduction

Diffusion models are part of the the latest advancements in computer vision and are used for tasks involving image enhancement, completion, denoising, restoration and most notably image synthesis. Popular generative models include DALL-E 2 [Ramesh et al., 2022], Midjourney and **Stable Diffusion** (SD) [Rombach et al., 2021], with the latter offering free access to this kind of image generation software.

A big challenge for these generative models remains the computational intensity when image sizes increase. The computational demands square with the number of pixels (and tokens), due to the models reliance on a transformer backbone.

A promising remedy for this issue is the concept of Token Merging (ToMe) by [Bolya and Hoffman, 2023]. ToMe for SD exploits redundancies in the input by merging similar tokens to reduce the number of tokens processed by the transformer. Unlike token pruning methods, ToMe retains the original image size by unmerging the tokens after they were processed by a computational unit. Additionally, ToMe is model agnostic and requires no specific training. The authors claim: "ToMe for Stable Diffusion minimally impacts visual quality while offering up to $2\times$ faster evaluation using $5.6\times$ less memory" [Bolya and Hoffman, 2023]. ToMe for SD is an continuation of the original ToMe [Bolya et al., 2023] and specifically tailored as an extension for SD. It can be used to accelerate both training and inference.

In this work, our goal is to replicate parts of their findings, while also investigating different configurations of ToMe and their effect on SD's performance in image generation tasks. Our setup involves creating datasets of images with and without ToMe and calculating their Fréchet-Inception-Distance (FID) to assess the images similarity while also measuring the algorithms speed.

The main contents of this thesis are presented across four sections. Section 2 will provide explanations for important concepts related to our studies. An short summary of alternative approaches to solving diffusion model's inefficiency problem is presented in Section 3. Section 4 holds an in-depth overview of the functionality of ToMe, explicitly laying out the merge and unmerge process. Finally, Section 5 explains the whole experimental process this thesis is built upon, describing in detail the setup and the results of our trials.

We will show that ToMe is indeed able to cut image generation time in half while preserving most of the information, thus producing images very close to the original. Moreover, we will show how altering the configuration of ToMe can further improve image quality, as well as speed (albeit marginally). We hope that our results can validate the legitimacy of ToMe and motivate further research.

2 Background

2.1 Stable Diffusion

Stable Diffusion is based on a special type of Diffusion Model called Latent Diffusion Model (LDM) [Rombach et al., 2022]. LDMs are trained to create a desired image by repeatedly denoising an image initialised with random gaussian noise; this process is called reverse diffusion. Forward diffusion, which applies more and more noise at each step, is used during training, so the model learns to identify noise in an image. What distinguishes LDMs from conventional Diffusion Models is that inference takes place within a lower-dimensional latent space, instead of the pixel-space. This greatly alleviates the need for extensive computational resources, especially when generating larger images. LDMs have three main components: 1) the autoencoder, 2) the U-Net and 3) the text-encoder.

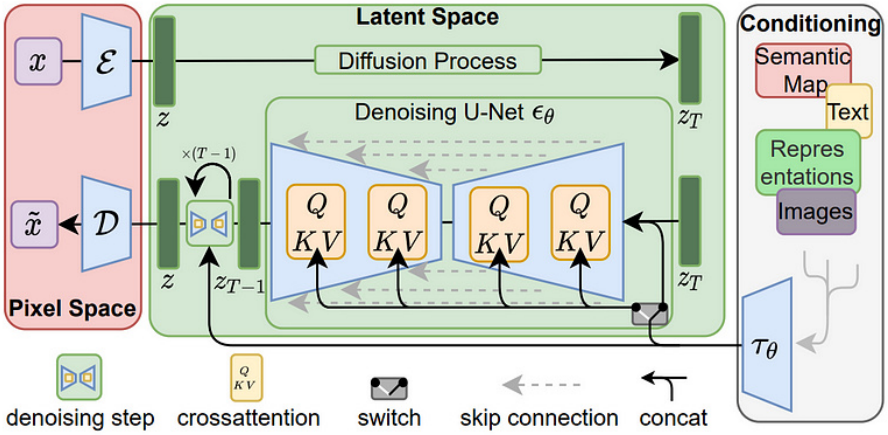


Figure 1: The LDM [Rombach et al., 2022, Fig. 3].

2.1.1 The Autoencoder

A Variational Autoencoder (VAE) consists of two parts: the encoder and the decoder. "During latent diffusion training, the encoder is used to get the latent representations (latents) of the images for the forward diffusion process" [Patil et al., 2022]. The decoder conversely transforms the denoised latents, created by the U-Net, back into the pixel space at the end of the reverse diffusion process. Only the decoder is used during image generation (see Fig. 1).

2.1.2 The U-Net

The U-Net [Ronneberger et al., 2015] is a type of Convolutional Neural Network (CNN) that is responsible for predicting the noise in the current sample image during

inference. The image generation process starts with a sample image of random gaussian noise. The U-Net then takes this sample $(latents)$ and predicts the noise residual $(noise\_pred)$ of the image. This noise residual is partially $(sigma)$ removed in every step, until it ends up with the final fully denoised image after n steps (see Fig. 1).

```
latens = latents - sigma * noise_pred
```

The U-Net consists of an encoder and a decoder with transformer-based blocks. "Thus, it first encodes the current noised image as a set of tokens, then passes it through a series of transformer blocks" [Bolya and Hoffman, 2023]. The inclusion of transformer-based blocks allows the model to not only preserve the spacial hierarchies but also the semantic structure of the image.
A further aspect that distinguishes the U-Net from an autoencoder are the skip connections that directly pass information from the encoder to the decoder, aiding the preservation of details during upsampling (see Fig. 1).

2.1.3 The Text-Encoder

The text-encoder is a transformer-based encoder that converts the input prompt from a string to an embedding vector that captures the semantic meaning of text and can be interpreted by the U-Net. Stable Diffusion uses the pre-trained text-encoder of the CLIPTokenizer [Radford et al., 2021] and thus avoids additional training of the text-encoder.

2.1.4 The Transformer Block

Every transformer block has a self-attention (self-attn), cross-attention (cross-attn) and multi-layer perceptron (mlp) module. The transformer also has residual connections around these modules to preserve important information and improve gradient flow, as well as multiple layer normalization blocks.

Self-Attention

Self-attention [Vaswani et al., 2017] takes a series of (token) vectors and computes outputs while attending to every other token in the sequence.
The input vectors (X) are firstly transformed into three different embedding matrices called Queries (Q), Keys (K) and Values (V), by multiplying the inputs with special transformation matrices (W_Q, W_K, W_V). Then the self-attention module computes the attention weights (A) for every pair of input vectors by taking the scaled dot-product of the Query and Key matrix and then applying the softmax-function. Lastly, the output vectors (Y) are generated by taking a weighted sum of the Value embeddings

with the attention weights (see Fig. 2).
The output vectors are transformed representations of the input vectors, considering their interactions with every other input vector. The outputs are then passed on to the subsequent layers of the transformer. Self-attention enables the transformer to capture global dependencies in images or text.

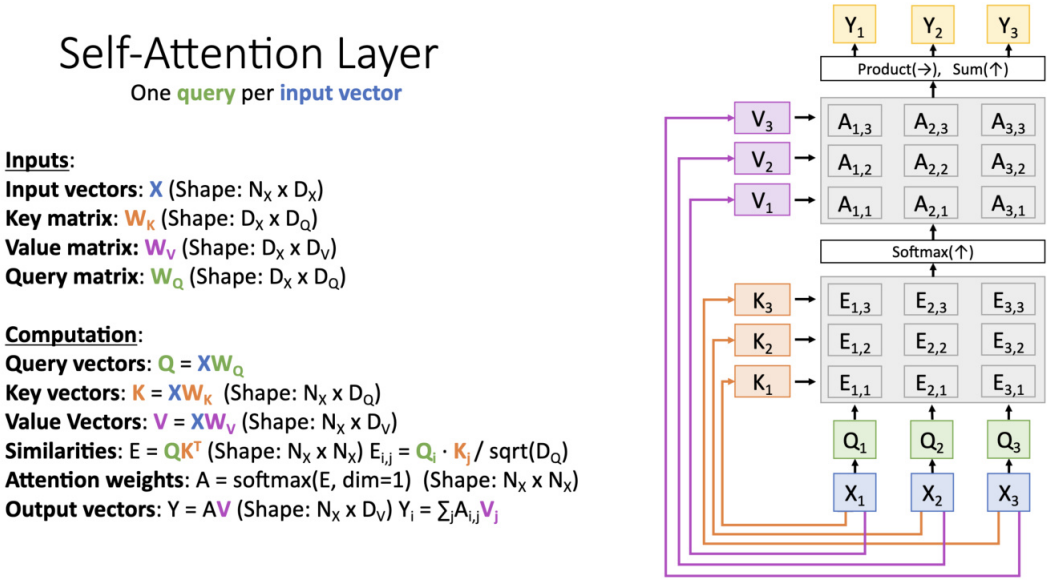


Figure 2: Self-attention [Johnson, 2019]

Cross-Attention

Cross-attention creates its output from two input embeddings (X_1, X_2) by generating Q from one input $(Q = X_1 W_Q)$, and K and V from the other $(K = X_2 W_K,\ V = X_2 W_V)$. The algorithm is identical to self-attention from this point onwards.
Cross-Attention allows the model to consider both visual and semantic information when generating sequences, which enables the U-Net to attend to both the image and the prompt at the same time.

Multi-Layer Perceptron

A multi-layer perceptron is a fully connected neural network. The mlp layer takes the outputs of the previous layer and applies an mlp network to each vector independently. This network consists of two linear transformations with a non-linear activation function (e.g., GELU [Hendrycks and Gimpel, 2016]) in between. Non-linearity exists exclusively in the mlp layer, allowing the transformer to capture more complex patterns in the data. It is particularly effective at modeling short-range patterns and local dependencies within a sequence.

2.2 Fréchet-Inception-Distance

The Fréchet Inception Distance (FID) is a metric that is used to evaluate the quality of generated images. It provides a quantitative measure of similarity by calculating the distance between the distributions of the feature vectors of two image datasets. The feature vectors are extracted from the Inception model and the mean and covariance of their distributions are calculated.
"We call the Fréchet distance $d(.,.)$ between the Gaussian with mean (m, C) obtained from $p(.)$ and the Gaussian with mean (m_w, C_w) obtained from $p_w(.)$ the "Fréchet Inception Distance" (FID), which is given by:

$$d^2((m, C), (m_w, C_w)) = ||m - m_w||_2^2 + Tr(C + C_w - 2(CC_w)^{\frac{1}{2}})"$$

[Heusel et al., 2017].
A lower FID value indicates greater similarity between the two image sets.

2.2.1 Inception Model

The Inception model [Szegedy et al., 2015] is a deep convolutional neural network developed by Google researchers that is pretrained on ImageNet [Deng et al., 2009]. The model improves performance and efficiency in image classification and computer vision tasks by using multiple convolutional filter operations at different scales within the same layer.
The Inception model is used in the context of FID for creating low-dimensional latent representations of the input images, utilizing its ability to capture meaningful image features. Current code implementations such as [Seitzer, 2020] use the model's third iteration Inception-v3 [Szegedy et al., 2016].

2.2.2 Caveats

FID is statistically biased, with its bias growing inversely proportional to the size of the image set, and the bias also depending on the model used for feature extraction [Chong and Forsyth, 2020]. This makes comparing specific FID values from different experiments impossible, when identical sample sizes or the use of the same model are not guaranteed.
A further limitation of FID is the Inception model's compression of images to 299×299 pixels [Szegedy et al., 2016]. Thus, assessing the quality of larger images is additionally hindered as information about the images is lost through the compression.

2.3 What are Tokens?

Tokens are semantic units that can be processed by transformer-based models such as Stable Diffusion. Text, such as the prompt in image generation, is tokenized into words or subwords, which are then mapped to an embedding vector by the text-encoder. Images are divided into a grid of patches. An image token captures discrete representations of a specific image patch's visual features, such as objects, shapes, textures, or colors. Positional tokens are also added to maintain the order or spatial relationships between tokens. This breakdown into smaller units is necessary for efficient computations and memory management, additionally enabling these algorithms to better capture patterns and relationships within the data. Tokenizers such as CLIP map images and text into a shared embedding space, allowing for multi-modal processing of tokens by the model.

3 Related Work

3.1 Token Pruning

Several works have tried to utilize the transformer's flexibility when it comes to handling inconsistent amounts of tokens by pruning tokens that are considered the least important [Meng et al., 2022, Yin et al., 2022]. Differing from ToMe, these methods require model-specific training. Additionally, they dynamically determine the number of pruned tokens, resulting in variances between images. The flexibility in token number may improve accuracy, but makes batching of different samples impossible and requires the use of masks for training. This eradicates, or at least limits, the speedup that can be gained during training.
ToMe does not suffer from these problems as the number of tokens always stays the same outside of transformer components and generally no training is required. Still, ToMe achieves speedups in both image generation and training of models [Bolya and Hoffman, 2023].

3.2 Combining Tokens

Other works try to combine tokens instead of pruning them. There have been different approaches ranging from fusing less informative tokens into a single package token [Kong et al., 2021, Liang et al., 2022], over using a multi-layer perceptron with spatial attention for token reduction [Ryoo et al., 2021], to using a learned deformable token merging module for adaptive token merging between stages [Pan et al., 2022]. The most similar method to ToMe is Token Pooling by [Marin et al., 2021], though it

does again require specific finetuning. Token Pooling reduces the number of tokens to K by using k-means to cluster the tokens into K clusters. In this case the cluster centers represent the new tokens. The k-means-algorithm is slow, therefore Token Pooling offers a bad speed-accuracy trade-off.
ToMe is the only algorithm that offers reasonable speed and accuracy, without requiring any training [Bolya et al., 2023].

4 Token Merging

In Token Merging for Stable Diffusion by [Bolya and Hoffman, 2023], the number of tokens within a transformer component is reduced by $r\%$ through the merging of similar tokens prior to processing. This is crucially followed by unmerging the tokens to restore the original image size.
The tokens are partitioned into a source (**src**) and a destination (**dst**) set, then the most similar tokens from the **src** set are continuously merged into their **dst** counterparts until the number of tokens has reduced by $r\%$.
The choice of r is a trade-off between image fidelity and image generation time as a lower amount of tokens requires a smaller computing time but more information about the image is lost in the merge process.
The merge can be applied in different components of the transformer (i.e., self-attn, cross-attn, mlp) (see Fig. 3).

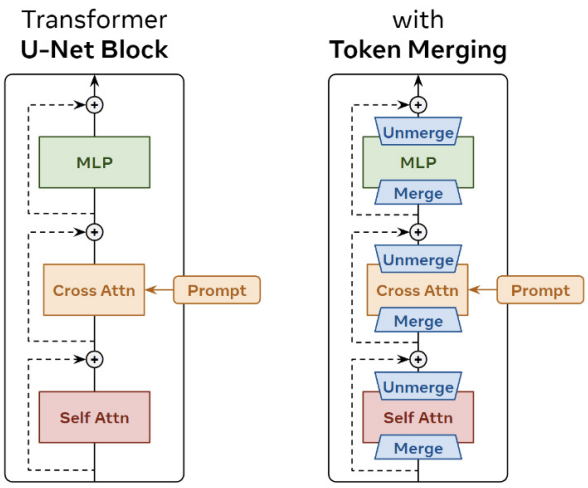


Figure 3: Transformer with ToMe [Bolya and Hoffman, 2023, Fig. 2].

4.1 Merge and Unmerge Algorithms

Merging.

The Merge algorithm uses Bipartite-Soft-Matching to determine the similarity of tokens between the **src** and **dst** set. The two most similar tokens are repeatedly merged into a new token until the overall number of tokens has reduced by $r\%$. Two tokens $x_1, x_2 \in \mathbb{R}^c$ would be merged into a new token $x^*_{1,2} \in \mathbb{R}^c$, by averaging their features:

$$x^*_{1,2} = \frac{x_1 + x_2}{2}$$

Unmerging.

The Unmerge algorithm takes an originally merged token $x^*_{1,2} \in \mathbb{R}^c$ and splits it up into its original tokens $x'_1, x'_2 \in \mathbb{R}^c$:

$$x'_1 = x^*_{1,2} \qquad x'_2 = x^*_{1,2}$$

in order to recreate the pre-merge amount of tokens. This naive approach does lose information, as the now unmerged tokens both have the average of their previous values. This loss though is often small (especially for smaller values of r), as token selection is based on similarity.

4.2 Bipartite-Soft-Matching

Bipartite-Soft-Matching involves two main steps. First, the tokens are split into two sets, holding the **src** and **dst** tokens respectively. Then, every token in the **src** set is matched with its most similar token in the **dst** set, creating a bipartite graph with edges between every **src** token and their closest match in the **dst** set.

Choosing src and dst

The grid of tokens is partitioned into a grid of $sx \times sy$ batches of tokens. Within every batch one token is chosen for the **dst** set (blue) with the rest now belonging to the **src** set (red). The choice is random by default, alternatively the top left token is always chosen when the parameter **use_rand=False** is set (see Fig. 4).
The number of mergeable tokens, which corresponds to the size of the **src** set, is limited by the choice of sx and sy.

$$r_{max} = 1 - \frac{1}{sx * sy}$$

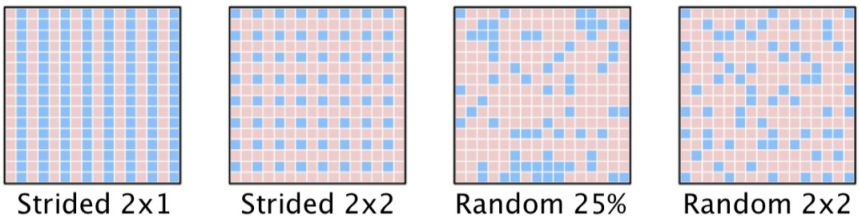


Figure 4: Partitioning of tokens into **src** and **dst** [Bolya et al., 2023, Fig. 5]

That means for $sx = 2$ and $sy = 2$, no more than 75% of tokens can be merged.
In the subsequent steps, the two most similar tokens are chosen greedily and merged into one, until $r\%$ of tokens are gone.

4.3 Token Similarity

It is also important to define what "token similarity" means in this context. Unlike the first iteration of ToMe which used key-based similarity [Bolya et al., 2023], ToMe for SD simply partitions the inputs of block x into to **src** and **dst** and then computes the cosine similarity between the two sets. The similarities are only computed once at the beginning of each transformer block [Bolya and Hoffman, 2023].

Computing cosine similarity

The token tensor (M) is normalized and then $split$ into two tensors (S and D), separating the **src** and **dst** tokens. Continuing from there, the cosine similarity ($scores$) between every **src** and **dst** token is computed by taking the dot product of S and D.

```
M = M / M.norm(dim=-1, keepdim=True)
S, D = split(M)
scores = S @ D.transpose(-1, -2)
```

The r most similar tokens of **src** and **dst** can now be identified by the indices of the largest values of the $scores$ tensor.

4.4 Additional Technical Information

ToMe, in its basic form, only merges tokens in the self-attn layer and defaults to using 2×2 batches for token partitioning, with a randomly chosen **dst** token for every batch, resulting in a 75% **src** and 25% **dst** split.
ToMe is also not applied in every U-Net block, but only the ones with the most tokens. Bolya and Hoffman argued: "We try restricting ToMe to only blocks with some minimum number of tokens and find that only the blocks with the most tokens need ToMe applied to get most of the speed-up" [Bolya and Hoffman, 2023].

Lastly, r remains consistent throughout the whole image generation process, as Bolya and Hoffman found that merging more tokens during earlier diffusion steps and fewer during later steps does not provide a significant improvement to performance.

Image inconsistencies

The usage of ToMe curiously prevents exact reproduction of the subsequent experiments, as minor randomness is involved in the image generation, even when **use_rand =False** is set. We were not able to determine why this randomness occurs. The variances are detectable by FID, but invisible to the human eye.

$r = 0\%$ 20% 50%

Table 2: 768×768 images created with the default configuration of ToMe

5 Experimental Procedures

We perform several experiments involving the creation of image datasets using the model Stable Diffusion v1.5 by runwayml [Rombach et al., 2022] and applying ToMe in different ways, then assessing the performance. The performance is defined by
1) speed: the average generation time for every image of a set and
2) image quality: the FID-value between sets of images that had token merging applied and their counterparts (that is same prompt, seed and image size) that didn't use any token merging.

5.1 Setup

Creating image datasets

The experiments always compare how image generation time and image quality change across a spectrum of different volumes of tokens (r) removed. The images were generated with a DiffusionPipeline from HuggingFace's diffusers library [von

Platen et al., 2022], using the Stable Diffusion v1.5 model.
We sampled multiple sets of 500 prompts from the library "Stable-Diffusion-Prompts" on HuggingFace which has 80,000 prompts filtered and extracted from the image finder for Stable Diffusion: Lexica.art, and generated a corresponding set of random seeds. This prompt library was chosen to closely simulate "real world" usage of SD. Prompts appearing multiple times within a dataset was not specifically prevented, due to the extremely low probability of a specific prompt-seed-pair appearing multiple times within a sample. Same prompts with different seeds result in different images and therefore do not corrupt the validity of a dataset by our assessment.
Multiple images were created per prompt, with a 0%, 10%, 20%, 30%, 40%, 50%, and 60% (if possible) merge applied respectively, creating a set of 3,500 (or sometimes 3,000) images which can be split up into subsets of 500 images each for every merge volume.

Measuring results

For image quality, the FID between a subset with $r > 0\%$ and the subset with $r = 0\%$ of the same dataset is computed to examine how ToMe performs across different values for r.
The speed of image generation is assessed by recording the time taken to create each individual image (in s/im) and then calculating the average across all subsets.

```
start = time.time()
image = pipeline(prompt, x, y, ...)
end = time.time()
time = end - start
```

Hardware

All experiments were conducted on a high performance computing cluster (HPC) provided by the Heinrich-Heine-Universität Düsseldorf. GeForce GTX 1080 Ti GPUs by Nvidia were used for creating the image datasets. Individual images were always generated on a single GPU.

5.2 Adjustments

FID

We use our own fork of pytorch-fid [Seitzer, 2020] for FID calculation, in order to accommodate for the HPC not being connected to the internet and therefore not being able to download the weights of the Inception model. Our version loads these

weights from a local directory to avoid any connection to the internet and requires the user to have them pre-installed. Some path variables also needed adjustment for usage on the HPC.

Prompts

We shortened every prompt that exceeds 300 characters, in order to ensure that CLIP's token limit is not breached, as the CLIPTokenizer can only handle up to 77 tokens. This is accomplished by determining the index (idx) of the last comma in the first 300 characters of every oversized prompt and then cutting off everything from this point onwards.

```
prompt = prompt[:idx]
```

5.3 Comparison to Original Setup

[Bolya and Hoffman, 2023] also used the Stable Diffusion v1.5 model to generate their images. They created 2,000 512×512 images (2 per ImageNet-1k class [Deng et al., 2009]) and then computed the FID values between their images and a class-balanced selection of 5,000 images from the ImageNet-1k dataset using pytorch-fid [Seitzer, 2020]. Their images were generated with 50 diffusion steps and a cfg scale of 7.5, matching our setup. Speed measurements were taken by averaging the diffusion time over all 2,000 samples. All their images were created on a single 4090 GPU.

The most notable difference to our setup is that we compare a set of merged images with their own unmerged versions, while Bolya and Hoffman's approach compares image sets of the same categories but does not directly try to measure alterations of images created by ToMe. Their GPU hardware also differs from ours, resulting in overall faster diffusion times in their experiments. The larger number of images in their trials also reduces the impact of FID's bias and leads to their results consistently showing lower FID values.
Additionally, Bolya and Hoffman measured and analyzed the memory usage during image generation, while we left memory consumption completely out of the scope of this work.

5.4 Results

The experiments can be split up into three different parts inspecting how ToMe affects performance

1) when applied to different parts of the transformer,
2) when applied to smaller images, and
3) when different settings for partitioning the **src** and **dst** sets are used.
The results of every experimental unit are presented in two figures. The first figure focuses on image quality, displaying FID values, while the subsequent one emphasizes image generation speed, by showcasing how the average image generation time of each subset compares to the $r = 0\%$ subset. Both figures have $r\%$ on the x-axis showing how the metrics change with an increasing amount of tokens merged. Datasets that are compared in an experimental unit, were always created with the same set of prompts and seeds, to ensure comparability of the results.

1): Experimenting with different components of the transformer

ToMe's default configuration involves token merging solely within the self-attention module. Our first set of experiments aims to gauge how the performance metrics are affected when creating sets of 768×768 images, by extending token merging to different combinations of transformer components.

1.1): default (only self-attn) vs all (self-attn, cross-attn and mlp)

The first experiment of part 1 compares the default setting of ToMe where token merging is only applied in the self-attn layer (black), with a different configuration that has token merging applied in the self-attn, cross-attn and mlp layer (red).

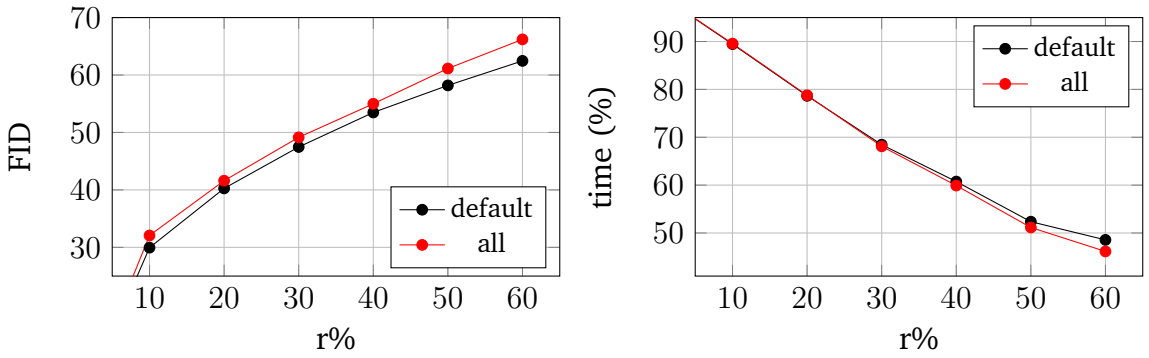


Figure 5: FID and relative time compared to r=0% for 1.1)

Firstly, the default setup of ToMe does make inference up to $2\times$ faster for $r = 50\%$ (see Tab. 7), matching Bolya and Hoffman's description.
The naive approach of improving performance by using token merging in every transformer layer, does not yield significant improvements. Image generation speed noticeably increases when $r > 30\%$, albeit at a clear cost of image quality with FID being consistently larger for this configuration (see Fig. 5).

ToMe, in its default setup, consistently produces images closer to their no-ToMe original, so extending token merging to both the cross-attn and mlp layer does not appear beneficial. This approach can therefore be disregarded.

1.2): [default] vs [self-attn & cross-attn] vs [self-attn & mlp]

After 1.1, the question remains whether performance drops were caused by using ToMe in both the cross-attn and mlp layer or only one of them, so the second experiment of part 1 attempts to delve deeper into that matter. This time token merging is only extended to either the cross-attn (red) or the mlp module (blue). Then a comparison is made with the results of the default ToMe settings (black) from the previous trial.

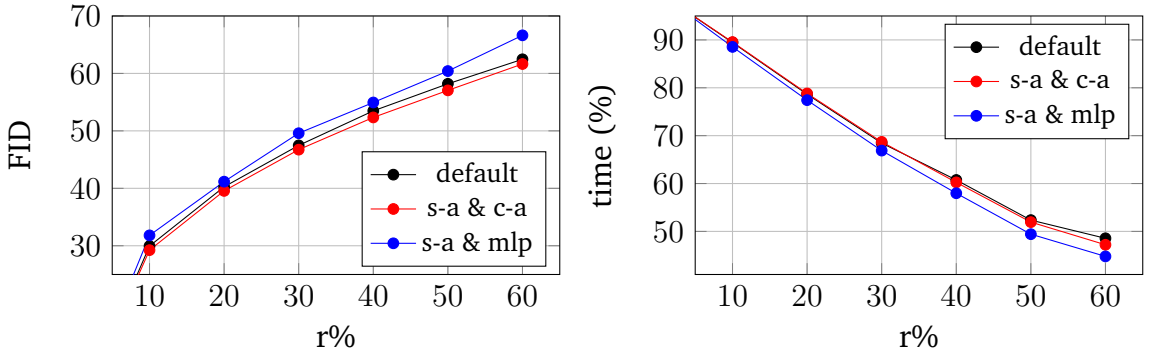


Figure 6: FID and relative time compared to r=0% for 1.2)

This time it's clearly visible that merging tokens within the self-attn and cross-attn module performs the best, as it slightly improves the ToMe default both in terms of image quality and image generation speed when $r > 30\%$ (see Fig. 6).
Token merging in the self-attn and mlp module on the other hand might offer a larger speedup for image generation, but clearly performs the worst in terms of image quality across the board (see Tab. 8).
We can therefore conclude that extending token merging to the cross-attn module might have positive effects on the performance, while extending it to the mlp module clearly worsens image quality.

1.3): [default] vs [cross-attn & mlp] vs [only cross-attn]

The common denominator of the previous examinations was the application of token merging in the self-attn layer. This time we are explicitly avoiding token merging in the self-attn module and instead apply it in only the cross-attn (blue) and both the cross-attn and mlp component (red). The results of the default ToMe settings (black) again carry over from the previous trial.

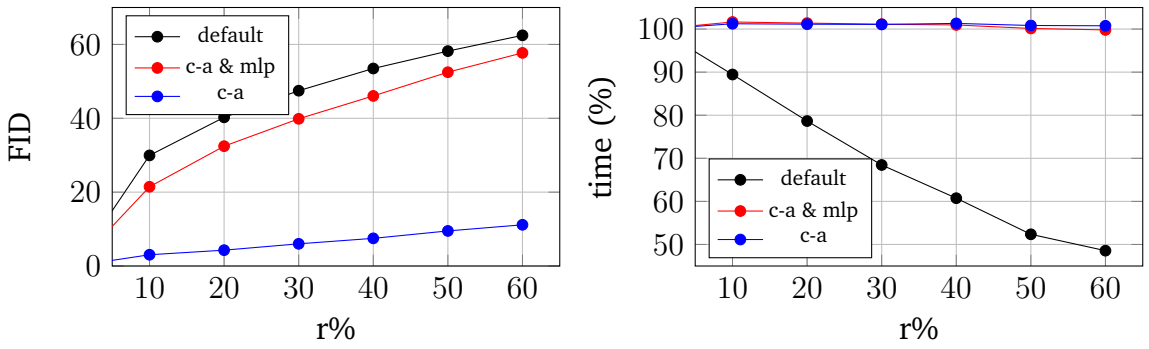


Figure 7: FID and relative time compared to r=0% for 1.3)

The most striking result here is that no token merging in the self-attn layer corresponds to no image generation speedup at all, rather slowing the process down (see Tab. 9). This strongly indicates that self-attention is the computational bottleneck of the transformer.
The apparent improvements to image quality compared to the ToMe default consequently become negligible without any speed benefits, though it can be noted that token merging in only the cross-attn layer greatly outperforms token merging in both the cross-attn and mlp layer in terms of image quality (see Fig. 7).
We conclusively derive that ToMe without involvement of the self-attn layer does not accelerate the image generation process at all and can be considered redundant and therefore be disregarded. It can additionally be derived from $1.1 - 1.3$ that token merging in the mlp layer has strong negative effects on image quality and, thus, can not be recommended.

1.4): [default] vs [self-attn & cross-attn] (the second time)

The most prominent takeaway of the first three experiments is that token merging in both the self-attn and cross-attn layers improves the performance of SD both in terms of image quality and image generation speed.
We want to further examine this discovery by repeating the experiment with a new set of 500 different prompt-seed pairs and then average the results of both trials.
It is again visible that the use of ToMe in both the self-attn and cross-attn layer improves image quality in any case and improves inference speed when $r > 30\%$, compared to the ToMe default (see Fig. 8). Image generation speed seems unaffected by ToMe when $r \leq 30\%$ (see Tab. 10). These results motivate token merging in both the self-attn and cross-attn modules as the new default for 768×768 images going forward.

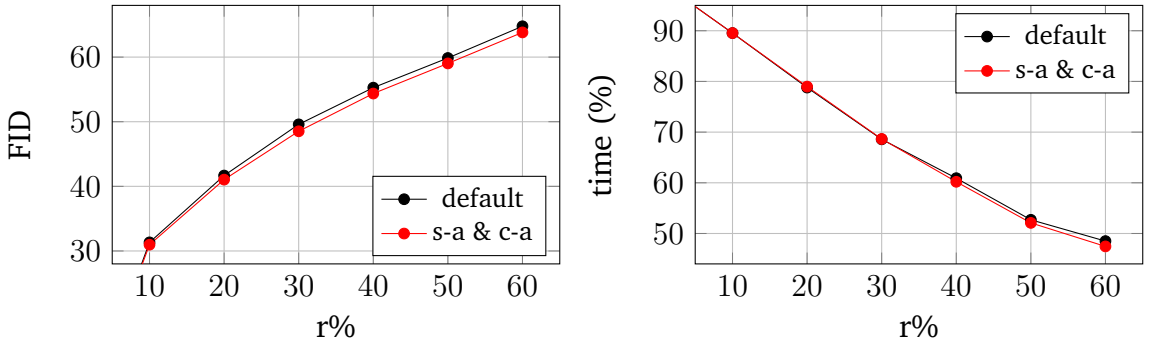


Figure 8: FID and relative time compared to r=0% for 1.4)

$r = 0\%$ 50% 50%

Table 3: 768×768 images created with ToMe default (middle) and ToMe expanded to cross-attn (right)

2): Exploring smaller image sizes

In this section, we want to expand the scope of our trials to smaller image sizes. Precisely, we want to repeat looking at token merging in the self-attn layer (black) and both the self-attn and cross-attn layer (red), but this time with smaller 512×512 images. Again a set of 500 prompts is randomly sampled and a corresponding set of random seeds is generated.

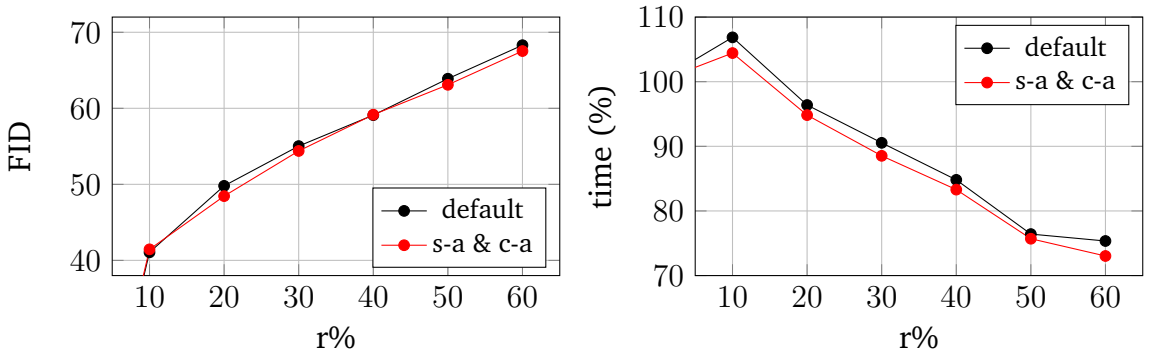


Figure 9: FID and relative time compared to r=0% for 2)

Generally, the speedup provided by ToMe noticeably decreases to ~25% from ~50% in the previous trials with 768×768 images. ToMe even worsens image generation

speed when $r < 20\%$, as the process actually becomes slower by $\sim$1 s/im at $r = 10\%$ (see Tab. 11).
The inclusion of the cross-attn layer provides a consistent time improvement, but image quality gains are quite inconsistent. ToMe in self- and cross-attn yields better quality for $r > 10\%$ but the gap closes completely at $r = 40\%$ (see Fig. 9).
We conclude that using ToMe for such small images (e.g. 512×512 or smaller) is less practical than it is for larger images. It is especially inadvisable to be used with a small merge volume ($r < 20\%$), as there is no time gained in the generation process, but information is lost during merging.
In consequence, we will not investigate the usage of ToMe on this scale any further. Neither will we investigate ToMe's effect on larger images due to hardware limitations.

3): Exploring different batch sizes

The third part of the experimental section examines how different values for sx and sy influence the performance of ToMe on 768×768 images. As a reminder, sx and sy determine the size of the token batches, which are used to spread the **dst** tokens somewhat evenly across the image, with larger batches resulting in a less consistent distribution. Our new default configuration with token merging applied in both the self-attn and cross-attn layer will be used throughout this section.

3.1): 3×3 batches

The first choice was scaling up from 2×2 (black) to 3×3 batches (red). This setup strongly tilts the ratio of src and dst tokens toward the former and theoretically allows for a larger number of tokens to be merged.

$$r_{max} = 1 - \frac{1}{3 * 3} = \frac{8}{9} \approx 88.89\%$$

We will, nevertheless, not exceed a merge rate of $r = 60\%$, as image quality already exhibits clear deterioration at that level and also due to the absence of other data available for comparison.

3.2): 1×2 vertical batches

Scaling down the same way to 1×1 batches is impossible, as it would result in every token landing in the **dst** set and $r_{max} = 0\%$.
We therefore chose to cut the batches in half horizontally, creating 1×2 batches. This

decision led to a balanced 50-50 distribution between **src** and **dst**.

$$r_{max} = 1 - \frac{1}{1 * 2} = 50\%$$

This limits our merge volume to 50% but allows for better matches during the merge process. Compared to the 2×2 default, every **src** token now has double the number of **dst** tokens to be matched with.

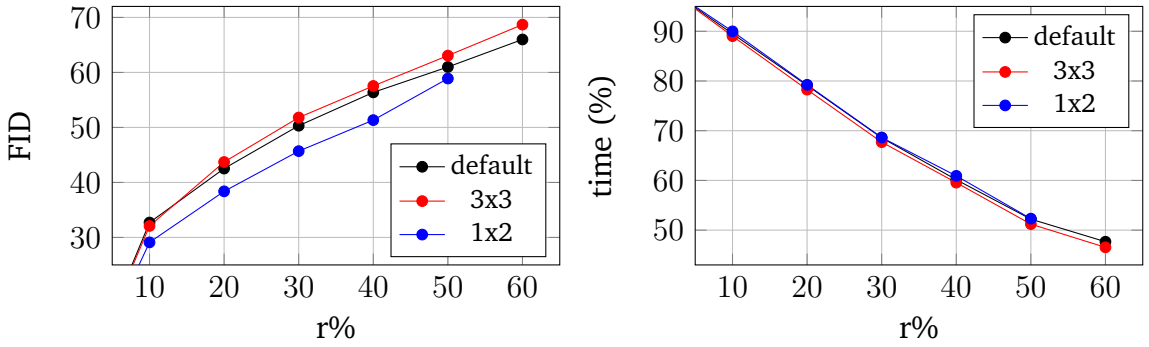


Figure 10: FID and relative time compared to r=0% for 3)

Resizing the batches only has minor effects on image generation speed, as all three configurations consistently remain within one second of each other, although the setup with 3×3 batches does perform the best in that regard (see Tab. 12).
Image quality shows a noticeable improvement with smaller batch sizes. FID values for the ToMe setup using 3×3 batches slightly exceed the ones of the setup with 2×2 batches (when $r > 10\%$), which in turn clearly exceed the ones of the version with 1×2 batches. (see Fig. 10). The gap closes towards $r = 50\% = r_{max}$, as every **src** token has to be merged when using 1×2 batches, regardless of how good of a match from **dst** is available.
It can still be concluded that using the 1×2 batches to create an equal number of **src** and **dst** tokens yields the best performance (even at a minor speed discount), especially when r is not close to 0% or r_{max}, while the usage of larger batches with a smaller **dst%** is not advisable.

4): Putting it all together

Finally, we are comparing the most successful configurations from the past experiments. That means we are looking at **setup 1**: the original ToMe default by Bolya and Hofmann (black), **setup 2**: our new default with token merging expanded to the cross-attn layer from 1.2 and 1.4 (red), and **setup 3**: the new default but with 1×2 batches for token partitioning from 3.2 (blue).

$r = 0\%$ 50% 50%

Table 4: 768×768 images created using 3×3 batches (middle) and 1×2 batches (right)

We expanded the image sets for **setup 1** and **2** to 1000 images per merge volume in 1.4, so we will do the same for **setup 3** to enable examinations on a larger scale.

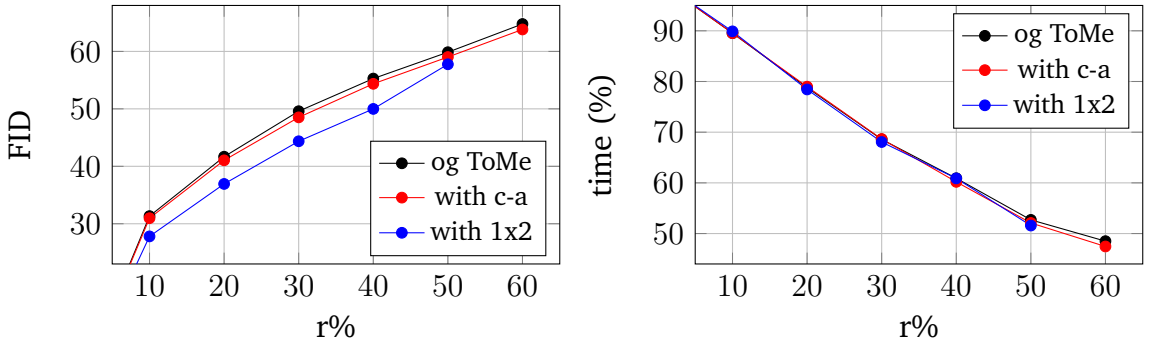


Figure 11: FID and relative time compared to r=0% for 4)

Speed is very close across the board, with the speedup offered by every version always differing by less than 1.2%. Within these thin margins, **Setup 2** consistently beat **Setup 1** when $r > 30\%$ and **Setup 3** was the quickest of the three most of the time (see Tab. 13).
Our discoveries regarding image quality are reinforced, with **setup 3** noticeably outperforming **setup 2**, which in turn performs slightly better than **setup 1** (see Fig. 11).

Summary

Firstly, the experiments we conducted support ToMe's effectiveness regarding image generation speedup. Inference time was cut in half while image quality remained great, although in some cases details and textures started to deteriorate for larger values of r.
Our results suggest that ToMe's performance can be improved by applying token merging in both the self-attn and cross-attn layer of the transformer. Furthermore, the best performance can be achieved by using 1×2 batches for token partitioning,

bringing particularly great improvements to image quality when $r \leq 40\%$.
Another important discovery is that token merging in the self-attn module is essential to unlocking speed improvements for image generation. On the other hand, using ToMe in the mlp layer seems to exclusively come at a disadvantage for image quality.
Moreover, it is advantageous to create sufficiently large images when using ToMe in order to reap greater speed benefits. We showed that the time improvements with token merging noticeably decrease from >50% for 768×768 images to only about 25% for 512×512 images at $r = 60\%$ (see Tab. 10, 11).
Additionally, it was shown that strongly skewing the **src**-**dst**-ratio away from an equal distribution has negative effects on image quality as well (see Tab. 12).
Another thing that stands out is that the speed advantage increases less when r becomes larger. E.g., switching from $r = 10\%$ to $r = 20\%$ usually results in a speed increase of more than 10%, while switching from $r = 50\%$ to $r = 60\%$ only gains about 5% most of the time.

$r = 0\%$ 20% 50%

Table 5: 768×768 images created with the ToMe configured according to our best setup from 3.2

5.5 Comparison to Original Results

The great differences in experimental setups and hardware make it impossible to compare specific numbers. The general promises of ToMe for SD by Bolya and Hoffmann can be confirmed though, as our results show that ToMe makes image generation up to $2.2\times$ faster while often maintaining great image quality even at $r = 60\%$. Though complex prompts and images were more susceptible to deterioration when large amounts of tokens were merged, which in some cases even lead to images completely falling apart. Image complexity and detail found little to no consideration in Bolya and Hoffman's work.

Our findings suggest different configurations of ToMe yield better results than the

original default. Firstly, we found that expanding token merging from only the self-attn layer to the cross-attn layer improves both speed and quality. There was no data for this setup listed by Bolya and Hofmann, although they mentioned: "Note that FID does not consider prompt adherence, which is likely why merging the cross-attn module actually reduces FID" [Bolya and Hoffman, 2023].
The implicit assertion that extending token merging to the cross-attn module lowers image quality outside of the scope of FID seems less plausible, as our setup indirectly considers prompt adherence by measuring the similarity between identical images (same size, prompt and seed), instead of larger sets of images that are merely class-balanced.
Our results do confirm Bolya and Hoffman's observation that token merging in the mlp layer is clearly detrimental to image quality and thus not recommended.

Furthermore, our findings regarding token partitioning clearly contradict Bolya and Hoffman's. Their partition experiments found notably stronger deteriorations to image quality for 1×2 strides than for 2×2 strides, with FID being almost 10% larger at $r = 50\%$ for the former [Bolya and Hoffman, 2023, Tab. 2 (a)]. For us on the other hand, FID was about 2% smaller at $r = 50\%$ and almost 10% smaller at $r = 30\%$ for 1×2 batches (see Tab. 13).

6 Further Exploration

The results of our experiments could be improved by expanding research on the influence of different elements of ToMe's or SD's configuration.
The larger 768×768 images benefited noticeably more from token merging than the 512×512 images, so exploration whether this positive effect further scales up with image size could become relevant.
Another aspect we rather briefly touched upon is the selection of **dst** tokens. ToMe currently only selects one **dst** token per batch, so experimenting with multiple **dst** tokens per batch and differently sized bacthes might improve performance as well.

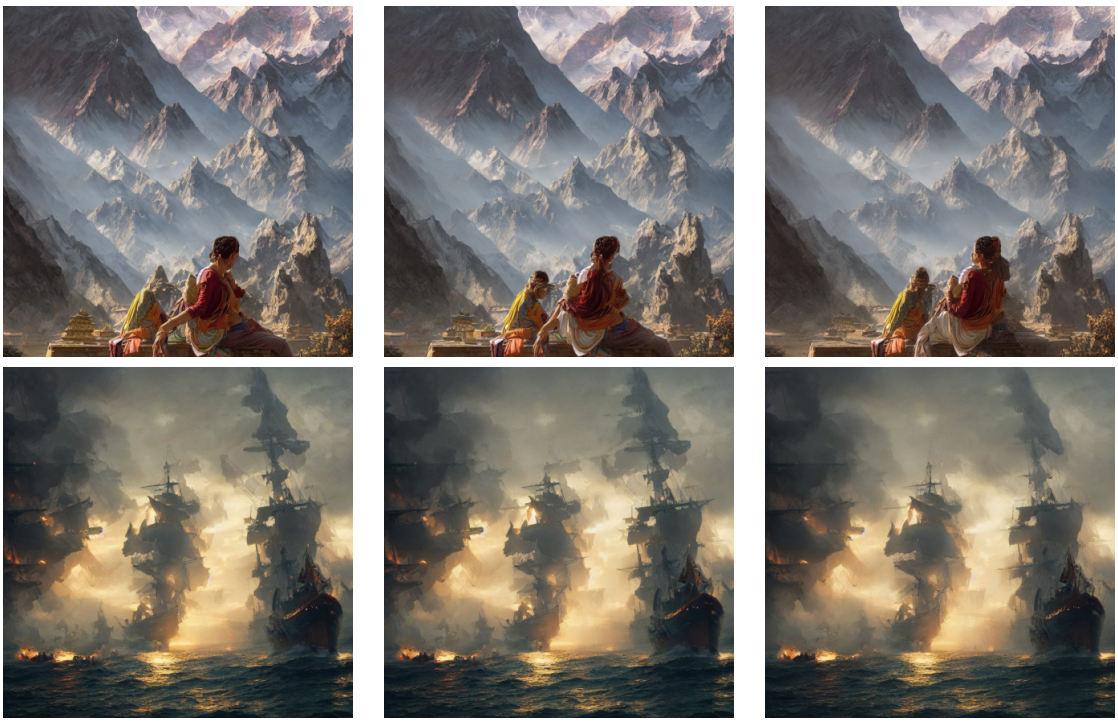
Bolya and Hoffman also mentioned that further improvements could also be achieved by exploring better unmerging strategies or whether proportional attention or key-based similarity are useful for inference. The current unmerge algorithm and token similarity metric are quite naive, so exploring ways to keep some of the information currently lost during the merge process could also greatly benefit the image quality.

7 Conclusion

In this work, we presented an exploration of the functional quality of Token Merging (ToMe) for Stable Diffusion by [Bolya and Hoffman, 2023]. ToMe offers accelerated image generation by merging similar tokens to reduce their overall number, notably without training.

In this thesis we conducted extensive experiments on images, obtaining speeds and fidelity superior to those of Bolya and Hoffman's default configuration.
ToMe has proven itself again as a practical extension to Stable Diffusion to reduce computational resources, cutting image generation time in half while mostly maintaining great image quality.
The application of ToMe can also be used to decrease the training time of models, though our experiments were only concerned with improving inference.

We hope this work can expand the understanding of Token Merging and reinforce its viability as a tool for improving the performance of diffusion models and inspire further research into the efficiency of transformers and generative models.



$r = 0\%$ 20% 50%

Table 6: 768×768 images created with the ToMe configured according to our best setup from 3.2

A Data Tables

Table 7: default (only self-attn) vs all (self-attn, cross-attn and mlp)

r%	FID	
0	0	0
10	29.95	32.07
20	40.26	41.60
30	47.74	49.15
40	53.48	54.98
50	58.19	61.13
60	62.46	66.21

r%	s/im (% to r=0%)	
0	70.74 (100)	69.91 (100)
10	63.28 (89.45)	62.61 (89.56)
20	55.64 (78.65)	55.06 (78.76)
30	48.42 (68.45)	47.61 (68.10)
40	42.97 (60.74)	41.88 (59.91)
50	37.04 (52.36)	35.76 (51.15)
60	34.35 (48.56)	32.26 (46.15)

Table 8: [default] vs [self-attn & cross-attn] vs [self-attn & mlp]

r%	FID		
0	0	0	0
10	29.95	29.23	31.81
20	40.26	39.55	41.16
30	47.74	46.73	49.59
40	53.48	52.34	54.94
50	58.19	57.05	60.41
60	62.46	61.64	66.63

r%	s/im (% to r=0%)		
0	70.74 (100)	69.75 (100)	73.88 (100)
10	63.28 (89.45)	62.46 (89.55)	65.41 (88.54)
20	55.64 (78.65)	54.98 (78.82)	57.20 (77.42)
30	48.42 (68.45)	47.92 (68.70)	49.41 (66.88)
40	42.97 (60.74)	42.02 (60.24)	42.83 (57.97)
50	37.04 (52.36)	36.23 (51.94)	36.51 (49.42)
60	34.35 (48.56)	32.93 (47.21)	33.08 (44.78)

Table 9: [default] vs [cross-attn & mlp] vs [only cross-attn]

r%	FID		
0	0	0	0
10	29.95	21.46	3.05
20	40.26	32.45	4.29
30	47.74	39.86	6.01
40	53.48	46.06	7.49
50	58.19	52.48	9.50
60	62.46	57.71	11.16

r%	s/im (% to r=0%)		
0	70.74 (100)	70.78 (100)	69.99 (100)
10	63.28 (89.45)	71.94 (101.64)	70.85 (101.23)
20	55.64 (78.65)	71.76 (101.38)	70.78 (101.13)
30	48.42 (68.45)	71.58 (101.13)	70.74 (101.07)
40	42.97 (60.74)	71.45 (100.95)	70.91 (101.31)
50	37.04 (52.36)	70.88 (100.14)	70.57 (100.83)
60	34.35 (48.56)	70.64 (99.80)	70.52 (100.76)

Table 10: [default] vs [self-attn & cross-attn] (the second time)

r%	FID	
0	0	0
10	31.33	30.97
20	41.67	41.04
30	49.59	48.52
40	55.27	54.36
50	59.86	59.02
60	64.77	63.82

r%	s/im (% to r=0%)	
0	70.76 (100)	69.88 (100)
10	63.36 (89.54)	62.57 (89.54)
20	55.75 (78.79)	55.19 (78.98)
30	48.53 (68.58)	47.97 (68.65)
40	43.10 (60.91)	42.07 (60.20)
50	37.29 (52.70)	36.40 (52.09)
60	34.32 (48.50)	33.16 (47.45)

Table 11: [default] vs [self-attn & cross-attn] (512×512)

r%	FID	
0	0	0
10	41.06	41.45
20	49.80	48.45
30	55.03	54.39
40	59.10	59.16
50	63.39	63.09
60	68.30	67.53

r%	s/im (% to r=0%)	
0	16.92 (100)	17.61 (100)
10	18.08 (106.86)	18.39 (104.43)
20	16.31 (96.39)	16.70 (94.83)
30	15.32 (90.54)	15.59 (88.53)
40	14.35 (84.81)	14.67 (83.30)
50	12.93 (76.42)	13.33 (75.70)
60	12.75 (75.35)	12.86 (73.03)

Table 12: default (2×2) vs 3×3 vs 1×2

r%	FID		
0	0	0	0
10	32.71	32.09	29.10
20	42.52	43.71	38.38
30	50.31	51.80	45.69
40	56.37	57.54	51.33
50	60.99	63.06	58.89
60	65.99	68.69	-

r%	s/im (% to r=0%)		
0	70.02 (100)	69.56 (100)	69.34 (100)
10	62.67 (89.50)	61.92 (89.02)	62.39 (89.98)
20	55.39 (79.11)	54.44 (78.26)	54.94 (79.23)
30	48.01 (68.57)	47.09 (67.70)	47.59 (68.63)
40	42.11 (60.14)	41.43 (59.56)	42.24 (60.92)
50	36.56 (52.21)	35.62 (51.21)	36.27 (52.31)
60	33.38 (47.67)	32.37 (46.54)	-

Table 13: [only self-attn] with 2×2 batches vs [self-attn & cross-attn] with 2×2 vs [self-attn & cross-attn] with 1×2

r%	FID		
0	0	0	0
10	31.33	30.97	27.80
20	41.67	41.04	36.93
30	49.59	48.52	44.36
40	55.27	54.36	49.99
50	59.86	59.02	57.76
60	64.77	63.82	-

r%	s/im (% to r=0%)		
0	70.76 (100)	69.88 (100)	71.25 (100)
10	63.36 (89.54)	62.57 (89.54)	64.05 (89.89)
20	55.75 (78.79)	55.19 (78.98)	55.88 (78.43)
30	48.53 (68.58)	47.97 (68.65)	48.49 (68.06)
40	43.10 (60.91)	42.07 (60.20)	43.38 (60.88)
50	37.29 (52.70)	36.40 (52.09)	36.75 (51.58)
60	34.32 (48.50)	33.16 (47.45)	-

B Images

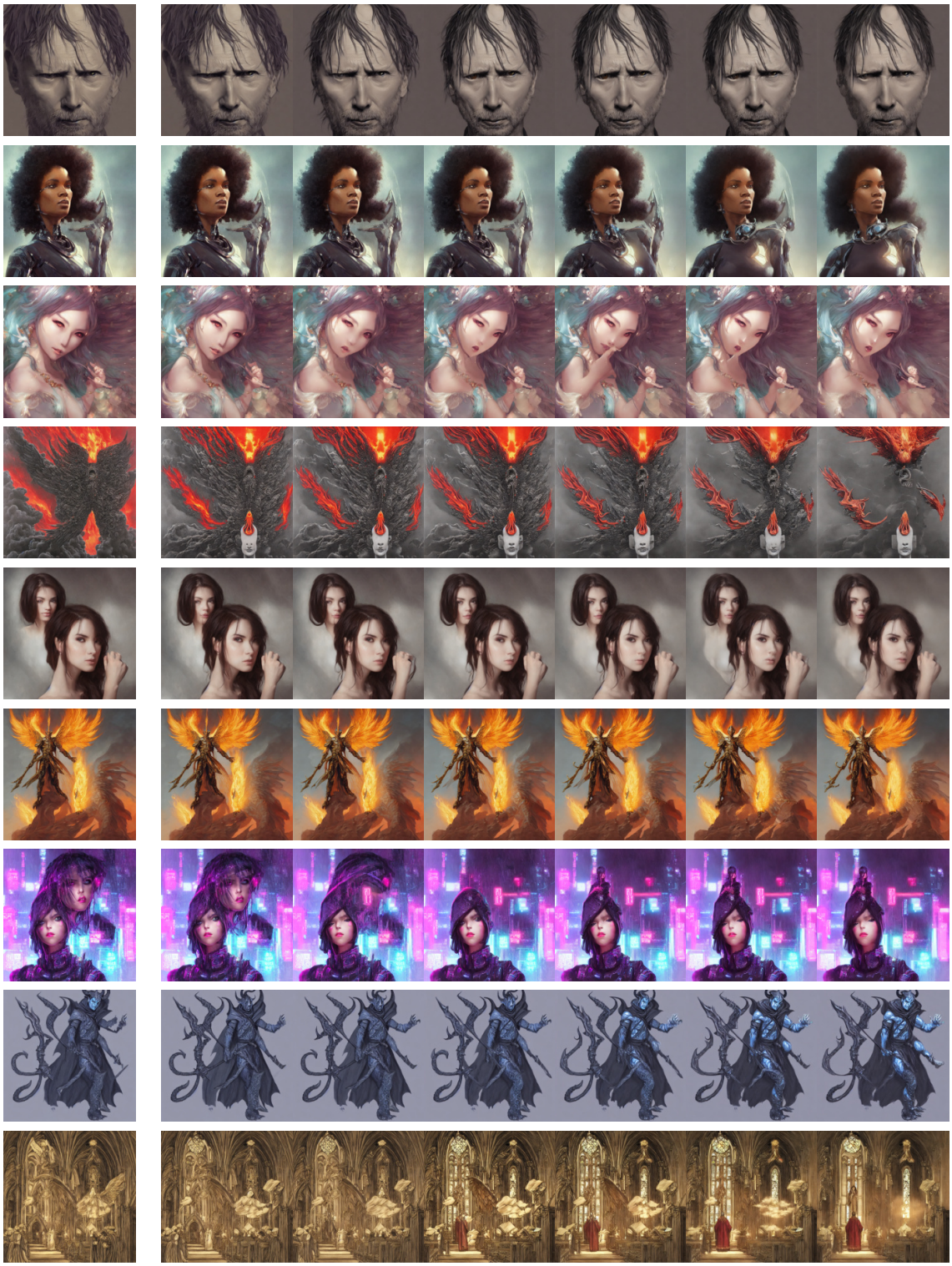


Table 14: 768×768 images created with the original default configuration of ToMe

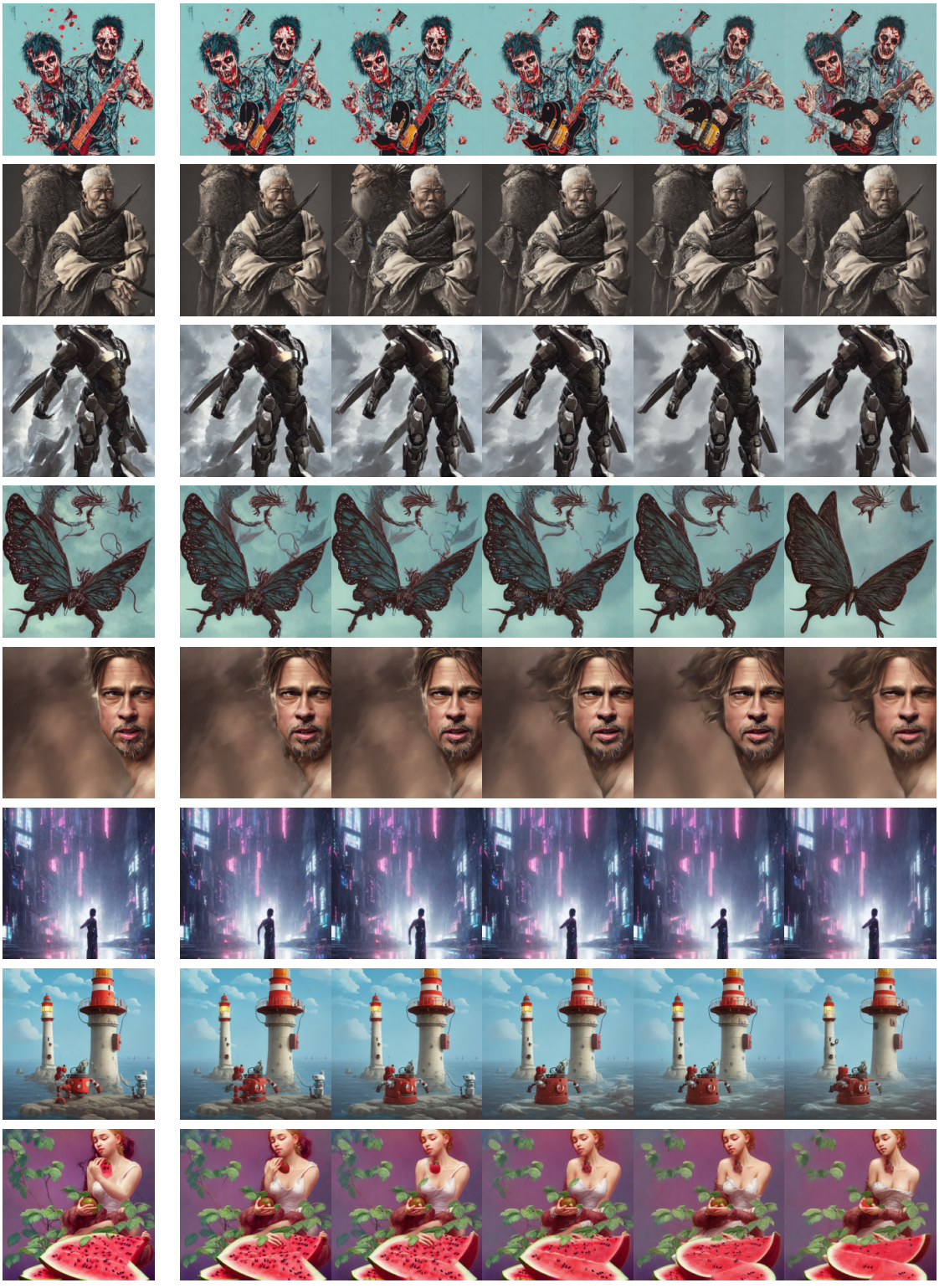


Table 15: 768×768 images created with the ToMe configured according to our best setup

C Libraries and Code

Python: 3.8.3
PyTorch: 1.13.0
Diffusers: 0.18.2
Transformers: 4.30.2
NumPy: 1.19.5
SciPy: 1.10.1
tomesd: 0.1.3
pytorch-fid (fork): 0.3.0
CUDA Toolkit: 11.3.1
cuDNN: 8.2.0

Software on HPC: CentOS 7.9.2009 Linux with 3.10.0-1160 kernel

Code for image generation (file: src/gen_imgs.py):

```
import sys
sys.path.insert(1, '/gpfs/project/hebal100/ba-code')
from diffusers import DiffusionPipeline
import torch, time, random, string, tomesd, numpy as np, pandas as pd

HPC_PATH = "/gpfs/scratch/hebal100"

# save command line args
assert len(sys.argv) >= 5
MAIN_DIR = sys.argv[1]
sample_size = int(sys.argv[2])
x, y =  int(sys.argv[3]), int(sys.argv[4])
try:
    src_file = sys.argv[5]
except IndexError:
    src_file = None

# load prompts and generate seeds
if src_file == None:
    all_prompts = pd.read_csv('data/prompts.csv')['columnn'].values
    idcs = np.random.randint(0, len(all_prompts), sample_size)
    prompts = all_prompts[idcs]
    seeds = np.random.randint(0, 4294967295, len(prompts))
# load prompts and seeds from previous run
else:
```

```
    prompts = pd.read_csv(f'{HPC_PATH}/data/{src_file}')['prompt'].values
    seeds = pd.read_csv(f'{HPC_PATH}/data/{src_file}')['seed'].values

# method for cutting oversized prompts
def cut_prompt(prompt, max_len=300, delimiter=','):
    if len(prompt) <= max_len:
        return prompt

    idx = prompt[:max_len].rfind(delimiter)
    # catch if delimiter isn't used
    if idx <= 0:
        idx = prompt[:max_len].rfind(' ')

    return prompt[:idx]

# build pipeline
assert torch.cuda.is_available()
pipeline = DiffusionPipeline.from_pretrained('pipelines/SD-v1-5').to('cuda')
pipeline.enable_attention_slicing()
def dummy(images, **kwargs):
    return images, [False]
pipeline.safety_checker = dummy

# set up lists
merge_volumes = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
directories = ['images_0', 'images_10', 'images_20', 'images_30', 'images_40', 'images_50', 'images_60']
logger = []

# run image generation loop to create image sets
for r, dir in zip(merge_volumes, directories):
    tomesd.apply_patch(pipeline, r, sx=2, sy=2, merge_attn=True, merge_crossattn=False, merge_mlp=False)
    for i in range(sample_size):
        prompt, seed = cut_prompt(prompts[i]), seeds[i].item()
        # create image and measure time
        start = time.time()
        image = pipeline(prompt, x, y, generator=torch.Generator().manual_seed(seed), ).images[0]
        end = time.time()
        diff_time = end - start
        # save image
        name = 'img_' + ''.join(random.choices(string.ascii_letters + string.digits, k=10))
```

```
        image.save(f'{HPC_PATH}/{MAIN_DIR}/{dir}/{name}.png')
        # create new log entry
        logger.append([prompt, seed, r, diff_time, name])
tomesd.remove_patch(pipeline)

# save log
log = pd.DataFrame(logger, columns=['prompt', 'seed', 'm_vol', 'time', 'name'])
name = 'log_' + ''.join(random.choices(string.ascii_letters + string.digits, k=5))
log.to_csv(f'{HPC_PATH}/{MAIN_DIR}/logger/{name}.csv', index=False)
```

Code for calculation of FID and average time (file: src/logging/gen_perf_log.py):

```
import sys
sys.path.insert(1, '/gpfs/project/hebal100/ba-code')
import pandas as pd
# pytorch_fid was manually cloned into the project
from libs.pytorch_fid.src.fid.fid_score import calculate_fid_given_paths

HPC_PATH = "/gpfs/scratch/hebal100"
DATA_PATH = sys.argv[1]
# directories where images of certain merge volume are stored
directories = ['images_0', 'images_10', 'images_20', 'images_30', 'images_40', 'images_50', 'images_60']
m_vols = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
perf_values = []

for m_vol, dir in zip(m_vols, directories):
    fid = calculate_fid_given_paths((f'{HPC_PATH}/{DATA_PATH}/images_0',
                                     f'{HPC_PATH}/{DATA_PATH}/{dir}'), batch_size=50, device='cuda', dims=2048)
    df = pd.read_csv(f'{HPC_PATH}/{DATA_PATH}/img_log.csv')
    df = df[df.m_vol == m_vol]
    avg_time = df['time'].values.mean()
    perf_values.append([m_vol, fid, avg_time])

perf_log = pd.DataFrame(perf_values, columns=['m_vol', 'fid', 'time'])
perf_log.to_csv(f'{HPC_PATH}/{DATA_PATH}/performance_log.csv', index=False)
```

D References

[Bolya et al., 2023] Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. (2023). Token merging: Your ViT but faster. In *International Conference on Learning Representations*.

[Bolya and Hoffman, 2023] Bolya, D. and Hoffman, J. (2023). Token Merging for Fast Stable Diffusion. *CVPR Workshop on Efficient Deep Learning for Computer Vision*.

[Chong and Forsyth, 2020] Chong, M. J. and Forsyth, D. (2020). Effectively unbiased fid and inception score and where to find them. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Hendrycks and Gimpel, 2016] Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

[Heusel et al., 2017] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

[Johnson, 2019] Johnson, J. (2019). Lecture 13: Attention. https://web.eecs.umich.edu/~justincj/slides/eecs498/498_FA2019_lecture13.pdf. From the course Deep Learning for Computer Vision.

[Kong et al., 2021] Kong, Z., Dong, P., Ma, X., Meng, X., Sun, M., Niu, W., Shen, X., Yuan, G., Ren, B., Qin, M., et al. (2021). Spvit: Enabling faster vision transformers via soft token pruning. *arXiv preprint arXiv:2112.13890*.

[Liang et al., 2022] Liang, Y., Ge, C., Tong, Z., Song, Y., Wang, J., and Xie, P. (2022). Not all patches are what you need: Expediting vision transformers via token reorganizations. *arXiv preprint arXiv:2202.07800*.

[Marin et al., 2021] Marin, D., Chang, J.-H. R., Ranjan, A., Prabhu, A., Rastegari, M., and Tuzel, O. (2021). Token pooling in vision transformers. *arXiv preprint arXiv:2110.03860*.

[Meng et al., 2022] Meng, L., Li, H., Chen, B.-C., Lan, S., Wu, Z., Jiang, Y.-G., and Lim, S.-N. (2022). Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12309–12318.

[Pan et al., 2022] Pan, Z., Zhuang, B., He, H., Liu, J., and Cai, J. (2022). Less is more: Pay less attention in vision transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2035–2043.

[Patil et al., 2022] Patil, S., Cuenca, P., Lambert, N., and von Platen, P. (2022). Stable diffusion with diffusers. *Hugging Face Blog*. https://huggingface.co/blog/stable_diffusion.

[Radford et al., 2021] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

[Ramesh et al., 2022] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. (2022). Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3.

[Rombach et al., 2021] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2021). High-resolution image synthesis with latent diffusion models.

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

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). 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*, pages 234–241. Springer.

[Ryoo et al., 2021] Ryoo, M., Piergiovanni, A., Arnab, A., Dehghani, M., and Angelova, A. (2021). Tokenlearner: Adaptive space-time tokenization for videos. *Advances in Neural Information Processing Systems*, 34:12786–12797.

[Seitzer, 2020] Seitzer, M. (2020). pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid. Version 0.3.0.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[von Platen et al., 2022] von Platen, P., Patil, S., Lozhkov, A., Cuenca, P., Lambert, N., Rasul, K., Davaadorj, M., and Wolf, T. (2022). Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers.

[Yin et al., 2022] Yin, H., Vahdat, A., Alvarez, J. M., Mallya, A., Kautz, J., and Molchanov, P. (2022). A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10809–10818.