# LATENT VIDEO DIFFUSION

Arjun Swani
Michelle Cao
Nick Bratvold
Simon Ryu

Project Sponsor:
Curtis Huebner

Project 2361
ENPH 479
Engineering Physics Project Laboratory
The University of British Columbia

April 18, 2024

# 1 Executive Summary

This research paper presents an open-source video diffusion model that uses current advancements in generative AI to synthesize controllable video sequences. In recent years, generative AI models have revolutionized various domains, including text, image, music, and video. However, the dominance of proprietary models poses challenges to accessibility, transparency, and innovation in the field.

Curtis Huebner, Head of Alignment at EleutherAI, implemented a latent diffusion model for video generation. Latent diffusion integrates variational autoencoders (VAE) with diffusion models, combining the efficient encoding and latent space learning by the former with the high-fidelity generation capabilities of the latter. As training these models at a large scale is extremely expensive, this paper focuses on small-scale experiments, with the aim of optimizing and demonstrating scalability of the model for future large-scale training runs. The code, datasets, and model weights are open for anyone to use.

For the VAE model, our experiments with batch sizes, learning rates, regularization, and normalization methods yielded improvements in model loss and image encoding quality. Additionally, we found that increasing model size and dataset size reduced loss and improved video reconstruction quality, showcasing scalability with additional compute resources. The diffusion transformer's performance improved as the model was scaled up and trained for longer periods. In training both of these models, training optimizations were made to increase training speed by over 10000 times.

The final model achieves promising results in generating video sequences, laying the groundwork for future developments in text-to-video generation, including user-conditioned generation. These advancements contribute to fostering transparency, collaboration, and ethical use of AI technologies, aligning with EleutherAI's mission to promote inclusivity and fairness in AI research and development.

# Contents

# List of Figures

# 2 Introduction

## 2.1 Project Background

### 2.1.1 Applications of Generative AI

Generative AI models learn the underlying patterns and structures of a dataset and use this knowledge to generate novel samples that resemble the original data. These models have seen significant development and innovation in recent years, driven by breakthroughs in deep learning models, such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and diffusion models.

Generative AI has found applications across a wide range of domains, including text generation, image generation, music composition, and video generation. In healthcare, generative AI facilitates medical image analysis, drug discovery, and personalized treatment planning. In education, it offers interactive learning experiences and adaptive tutoring systems. However, the dominance of proprietary and closed-source models in the field poses challenges to equitable access, transparency, and innovation. Leading models developed by companies like OpenAI and DeepMind are inaccessible to researchers and developers, hindering collaboration and limiting opportunities for advancements in the field. Efforts to promote open-source alternatives and foster transparency in AI model development are therefore crucial for ensuring inclusivity, fairness, and ethical use of AI technology.

### 2.1.2 About our Sponsor

Our project is sponsored by Curtis Huebner, who is the Head of Alignment at EleutherAI. EleutherAI is a non-profit research institute that focuses on the interpretability and alignment of large AI models, where alignment refers to the process of encoding human values and goals into ML models to make them as helpful, safe, and reliable as possible. To this end, their goals are to:

- Advance research on interpretability and alignment of foundation models
- Ensure equitable access to the study of foundation models
- Educate people about the capabilities, limitations, and risks associated with AI technologies

EleutherAI is currently aiming to develop an open-source text-to-video generation model that is on par with current benchmarks.

### 2.1.3 Project Objectives and Scope

The project aims to address several challenges inherent in creating a video generation model, focusing on key objectives that include:

- **Data Acquisition:** Careful sourcing of high-quality video datasets while navigating potential licensing issues to ensure ethical and legal use of the data.

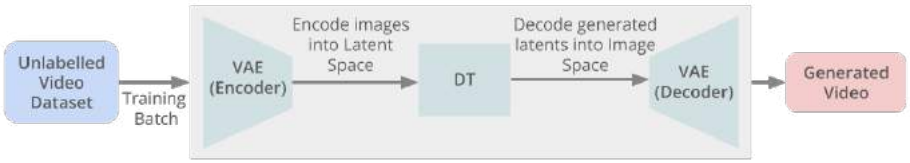



Figure 1: An overview of our full text-to-video generative model

- **Optimization:** While scaling up model size can yield improvements, the project will focus on exploring optimization techniques to enhance model performance and efficiency. Small enhancements in model performance can lead to substantial cost savings at scale, making optimization a priority.
- **Controllability:** Unlike text-to-image and language models, labeled training data for text-to-video is relatively scarce. Therefore, the project aims to address the challenge of controllability by exploring alternative, data-efficient methods for training video generation models.
- **Scaling:** Once an effective model architecture and training procedure have been established, the project will focus on scaling the code base to handle larger datasets and more complex models.

In order to streamline the project scope and focus on key objectives, we made the decision to limit the scope to unconditioned video generation to reduce complexity and resource requirements. We aimed to prioritize simplicity and feasibility, allowing us to concentrate our efforts on model development, dataset preparation, training optimization, and parallelization.

## 2.2 High Level Model Overview

The video generation architecture proposed for this project leverages a combination of variational autoencoder (VAE) and diffusion transformer models to achieve high-fidelity and controllable video synthesis. This section provides a high-level overview of the key components and operation of the proposed architecture.

**Variational Autoencoder (VAE)** The VAE serves as the foundational component for learning the latent space representation of video sequences. It comprises of the encoder and the decoder. The encoder takes input video frames and maps them to a latent space, where each point represents a compressed representation of the input data. The decoder then reconstructs the original video frames from the latent space representations. By training to minimize reconstruction error, the model learns to capture the underlying structure and dynamics of the input video data.

**Diffusion Transformer** A diffusion transformer model is employed to generate video sequences from the latent representations produced by the VAE encoder. The diffusion transformer extends upon the traditional transformer architecture by

introducing a diffusion process, where noise is iteratively added to the input data at each time step. This noise diffusion mechanism helps stabilize the generation process and improve sample quality by progressively refining the generated outputs. Additionally, the diffusion transformer incorporates self-attention mechanisms to capture long-range dependencies and contextual information across video frames, facilitating a more accurate and coherent video synthesis.

**Integration** The VAE and diffusion transformer models are trained and integrated into a unified architecture, where the VAE serves as the encoder for learning latent representations, and the diffusion transformer refines the generated video sequences based on these representations.

# 3 Discussion

## 3.1 Theory

### 3.1.1 Variational Autoencoders

**Overview** Variational Autoencoders (VAEs) are probabilistic, latent models comprised of encoder and decoder neural networks.

The VAE trains by learning how to encode the training dataset to a latent space. A sample from a VAE is generated by sampling a latent variable $z$ through the learnt distribution and reconstructing it through the decoder network.

VAEs are trained by minimizing a variational lower bound which makes easier to compute than diffusion networks and more stable than GANS [13]. Further, they train a regularised and controlled latent space which allows for improvements in sampling speed and compute requirements with proper tuning. A restricted latent space is especially important in video generation where pixel and frame representations can be fairly redundant [15]. Further, latent representations are often used for downstream learning, motivating to widespread adoption of VAEs in intensive generative tasks [16].

VAE models are usually implemented with Gaussian priors for latent distribution often leading to visually blurry image reconstructions and less detailed results.Further, the L2-norm component in the VAE loss function suffers from the curse of dimensionality. Depending on the application, different variations of VAEs are used in conjunction with other models to achieve desired results [13].

**Model** The VAE encoder and decoder components are implemented as neural networks. The decoder neural network represents the distribution $p_\theta(X|z)$, translating an input latent variable $z$ to reconstructed data $x)$. The encoder on the other hand learns a probability distribution $q_\phi(z|X)$, by encoding training data $X$ into latent representations, such that

$$q_\phi(z|x) \approx P_\theta(z|X)$$

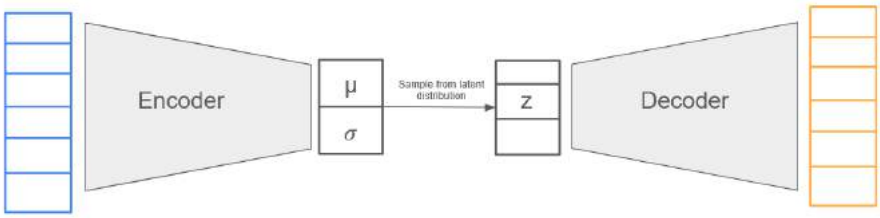



Figure 2: Functional overview of a VAE auto-encoder

allowing the computation of the marginal likelihood and the approximation of the intractable posterior.Moreover, the inference step does not have to be retrained for each data point. The inference network $q_\phi(z|X)$ can be used across the data set.

**Objective Function** The optimisation objective for the VAE is evidence lower bound given by

$$L(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta \left(\mathbf{x}|z\right) - D_{KL} \left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right)\right] \tag{1}$$

Since the $D_{KL}$ term is always positive, the following lower bound is established [9]

$$\log p_X \geq E[\log p(X|Z)] - D_{KL}[q(Z|X)||p(Z)] \tag{2}$$

By maximising $L(\theta, \phi)$, The VAE simultaneously

- Maximises the expectation of marginal likelihood $E[p_\theta(x)]$ making the generative model better. This is akin to $L_2$ loss between the sampled data and the observed data.
- Minimize the KL divergence to make the approximation $q(Z|X)$ as close to the target distribution $p(Z)$.

To make training compatible with stochastic gradient descent it is assumed that $q_\phi$ obeys the Gaussian distribution [13].
A popular variation of VAE encoders, $\beta$-VAEs scale the $D_{KL}$ divergence term by a hyper-parameter $\beta$. The scaling terms allows control over the regularization effect of the term, by tweaking the degree to which matches the Gaussian prior. A correctly optimized $\beta$ term have shown to provide a high degree of disentanglement of the latent space which allows for better downstream use [3].
The training algorithm for minimising the VAE objective function can be found in the appendix.

### 3.1.2 Diffusion Models

Diffusion models are a type of probabilistic generative model. They learn to denoise images through noise segmentation, where segmented noise can be iteratively subtracted from the original images to produce denoised images. As achieving full

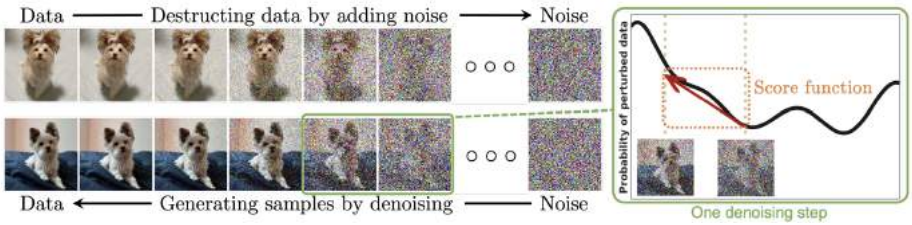



Figure 3: Noising and denoising training process in diffusion models. Adapted from [16].

segmentation in a single step yields low quality results, diffusion models are trained on images with varying noise levels. The level of noise at each step is dictated by a predetermined noise schedule. This noising and denoising training process is illustrated in Figure 3.

**Transformers** Transformers, a neural network architecture, are capable of implementing diffusion models, with their distinctive self-attention mechanism as a notable feature. Compared to other architectures that model sequential data, e.g. recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), attention mechanisms allow modeling of dependencies independent of their distance in the input or output sequences. Further, they are computationally efficient as they are extremely parallelizable. The transformer model architecture is pictured in Figure 4.

**Self-attention** The goal of self-attention is to identify and attend to the most important features in an input sequence. As an analogy to the more well-known convolution, which extracts features from images using relationships between pixels where the amount that pixels influence each other depends on their relative spatial position, self-attention extracts features from sequences using relationships between patches where the amount that patches influence each other depends on their embedding vectors. A summary of the key steps can be found in the appendix.

**Multi-head attention** Multi-head attention applies self-attention in parallel with different sets of learned weights, with the aim that each attention head focuses on different aspects or relationships within the input data. The outputs of multiple attention heads are concatenated and linearly transformed to produce the final attention output. This process is visualized in Figure 5.

**Encoder-decoder architecture** The encoder's objective is to create an interesting learned representation of the input for the decoder. The decoder predicts the next patch using encoder and previous decoder output.

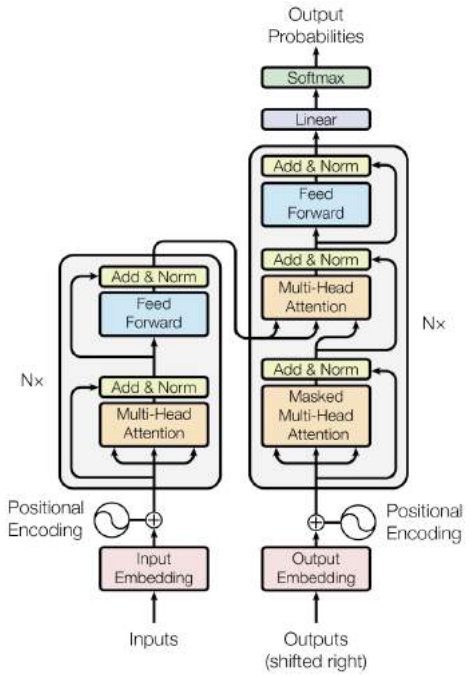



Figure 4: Transformer model architecture. Adapted from [12].

**Loss** The diffusion loss for a discrete noise schedule can be represented as:

$$L_T = \sum_{i=1}^{T} E_{q(z_{t(i)}|z)} D_{KL}[q(z_{s(i)}|z_{t(i)}, x)||p(z_{s(i)}|z_{t(i)})]. \tag{3}$$

where $U\{1, T\}$ is the uniform distribution on the integers $\{1, \ldots, T\}$, and $z_t = \alpha_t x + \sigma_t \epsilon$ [11]. In brief, it is a sum of the reconstruction loss between each model's denoising step compared to what the actual denoising should be.

This can be simplified by taking the time steps $T$ to infinity and modeling it as a continuous-time model

$$L_{\text{inf}} = \frac{1}{2} E_{\epsilon \sim \mathcal{N}(0,I), t \sim \mathcal{U}(0,1)} \left[ \gamma'_\eta(t) \left\| \epsilon - \hat{\epsilon}_T(z_t; t) \right\|_2^2 \right], \tag{4}$$

[11]

As in the Variational Diffusion Model paper, we use the Monte Carlo estimator of this loss for evaluation and optimization [11].

### 3.1.3 System Overview

Our full generative model integrates diffusion models with VAEs, which is known as latent diffusion.The encoder of the VAE compresses the input images into a latent

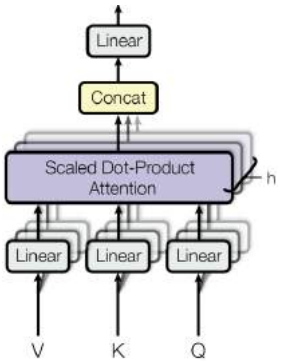



Figure 5: Multi-Head Attention. Adapted from [12].

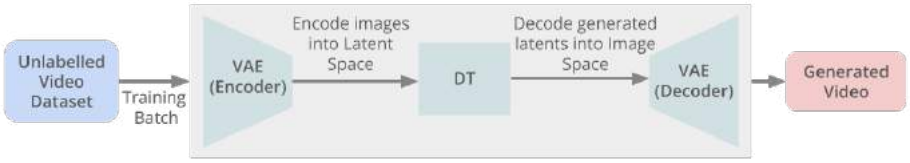



Figure 6: System overview.

representation. The diffusion model is trained on this learned latent space, and its output is decoded back into the original video space using the VAE's decoder. This process is illustrated in Figure 6. Training the diffusion model on a compressed latent space offers significant computational advantage over training on the raw, uncompressed space, resulting in faster training times and larger models. Latent diffusion thus leverages the efficient sampling capabilities of VAEs and the high-fidelity generative performance of diffusion models.

## 3.2 Design, approach, method, etc.

### 3.2.1 Machine learning framework

In this project, we employed JAX as the primary machine learning framework for developing our generation model. JAX is a Python mathematics library with a NumPy interface developed by Google. One of the key distinguishing features of JAX is its focus on functional programming and composable transformations [2]. Unlike more popular frameworks like TensorFlow or PyTorch, which rely on object-oriented approach, JAX adopts a functional and composable approach using transformations such as `jit`, `grad`, and `vmap`. These transformations enable efficient automatic differentiation, just-in-time compilation, and vectorization of numerical computations, making it well-suited for our aim to achieve large-scale computational performance. Another reason why JAX was chosen as our primary framework is because of its

ecosystem. JAX provides freedom to combine high-level neural network libraries and low-level modules to build a custom solution, making it ideal for the development of novel components and techniques.

### 3.2.2 Training infrastructure

We trained the models on a TPU v3-8 Pod, a high-performance computing platform provided by Google Cloud. TPUs, or Tensor Processing Units, are hardware accelerators specifically designed for deep learning tasks, offering significant speed and efficiency advantages over traditional CPU or GPU-based training. The TPU v3-8 configuration provides eight TPU cores, with each having a traditional vector processing part (VPU) as well as dedicated matrix multiplication hardware capable of processing 128x128 matrices. TPUs are equipped with 128GB of high-speed memory allowing larger batches, larger models and also larger training inputs [5].

### 3.2.3 Dataset

Choosing the appropriate dataset for our text-to-video generation project involved a consideration of permission issues, licensing constraints, and ethical concerns. we chose to utilize the YouTube User-Generated Content (UGC) Dataset that is shared under the Creative Commons license. The dataset contains around 1100 video clips with a duration of 20 seconds each, with labelled categories such as animation, gaming and sports and of various resolutions [17].

### 3.2.4 Training speed

In the early stages of our project, training speed emerged as a major bottleneck, hindering the efficiency and scalability of our model. In this section, we discuss training pipeline optimization and various parallelization strategies we explored in the development of the VAE.

**Parallel Data Loading** Initially, loading of data batches and training models were done separately in a single process which increased the total training time of the model. To better sequence the repetitive process of loading data, we implemented the producer-consumer pattern using Python's multiprocessing module which enables the processing of multiple processes by multiple processors. The data loading and training of the model was divided into separate producer and consumer processes, where the producer extracts samples of video frames and the consumer trains the model on the extracted batch of frames to update the parameters. While the processes are executed in parallel, queues are used as a communication channel to share the data. Figure 7 shows the comparison between the single-process and multi-process data loading with producer-consumer pattern.

**Data Parallelism** Data parallelism focuses on distributing the data across different nodes, which operate on the data in parallel. This approach is particularly useful

when the batch size is too large to fit on a single machine. A data parallelism framework achieves the following tasks:

1. It creates copies of the model, with one copy per each device.
2. It shards the data and distributes it to the devices.
3. Each device trains the model on their sharded data, and all results are aggregated together in the backpropagation step.

In JAX, `jax.Array` is a unified datatype for representing arrays that can be represented with physical storage spanning multiple devices. By describing the distributed memory layouts, we sharded the batch data into eight TPU devices.

**Model Parallelism** Unlike data parallelism that replicates the same model for all cores, model parallelism shards the tensors such as weight matrices and activations of a layer across multiple devices. It is therefore particularly useful when training a model that does not fit into a single GPU, which data parallelism cannot handle. Figure 8 shows the high level diagrams of data and model parallelism. Although we did not implement this strategy, it is more scalable than data parallelism for large networks and should be explored further. It is also possible to combine both model and data parallelism, discussed later in Recommendations.

**Result** Most training speed improvements came from various optimizations of data loading and training process. Figure 9 shows the time to train on a single image as these changes were introduced. Data parallelism did not significantly reduce the training time and we could not verify its implementation. As a final result, a first optimization provided a 375x speed up and further improvements in our batch training iterations saw a reduction in time from 30 seconds per iteration to less than a second per iteration. Overall, our training speed improved by over 10,000%.

# 4 Results

## 4.1 Variational Autoencoder

To train the VAE, we adjust a number of hyperparameters and training specifications to improve the model loss defined by Equation 1 and the visual fidelity of the encoder-decoder process. The objective is to enhance the performance of the model when it's relatively small and demonstrate its scalability as the model size grows. Below is a summary of the hyperparameters that were tracked and a brief intuition behind why we tune them.

A training run is carried out by setting model parameters and training for a set number of iterations. The final loss for validation and training filtered loss is recorded

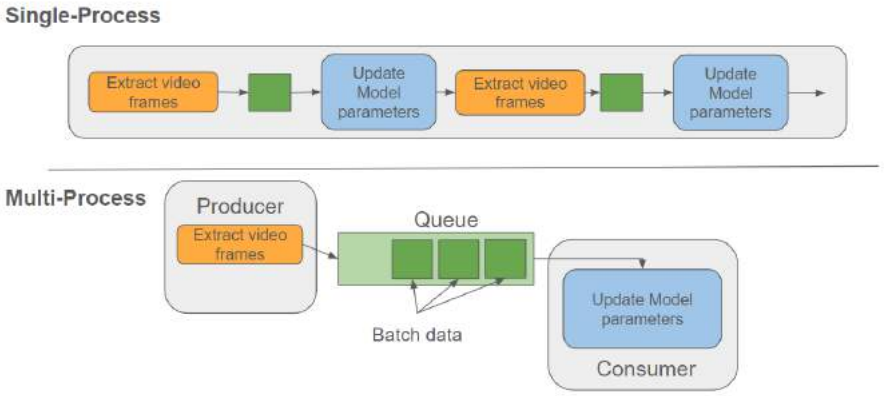



Figure 7: Single-Process and Multi-Process data loading.

| **Model Specifications** | **Intuition** |
|---|---|
| Dataset Size | Number of videos to train from |
| Learning Rate | Step size during training |
| Batch Size | Number of video frames processed before updating model weights. |
| Latent Vector | Number of latent vectors to represent a video frame. |
| KL Divergence Weight | Impact of regularization in model. |
| Normalization Technique | Method used to normalize model weights. |

Table 1: Hyperparameters and their Intuition

and the reconstruction quality is visually assessed. Figure 10 and Figure 11 show examples of training run and the reconstruction respectively.

The loss plot in Figure 10 is desired since the model shows obvious improvement then reaches a floor with no training instabilities. We attribute this noise floor to the noise in the dataset that VAE model can not constrain with its current settings.

It is important to note that while the training curves are ideal, the blue blob artifact shown in Figure 11 is not captured because the loss is calculated across the entire image. Future considerations could split the image into chunks and track each chunk's loss. This would allow identification of the artifact and the training iteration it occurs in.

A high quality VAE is important to create since the quality of the diffusion transformer is limited by the quality of the VAE's ability to encode and decode data. As such, we spent most of our effort to create a high quality VAE for the diffusion transformer.

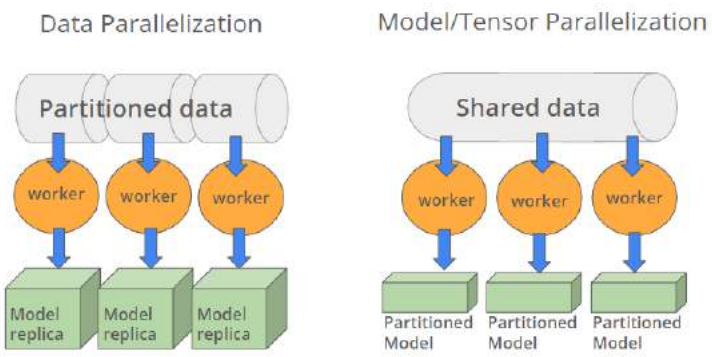



Figure 8: Data parallelism and model parallelism.

#### 4.1.1 Verification

To assess the feasibility of the model architecture, we trained the model on a single video to verify that we can create a high quality reconstruction of the input. This process revealed several bugs in the code implementation, the resolving of which had to be prioritised.

Figure 12 shows a few iterations of improving the VAE over several months and a final reconstruction that we used to verify the diffusion transformer.

#### 4.1.2 Dataset Size

The same VAE was then trained with a growing dataset size. There is improvement in the model loss and reconstruction quality as seen in Figure 13 and Figure 14. The VAE can extract more robust and high quality features to represent an image from more data. The results from these trends allowed us to scale our dataset size to 1100 videos when more storage became available. The data from 1100 videos are unavailable because many changes to the architecture and hyperparameters were made so no fair comparison can be made.

#### 4.1.3 Batch Size

By increasing the batch size the variance between each iteration decreased resulting in more stable gradient estimates. This allowed for better generalization of the data, but it does linearly increase the memory and time needed to train.

#### 4.1.4 Learning Rate

Two learning rate effects were observed. First, a learning rate that is too high will lead to loss divergence near the noise floor. Second, lowering the learning rate before a noise floor is reached results in a better model loss.


Figure 9: Batch training speed improvements.

The first effect is observed in Figure 15, where a learning rate of 0.0002 was used and and the loss blew up as it reached the noise floor. A model check point was saved before the divergence and was repeated with a learning rate of 0.00005. The model continued to learn, although overfit, and no loss blow up was observed.

The second effect was observed by training a model until it reached its noise floor then lowering the learning rate. Surprisingly, this caused zero improvement in the model loss and it ended up at [training:3.97, validation:4.06]. The same model was trained but the learning rate was lowered just before the noise floor was reached and an improvement was observed, [training:3.88, validation:3,99].

From these two effects, a two-stage training schedule is recommended. First, a constant high learning rate, then when the training slows down a exponential decaying learning rate should be implemented. This sort of schedule has been seen in training Dalle Mega [6].

#### 4.1.5 Latent Vector Size

By increasing the number of latent vectors to represent an image in latent space we see improved model loss and reconstruction quality. The total size of the VAE model increases when more latent vectors are added. These trends show that the model is capable of scaling with more compute. Figure 16 shows strong evidence that the model loss decreases with more latent vectors. Figure 17 shows a series images where the resolution of the reconstructions increase with more latent vectors. At 4096 latent vectors a single core of the TPU is maxed out.

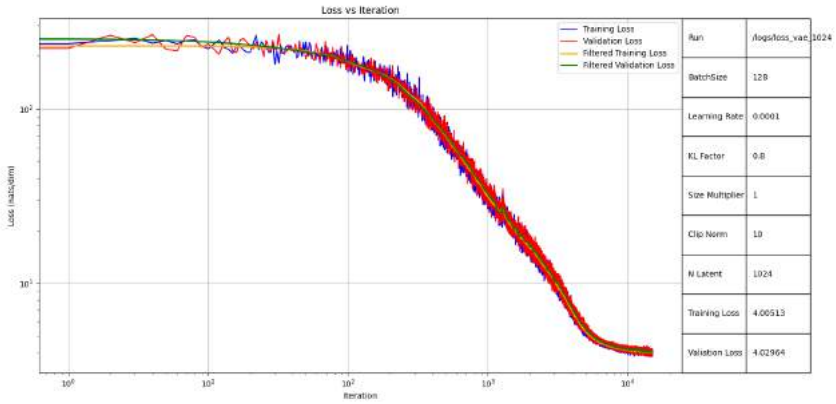
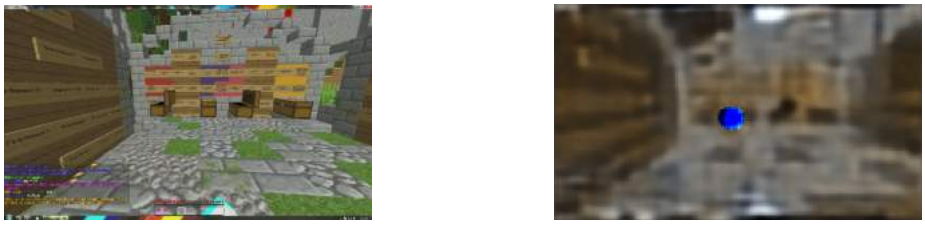



Figure 10: A plot of model loss vs iteration number. The training and validation curves are shown in blue/yellow and red/green respectively. A filtered average of each is shown to smoothly track the losses. On the right are the hyper-parameter settings of the training run.

### 4.1.6 Kullback–Leibler Divergence

The Kullback-Leibler Divergence (KL) is used to regularize the data. It is added into the loss function Equation 1 to promote a normal distribution of the latent space.Since the validation loss across all the training runs have rarely shown evidence of over fitting a reduction in the KL term will allow the model to focus more on quality reconstructions than a normal latent space. By lowering the weight of the KL term we see immense improvement in reconstruction quality, especially in colour accuracy, as shown in Figure 18.



(a) The base image. (b) The reconstructed image.

Figure 11: The base image is encoded and decoded with a trained VAE model. The reconstruction quality is poor in resolution and contains a blue artifact.

(a) Base Image

(b) VAE Attempt

(c) Final VAE

(d) Base Image

(e) VAE Attempt

(f) Final VAE

Figure 12: Two VAEs trained on a single video and their respective reconstructions. The top set is a mountain biking video and the second is a dog eating his favourite treat, Timbits. After tuning these models high quality reconstructions are observed.

#### 4.1.7 The "Blob"

We observed that the images generated by our VAE exhibited blob-shaped artifacts. An example is pictured in Figure 19. This is a known failure mode of VAEs that affects even widely-used models. It is hypothesized to be a result of the model trying to sneaking global information about the image through the latent space, by creating a strong, localized spike that dominates the statistics, This allows the model to effectively scale the signal as it likes elsewhere [8, 7]. In StyleGAN, the issue was pinpointed to be instance normalization, as normalizing the mean and variance of each feature map separately potentially destroys any information found in the magnitudes of the features relative to each other [8]. Our VAE uses layer normalization, which is similar to instance normalization except it normalizes across all channels rather normalizing across each channel separately. A visual comparison of common normalization methods is provided in Figure 20. In the VAE used for Stable Diffusion, the issue is suspected to be under-weighting of the KL divergence loss term [7]. To eliminate the blob-shaped artifacts, we then needed to either prevent the normalization layer from destroying global information or create an explicit mechanism to encode global information. To this end, we experimented with:

- Increasing the size of the latent space, as increasing the number of learnable parameters may allow the model to better encode global information
- Increasing the weight of the KL divergence term to further encourage the latent space distribution to match the prior distribution (Gaussian)

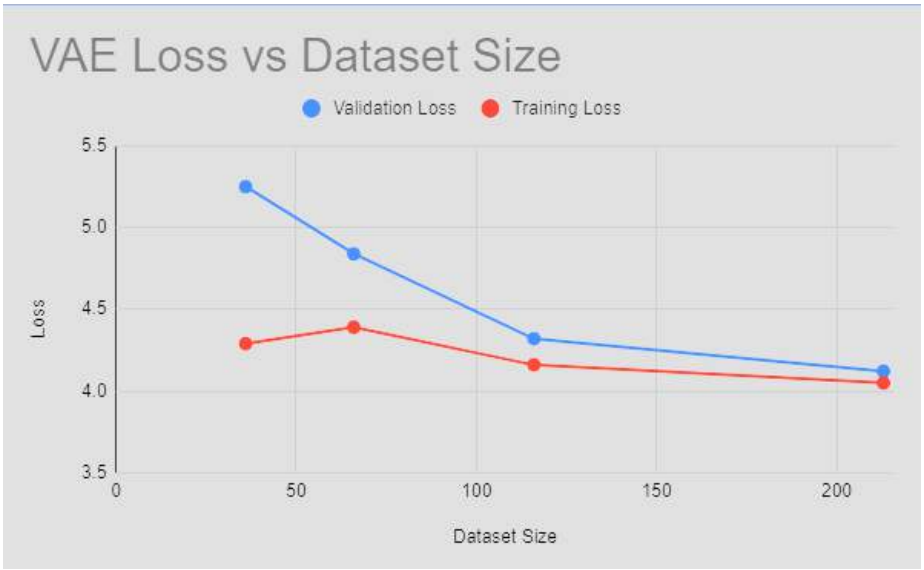



Figure 13: A plot of model loss vs dataset size. The training and validation loss decrease as more data is trained on.

(a) The base image. (b) 36 Videos (c) 66 Videos (d) 213 Videos

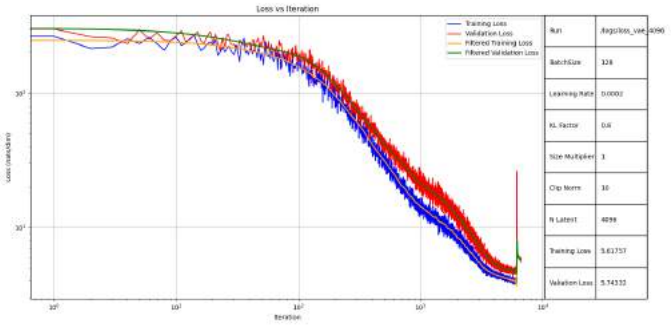
Figure 14: Reconstruction quality improves as the model has more data to extract features from.

- Increasing the convolution kernel size to increase the maximum relative distance at which pixels can influence each other
- Replacing layer normalization (all pixel locations across all channels) with channel normalization (each pixel location across all channels) to prevent values at different pixel locations from influencing each other within the normalization layer

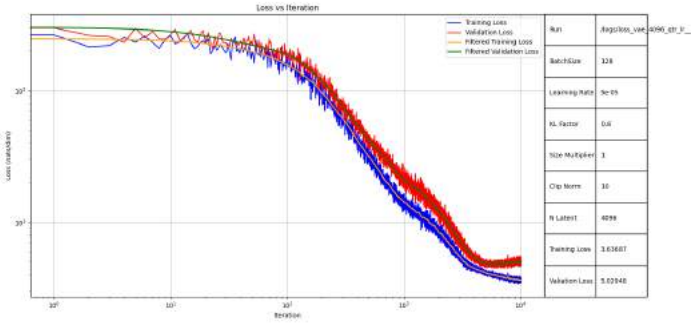
In our results, only replacing layer normalization with channel normalization eliminated the blob-shaped artifacts.

## 4.2 Diffusion Transformer

As with the VAE, we track a number of hyperparameters and training specifications to improve the model loss Equation 4 and visual fidelity of the diffusion process.




(a) Crashed training run with a learning rate of 2e-4.




(b) Stabilized training run with a learning rate of 5e-5.

Figure 15: Two training runs with different learning rates near end of training. Lowering the learning rate eliminated the model crash.

The end goal is to optimize the model performance of the model at the same model size and show scaling capabilities as the model size is increased. As stated earlier, the quality of the VAE limits the quality of the diffusion transformer because of this we spent the majority of the time working on creating a quality VAE with no artifacts which left us with little time to test the diffusion transformer. We looked at the learning rate and increasing the size of the model and displayed evidence of scalability.

A training run is carried out by setting model parameters and training for a set number of iterations. The final loss for validation and training filtered loss is recorded and the generation quality is visually assessed. In Figure 21 and Figure 22 an example training run and the unconditioned generation is shown respectively.

The training curve shows evidence of training, but the steep learning curve and plateau is less obvious as in Figure 10. The training run could likely continue

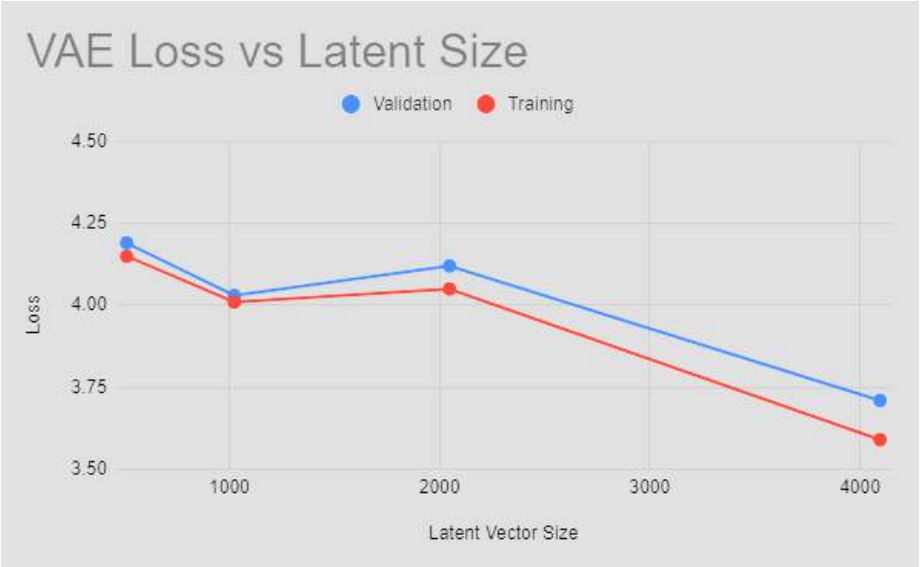



Figure 16: A plot of model loss vs latent size. The training and validation loss decrease as more vectors are used to represent an image.

training but the run was already at over 80 hours. The generated samples are created from random noise, the frames have similar colours but show no evidence of temporal consistency.

### 4.2.1 Verification

Since the frames in Figure 22 are quite poor it is a good idea to verify the model by over training it on a single video. We used the over trained bike VAE Figure 12 to encode the video. The encoded latent video was used to train the diffusion transformer and generation was made. The frames in Figure 23 show strong resemblance to the original biking video and displays evidence that the diffusion transformer architecture does in fact work.

(a) The base image. 1:1 compression (b) 512 latents, 450:1 compression (c) 1024 latents, 225:1 compression (d) 4096 latents, 56:1 compression

Figure 17: Reconstruction quality improves as the model has more latents to represent features.

(a) Full KL weight. (b) 0.1 KL weight.

Figure 18: Two reconstructions comparing the effect of KL divergence on the VAE. Improved reconstructions are shown by lowering the KL divergence weight in the model loss.

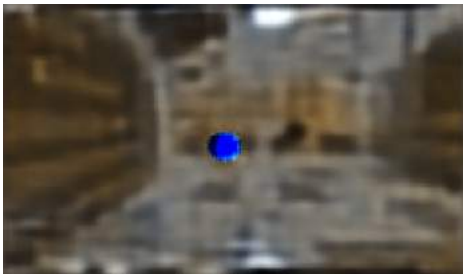


Figure 19: VAE reconstruction with blob-shaped artifact.

### 4.2.2 Learning Rate

Similar to the VAE increasing the learning rate too much causes model collapse. In Figure 24 we see three models at increasing learning rates, 5e-5, 1e-4 and 5e-4. The model ends up blowing up as the learning rate is increased.

### 4.2.3 Model Size

Since the diffusion model needs to process many encoded frames at once, its total model size must be much larger than the VAE. As such, we increased many parameters to increase the total model size from 1.5GB to 4.5GB. We used the same 4096 latent vector trained VAE and saw a large reduction in loss from the two diffusion models. The 1.5GB model had a training loss of 7.60 and a validation loss of 7.66 while the 4.5GB model had a training loss of 5.53 and a validation loss of 5.72, although the generation from each were still lackluster. Figure 25 shows the two diffusion models and a generated image frame. The resolution of the noise is much higher in the larger model.

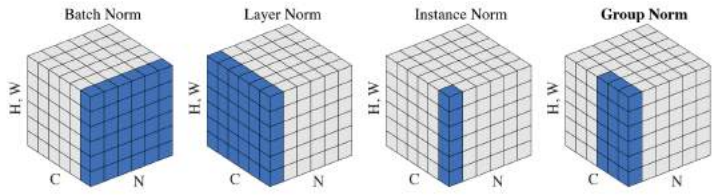



Figure 20: Common normalization methods, where N is the batch axis, C is the channel axis, and (H,W) are the spatial axes. The pixels in blue are normalized by the same mean and variance, which are computed by aggregating the values of these blue pixels. Adapted from [14].

### 4.3 Generative Video

A smaller VAE was trained with 1500 latent vectors. This data was encoded and a 4.5GB diffusion transformer was trained for 350000 iterations which took 5 days. Figure 26 shows this training run with a final training loss of 3.2 and a validation loss of 3.7. Some overfitting is observed.

To generate the videos, a random encoded video was selected and 5 frames were used to prompt the generation. Using the frames to provide context, 5 additional frames were generated. In Figure 27, a collection of video generations are shown. These results are promising, and showcase the capability to generate video given a context. In this case it was a few frames but in the future this context could be text, image, or audio to produce video.

## 5 Conclusions

### 5.1 Variational Autoencoder

Based on our tests, a series of recommendations are listed in Table 2. It was shown that the model will scale with compute and additional resources through the dataset size and latent vector tests. The model can be optimized by creating a learning rate schedule, increasing batch size, lowering the weight of the KL divergence term, and applying channel normalization.

### 5.2 Diffusion Transformer

Unfortunately, since a VAE without artifacts was not created until the end, we were unable to thoroughly test the diffusion transformer. Evidence of model scaling was shown by increasing the total model size and seeing a strong reduction in the end loss of the training. The ability to generate video for small lengths was proven. Future work is needed.

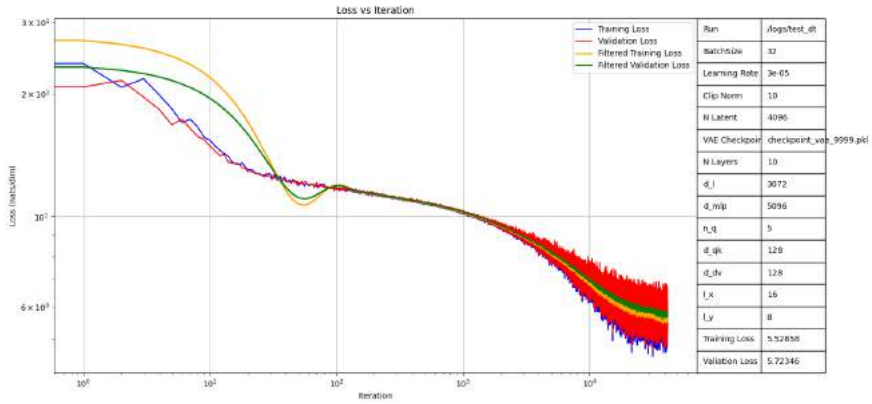



Figure 21: A log log plot of model loss vs iteration number. The training and validation curves are shown in blue/yellow and red/green respectively. A filtered average of each is shown to smoothly track the losses. On the right are the hyperparameter settings of the training run.

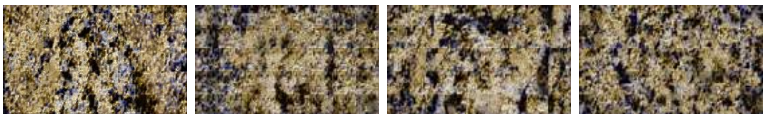


Figure 22: Four sequential generated frames from random noise.

# 6 Recommendations

## 6.1 Distributed Training

Despite the improvements achieved through parallelization and optimization strategies such as JIT compilation and parallel data loading, we encountered challenges associated with the size of the neural network that could be accommodated on a single TPU core. To overcome this limitation, we recommend several strategies that could be implemented:

1. **Padding aware architecture.** Cloud TPUs contain two systolic arrays of 128 x 128 ALUs on a single processor. When a matrix computation cannot occupy an entire MXU, the compiler pads tensors with zeroes, under-utilizing the TPU core and increasing the amount of on-chip memory storage required for a tensor [5]. A padding aware architecture can be adapted to minimize unused memory by avoiding padding.

2. **Activation check-pointing.** Activation checkpointing is a technique used to reduce GPU memory usage during training by clearing activations of cer-

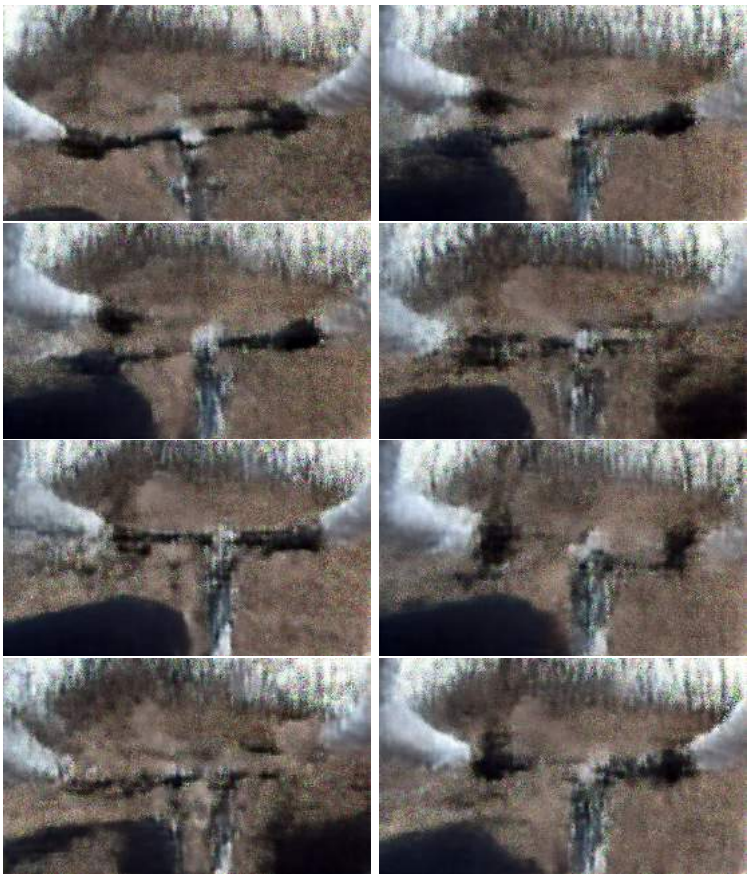


Figure 23: Eight sequential, left to right top to bottom, video frames from a diffusion transformer over trained on a single video.

tain layers and recomputing them during a backward pass. `jax.checkpoint` allows for recomputation of internal linearization points when differentiated.

3. **Fully Sharded Data Parallel (FSDP).** Fully Sharded Data Parallel (FSDP) is a hybrid of model parallelism and data parallelism that shards a model's parameters, gradients, and optimizer states across data-parallel workers. This distribution of model components allows for more efficient use of computing resources and enables training with larger batch sizes and models. FSDP also provides the option to offload the sharded model parameters to CPUs when they are not actively involved in computations.

| Model Specifications | Observations | Recommendations |
|---|---|---|
| Dataset Size | Increasing dataset size decreases loss | Increase storage limit |
| Learning Rate | Decreasing learning rate as training slows down results in decreased loss | Use constant high learning rate then exponential decay near end of training |
| Batch Size | Decrease loss variance | Increase if observing noise spikes or anomalies in training |
| Latent Vector | Increasing latent size decreases loss | Increase until diffusion transformer can't constrain the latent vector representation |
| KL Divergence Weight | Lowering weight improves reconstruction and loss | Reduce if reconstruction quality is poor |
| Normalization Technique | Channel normalization removes "blob" artifact | Apply channel normalization if artifacts occur |

Table 2: Variational Autoencoder recommendations

## 6.2 Diffusion Transformer Finetuning

Due to time constraints, we were unable to thoroughly test and finetune the diffusion transformer. With more time, we would experiment with:

1. Different noise schedules using [4] as a starting point
2. Learning rates

## 6.3 Model Architecture

Further experimentation with the model architecture is recommended in developing the final model. These experiments may include:

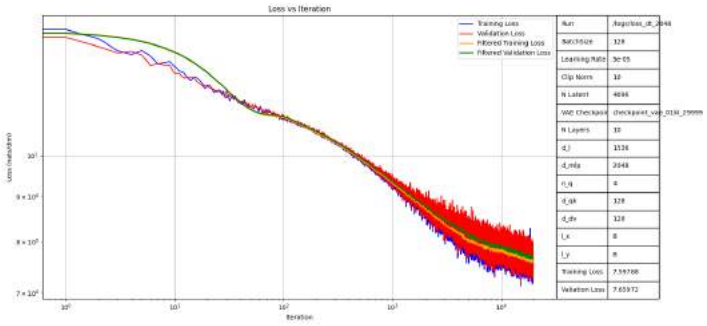
1. Replacing the VAE decoder with a second diffusion transformer
2. Replacing channel normalization in the VAE with batch normalization

## 6.4 Scaling

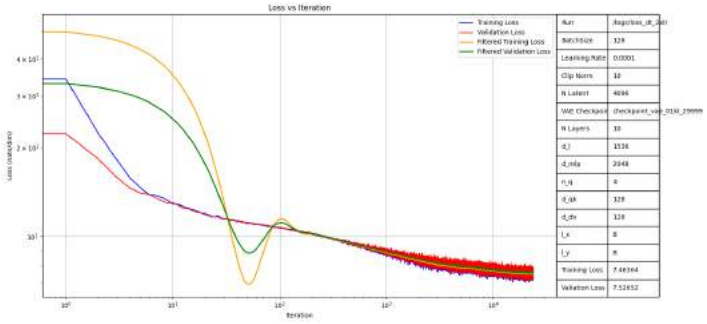
With more compute and storage, we would be able to make our models larger and train on larger datasets. This is essential for developing a performant model.

# 7 Deliverables

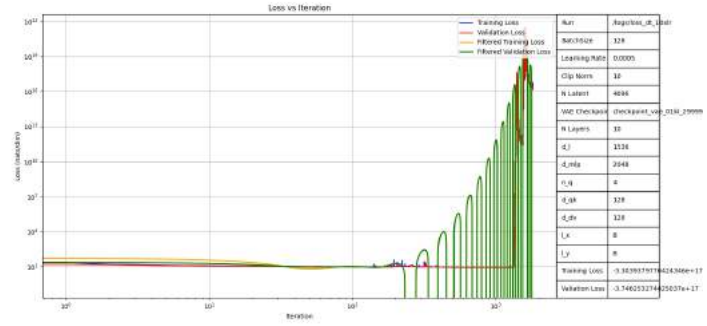
1. Github Repository
2. Training log files
3. Training log tracking sheet




(a) 5e-5 learning rate.

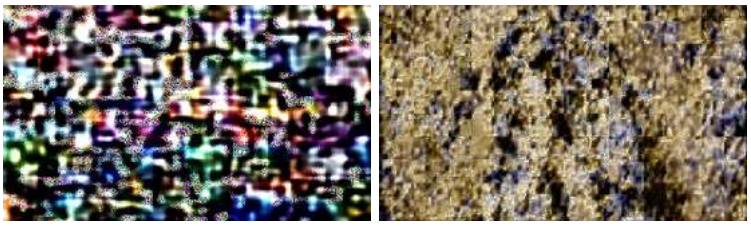



(b) 1e-4 learning rate.




(c) 5e-4 learning rate

Figure 24: Three diffusion training runs with increasing learning rates. The training becomes unstable when the learning rate is sent to 5e-4.



(a) 1.5GB (b) 4.5GB

Figure 25: A comparison of two different diffusion models by total model size and their respective generations.

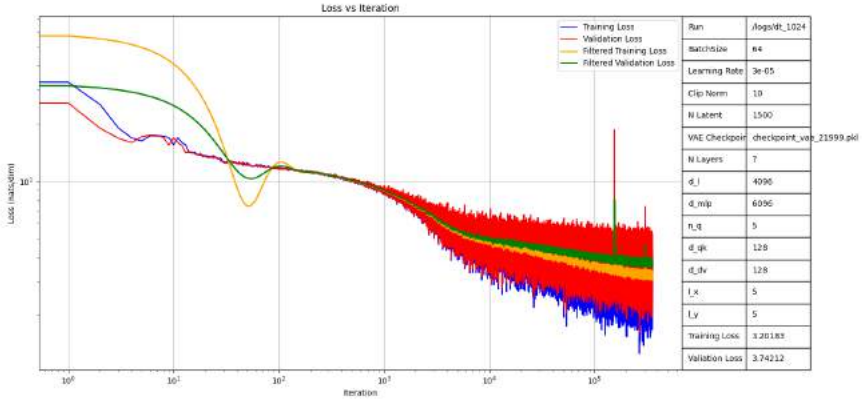



Figure 26: A log log plot of model loss vs iteration number. This diffusion model was trained with 1500 latent vectors over 5 days.

Figure 27: A series of generated videos using 5 encoded frames as context and generating the next 5. An artifact from the encoding process appears in all video frames.

# 8 Appendix

## 8.1 Key steps in self-attention

Self-attention consists of the following key steps:

1. **Input embedding.** The input sequence is broken up into smaller units (typically referred to as "patches" for images and videos and "tokens" for text). Each patch is assigned an embedding vector that is learned during training.

2. **Position embedding.** As patches are fed into a transformer as an unordered set, positional encodings are added to input embeddings to capture the absolute or relative positions of tokens in the patches. In our model specifically, we use rotary position embeddings (RoPE), which applies a rotation to the input embedding, where the amount of rotation is an integer multiple of the position. RoPE has been found to perform as well or better than other approaches [1] and has the advantages that:

    - Adding more patches to the end of a sequence does not change the embeddings of previous patches (in contrast to some relative positional encoding approaches)
    - Relative positional information is preserved by the angle between vectors (in contrast to absolute positional encoding approaches)

3. **Compute Query, Key, Value vectors.** For each patch in the input sequence, its query, key, and value vectors are computed by matrix multiplication with learned query, key, and value weight matrices. These vectors each represent the following:

    - Query vector (Q): the patch for which you want to compute attention scores
    - Key vector (K): the patches in the input sequence that you want to compare the query with
    - Value vector (V): information or features associated with each patch in the input sequence

4. **Compute attention weighting.** The pairwise similarity between each query and key is known as its attention score. It can be computed using cosine similarity (i.e. dot product). The level of similarity between query-key pairs is known as the attention weighting, and is computed as the softmax of the attention scores.

5. **Extract features with high attention.** The features that are relevant to the query vector are extracted by matrix multiplying the attention weighting with the value vectors.

Figure 28 visualizes these key steps.

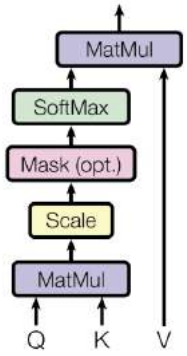



Figure 28: Scaled Dot-Product Attention. Adapted from [12].

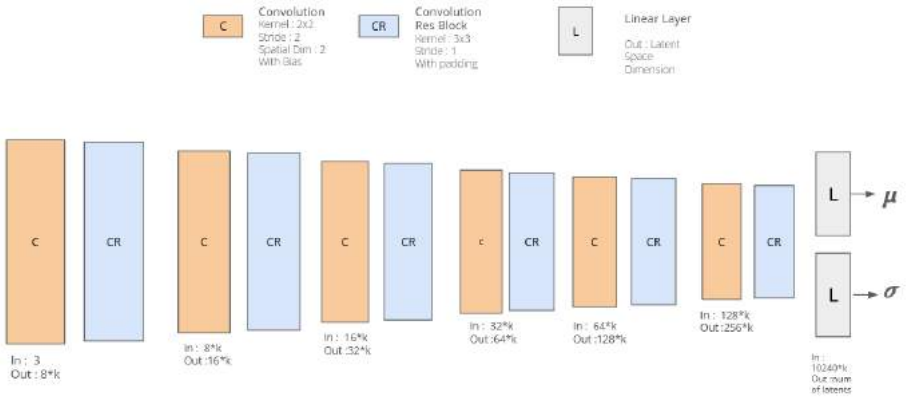



Figure 29: VAE encoder Neural Network representation

## 8.2 VAE : Model

The encoder network is represented in Figure 29. The decoder architecture follows a similar but inverted output. Note that the spatial dimensions in successive layers decrease however the number of channels increase. A training algorithm proposed by Kingma and Welling[10] is shown in Figure 30.

## 8.3 Probabilistic Models

Probabilistic models are based on the assumption that for a data set $X$ there is an underlying probability distribution $P(X)$ that describes each data point [10]. Probabilistic models output a probability distribution $P_\theta(X)$ that after training is

**Algorithm 1:** Stochastic optimization of the ELBO. Since noise originates from both the minibatch sampling and sampling of $p(\boldsymbol{\epsilon})$, this is a doubly stochastic optimization procedure. We also refer to this procedure as the *Auto-Encoding Variational Bayes* (AEVB) algorithm.

**Data:**

$\mathcal{D}$: Dataset

$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model

$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$: Generative model

**Result:**

$\boldsymbol{\theta}, \boldsymbol{\phi}$: Learned parameters

$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters

**while** *SGD not converged* **do**

$\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)

$\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)

Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$

Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer

**end**

Figure 30: Training for VAEs

close to the datas underlying distribution.

$$P_\theta(X) \approx P(X)$$

A subclass of probabilistic models are latent variable models that learn the distribution $P_\theta(X)$ through "hidden" or latent variables $z$ such that

$$P_\theta(X) = \int P_\theta(X, z)dz$$

These models offer added flexibility in specifying model constraints for data efficiency and interoperability, and have been demonstrated to outperform their deterministic counterparts in some contexts. Often, the barrier in these models is the inference step i.e finding $P_\theta(z|X)$ which is often intractable. A variety of approximate inference techniques have been proposed to overcome this. [10]

# References


[1] Stella Biderman et al. *Rotary Embeddings: A Relative Revolution*. blog.eleuther.ai/. [Online; accessed April 18, 2024]. 2021.

[2] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[3] Christopher P. Burgess et al. *Understanding disentangling in $\beta$-VAE*. 2018. arXiv: 1804.03599 [stat.ML].

[4] Ting Chen. *On the Importance of Noise Scheduling for Diffusion Models*. 2023. arXiv: 2301.10972 [cs.CV].

[5] Google Cloud. *Introduction to Cloud TPUs*. [Online; accessed April 18, 2024]. URL: https://cloud.google.com/tpu/docs/intro-to-tpu.

[6] Boris Dayma. *DALLE Mega - Training Journal*. [Online; accessed April 18, 2024]. URL: https://wandb.ai/dalle-mini/dalle-mini/reports/DALL-E-Mega-Training-Journal--VmlldzoxODMxMDI2.

[7] drhead. *The VAE used for Stable Diffusion 1.x/2.x and other models (KL-F8) has a critical flaw, probably due to bad training, that is holding back all models that use it (almost certainly including DALL-E 3)*. [Online; accessed April 18, 2024]. URL: https://old.reddit.com/r/StableDiffusion/comments/1ag5h5s/the_vae_used_for_stable_diffusion_1x2x_and_other/.

[8] Tero Karras et al. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV].

[9] Yoon Kim, Sam Wiseman, and Alexander M. Rush. *A Tutorial on Deep Latent Variable Models of Natural Language*. 2019. arXiv: 1812.06834 [cs.CL].

[10] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[11] Diederik P. Kingma et al. *Variational Diffusion Models*. 2023. arXiv: 2107.00630 [cs.LG].

[12] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[13] Ruoqi Wei et al. "Variations in Variational Autoencoders - A Comparative Evaluation". In: *IEEE Access* PP (Aug. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.3018151.

[14] Yuxin Wu and Kaiming He. *Group Normalization*. 2018. arXiv: 1803.08494 [cs.CV].

[15] Wilson Yan et al. *VideoGPT: Video Generation using VQ-VAE and Transformers*. 2021. arXiv: 2104.10157 [cs.CV].

[16] Ling Yang et al. *Diffusion Models: A Comprehensive Survey of Methods and Applications*. 2024. arXiv: 2209.00796 [cs.LG].

[17] Youtube. *UGC Dataset*. [Online; accessed April 18, 2024]. URL: https://media.withyoutube.com/.