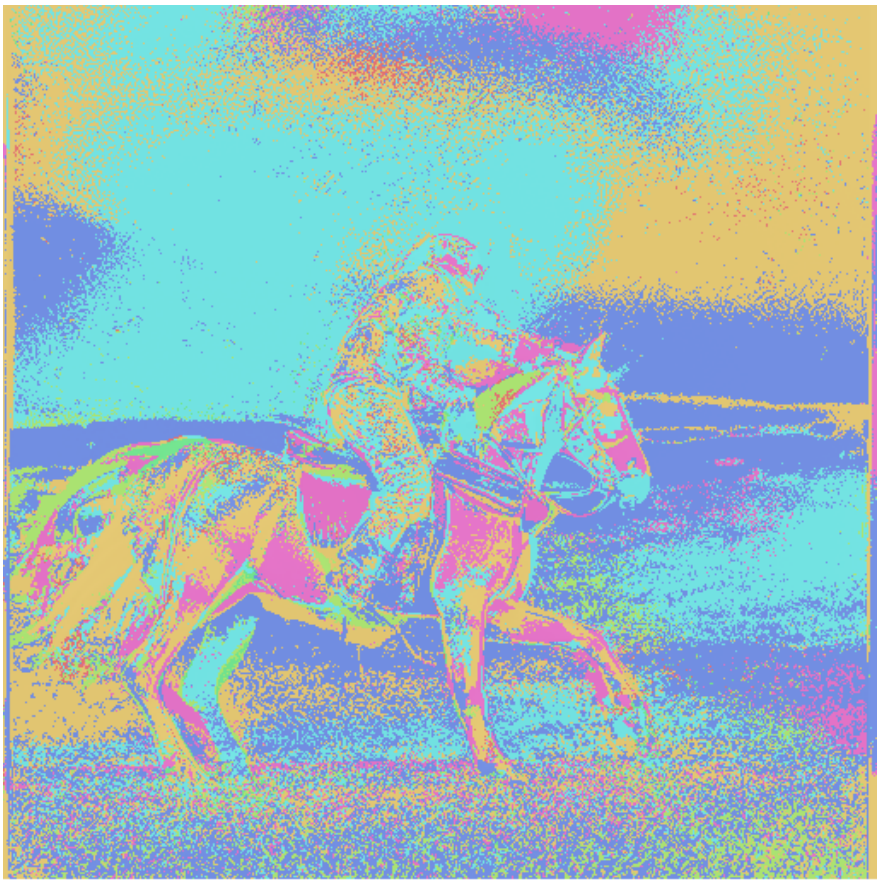


Controlling Diffusion

Input-Output Mapping of the Components of a Diffusion Model as a Potential Approach for Enhanced Model Control

Master's thesis in Complex adaptive systems

Philip Gard

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Controlling Diffusion

Input-Output Mapping of the Components of a Diffusion Model as a Potential Approach for Enhanced Model Control

Philip Gard

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Controlling Diffusion
Input-Output Mapping of the Components of a Diffusion Model as a Potential Approach for Enhanced Model Control
Philip Gard

Supervisors: Mats Granath, Physics and Hampus Linander, Mathematical Sciences
Examiner: Mats Granath, Physics

Master's Thesis 2023
Department of Physics
Division of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization showing the effect of perturbations on the prompt "a photograph of an astronaut riding a horse", the perturbation size was +15% in this image.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Controlling Diffusion
Input-Output Mapping of the Components of a Diffusion Model as a Potential Approach for Enhanced Model Control
Philip Gard
Department of Physics
Chalmers University of Technology

Abstract

In this thesis, the primary focus is the exploration and evaluation of input-output mappings within the components of diffusion models, extending conventional methods of control like prompt engineering. The objective is not to propose a definitive solution for control within diffusion models, but rather to probe the underlying mappings that drive these processes.
The project examines two main components: The attention maps within the CLIP model, and the input-output relationships in the diffusion process itself. By doing so, the intention is to increase understanding of the inherent complexity of the models and identify potential control opportunities.
In the intricate domain of diffusion models, the evaluation of input-output maps might not always offer a clear-cut measure of success. This project contributes by examining the interplay between various components of the model, as viewed through the lens of input-output mappings. Instead of presenting conclusive control methods, it examines these components, setting a possible path for further exploration.

Keywords: diffusion model, stable diffusion, CLIP, input-output mapping, attention map, saliency map.

Acknowledgements

I want to thank Mats Granath and Hampus Linander at Chalmers University of Technology for the instrumental guidance and support through this project. Computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) and the Swedish National Infrastructure for Computing (SNIC) at Chalmers Centre for Computational Science and Engineering (C3SE), partially funded by the Swedish Research Council through grant agreements no. 2022-06725 and no. 2018-05973. I would also like to acknowledge that this project would not have been possible without the open-source model Stable Diffusion by Runway, CompVis, and Stability AI, made available through Hugging Face.

Philip Gard, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BPE	Byte Pair Encoding
C3SE	Chalmers Centre for Computational Science and Engineering
CNN	Convolutional Neural Network
DDIM	Denoising Diffusion Implicit Model
DDPM	Denoising Diffusion Probabilistic Model
GAN	Generative Adversarial Networks
GLIDE	Text-conditional image generation
LDM	Latent Diffusion Models
LLM	Large Language Model
LMS	Linear Multistep Schedulers
ML	Machine learning
MSE	Mean Squared Error
NAISS	National Academic Infrastructure for Supercomputing in Sweden
NLP	Natural Language Processing
PE	Positional Encoding
PNDM	Pseudo Numerical Diffusion Methods
SDE	Stochastic differential equation
SNIC	Swedish National Infrastructure for Computing
VAE	Variational Autoencoder
ViT	The Vision Transformer

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

In recent years, the machine learning field has been revolutionized by the advent of very large models capable of undertaking a multitude of tasks. Notable examples include GPT-4, adept at text generation, and DALL-E 2, which excels in image synthesis. These models demonstrate an intriguing blend of scale and generality, setting new standards in areas such as natural language processing and computer vision.

However, their widespread use in practical applications where initially limited. While potentially powerful, the outputs of these models often require specialized knowledge for control and effective use.

While models like GPT-4 exhibit impressive capabilities, their direct application can be challenging, particularly in tasks like chatbot creation. For instance, unless properly prompted, these models tend to generate the bot's response and the anticipated user's next question. This highlights the complexity of controlling the model's output and has prompted the development of more refined approaches. An example of such is ChatGPT. This tool, built on GPT-4 or GPT-3, depending on the version, implements a response-to-answer structure that limits the model's responses, thus simplifying its use as a chatbot.

This problem, however, seems to persist in image generation models where the output of the models does not seem to be much more useful in commercial production compared to what an internet search might produce, as most commercial products appear to still employ largely the same methods of asset generation as before. This seemingly implies that the image generation models are at most used as a step in this generation, likely concept image generation, which internet searches are already relatively good at.

1.2 Aim

This project aims to explore possible paths for increasing control of the output from large image generation models based on Diffusion through input-output mappings, i.e., Determining which parts of input have which effects on the output. The focus is on the Stable Diffusion models created by Stability AI.

1.3 Problem description

1.3.1 Limitations

In the interest of time, this project has been limited to mostly using pre-trained models to decrease the need for hyperparameter tuning. This project has also not explored fine-tuning of the methods evaluated unless strictly necessary for the evaluation, i.e., the project has focused on proof-of-concept.

1.3.2 Specification of issues under investigation

This master thesis aims to handle the following questions:

1. Can input-output mappings be constructed for the modules, i.e., The sub-models of a large ML-model, specifically the Stable Diffusion model?
2. Can input-output mappings be leveraged to gain insights into possible control opportunities?
3. Can these insights be extended to actual proof-of-concept control methods?

1.4 Ethical Considerations

Using diffusion models in image generation introduces ethical issues: Potential risk of misuse and impact on artistic professionals.

1.4.1 Misuse Risk

High-fidelity images generated through diffusion models hold some potential for harmful content production, for example, misinformation or violent content. Avoiding this might require robust content moderation or, more likely, algorithmic filtering systems.

1.4.2 Impact on Artistic Professionals

Diffusion models can both aid and challenge artists. They could enhance creativity but potentially undermine the work of artists specializing in traditional tools. This needs to be accounted for while not overlooking the potential creativity boost for all individuals who stand to benefit from the employment of more powerful tools, like AI.

2
Theory

The basic concept behind neural networks is a model consisting of a large number of tunable parameters, employing matrices to create a structure reminiscent of biological neural networks. This, combined with optimization methods, creates a model capable of generalized learning [1].
Some extensions on this base model on the level of the artificial neuron exist, for example, Convolutional Neural Networks (CNNs) [2]. However, many models simply combine these basic layers and the extended layers, in novel and innovative architectures, training methods, and interference procedures, for example, Generative Adversarial Networks (GANs), which use a distinctive adversarial training process [3].
Many of the models that have revolutionized the machine learning field in recent years represent an extra level of abstraction. These model's architectures employ multiple models, which themselves are based on many types of layers. For example, the Stable Diffusion model includes text embedding networks based on transformers as well as Autoencoders [4].
The following sections provide a description of each of the parts that make up the Stable Diffusion model, which is heavily utilized throughout this project, with the only exception consisting of the CLIP image encoder (which is not directly a part of the Stable Diffusion model).

2.1 Overview and Context

This section walks through the Stable Diffusion architecture at a high level. This can be viewed as the context for the models treated in this chapter.

2.1.1 Stable Diffusion architectural overview

The core of Stable Diffusion is the denoising model, i.e., the (diffusion) model that turns noise into an image. In the case of Stable Diffusion, this is referred to as the "Denoising U-Net". Stable Diffusion does not operate on images directly but instead on a lower-dimensional representation of an image. The model that turns this representation into an image is the "VAE decoder". Lastly, the denoising model is prompt-directed, i.e., the model uses a vector representation of a prompt to direct the image generation. The "CLIP text encoder" (a transformer model) produces this prompt vector representation.

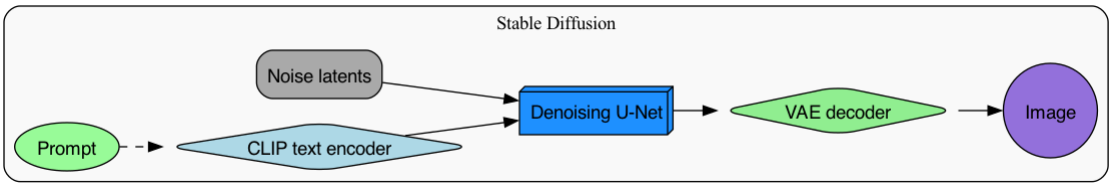


Figure 2.1: A simplified overview of the Stable Diffusion architecture. "Prompt" is the main input, and "Noise latents" are the secondary input, often represented as a seed. The "CLIP text encoder" is the model that converts a prompt into a vector, "Denoising U-Net" is the actual image generation model, and the "VAE decoder" is the model that turns the vector representation of an image into a pixel image

2.2 Theoretical framework

The components in the Stable Diffusion model can be viewed from the base theoretical framework provided by Autoencoders, i.e., models or processes representing encoders and decoders.

2.2.1 Autoencoders

Autoencoders are a type of neural network designed for data compression and noise reduction. Their architecture consists of two key components: an encoder and a decoder.
The encoder function, denoted as f, takes an input x, converting it into a compressed latent representation, h, i.e., $h = f(x)$. This latent space encapsulates the essential features of the input while shrinking the dimensionality.
The decoder function, g, reconstructs the input data from the latent space. It works by mapping h back into the original high-dimensional input space, denoted as x'. Thus, the reconstructed input is given by $x' = g(h)$.
The goal during training is to minimize the difference between the input data x and the reconstructed data x', i.e., $x \approx x'$. This is achieved using a loss function, typically the Mean Squared Error (MSE), which measures the average squared differences between x and x'. This process results in a model that can compress x into h (using the encoder) and then reconstruct an approximation of x, i.e., x' (using the decoder) [5].

2.3 Converting prompts into Text embeddings

The Stable Diffusion model employs text embeddings to direct the Diffusion process, i.e., allowing prompt-directed image generation.

2.3.1 Tokenization

Tokenization is the first step in the processing pipeline of Natural Language Processing (NLP). It entails dissecting a text string into a sequence of individual fragments

represented as an integer, referred to as "tokens". The granularity of these tokens can vary, spanning from single characters to whole words or even larger chunks of text, depending on the specifics of the tokenization approach.
The goal of the tokenizer is to represent any text with as few distinct tokens as possible in order to minimize the dimensionality of the vocabulary that is used in the next step of the NLP pipeline, i.e., the Word Embeddings, while at the same time capturing the semantic structure inherent in words and phrases.
There are two 'naive' approaches to tokenization: character-level and word-level. Character-level tokenization treats every character as a token, effectively minimizing the vocabulary size, which is beneficial for computational efficiency. However, it falls short in capturing semantic structure, i.e., part of the information contained in the text [6].
Word-level tokenization, on the other hand, treats every word as a token, preserving semantic structure and information. This method, however, results in a large vocabulary. For example, in languages where words can have many forms, such as English, word-level tokenization ends up treating "run", "runs", "running", and "ran" all as different tokens, thus increasing the size of the vocabulary.
A more practical middle-of-the-road approach is subword-level tokenization, i.e., breaking down words into frequently occurring subwords. This approach balances minimizing the vocabulary and capturing the semantic structure. A common method for achieving this is Byte Pair Encoding (BPE), a technique used in the CLIP model. BPE begins with a vocabulary of individual characters. It then iteratively merges the most frequent adjacent character pairs in the dataset to form new symbols, thereby extending the vocabulary. This process continues until a specified vocabulary limit (a hyperparameter) is reached. BPE's strength lies in its ability to handle out-of-vocabulary words by breaking them down into recognizable subword units, efficiently capturing semantic information while managing vocabulary size [7].

2.3.2 Word Embedding

After tokenization, the next critical step in the NLP pipeline is word embedding, a technique that converts discrete tokens into a continuous vector representation. This process involves mapping each token, represented by a unique integer ID from the tokenization step (see 2.3.1 above), into a high-dimensional vector space [8].
Word embeddings are created using an embedding layer, which operates as a lookup table that links unique integer IDs to dense vectors. These vectors are initially filled with random values. During training, these values are fine-tuned to reduce a cost function that measures how well the model predicts contextual words for a given target word. The better the model's predictions, the more effectively the embeddings capture the relationships between words. As a result, semantically similar words end up with similar embeddings, reflecting their contextual similarities [9].

2.3.3 Positional Encoding

The next step, after obtaining the Word Embeddings, is to add the Positional Encoding (PE). Positional Encoding is a way to encode the information about where

a word is in a text. This is crucial, as a word's position significantly influences the semantic information in a text.
The Positional Encoding scheme addresses a fundamental limitation in transformer models: their inability to inherently capture the order of words due to the self-attention mechanism (2.3.4). It does so by creating a set of vectors representing each word's position in the sequence according to 2.1.

A frequent method of Positional Encoding is to use sinusoidal functions, which are commonly defined as:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \end{aligned} \tag{2.1}$$

pos is the position of the word, or token, in the sequence of words that is the text, and i is the dimension in the embedding vector. That is, each dimension of the positional encoding corresponds to a sinusoid with a different frequency. Here d_{model} is the dimension of the model [10].

Positional Encoding is added directly to the word embeddings, resulting in encoded vectors that carry both the semantic meaning of the words and their positional information. This amalgamated information becomes the input for the subsequent layers of the model.
This technique allows the model to capture a unique encoding for each position and a relative representation of the distance between different positions in the sequence. The model, hence, becomes capable of recognizing patterns based on the order of words, a critical requirement for understanding and generating human language [10].

2.3.4 Attention

The main part of the Transformer architecture is the Attention mechanism. Attention, in the context of neural networks, refers to the model's ability to focus on specific parts of the input when generating the output. It is inspired by human attention, where we focus more on certain aspects of our environment while paying less attention to others. In transformers, attention is commonly computed using the Scaled Dot-Product Attention mechanism. This process involves three main components: Query (Q), Key (K), and Value (V), each of which corresponds to different aspects of the input data. These are typically derived from the input sequence through a linear transformation, a trainable part of the model [10].

The idea of the Query, Key, and Value can be understood from the context of search:

Term	Analogy in Search	Transformer's Meaning
Query (Q)	Search Query	Current context used to select relevant input
Key (K)	Search Keywords	Elements in the input sequence used for matching
Value (V)	Search Results	Information pieces used to build the output

Table 2.1: Explanation of Query, Key, and Value

The Query (Q) and Key (K) components are used to compute the attention scores. This involves taking the dot product of Q and K, scaling it by the square root of their dimension size, and applying a Softmax function. This results in a distribution that indicates the amount of "attention" each word in the input should receive when generating a new word in the output sequence. The Value (V) component corresponds to the actual content associated with each word. The computed attention distribution is then multiplied by V, essentially deciding the information to pass onto the next layer based on the calculated attention.

To summarize, the attention mechanism can be represented as:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.2}$$

In this equation, Q, K, and V are matrices representing the Query, Key, and Value vectors for all words in the input sequence. The trainable parameters in this attention mechanism live within the linear transformations that derive Q, K, and V from the input sequence [10].

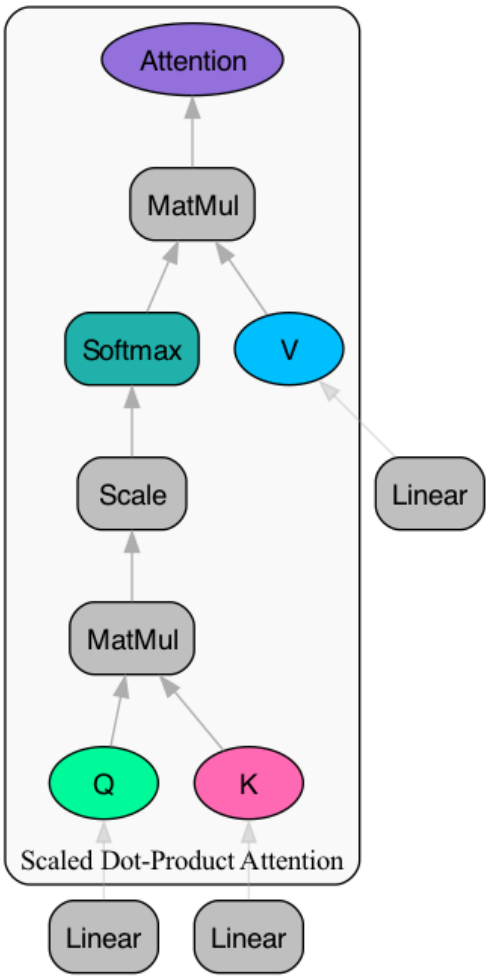


Figure 2.2: A overview of the Scaled Dot-Product Attention operation, the first "MatMul" represents the matrix multiplication QK^T and "Scale" is the division be $\sqrt{d_k}$. The graph was displayed through Graphviz.

2.3.5 Multi-Head Attention

The transformer extends the concept of attention into what is known as Multi-Head Attention. Operating at the same level in the network as traditional attention, this enhanced mechanism allows the model to generate multiple "interpretations" of the input, focusing on different words for each interpretation.

In Multi-Head Attention, the inputs (query, key, and value) are first linearly transformed into multiple sets. Each of these sets is then fed into its own Scaled Dot-Product Attention mechanism, creating multiple "heads". The outputs of all the attention heads are then concatenated and linearly transformed to result in the final output [10].

This approach enhances the capacity of the model to focus on different parts of the input simultaneously, thereby capturing a more comprehensive understanding of the input data. The more attention heads, the greater the number of interpretations, leading to a more nuanced understanding of the sequence.

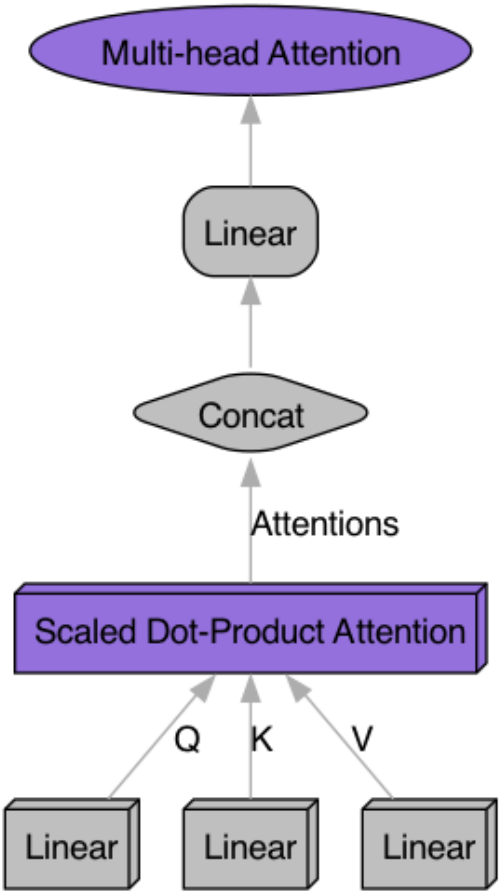


Figure 2.3: A overview of the Multi-Head Attention mechanism the "3D boxes" represents a number of "heads", "Concat" is the concatenate operation. This was displayed through Graphviz.

2.3.6 Attention Maps

Attention maps portray the attention weights between tokens in a specific input sequence, offering a detailed look into the internal decision-making process within a transformer model. These maps are presented as square matrices, Attention $\text{Map}_{i,j}$, with indices i and j denoting the positions of query and key tokens, respectively, for a particular input sequence [10].

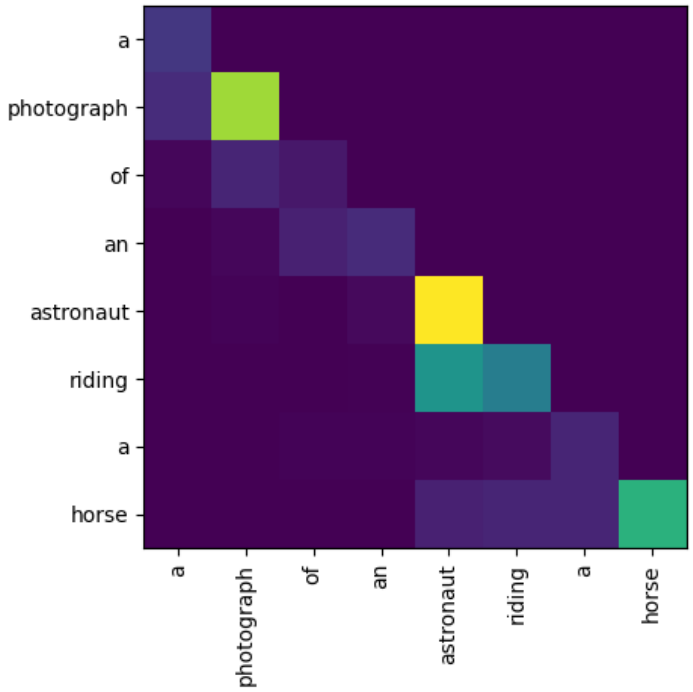


Figure 2.4: This image displays an example of an Attention Map for a transformer, in this case, the indices i and j correspond to the words, or tokens: "a", "photograph", "of", "an", "astronaut", "riding", "a", "horse". This specific example is a sub-set of an Attention Map from the transformer model CLIP

However, interpreting attention maps should be approached with care. High attention scores may not necessarily equate to a significant influence on the model's final output, as the actual impact of these attention scores depends on the intricate network dynamics [11].

2.3.7 Transformer

The transformer, a significant architecture first introduced in the paper "Attention is All You Need" [10], ingeniously assembles the previously discussed components - tokenization, word embeddings, positional encoding, and attention mechanisms - into a model for sequence processing.
The transformer architecture consists of an encoder and a decoder, each composed of multiple identical layers stacked on top of each other. Each of these layers in the encoder contains two main sub-layers: a Multi-Head Attention mechanism and a fully connected feed-forward network. Around each of these sub-layers, a residual connection is employed, followed by layer normalization i.e., shifts the activations within the layer to have a mean of zero and a standard deviation of one [10].
Initially, the input sequence is tokenized (see 2.3.1), and each token is embedded into a high-dimensional vector using an embedding layer (see 2.3.2). Positional encoding is then added to these word embeddings to incorporate the order of the words (see 2.3.3).
These encoded vectors are passed into the encoder, the gray box on the left in figure

2.5, which uses the Multi-Head Attention mechanism to establish a weighted representation of the input sequence, considering the interaction of each word with every other word. The feed-forward network processes these weighted representations independently to produce the encoder output.

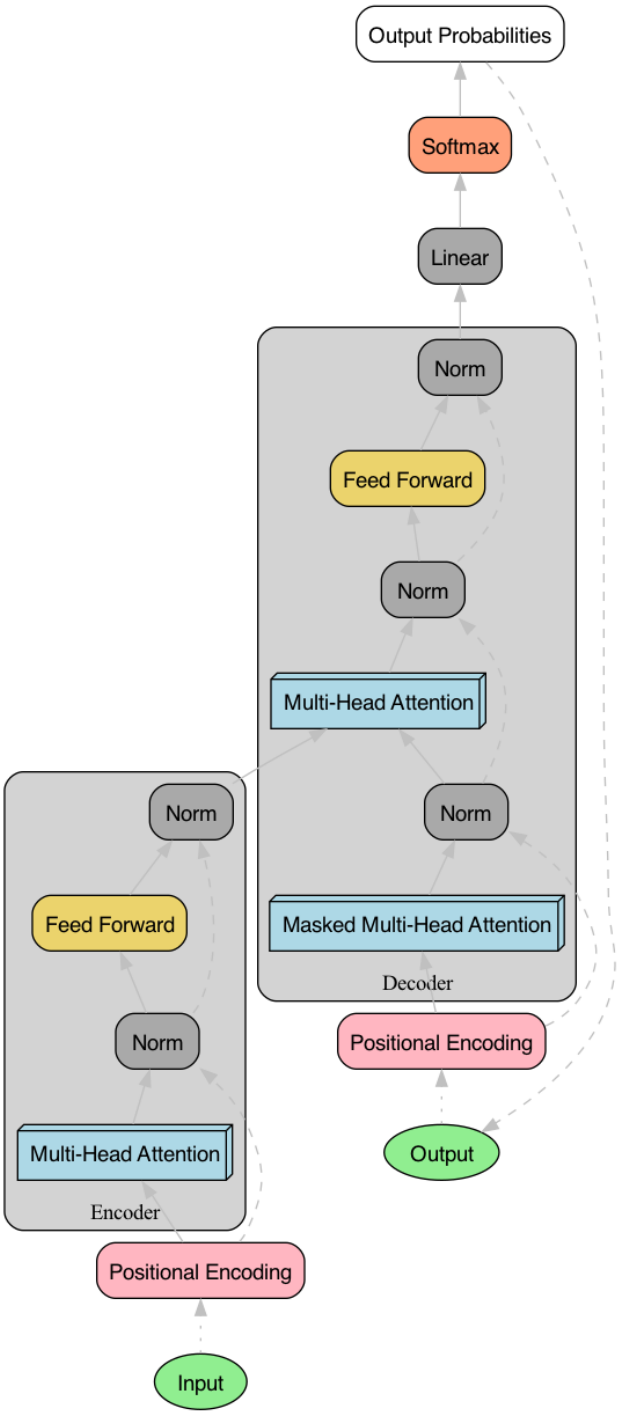


Figure 2.5: A overview of the Transformer architecture. "Norm" corresponds to layer normalization, and "Output Probabilities" is the output of the decoder part of the model in the context of a text generator; this is commonly an index to a token, or word.

The decoder, the gray box on the right in figure 2.5, in a transformer model, includes an additional sub-layer compared to the encoder. This sub-layer uses attention, specifically multi-head attention, on the encoder's output, enabling tasks that require correlation between sequences, such as translation. The final decoder layer's output is then converted into output probabilities for each token in the target vocabulary. Unlike RNNs and LSTMs, Transformers are particularly effective at capturing long-range dependencies in sequence data due to their comprehensive utilization of attention mechanisms. This makes them a robust and powerful approach for sequence-to-sequence tasks [10].

2.3.8 Vision Transformer

Much like how traditional transformers handle sequential data, such as text, the Vision Transformer (ViT) deals with visual data and images. Its distinctive feature is treating an image as a sequence of patches, much like how words form a sequence in a sentence. This approach helps to identify and utilize the inherent structure in the visual data, making the Vision Transformer a highly effective model for image classification and related tasks.
An image, in the context of a ViT, is divided into a grid of non-overlapping patches analogous to tokens in text processing. These patches are then linearly embedded to form a sequence of vectors (analogous to how word embeddings form prompt embedding). Following the practice of text processing, a specific "start token" is commonly appended to the beginning of this sequence. This token provides context and some positional information for the whole image, similar to how it is used in text sequences [12].
Positional embeddings, analogous to the ones in the transformer model for NLP tasks, are also added to each patch embedding to encode their relative positions in the image grid. The combination of patch embeddings, positional embeddings, and the "start token" creates the final sequence of vectors that forms the input to the transformer.
The transformer layers in the Vision Transformer model are the same as those in the classic transformer: multi-head self-attention and feed-forward neural networks, coupled with layer normalization and residual connections. The model, through these layers, learns to relate patches not just with their immediate neighbors but with distant ones as well, ultimately capturing global context of an image [12].
The ViT serves as a compelling exemplification of how transformer architectures can effectively be applied to computer vision. By reframing image analysis in terms of sequential data processing. The ViT ushers in a new avenue of image interpretation that prioritizes global context and long-range interdependencies. This move away from the localized focus of traditional CNNs enhances the model's capability to discern complex, interwoven visual narratives within an image [12] [13].

2.3.9 CLIP

The CLIP model is a widely used model for connecting text and images through embeddings. The concept is similar to Autoencoders with the fundamental differ-

ence being that CLIP is not trained to reproduce a text prompt from a image, or vice versa. Instead, the objective function consists of minimizing the cosine similarity between non-matching text-image pairs while maximizing the cosine similarity between matching text-image pairs, thus the model employs two encoders. The models used to achieve this are one Transformer for the text-encoder and a Vision Transformer for the image-encoder [14].

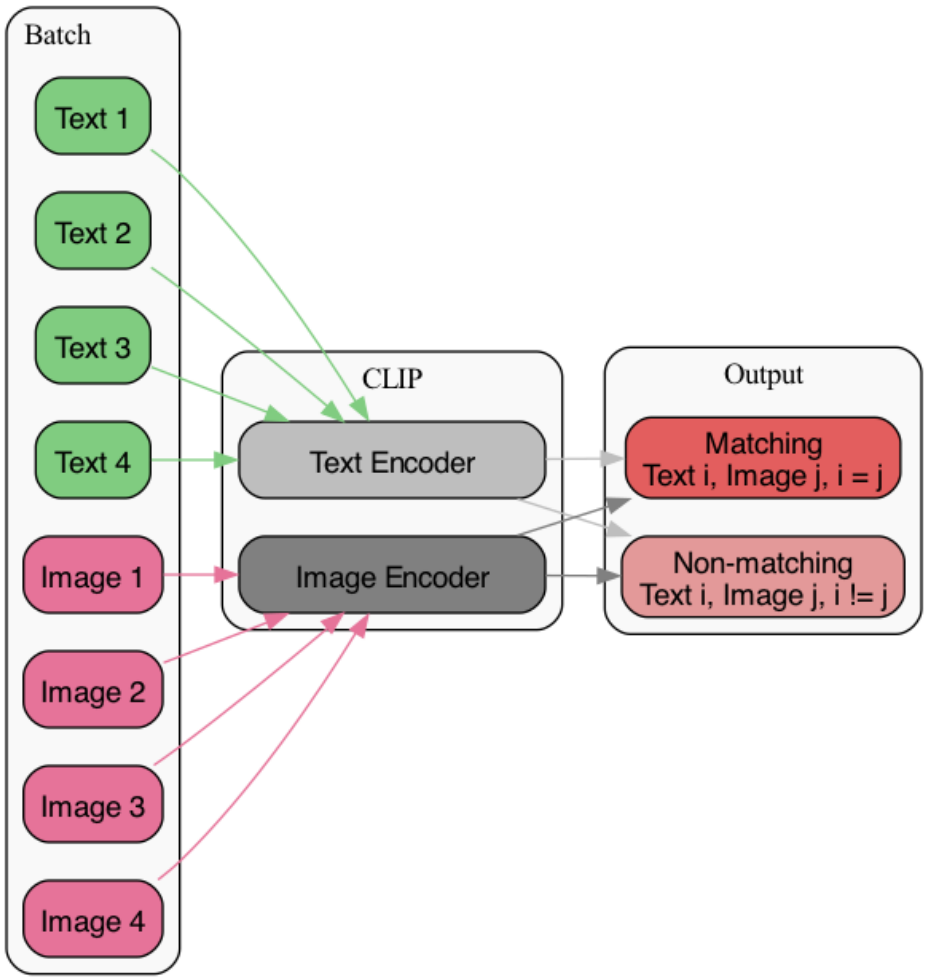


Figure 2.6: A overview of the unique training process that results in two encoders used in the CLIP model. Displayed through Graphviz.

The cosine similarity between a text embedding, t, and an image embedding, i, can be computed as:

$$\text{Cosine Similarity}(t, i) = \frac{t \cdot i}{||t||_2 \times ||i||_2} \tag{2.3}$$

Where $t \cdot i$ is the dot product of the text and image embeddings, and $||t||_2$ and $||i||_2$ are their respective L2 norms.

2.4 Diffusion Models

The core of the Stable Diffusion model is the Diffusion module. This is the module that causes the apparent "creativity" displayed by the model.

2.4.1 Base Diffusion Model

Diffusion models can be understood as stochastic iterative versions of Autoencoders. The "encoder" in a diffusion model is represented by a stochastic differential equation (SDE) which progressively corrupts the data with noise, driving it towards a simpler distribution (usually a standard multivariate Gaussian). This can be thought of as a noisy generalization of an Autoencoder's encoder mapping the data into a latent space.

The "encoding" process in a diffusion model is mathematically represented by a stochastic differential equation (SDE) that resembles Brownian motion. The SDE is given by:

$$dx_t = \sqrt{2\alpha}\, dW_t, \tag{2.4}$$

Where x_t represents the state of the data at time t, α signifies the timestep and W_t is a standard Wiener process. At each timestep, Gaussian noise is added to the data, gradually driving it towards a simpler distribution, typically a standard multivariate Gaussian [15].

The "decoder" in a diffusion model corresponds to the reverse process that starts from samples drawn from the simpler distribution and progressively removes the added noise, reconstructing the original data. This mirrors an Autoencoder's decoder, which attempts to map from the latent space back to the original data space [15].

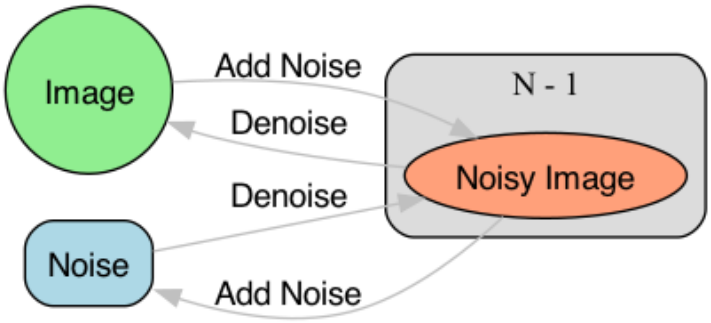


Figure 2.7: An overview of the training process that is core to Diffusion models. "Image" corresponds to the training data or the reconstructed image depending on if noise is added or removed, corresponding to training or interference, and "N" is the number of iterations of adding or removing noise. Displayed through Graphviz.

However, unlike a standard Autoencoder, the reverse or "decoding" process in a diffusion model is not deterministic. It involves learning a Markov transition kernel that captures the reverse of the noise addition process described above.

In mathematical terms, this involves learning a reverse-time SDE of the form $dx_t = -\sqrt{(2\alpha)}dW_t + \beta dt$, where β is a function that is learned from the data. This learned function serves as a "denoising" operation that removes noise from the data. Training a diffusion model involves optimizing this function to minimize the difference between the original and reconstructed data, similar to minimizing the reconstruction error in an Autoencoder.

2.4.2 Schedulers

In the context of diffusion models, schedulers play a crucial role in determining how the noise is added and removed over the course of the diffusion process. Essentially, they govern the evolution of the noise variance over time, which directly influences the behaviour of the stochastic differential equations (SDEs).
A noise schedule is a sequence of noise levels (α_t) at each timestep. During the forward or "encoding" process, the scheduler determines how much noise is introduced at each timestep. This is usually done in such a way that the variance of the noise increases over time, moving the data closer to the target distribution (often a standard multivariate Gaussian) [16].
During the reverse or "decoding" process, the scheduler dictates the "denoising" process. It governs how much of the noise is subtracted at each timestep, guiding the data's trajectory back to the original distribution. In this case, the noise levels typically decrease over time.
Schedulers are usually chosen based on empirical performance. There is a wide range of possible schedules, from linear ones where the variance increases or decreases linearly with time to more complex non-linear schedules. The optimal schedule depends on the specific dataset and task at hand, and discovering good noise schedules is an active area of research in diffusion models.

2.4.2.1 Denoising Diffusion Implicit Models

The widely employed Denoising Diffusion Probabilistic Models (DDPMs) operate as standard diffusion processes in the domain of diffusion models, following a noise schedule set by the scheduler. However, Denoising Diffusion Implicit Models (DDIMs) offer a key alternative that extends the diffusion process beyond the standard Markovian (depending only on the preceding state) framework.
DDIMs address the computational inefficiencies of DDPMs by employing non-Markovian (depending on more than the preceding i.e. the path matters) diffusion processes. Both models share similar training procedures but diverge in their generative processes and diffusion mechanisms [17].
In contrast to DDPMs, where the generative process is the reverse of a Markovian diffusion process, DDIMs facilitate a deterministic generative process, significantly improving sample speed.
The extended diffusion process in DDIMs involves designing reverse generative Markov chains. This results in a training objective identical to DDPMs, but it permits a wider array of generative models. These non-Markovian diffusion processes generate "short" Markov chains, improving sample efficiency with only a minor compromise in sample quality [17].

Finally, DDIMs enhance sample generation quality compared to DDPMs, particularly under accelerated sampling conditions. DDIMs also incorporate a "consistency" property, enabling semantically meaningful image interpolation capabilities [17].

2.4.2.2 Linear Multistep Schedulers

Scheduling of noise levels is pivotal in controlling diffusion processes, especially in Denoising Diffusion Probabilistic Models (DDPMs). To enhance this, Linear Multistep Schedulers (LMS) have been designed, offering a nuanced approach to noise management.
In stark contrast to the conventional linear or straightforward schedules, LMS employs a more flexible approach, enabling the tailoring of the β or noise values used at each step of the diffusion process. This adaptability ensures the potential to optimize noise schedules according to the specific requirements of a given task or dataset [18].
The critical parameters under LMS control include the starting and ending noise values, the total number of training timesteps, and the type of prediction. The ability to manipulate these parameters allows for an informed and strategic approach to noise management in DDPMs, contributing to enhanced model performance.
In essence, the LMS offers a more dynamic and customizable solution for noise scheduling within DDPMs, serving as an instrumental tool for potential improvements in model efficiency and sample generation quality [18].

2.4.2.3 Pseudo Numerical Methods

Pseudo Numerical Methods for Diffusion Models (PNDMs) is a novel approach to further enhance the effectiveness and efficiency of generative diffusion models. This methodology operates under a unique perspective, viewing Denoising Diffusion Probabilistic Models (DDPMs) as a process of solving differential equations on manifolds i.e., solving the differential equation locally.
Accelerating DDPMs traditionally involves adjusting the variance schedule or modifying the denoising equation. However, these changes often lead to a compromise in sample quality and could potentially introduce new noise at high speedup rates, thereby limiting their practical utility. To address this, PNDMs pioneer the introduction of pseudo-numerical methods specifically designed for diffusion models [19].
Pseudo-numerical methods represent a variant of classical numerical methods adapted to operate efficiently on manifolds. By incorporating these methods, PNDMs can markedly speed up the inference process while maintaining high sample quality. PNDMs have indeed demonstrated their potential by producing superior-quality synthetic images in significantly fewer steps compared to Denoising Diffusion Implicit Models (DDIMs).
The application of PNDMs is versatile and accommodates a variety of DDIM-like models. These include DDIM, Score-Based Generative Modeling through Stochastic Differential Equations (SDE), and Improved Denoising Diffusion Probabilistic Models (DDPM). This adaptable framework provides an array of choices for the numerical method, the noise addition schedule, and the neural network used for fitting

noise, facilitating adaptability to diverse tasks and datasets [19].
In conclusion, PNDMs represent a promising direction in the domain of diffusion models. They offer an efficient and effective approach to sample generation while maintaining a high level of quality. Their flexible framework and seamless integration with other libraries make them a versatile tool in the field of generative modeling.

2.4.3 GLIDE

An important component in the diffusion modules in the Stable Diffusion model is text-conditional image generation i.e. directing the image denoising with text. This was introduced in the GLIDE model [20], and is simply done by giving the diffusion decoder a vector representing the caption (text embedding) of the image during the training. After training this produces a decoder that will produce an image from noise consistent with the text embedding it is given alongside. The Stable Diffusion model uses the CLIP encoder to produce this text embedding from the prompt.

2.5 Efficient image representations

The Stable Diffusion employs a latent image space enabled by a Variational Autoencoder. This is the main difference between the Stable Diffusion model and, for example, DALL-E-2 and the reason this model can run on relatively light hardware.

2.5.1 VAE

Variational Autoencoders (VAEs) represent an intriguing twist to the conventional autoencoder structure. While traditional autoencoders encode input data into a deterministic latent representation, Variational Autoencoders, in contrast, model the input data as a probability distribution within the latent space. This fundamental shift enables VAEs to generate new data instances that closely resemble the input data.
In VAEs, the encoder doesn't directly produce a single latent vector. Instead, it outputs parameters of a Gaussian distribution: a mean vector (μ) and a standard deviation vector (σ). The *reparameterization trick* is then employed, using an auxiliary noise variable ϵ sampled from a standard normal distribution to derive the latent vector: $h = \mu + \sigma * \epsilon$ [21].
One significant advantage of VAEs over traditional autoencoders is their ability to model complex and multi-modal data distributions. By representing the latent space probabilistically, VAEs can handle a broader range of data distributions, which often results in better representations of the input data. Moreover, VAEs can generate new instances of data that bear a strong resemblance to the input data, thus serving as a potent tool for generative tasks.
Despite these advantages, VAEs do have their limitations. The generated data from VAEs often exhibit a certain "blurriness". This stems from the decoder's expectation operation, which averages over the sampled latent variables, leading to less sharp reconstructions. Additionally, the computational cost of training VAEs is higher

than that of traditional Autoencoders due to the need for Monte Carlo sampling to estimate the gradients during backpropagation.

2.5.2 Latent Diffusion Models

The main thing that separates Stable Diffusion from other large and effective diffusion models like DALL-E-2 is that the diffusion is done in a highly compressed space. This is achieved by training the diffusion model with a vector representation of an image (latent vector) instead of an actual full or close to the full-resolution image; this is the concept behind Latent Diffusion Models (LDMs). This compressed latent space is commonly achieved by training a Autoencoder on a data set of images [22].

2.6 U-Net

The U-Net is a specific type of convolutional neural network (CNN) that follows the architectural principles of autoencoders, designed with a symmetric "U" shaped structure. U-Nets adopt the encoding-decoding philosophy of autoencoders, which has led to their successful application in various image processing tasks, notably in biomedical image segmentation [23]. In U-Nets, the encoding (contracting) pathway is responsible for capturing the context in the image, with a series of convolutional and max-pooling layers that progressively reduce the spatial dimensions of the input. The decoding (expanding) pathway, consisting of up-convolutional layers, serves to increase the spatial dimensions while reconstructing the original image's details. A distinguishing feature of U-Nets is the presence of skip connections between the encoding and decoding pathways, reminiscent of autoencoders' bottleneck design. These lateral connections pass detailed spatial information directly from the encoding path to the decoding path, allowing high-level and low-level features to be fused. This mechanism enables U-Nets to better localize and delineate features of interest, making the architecture highly effective for tasks that require precise spatial information.

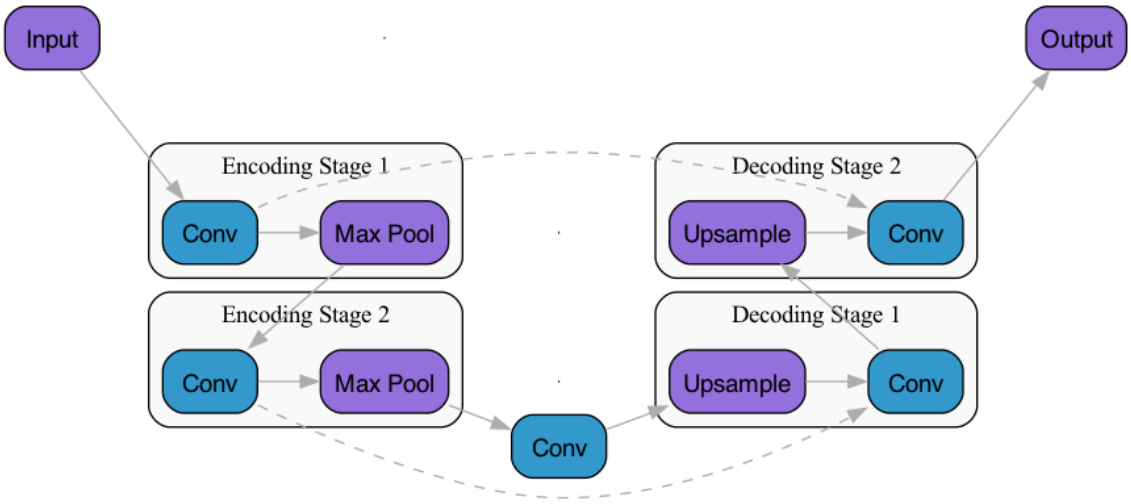


Figure 2.8: A overview of the U-Net architecture. Displayed through Graphviz.

In the context of diffusion models, U-Nets offer several compelling advantages. Their ability to handle image noise, retain spatial information, and produce highly detailed and coherent outputs aligns with the requirements of denoising functions within diffusion models. As such, U-Nets have found application in this domain, further illustrating the versatility and robustness of their autoencoder-inspired design (The U-Net for Diffusion models was introduced in [24]).

2.7 The Stable Diffusion Model

Stable Diffusion is one of a few large diffusion models; it is capable of generating high-quality images. Unlike other large diffusion models, for example, DALL-E-2 Stable Diffusion is open-source and can run on relatively light hardware. Stable Diffusion is an amalgamation of the concepts covered in the preceding sections, including the CLIP text encoder (see 2.3.9), Text-conditional image generation (see 2.4.3), VAEs (see 2.5.1), and LDMs (see 2.5.2). The model initiates by employing the CLIP text encoder (a transformer, see 2.3.7) to obtain the text embedding, which serves as a vector representation encapsulating the semantic information embedded in the text prompt [4]. Following the text embedding, the model's core component, the diffusion model. The diffusion model is of a U-Net architecture but includes connections that add the text embeddings at every encoding and decoding stage (see 2.6). However, rather than operating directly on pixel-level image data, the U-Net operates on a latent image representation, i.e., an LDM. This latent image representation is produced through a VAE, which during interference, decodes the latent image representation into the pixel space. During training, VAE compresses the high-dimensional images of the training dataset into a lower-dimensional latent space, enabling the model to work with more abstract and dense representations of the input data. This results in the relatively low computational requirements of the model [4]. Likely because the model is open-source, there is a large number of features and applications that have been developed for this model, for example, a large number of schedulers and pipelines, i.e., complete model flows, for example, image-to-image [25] [26].

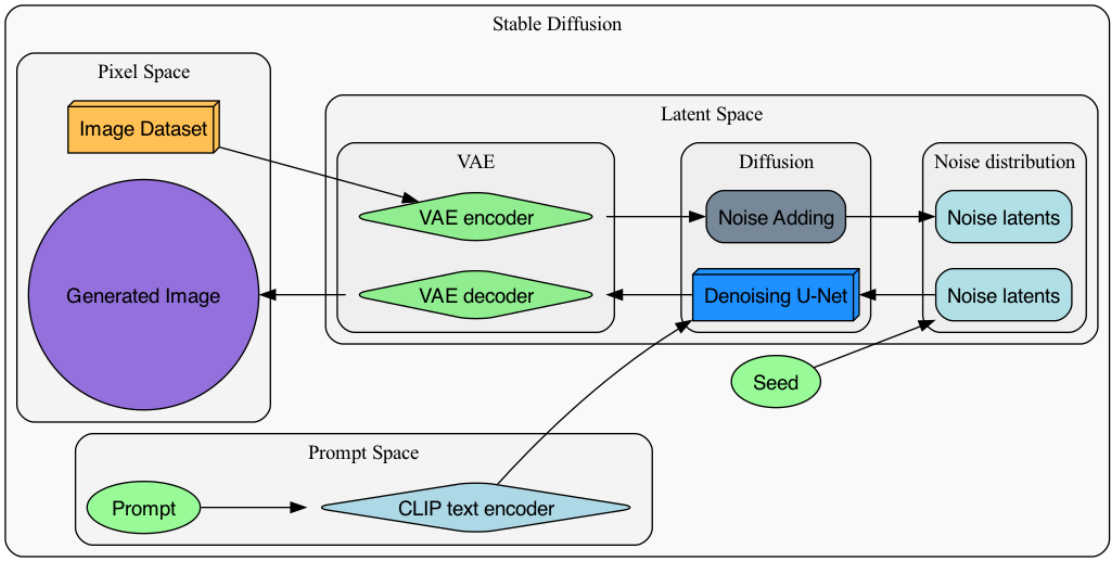


Figure 2.9: A overview of the Stable Diffusion model architecture. The "Latent Space" is the lower dimensional vector representation of images, "Pixel Space" is the pixel representation, and "Prompt Space" is the string representation of the prompt. There are two paths in "Latent Space"; one is for interference, and one is for training. The interference path starts from the seed-generated noise and the prompt, whereas the training path starts with an image from the training dataset

3
Methods

This chapter motivates and describes the implementations of the input-output mappings in this work. A proposed method of control is also presented in this chapter.

3.1 Software environment

This project utilized Python, an industry-standard language in machine learning (ML) and data science, primarily due to its vast array of robust packages and its simple, versatile syntax.
The Stable Diffusion model, an open-source deep learning model, was leveraged in this project. Opting for this model was driven by several factors. First, its open-source nature grants complete access to all sub-components of the model, promoting a deeper understanding of its workings and enabling customization for greater flexibility. Second, as a large pre-trained model, Stable Diffusion is capable of generating realistic images. This allowed the project to focus on exploring and refining the input-output mappings, rather than investing time and resources in training a new model from scratch.
PyTorch was selected as the primary ML library primarily for its seamless integration with the Stable Diffusion model via the Hugging Face platform. Although PyTorch offers a flexible and efficient environment for implementing various ML models, its specific advantages in this project revolve around its compatibility with the chosen model and platform.
Lastly, the Hugging Face platform was employed as it offers convenient access specifically to the Stable Diffusion model. The use of Hugging Face simplified the process of leveraging the model in this project and provided a comprehensive set of tools for fine-tuning and experimentation.
The chosen software environment, in its entirety, was designed to promote a transparent, efficient, and effective execution of the project tasks.

3.2 Creation of an Object-Oriented Pipeline

The initial stage of the project was devoted to constructing an object-oriented pipeline for generating images from text using the Stable Diffusion model. The design of this pipeline was geared towards enabling convenient access to all intermediate stages of the model's input/output. The pipeline was architectured around two fundamental objects: "Prompt" and "ImageNoise".

Object	Description
Prompt	The Prompt object encapsulates both the string and vector representation of the text input. The machine-encoded form of the prompt that guides the Stable Diffusion model is included in this encapsulation. The main reason for creating this object was to provide easy access and the ability to manipulate the prompt during different stages of the pipeline.
ImageNoise	The ImageNoise object represents the latent variable that the Stable Diffusion model operates upon. This latent representation can either be a pre-existing image, the output image, or a noise pattern, depending on the specific step in the latent diffusion process. The creation of the ImageNoise object aimed to provide an efficient method for accessing and manipulating this latent representation during different stages of the pipeline.

Table 3.1: Key objects in the pipeline

These objects form the backbone of the pipeline, providing a transparent and flexible framework for the project. This structure not only demystifies the inner workings of the Stable Diffusion model but also facilitates granular access to critical stages of the image generation process, supporting a more effective exploration of input-output mappings.

3.3 Input-Output Mapping of the Diffusion Model modules

In order to gain a deeper understanding of the image generation process facilitated by the Stable Diffusion model, an assessment of each module or stage was undertaken. This was accomplished by generating an input-output mapping for each distinct phase of the model's operation.
Input-output mappings serve to illuminate the functionality of each stage, tracing the transformation of data as it passes through a particular module. This provides insights into the module's contribution to the final image generation. These mappings may hint at potential avenues for improving control of the model.
The primary goal of this exploration is to enhance the understanding of the Stable Diffusion model. Additionally, it could potentially guide targeted improvements in the model, should the mappings suggest a feasible path for such enhancements.

3.3.1 Word Attention

To analyze the model's attention at the word level, the subword-level attention scores were first aggregated into a word-level attention map. This map encapsulates how much attention each word in the prompt receives from other words i.e. the token or

subword representation was converted into words:

$$\text{Word-level Attention Map } i, j = \text{Combine Tokens}(\text{Attention Map } i, j)$$

Here, Attention $\text{Map}_{i,j}$ represents the original attention matrix from the transformer model. The indices i and j correspond to the respective words in the input.
The word attention was then computed for each word i by summing the attention it receives from all other words, and itself, across a single head:

$$\text{Word Attention } i = \text{Sum}(\text{Word-level Attention Map } i, j)$$

To gain insight into the total average attention each word receives across all heads in all layers of the transformer model; the mean of the word attention, described above, was then computed:

$$\text{Average Word Attention } i = \text{Mean}(\text{Word Attention } i)$$

Finally, these average word attentions were visualized as a bar plot. This representation allows us to understand more clearly how much attention each word is given by the model when generating images. This could potentially guide us in crafting more effective prompts.

3.3.2 Smoothly adjusting the prompt

The U-Net that is the core of the diffusion model does not directly determine the image and is instead applied iteratively to make a prediction of the noise in the image by the scheduler. This introduces complexity when it comes to visualizing the effect of the prompt on the image; to circumvent this complexity, the vector representation of the word token was manipulated before inputting it into the U-Net. To do this, the vector corresponding to the position of a given word was adjusted. This approach assumes that the CLIP encoder does not change the position of the information connected to a word (which is supported by the results of Word Attention, see 3.3.1 and 4.1.2). Note that although the primary focus of these adjustments is for output control, this also serves as a test of the assumption.

3.3.2.1 Adjustments for control

First, this approach was evaluated as a means for smooth control of the generated image. This was done by increasing the norm of the word vector representation for different words and with different percentage increases.

$$\text{New Word Embedding } i = (1 + \epsilon)\text{Prompt Embeddings } i$$

Where i is the word index, Prompt Embeddings is the vector representation of the prompt indexed by word, and ϵ is the change.
Note that these embeddings can potentially be adjusted in different ways; for example, the encoder often learns to map similar words to similar embeddings, so one might move the vector toward a specific word. This is however outside the scope of

Hyperparameter	Value
Number of steps	250
Guidance Scale	7.5
Height of image	512
Width of image	512
Stable Diffusion version	CompVis/stable-diffusion-v1-4
CLIP version	openai/clip-vit-base-patch32

Table 3.2: Hyperparameter used during the "Adjustments for control" evaluation.

this project so that the norm will be adjusted as a "neutral" choice of adjustment in this project.
This table shows the hyperparameter used during the image generation. For this word embedding adjustment.

3.3.2.2 Comparing schedulers

The chosen scheduler does also have an effect on the resulting image, as well as the generation speed. To investigate this, the above-mentioned adjustment was done with three different main types of schedulers: DiscLMS, DDIM, and PNDM.

3.3.3 Prompt-to-Image Mapping

For the purpose of understanding the effect of each word, in the prompt, on the image output. The smooth representation from above will be leveraged to create a prompt-to-image map. This was attempted with a perturbation-based mapping, i.e., changing the prompt by a small amount and mapping the difference, and gradient mapping through finite difference, i.e., making a very small change to the prompt and scaling up the result to approximate the gradient.

Settings	Value
Scheduler	PNDM
Number of steps	250
Guidance Scale	7.5
Height of image	512
width of image	512
Stable Diffusion version	CompVis/stable-diffusion-v1-4
CLIP version	openai/clip-vit-base-patch32

Table 3.3: Settings used for the diffusion model when computing the image for prompt-to-image map.

3.3.3.1 Perturbation Mapping

For the "Perturbation map", a smooth prompt adjustment was done in the same way as in the section 3.3.2, for each word in the prompt. The adjustment was done

with a $\epsilon = 0.15$. Then the image was computed using the diffusion module for both the un-adjusted and adjusted prompts. Then the operation difference from the PIL Python package ImageChops was used to find the difference between the image corresponding to the un-adjusted and the adjusted prompts. This operation calculates the absolute value of the pixel-by-pixel difference between the two input images.
To display this result the "difference" images, given by the operation difference were first turned black-and-white. Then the images were given an individual color using the HSV color wheel, and the alpha was set to the intensity of the black-and-white image. Lastly, these images were overlaid in two different ways, by setting the pixel to the color corresponding to the "difference" image with the highest intensity at that pixel, or simply all together.

3.3.3.2 Gradient Mapping through Finite difference

The gradient was approximated in the image space, where an image was represented as a PyTorch tensor of the dimensions 3×Image height in pixels×Image height in pixels. The difference to the input was applied as a small adjustment (i.e. 3.3.2), $\epsilon = 0.01$ (This can be compared to the $\epsilon = 0.15$ that was used in the perturbation map 3.3.3.1). The reason why a smaller ϵ was not used was that this seemed to make the mapping very noisy.

$$\text{Approximate Gradient } i = (\text{Adjusted Image } i - \text{Original Image})/\epsilon_{Size}$$

Approximate Gradienti is the approximate gradient for a given word with index i, Adjusted Imagei is the image adjusted for this word, and ϵ_{Size} was calculated using the Linalg norm opartor from PyTorch.
The approximate gradient was then converted to a Image height in pixels×Image height in pixels matrix by calculating the norm of the tensor at the pixel locations, and the values were set so the minimum value was 0 and the maximum 255 i.e. corresponding to the standard color range in r,g,b. The matrix was then converted into a black-and-white Python PIL image and displayed in the same way as in the section 3.3.3.1.

3.3.4 Noise-to-Image Mapping

A specific image generated by the Stable Diffusion model, and diffusion models, in general, are highly influenced by the exact starting noise, or in other words, the latent vector that the models start with i.e. the noise instance. This is what makes these models "creative". Thus to completely investigate input-to-output mappings for every module the effect of the noise should also be investigated.
Quantifying the effect of the noise given a specific prompt was relatively straightforwardly done by feeding the generated image sample into the images encoder. The similarity score was then calculated using the CLIP processor accessed through Hugging Face. The similarity score is the scaled dot product scores between the image embeddings and text embeddings. This score is not directly related to any similarity measure but higher values correspond to a more similar prompt and image, and can thus be used to compare different image samples, generated with the same prompt.

Finally to measure the "impact of noise", depending on the prompt a number of images were generated for 5 prompts of an increasing level of relative "abstraction", for example, "a photograph of an astronaut in space" is less abstract than "a photograph of an astronaut riding an elephant". The similarity score for the prompt and image sample was then computed and displayed as a moving average, i.e., the first score only included that score as the "average" and the second was the average of the first and second score, the standard deviation of the data was also included in the score plot.

3.4 Proposed method of control: Movement-to-Image

Motivated by the instability of the diffusion model, i.e., sensitivity to noise (and how well the relatively simple method of adjustments of word embedding worked, see 3.3.2 and 4.1.3). A potential method may be to, instead of trying to map the segments of the input to the output for the purpose of greater control; one could instead map in human understandable points into a space with large control opportunities. That is mapping in similar, for example, prompts into the vector embedding space, then reduce the dimensionality, and then move toward or away from these known points.

Algorithm 1 Movement-to-Image pipeline

1: Generate similar inputs - Generate similar prompts ex. through an LLM (For the proof-of-concept this was hardcoded)
2: Map all inputs to the embedding space - Generate the embeddings (This was done through the CLIP encoder)
3: Reduce dimensionality - Create a PCA plot
4: Move the input in the embedding - Through some joystick-like input get user input to adjust the input embedding (For the proof-of-concept this was hard-coded)

This gives a smooth interface for input manipulation leveraging other inputs as a sort of "lighthouse" giving meaning to the movements in the embedding space. Effectively circumventing any need to understand the values in the embedding space but still allowing meaningful and potentially predictable smooth adjustments.

4

Results

This section walks through all the main results obtained in this project. These include text-impact mapping, which refers to the impact that specific words had on the prompt representation. Additionally, it covers word-to-image mapping, i.e., the effect of the prompt on the image output, and noise-impact mapping, which relates to the effect of the second input, noise, and its corresponding impact.

4.1 Text-impact mapping

This section includes different attention measures to determine which words have a relatively large impact on the prompt vector representation. As well as, maybe more importantly, how the CLIP text encoder shifts and processes the information contained in the prompt.

4.1.1 Standard attention map

Starting off with a sanity check and characterization of how the CLIP text encoder handles words in different heads. We can see from the images 4.1 that the information contained in the words seems to stay roughly at the same place, i.e., the attention the model gives to each token seems to be preserved at the places occupied by word tokens, instead of empty tokens. However, like most methods for evaluating how an ML-model makes its decisions, this is a guess, although motivated.

Specifically, figure 4.1 shows the amount of attention placed on a token when evaluating another token. The different images in the figure correspond to different masks in different layers for the same prompt.

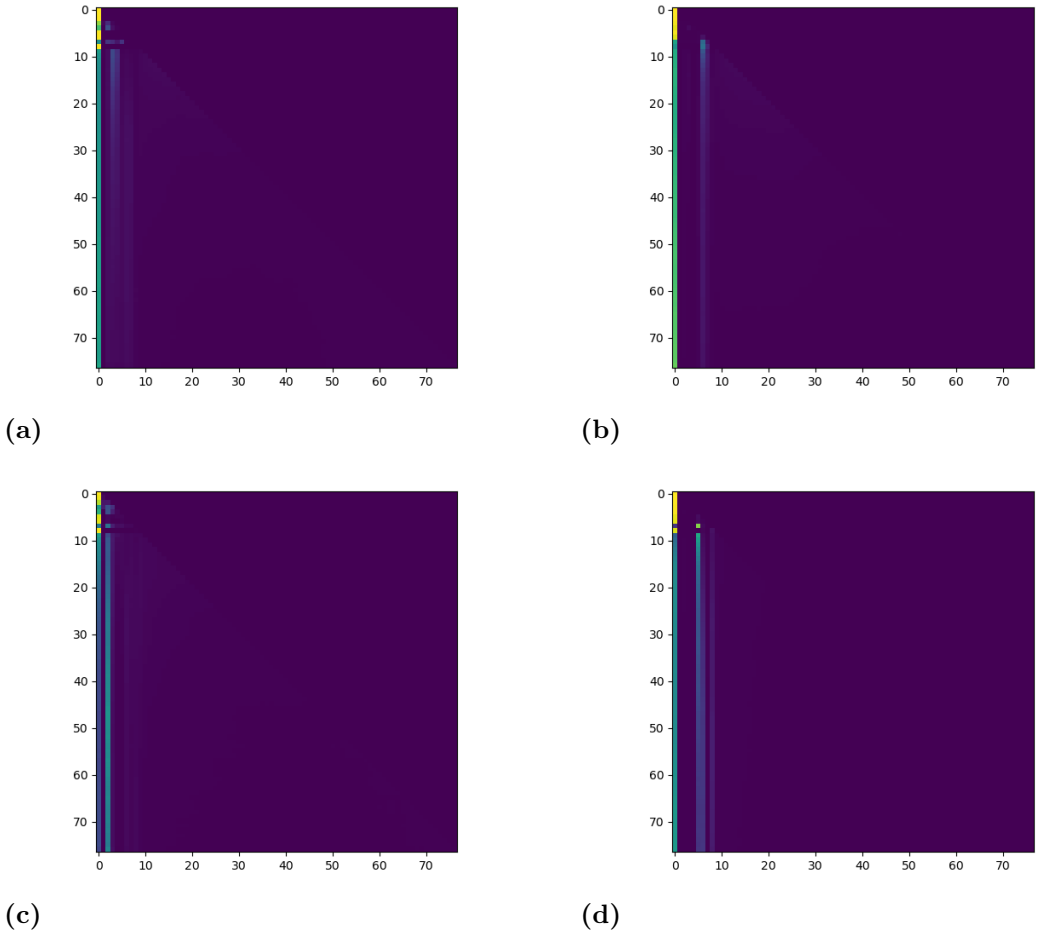


Figure 4.1: The full attention maps of four heads in different layers i.e. the cross attention for all tokens.

Zooming into the tokens that represent words, in the prompt "a photograph of an astronaut riding a horse", we can, of course, evaluate how much a word impacted another in a specific layer. We can also see that some words seem to be more important in general and not just for a single head.

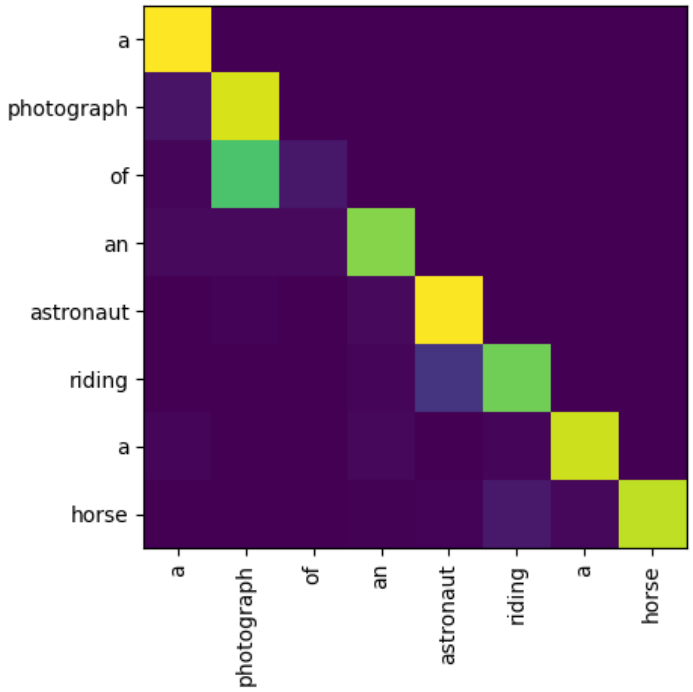
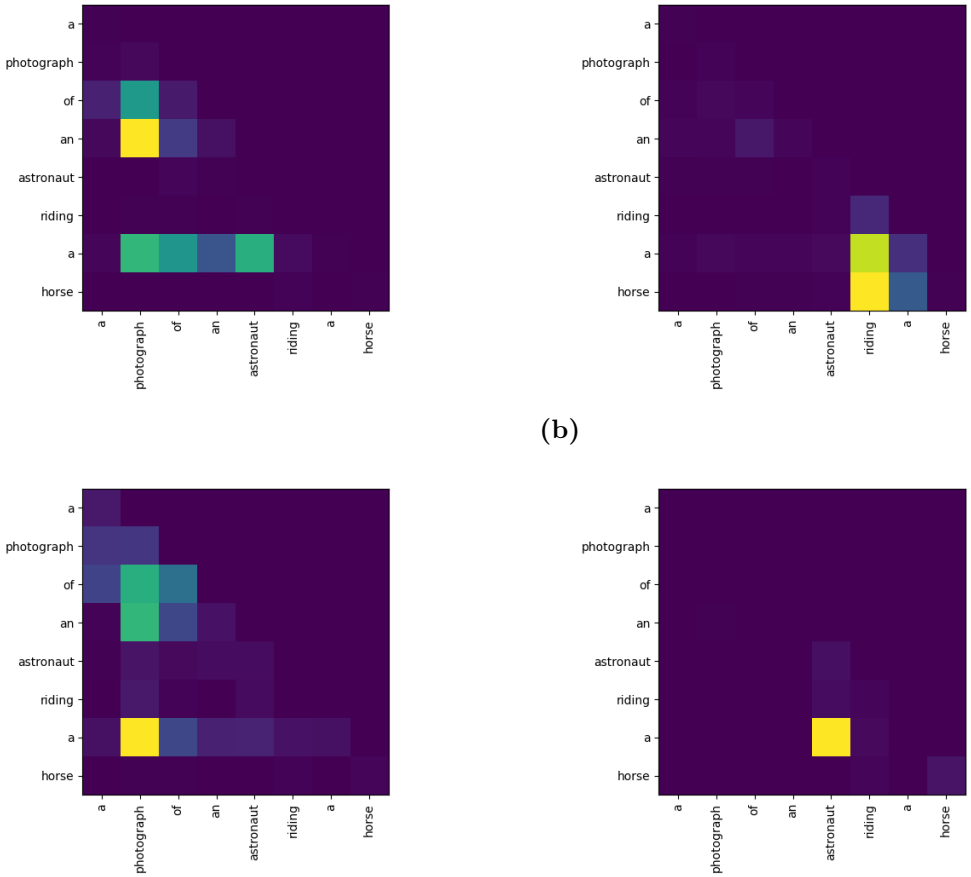


Figure 4.2: A zoomed and cropped attention map for one of the heads from the first layer.



(a) **(b)**

(c) **(d)**

Figure 4.3: The attention maps of four heads in different layers, the heads are the same as in 4.1.

4.1.2 Average word attention

A more useful way to view the attention maps when gauging the impact of a word on the prompt vector representation, and thus the image, maybe to sum up the cross attention. This gives a measurement showing how much other tokens and the token itself is influenced by the token. Figure 4.4 shows this for every layer for the prompt "a photograph of an astronaut riding a horse".

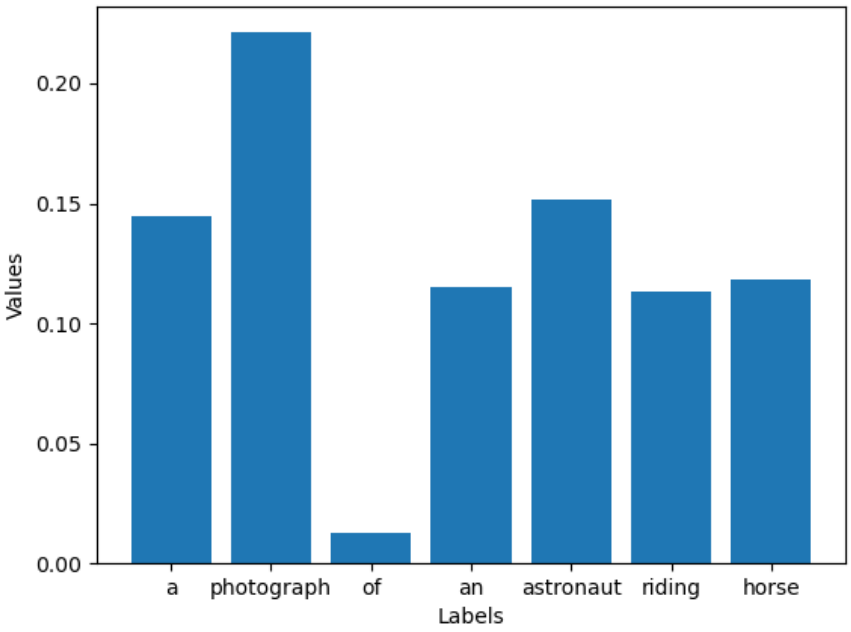
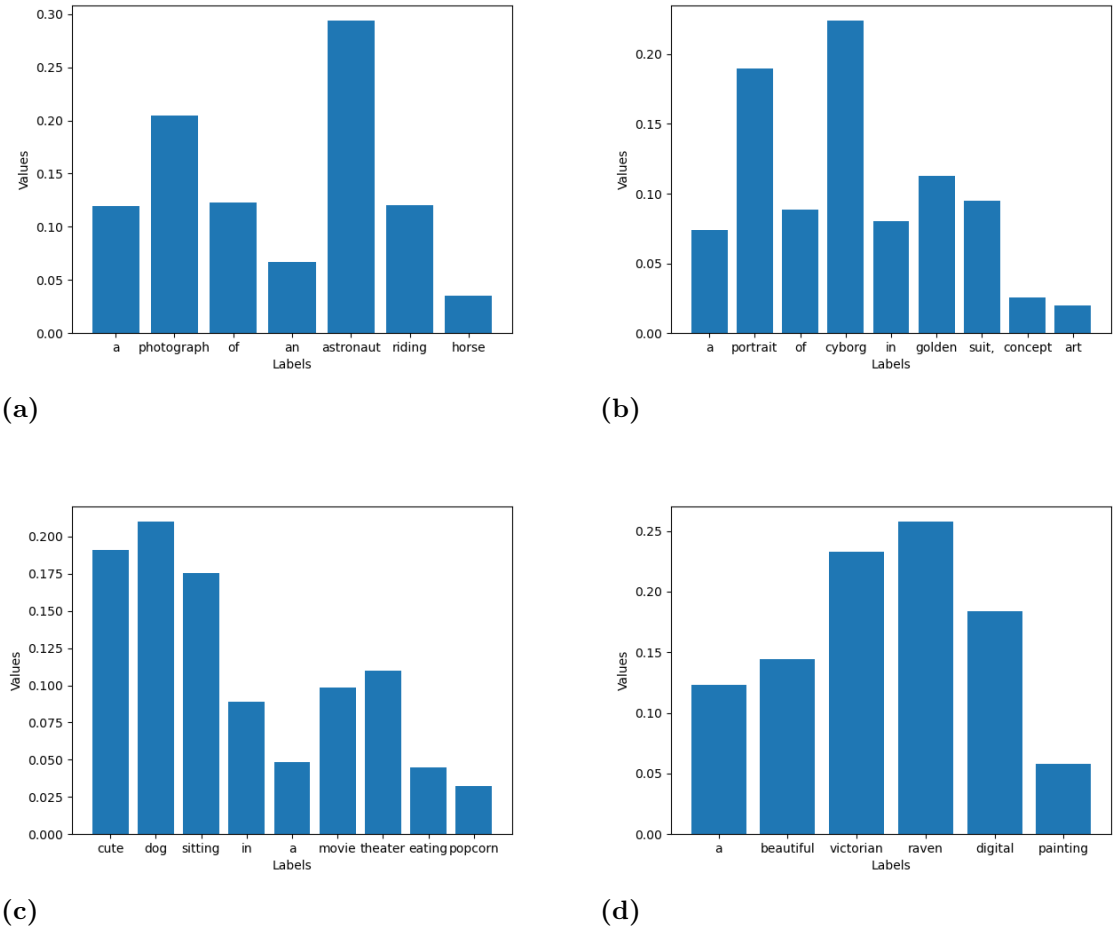


Figure 4.4: The attention score of one of the heads from the first layer.

Figure 4.5 shows the attention map or score averaged through all layers and heads for a few prompts. We can see from this that the score seems to align with the impact of each word token on the image for example, word tokens that can't be displayed in an image, for example, "a" seem to have low scores, and substantives that are included have high scores. Thus the average word token score seems to correspond roughly to the effect on an image. It should be noted that this does not apply to all words; for example, "horse" has a relatively low score, although "riding" and "horse" together do have a high score.



(a) **(b)**

(c) **(d)**

Figure 4.5: The average attention score of all heads and all layers.

4.1.3 Word weight adjustment

This section contains the results that build on the average word attention results presented above. It thus this by working from the assumption the word (tokens) thus not change position through the clip encoder. This is only a working assumption and seemingly, at best, an approximation. Using this assumption, the prompt was manipulated for different words. During this process, the effect of the choice of scheduler was also investigated.

4.1.3.1 Discrete LMS Scheduler

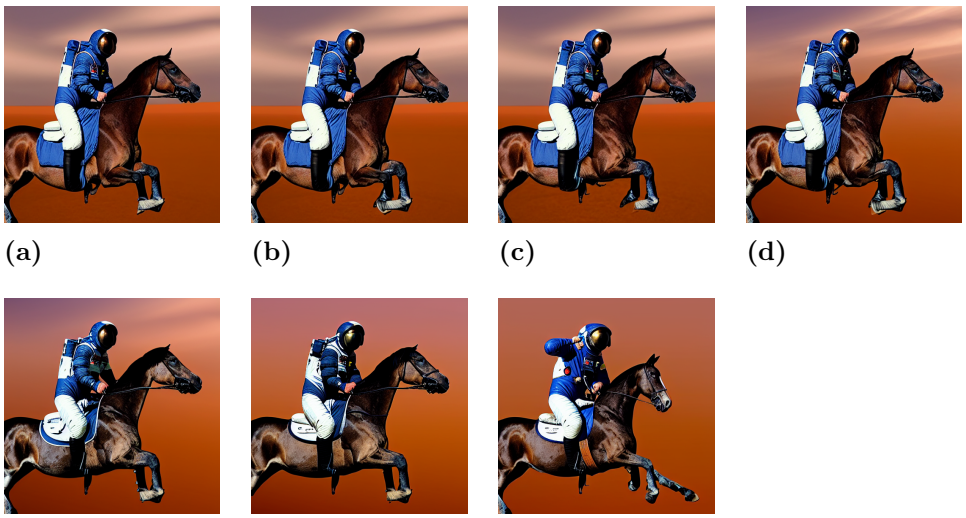
Below (4.6, 4.7 and 4.8), we can see that the embeddings manipulation seems to correspond well to what would be expected by the assumption. When increasing the "importance" of the word "suit" in the image with the golden cyborg, the body seems to become more "suit" like. When the "importance" of the word "suit" is decreased, the body seems to become more cyborg/robot-like, or in other words, less "suit" like.
Similar results can be seen for the words "photograph" and "astronaut" for the

prompt "a photograph of an astronaut riding a horse", so this seems to be a good avenue for increasing control of the model, seemingly allowing for smoothly adjusting the prompt until it fits what the user was looking for.

(a) **(b)** **(c)** **(d)**

(e) **(f)** **(g)**

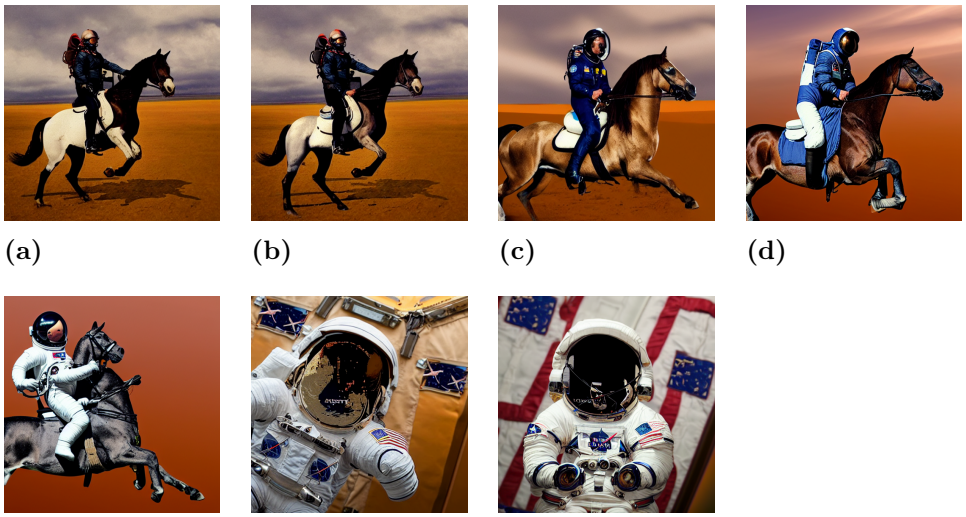
Figure 4.6: Image generation results for "a portrait of a cyborg in a golden suit, concept art", adjusted for the word: "suit". (a) $\epsilon = -0.6$. (b) $\epsilon = -0.4$. (c) $\epsilon = -0.2$. (d) $\epsilon = 0.0$. (e) $\epsilon = 0.2$. (f) $\epsilon = 0.4$. (g) $\epsilon = 0.6$ (Note that a light Stable Diffusion variant was used for computational efficiency, this does however reduce image quality).



(a) **(b)** **(c)** **(d)**

(e) **(f)** **(g)**

Figure 4.7: Image generation results for "a photograph of an astronaut riding a horse", adjusted for the word: "photograph". (a) $\epsilon = -0.6$. (b) $\epsilon = -0.4$. (c) $\epsilon = -0.2$. (d) $\epsilon = 0.0$. (e) $\epsilon = 0.2$. (f) $\epsilon = 0.4$. (g) $\epsilon = 0.6$.



(a) (b) (c) (d)

(e) (f) (g)

Figure 4.8: Image generation results for "a photograph of an astronaut riding a horse", adjusted for the word: "astronaut". (a) $\epsilon = -0.6$. (b) $\epsilon = -0.4$. (c) $\epsilon = -0.2$. (d) $\epsilon = 0.0$. (e) $\epsilon = 0.2$. (f) $\epsilon = 0.4$. (g) $\epsilon = 0.6$.

4.1.3.2 DDIM Scheduler

We can see here in the images below that the difference seems to be very small between LMS and DDIM, at least in the context of prompt adjustment (more images can be found in the appendix).

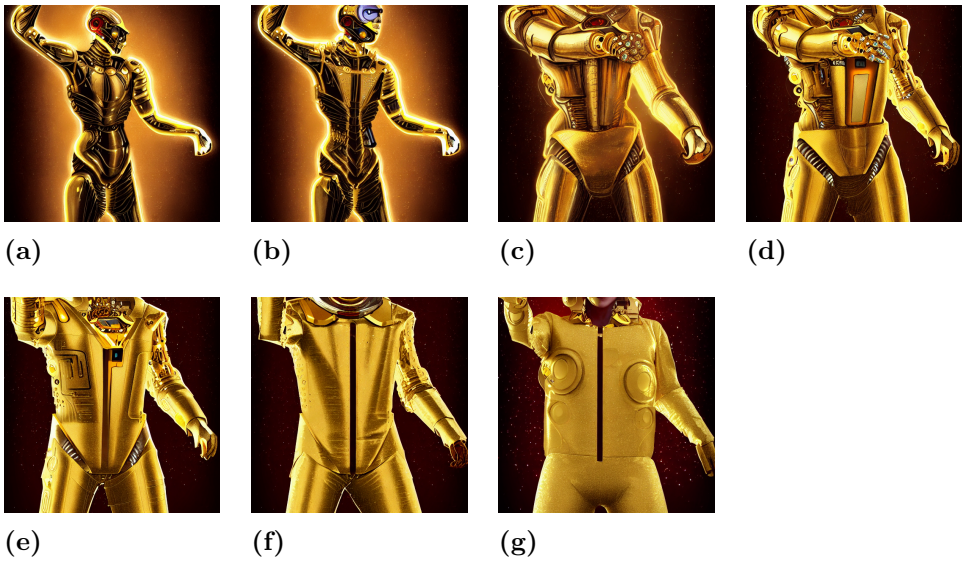
(a) (b) (c) (d)

(e) (f) (g)

Figure 4.9: Image generation results for "a portrait of a cyborg in a golden suit, concept art", adjusted for the word: "suit" with different epsilon values using the DDIM scheduler. (a) $\epsilon = -0.6$. (b) $\epsilon = -0.4$. (c) $\epsilon = -0.2$. (d) $\epsilon = 0.0$. (e) $\epsilon = 0.2$. (f) $\epsilon = 0.4$. (g) $\epsilon = 0.6$.

4.1.3.3 PNDM Scheduler

The PNDM scheduler seems to perform similarly to LMS and DDIM, at least in this context of prompt adjustment (more images can be found in the appendix).



(a) **(b)** **(c)** **(d)**

(e) **(f)** **(g)**

Figure 4.10: Image generation results for "a portrait of a cyborg in a golden suit, concept art", adjusted for the word: "suit" with different epsilon values using the PNDM scheduler. (a) $\epsilon = -0.6$. (b) $\epsilon = -0.4$. (c) $\epsilon = -0.2$. (d) $\epsilon = 0.0$. (e) $\epsilon = 0.2$. (f) $\epsilon = 0.4$. (g) $\epsilon = 0.6$.

4.2 Word-to-Image Mapping

This section includes the results of word-to-image mappings.

4.2.1 Perturbation Mapping

Below is an image showing the effect of a +15% perturbation to every word in the prompt; the prompt in question is "a photograph of an astronaut riding a horse". The image shows the color corresponding to the word with the largest change per pixel, i.e., for example, pink on a pixel if the difference image corresponding to an adjustment of the word "horse" had the largest norm for the color vector at that pixel. We can see that the words seem to be loosely related to the relevant objects in the image. However, one can also see that the words have effects on many "unrelated" parts of the image; for example, the word "astronaut" has large effects on the background.

Figure 4.11: Visualization showing the effect of perturbations on the prompt "a photograph of an astronaut riding a horse", the perturbation size was +15% in this image.

The next (4.12) shows the same perturbation-based mapping as in the above image

(4.11). However, unlike the above image here, the pixels are simply blended or added on top of each other but with an alpha value corresponding to the magnitude of change per pixel. The alpha value corresponds to how see-through a color is when a color is defined as [r, g, b, alpha]; thus, the color representing a word with little impact will be very see-through and won't have a large effect on the mapping.

Figure 4.12: Visualization constructed showing the effect of perturbations on the prompt "a photograph of an astronaut riding a horse", the perturbation size was +15% in this image, and the perturbation differences were directly overlaid.

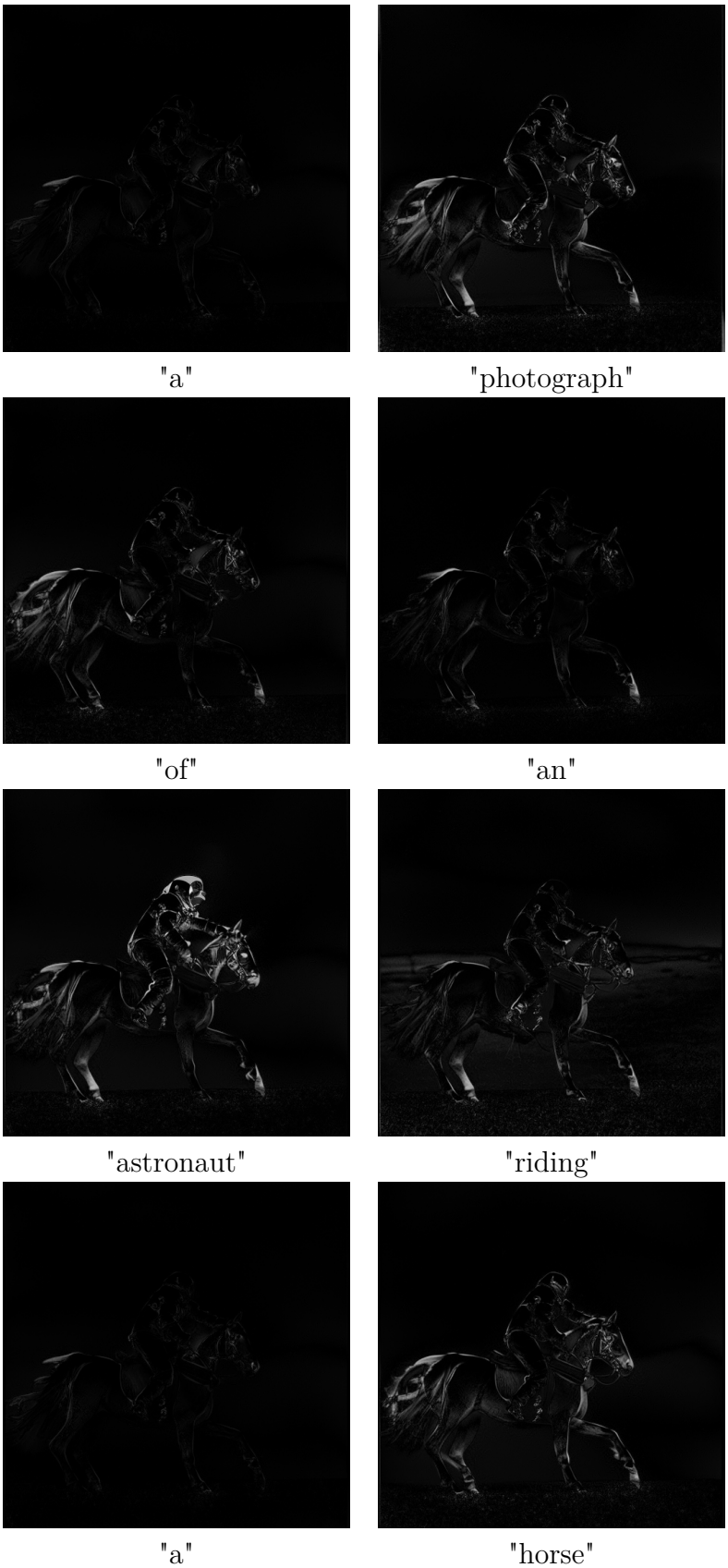


Figure 4.13: Images showing the effect of +15% size perturbations for each word in the text prompt "a photograph of an astronaut riding a horse".

4.2.2 Gradient Mapping

The image 4.14 shows the results of the gradient-based mapping, approximated through the finite difference method, for the prompt "a photograph of an astronaut riding a horse". The image shows the color of the word with the largest effect on each pixel. In the image, one can see that although there seems to be some structure to the word-to-image mapping, it is much more "random", i.e., the words have more effect on "unrelated" parts of the image.

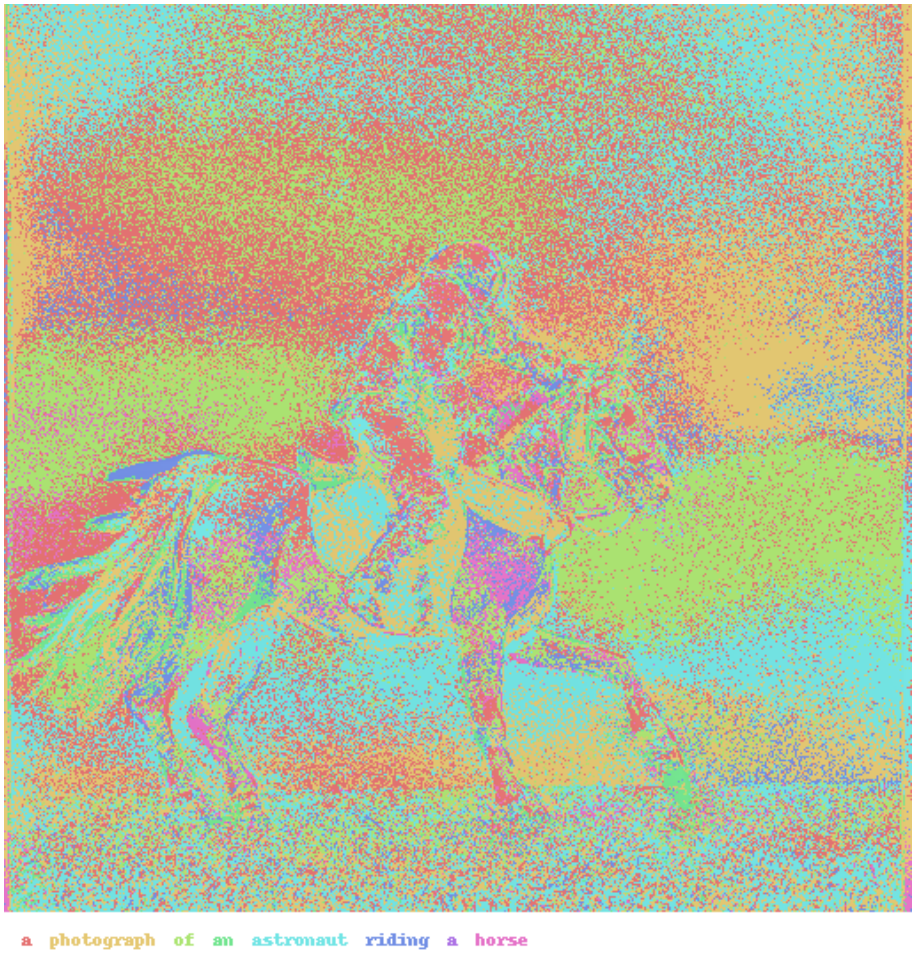


Figure 4.14: Visualization constructed in Python showing the "Saliency Map" of text prompts impact on the image output (approximated through the Finite difference method).

The image 4.15 is the result of the gradient-based mapping, but every color is blended on top of each other, with the alpha value corresponding to the effect a word had on a specific pixel. One can see in this image that although there are large effects on unrelated parts of the image, at least some words seem to correspond heavily to the expected part of the image. For example, the word "horse" has most of its effect on the horse in the image.

Figure 4.15: Visualization constructed showing the "Saliency Map" of text prompts impact on the image output, the differences are directly overlaid.

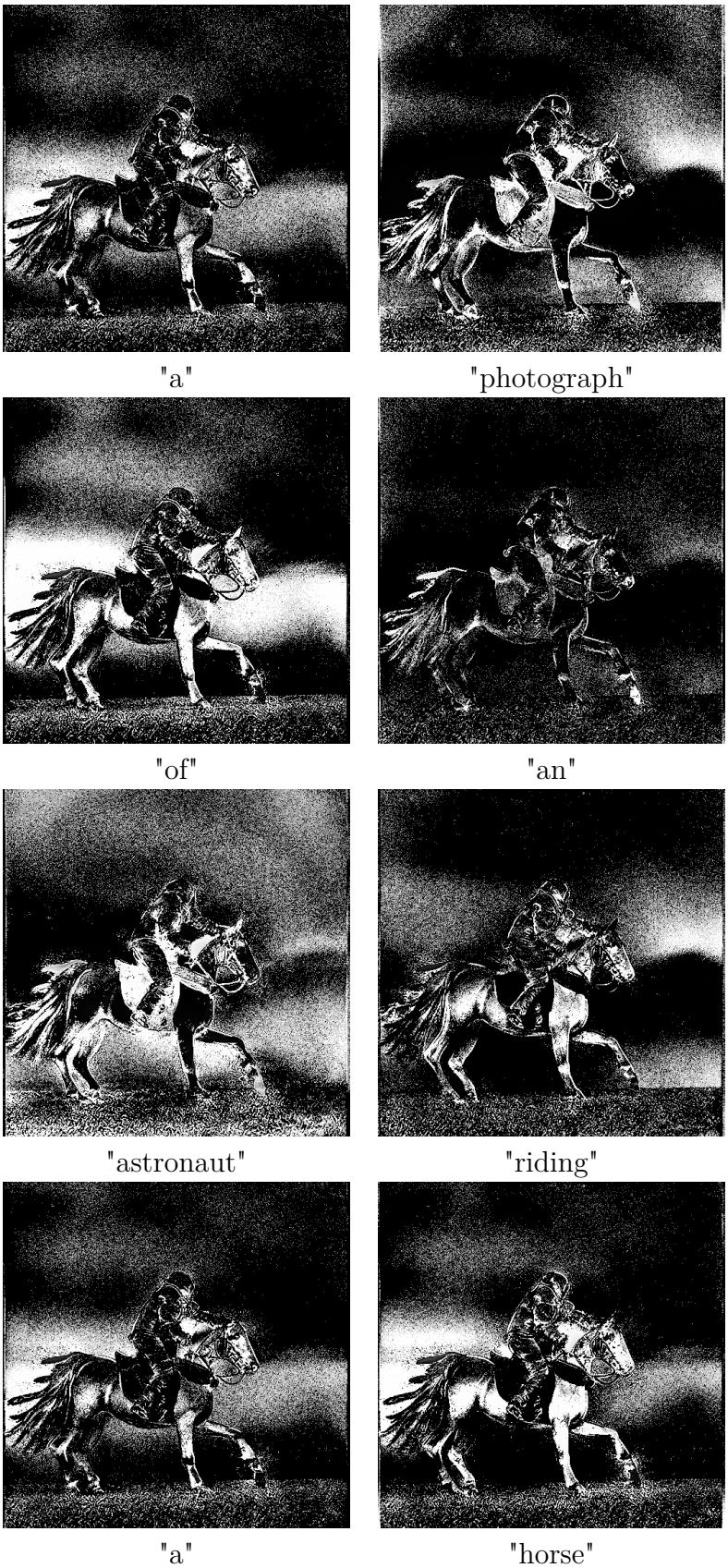


Figure 4.16: The "gradients" in image format computed during the Finite difference method for each word in the text prompt "a photograph of an astronaut riding a horse".

4.3 Noise-impact mapping

Below (4.17) is a plot consisting of the moving average (with an infinite window), as well as the standard deviation, of the similarity score of 10 for different images for 5 prompts. One can see that the average similarity score does not seem to correspond to anything representing some "structure" in the prompt; that is, all prompts have similar averages even though they were intentionally chosen to vary in how "abstract" they are, for example, the prompt "a photograph of an astronaut in space" is much less abstract than "a photograph of an astronaut riding a horse in the prism". Furthermore, the prompt "a photograph of an astronaut riding a horse in the prism" is in the middle of some of the more "normal" prompts. This suggests that the diffusion model is able to express the information continued in the CLIP embeddings regardless of the prompt, i.e., the embeddings for the generated image are close to the embeddings of the prompt despite some prompts being presumably harder to visualize.

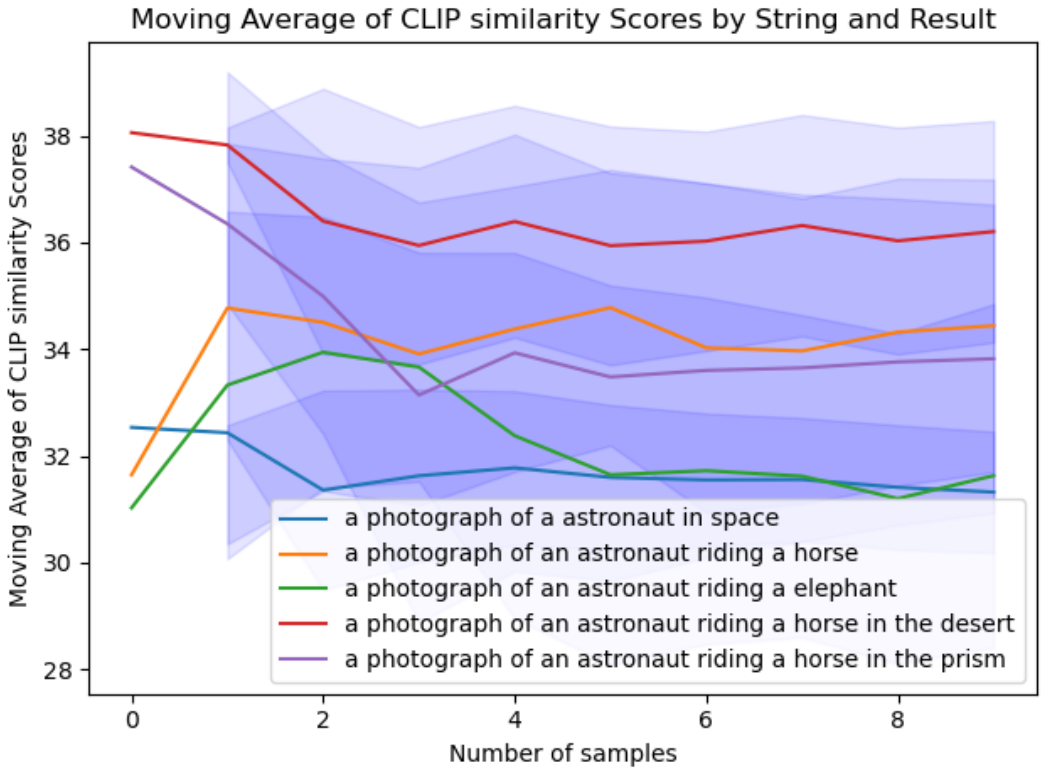


Figure 4.17: The moving average (the line) of the similarity score, as well as the standard deviation (with an infinite window), computed by the CLIP model.

Looking closer at the standard deviation in figure 4.18, one can see that it might correspond loosely to how "abstract" a prompt is having the most normal prompt at the bottom, although this number of samples is too few to make a judgment.

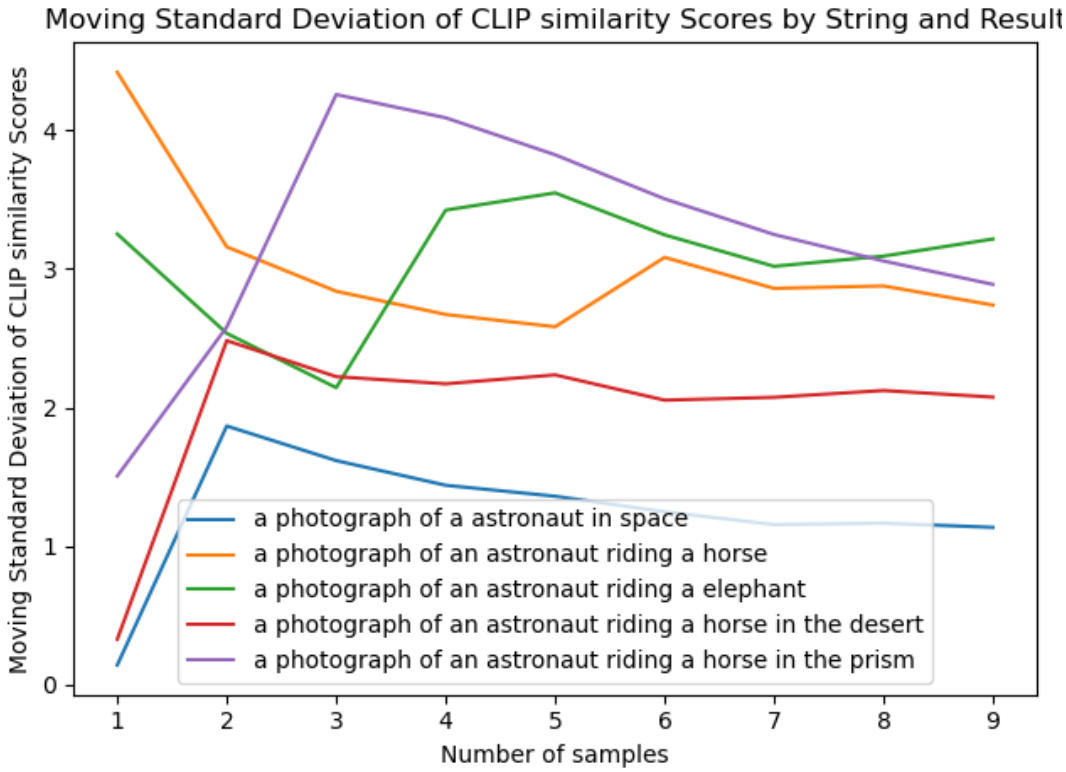


Figure 4.18: The moving standard deviation of the similarity score, computed by the CLIP model.

Evaluating if the noise-impact mapping illuminates something about the prompt that can be seen in the generated images, we have a few examples for the prompts in figure 4.19. One can see here that the standard deviation does seem to correspond to a "real" structure in the generated images for the different prompts, i.e., they are more varied for the prompts with higher standard deviations.

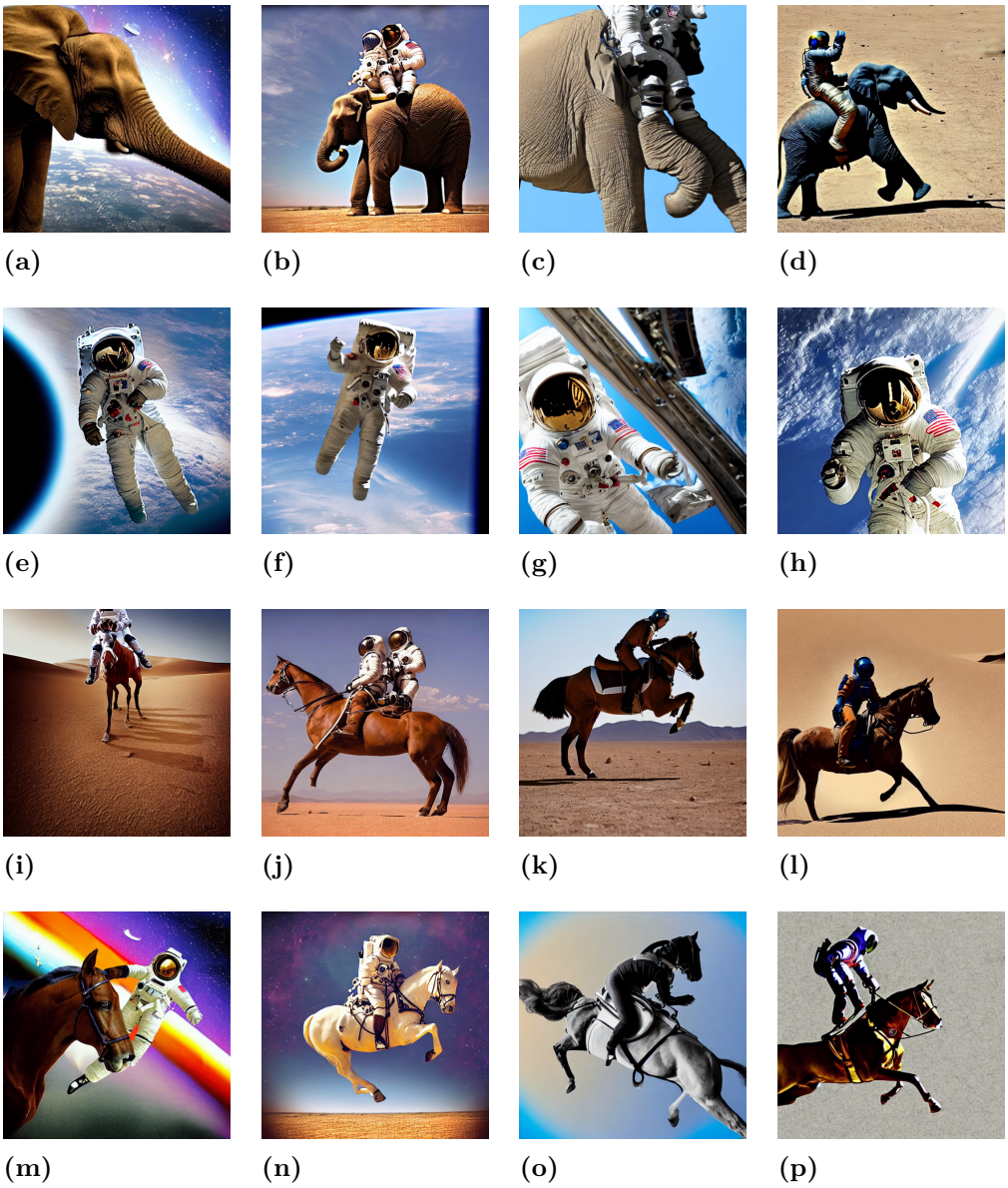


Figure 4.19: Examples of prompt-dependent variation. (a)-(d) Images generated for the prompt "a photograph of an astronaut in space". (e)-(h) Images generated for the prompt "a photograph of an astronaut riding an elephant". (i)-(l) Images generated for the prompt "a photograph of an astronaut riding a horse in the desert". (m)-(p) Images generated for the prompt "a photograph of an astronaut riding a horse in the prism" (Note that a light Stable Diffusion variant was used for computational efficiency, this does however reduce image quality).

We can thus, in total, see that the standard deviation of the similarity score seems to both be expressed in the generated images and may be caused by how "abstract" the prompt is. And also that the diffusion model seems to be able to express the information contained in the embeddings, which, together with the varying quality in the images, seen in 4.19 might suggest that the CLIP model is the "weak link" in the Stable Diffusion model, i.e., lower quality images may correspond to "lower"

quality prompt information in the embeddings.

4.4 Movement-to-Image pipeline Proof-of-Concept

Below we can see the results for the adjusted prompt embedding next to the relevant low-dimensionality representation, i.e., the interface. We can see that this proof-of-concept approach (see 3.4) seems to work relatively well; that is, the manipulation directly in the lower dimensional representation of the embedding space, which serves as the interface, does seem to correspond to the expected change in the image. In the images below (4.20 and 4.21), we can see the interface as well as the generated image for the input "a picture of a space explorer galloping on a horse" (the red dot).

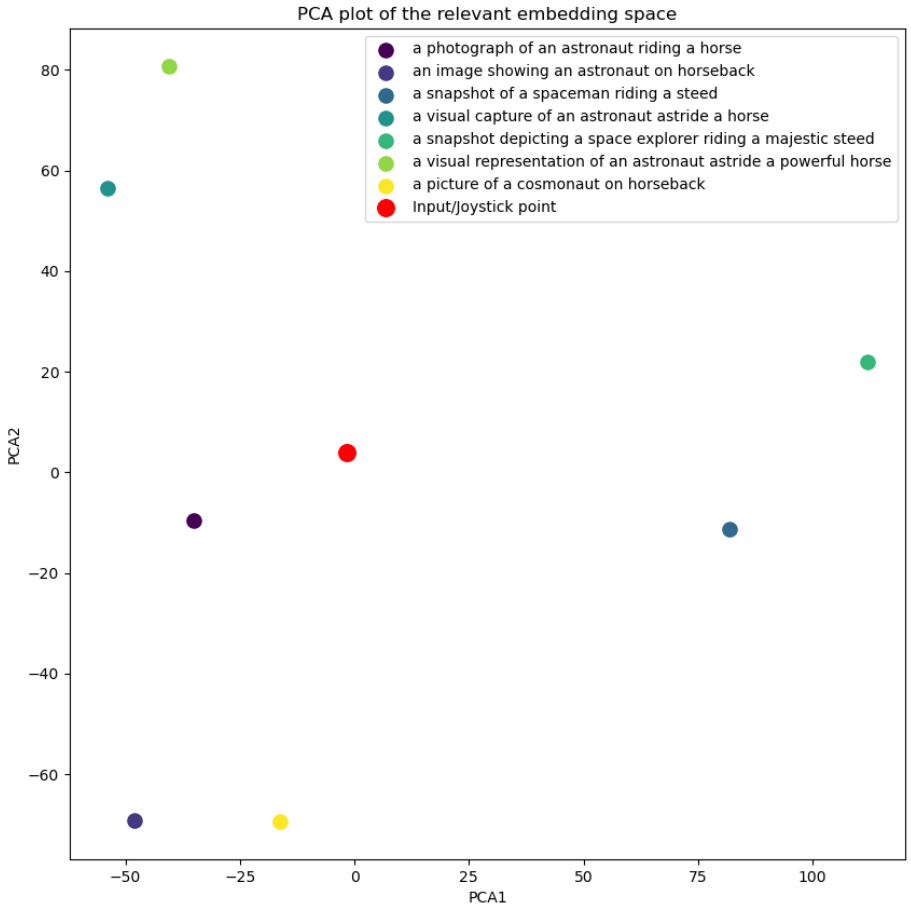


Figure 4.20: A image showing part of the movement-to-image pipeline proof-of-concept. The plot is the interface; in this case, a PCA plot on the embedding space, and the red dot is the input embedding.

Figure 4.21: A image showing part of the movement-to-image pipeline proof-of-concept. The image is the generated image corresponding to the interface 4.20 and the prompt "a picture of a space explorer galloping on a horse".

In the next image (4.22 and 4.23), we can see how making the starting input (the red dot) which was "a picture of a space explorer galloping on a horse" more similar to "a photograph of an astronaut riding a horse" actually makes the image include something similar to an "astronaut" while not changing the background much. This seems to correspond to a well-behaved interface as "astronaut" is the big difference in the prompts mentioned from the interface.

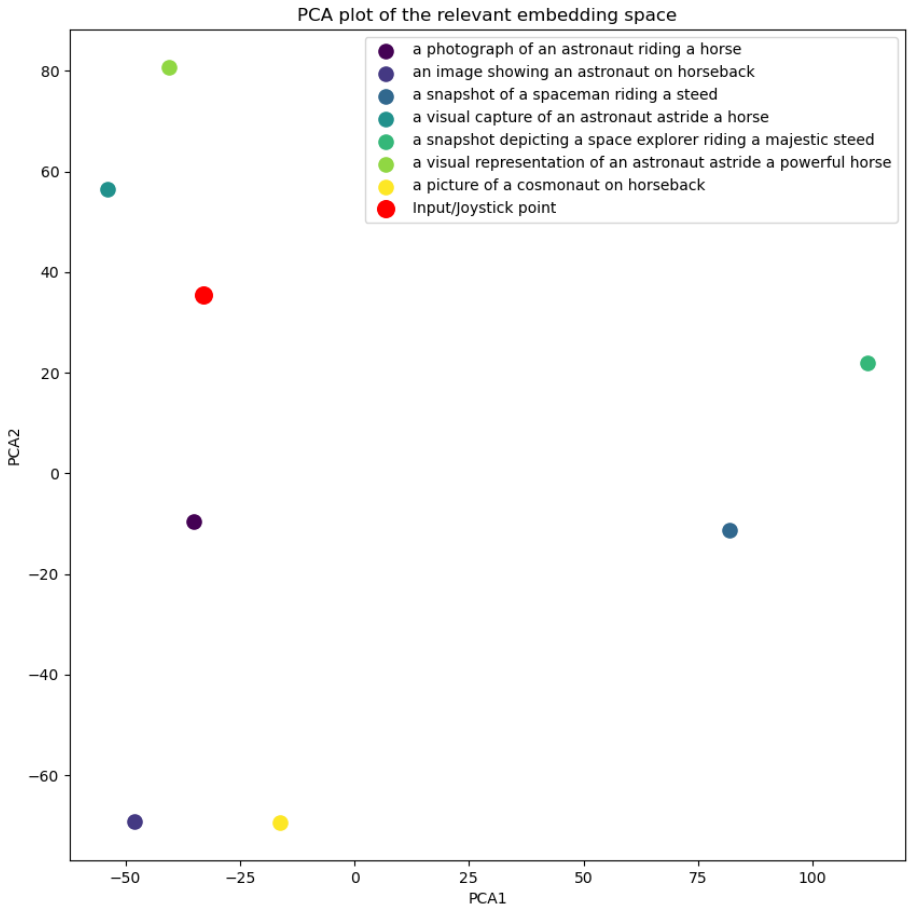


Figure 4.22: A image showing part of the movement-to-image pipeline proof-of-concept. The plot is the interface and the red dot is the adjusted input embedding.

Figure 4.23: A image showing part of the movement-to-image pipeline proof-of-concept. The image is the generated image from adjusted embedding corresponding to the interface 4.22 (Note that a light Stable Diffusion variant was used for computational efficiency, this does however reduce image quality).

5

Discussion

This section discusses some of the results that seem to be more significant.

5.1 Prompt consistency through the Encoder layers

To some degree, it is unexpected that the word attention score seems to reflect the amount of importance placed on a word (although from the results in the report, it is not known if this assumption is true). Normally neural networks are not expected to keep the placement of information in a way that reflects the input, and attention maps are not a direct reflection of the network output. But still, the words-related representation seems to have the same placement as in the input string i.e. average word attention 4.1.2 seems to reflect the expected importance of words and word weight adjustment 4.1.3 seems to result in the expected change. A possible explanation for this is the residual connection in the transformer architecture of the CLIP encoder. It seems reasonable that the model wouldn't be able to move the word information as the residual connection would interfere with the moved information.

5.2 Perturbation vs Gradient input-to-output mappings

The most significant characteristic difference between the perturbation and gradient input-to-output map seems to be the large word-specific region pixel changes i.e. the large changes not on the horse for the word "horse". This may be because it would be reusable for the CLIP word embeddings to only hold information on the content of the image without or with very little information on the composition, and thus the diffusion model is responsible for all information on the composition of the image. If we assume that this is true it would be reasonable that an infinitesimal change of the prompt embeddings would not change the content information much but would serve as a slightly different starting point for the diffusion model, and due to how sensitive the model is to noise i.e. different seeds produce very differently composed images. This slightly different starting point may still result in large composition information changes i.e. many pixel changes (caused by something like the outlines of the background moving one pixel). This may explain why the larger changes in the perturbation mapping seem to produce the biggest changes on the content.

5.3 Noise-impact mapping and Common sense

An interesting area of continuing research may be if the noise-impact mapping could be improved to the degree that it actually could measure how "difficult" it is for the model to construct an image for a given prompt this might be able to serve a more general purpose detecting something related to "how much sense the prompt makes" given how general these types of models are. Similar to how Autoencoders can be used for anomaly detection.

6
Conclusion

This project foremost aimed to investigate if input-to-output mappings could be produced for the Stable Diffusion model; this was successful from the viewpoint of the modules, which are of particular interest when it comes to control as they provide many intermediate representations. However, no complete input-to-output mapping for the diffusion model was produced in this project.
The next question was if control opportunities could be identified from the input-to-output mappings; in this area, it was found that the CLIP embeddings seemed to be remarkably consistent in structure with the prompt; this led to the possible control opportunity of changing the CLIP embeddings directly (3.3.2). It was also noticed from comparing Perturbation based embeddings-to-image mappings with Finite difference-based embeddings-to-image mappings that relatively large (smooth) changes seemed to correspond better to what may be expected compared to small (smooth) changes (4.2).
The last question was if these insights could be used to construct actual (proof-of-concept) control methods. In this area, smooth word importance adjustments were made and seemed to correspond to what was expected (4.1.3). A proof-of-concept for dimensionality reduction-based, general embedding smooth adjustments method was also constructed and showed some promise when considering how potentially general it is (4.4).
The proof-of-concept focus inherently means the project's most useful results may be as a catalyst for ideas in other works. To aid with this, the repository is public on GitHub at https://github.com/philipgrd/ControllingDiffusion.

Bibliography

[1] Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986;323(6088):533-6.

[2] Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86(11):2278-324.

[3] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial networks. Communications of the ACM. 2020;63(11):139-44.

[4] Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B. High-Resolution Image Synthesis with Latent Diffusion Models. CoRR. 2021;abs/2112.10752. Available from: https://arxiv.org/abs/2112.10752.

[5] Bank D, Koenigstein N, Giryes R. Autoencoders. CoRR. 2020;abs/2003.05991. Available from: https://arxiv.org/abs/2003.05991.

[6] Ribeiro E, Ribeiro R, de Matos DM. A Study on Dialog Act Recognition using Character-Level Tokenization. CoRR. 2018;abs/1805.07231. Available from: http://arxiv.org/abs/1805.07231.

[7] Toraman C, Yilmaz EH, Ş ahi
nuç F, Ozcelik O. Impact of Tokenization on Language Models: An Analysis for Turkish. ACM Transactions on Asian and Low-Resource Language Information Processing. 2023 mar;22(4):1-21. Available from: https://doi.org/10.1145%2F3578707.

[8] Mielke SJ, Alyafeai Z, Salesky E, Raffel C, Dey M, Gallé M, et al. Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. CoRR. 2021;abs/2112.10508. Available from: https://arxiv.org/abs/2112.10508.

[9] Almeida F, Xexéo G. Word Embeddings: A Survey. CoRR. 2019;abs/1901.09069. Available from: http://arxiv.org/abs/1901.09069.

[10] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention Is All You Need. CoRR. 2017;abs/1706.03762. Available from: http://arxiv.org/abs/1706.03762.

[11] Chefer H, Gur S, Wolf L. Generic Attention-model Explainability for Interpreting Bi-Modal and Encoder-Decoder Transformers. CoRR. 2021;abs/2103.15679. Available from: https://arxiv.org/abs/2103.15679.

[12] Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. CoRR. 2020;abs/2010.11929. Available from: https://arxiv.org/abs/2010.11929.

[13] Jamil S, Jalil Piran M, Kwon OJ. A comprehensive survey of transformers for computer vision. Drones. 2023;7(5):287.
[14] Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, et al. Learning Transferable Visual Models From Natural Language Supervision. CoRR. 2021;abs/2103.00020. Available from: https://arxiv.org/abs/2103.00020.
[15] McAllester D. On the Mathematics of Diffusion Models. arXiv preprint arXiv:230111108. 2023.
[16] Chen T. On the importance of noise scheduling for diffusion models. arXiv preprint arXiv:230110972. 2023.
[17] Song J, Meng C, Ermon S. Denoising Diffusion Implicit Models. CoRR. 2020;abs/2010.02502. Available from: https://arxiv.org/abs/2010.02502.
[18] Karras T, Aittala M, Aila T, Laine S. Elucidating the design space of diffusion-based generative models. arXiv preprint arXiv:220600364. 2022.
[19] Liu L, Ren Y, Lin Z, Zhao Z. Pseudo numerical methods for diffusion models on manifolds. arXiv preprint arXiv:220209778. 2022.
[20] Nichol A, Dhariwal P, Ramesh A, Shyam P, Mishkin P, McGrew B, et al. GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models. CoRR. 2021;abs/2112.10741. Available from: https://arxiv.org/abs/2112.10741.
[21] Kingma DP, Welling M. Auto-encoding variational bayes. arXiv preprint arXiv:13126114. 2013.
[22] Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B. High-Resolution Image Synthesis with Latent Diffusion Models. CoRR. 2021;abs/2112.10752. Available from: https://arxiv.org/abs/2112.10752.
[23] Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. CoRR. 2015;abs/1505.04597. Available from: http://arxiv.org/abs/1505.04597.
[24] Peebles W, Xie S. Scalable Diffusion Models with Transformers. arXiv preprint arXiv:221209748. 2022.
[25] Contributors HFSDM. Pipelines. Hugging Face; 2023. Accessed: 2023-05-29. Available from: https://huggingface.co/docs/diffusers/api/pipelines/overview.
[26] Contributors HFSDM. Schedulers. Hugging Face; 2023. Accessed: 2023-05-29. Available from: https://huggingface.co/docs/diffusers/api/schedulers/overview.

A

Appendix 1

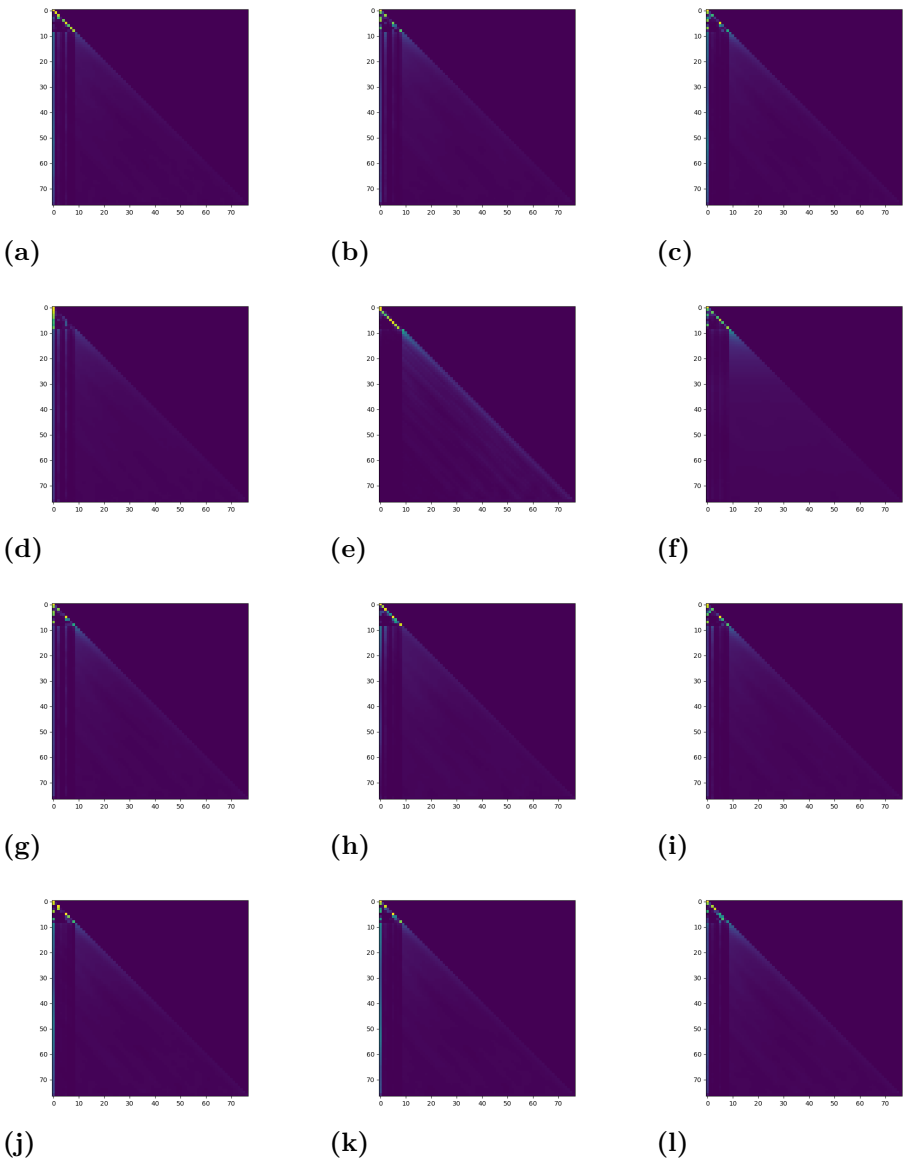


Figure A.1: Full attention maps of all heads in the first layer i.e. the cross attention for all tokens at every layer.

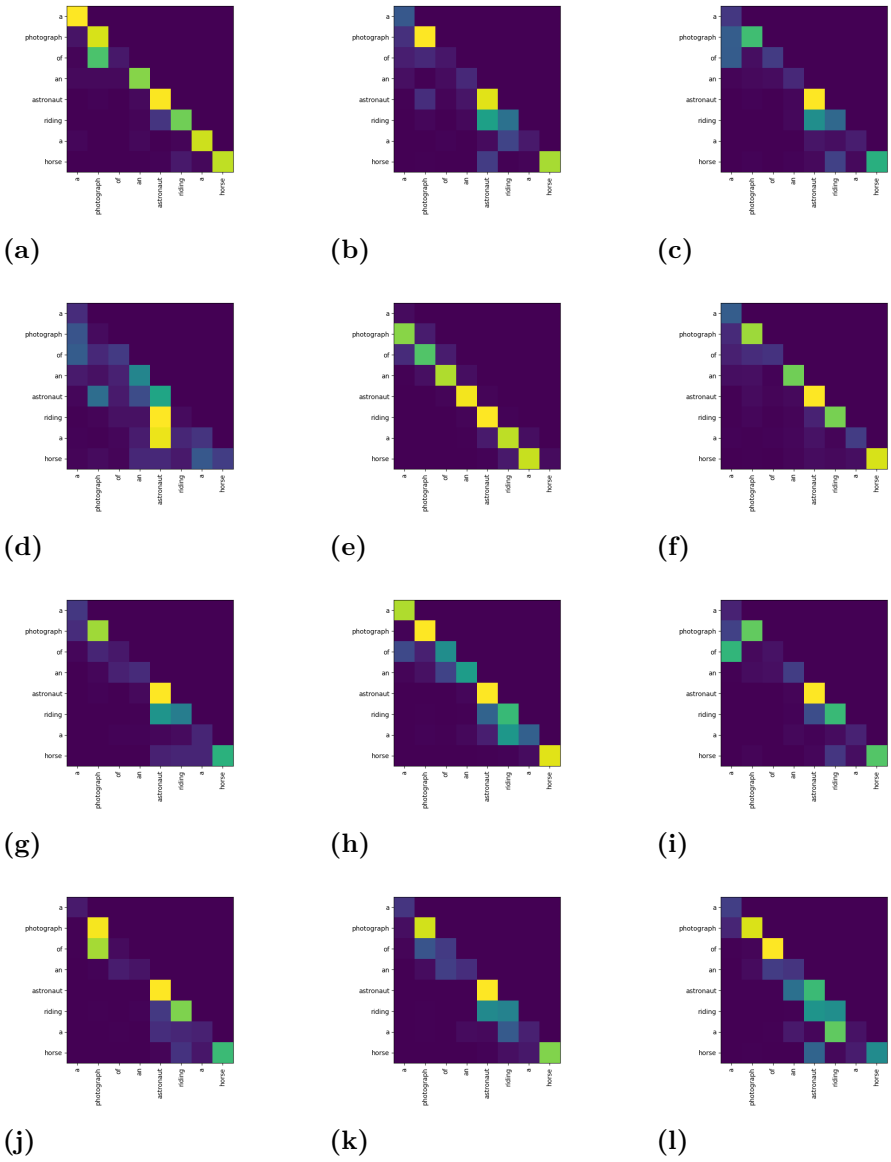


Figure A.2: Zoomed and cropped attention maps for all of the heads in the first layer, providing a comparison between heads.

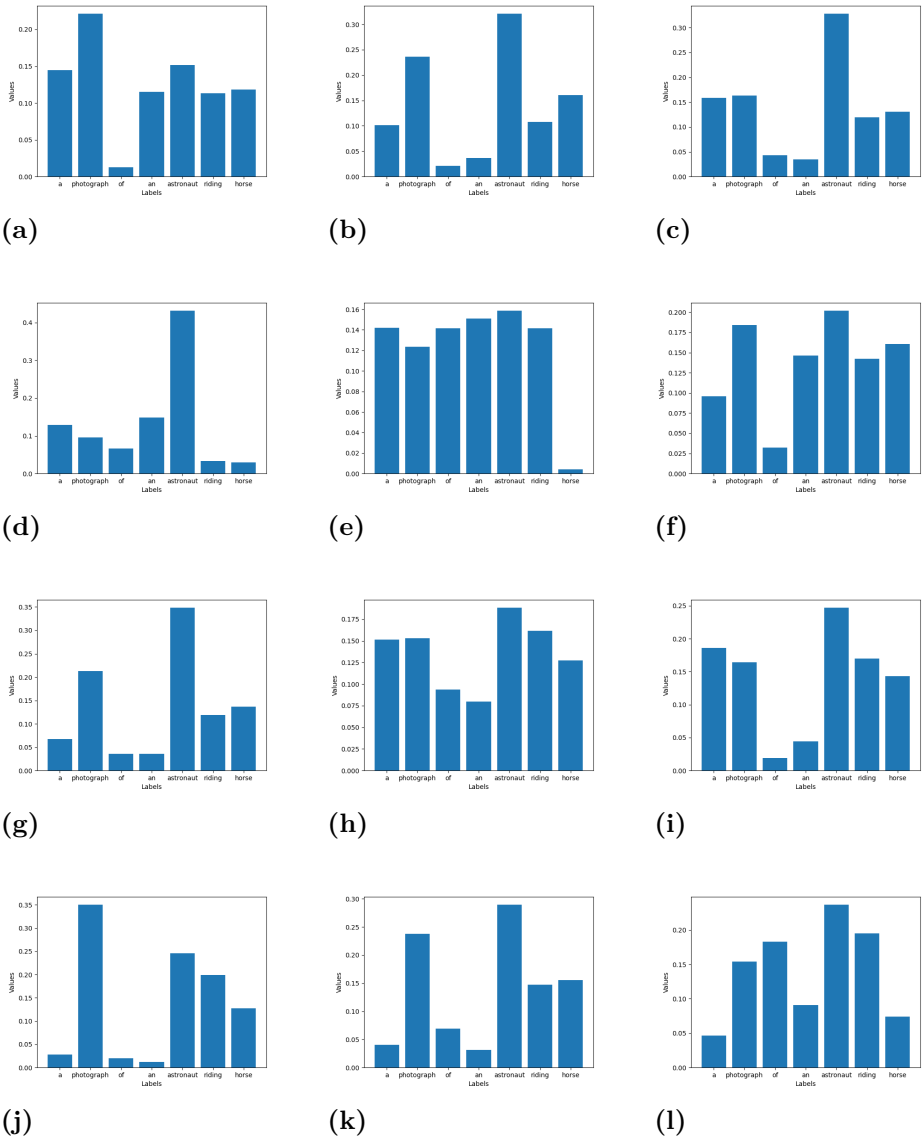


Figure A.3: The attention scores of all heads in the first layer.

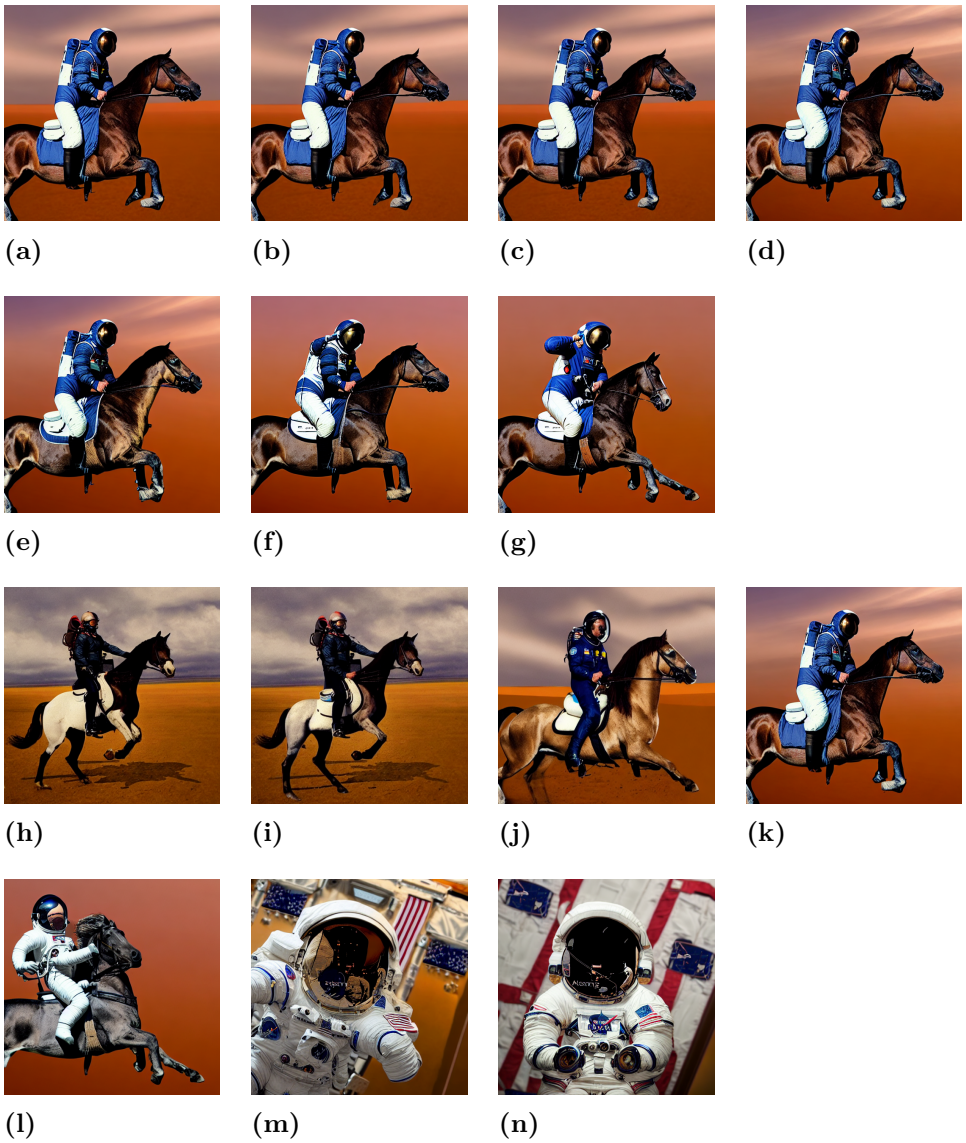


Figure A.4: Other image generation results with different epsilon values using the DDIM scheduler. The top two rows show the embeddings adjusted for the word: "photograph" and the bottom two rows show the embeddings adjusted for the word: "astronaut". For each set of images from left to right: (a) $\epsilon = -0.6$, (b) $\epsilon = -0.4$, (c) $\epsilon = -0.2$, (d) $\epsilon = 0.0$, (e) $\epsilon = 0.2$, (f) $\epsilon = 0.4$, (g) $\epsilon = 0.6$.

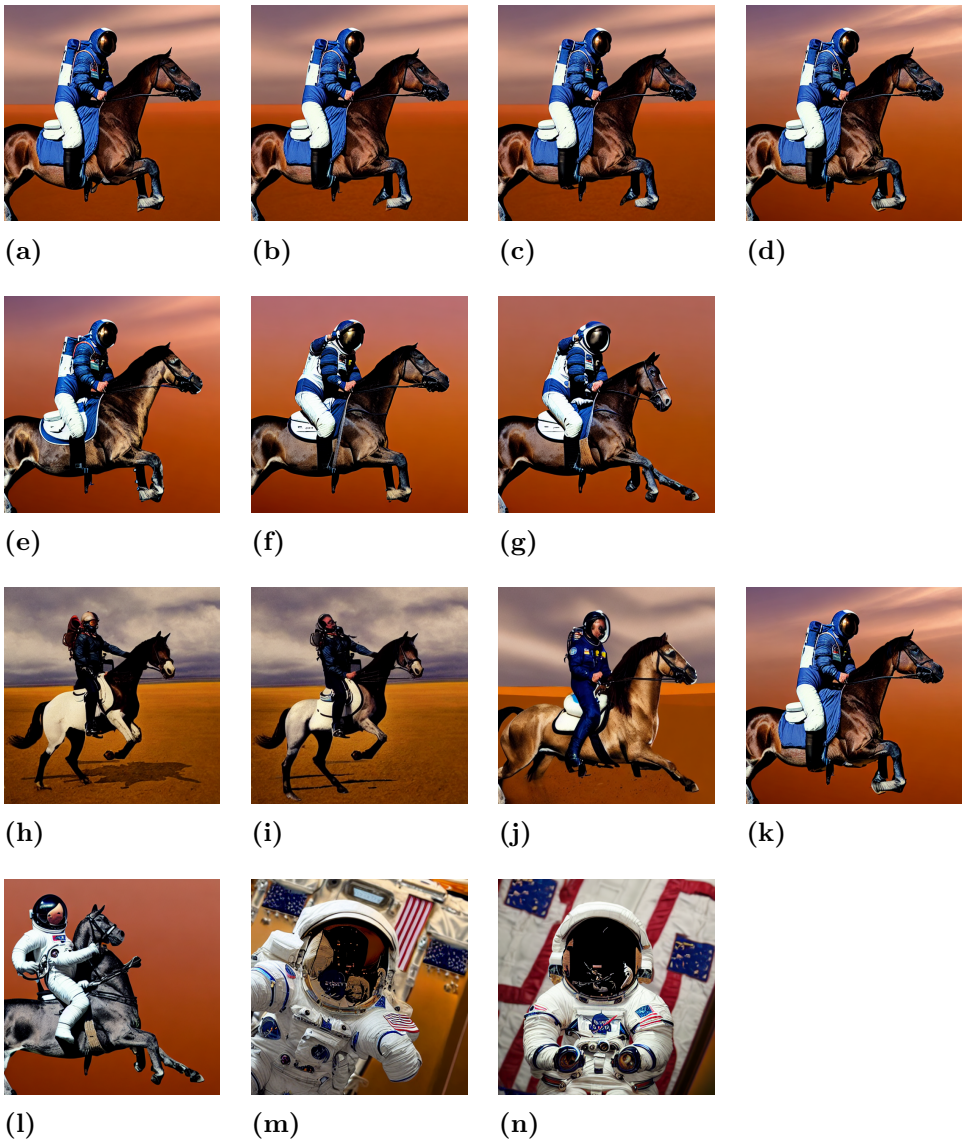


Figure A.5: Other image generation results with different epsilon values using the PNDM scheduler. The top two rows show the embeddings adjusted for the word: "photograph" and the bottom two rows show the embeddings adjusted for the word: "astronaut". For each set of images from left to right: (a) $\epsilon = -0.6$, (b) $\epsilon = -0.4$, (c) $\epsilon = -0.2$, (d) $\epsilon = 0.0$, (e) $\epsilon = 0.2$, (f) $\epsilon = 0.4$, (g) $\epsilon = 0.6$.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY