Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

# Visual Parameter Space Exploration for AI Art Design

Julian Hummel

| | |
|---|---|
| **Course of Study:** | Informatik |
| **Examiner:** | Dr. Steffen Koch |
| **Supervisor:** | Dr. Kuno Kurzhals,<br>Jena Satkunarajan |
| **Commenced:** | April 27, 2023 |
| **Completed:** | October 27, 2023 |

## Abstract

Generative AI gained great popularity in recent years and it is important that the users understand how the input parameters of such models relate to the generated output. The term *parameter space exploration* describes the systematic variation of model input parameters and the generation of the corresponding outputs which can then be used to better understand the relations between the parameters and the outputs. With this work, the *DiffusionExplorer* is proposed, incorporating a visual and interactive framework for parameter space exploration in the context of latent diffusion models. The *Iterative View*, the *Projection View* and the *Pipeline View* build up the major views of the DiffusionExplorer, while the *Iterative View* offers the user the iterative refinement of a given image by tuning the model parameters. The *Projection View* embodies the visualization of image samples by using a 2D projection based on similarity metrics. The last view, namely the *Pipeline View* provides the user with an image history consisting of nodes and edges representing the generation steps. In order to allow a seamless integration of drawing and other editing techniques, the tool is integrated into the painting program Krita via a plugin. After the implementation phase, the DiffusionExplorer has been put to the test, being evaluated with a user study, proving the effective application of the DiffusionExplorer in the context of AI art design. Therefore several participants successfully completed a given task using the tool in combination with Krita and accomplished the synthesis of images that could not be generated in a single synthesis step with a classical approach.

# Contents

# 1 Introduction

In recent years, the field of digital art design has undergone a significant transformation due to advancements in machine learning techniques. One notable breakthrough is the ability to synthesize new image content from text prompts, which has opened up exciting possibilities for generating images solely based on textual descriptions or in conjunction with existing images [RBL+21]. Recent developments, such as Latent Diffusion Models [RBL+22] make image synthesis more and more accessible due to the efficient generation of images. However, despite this technological leap, creating images that precisely meet the expectations of users remains a challenging task that often requires additional human involvement. The complexity arises from the multiple facets of parameter tuning that are essential for adjusting image results to achieve a more satisfying outcome. Each alteration of an individual parameter can potentially lead to significant changes in the resulting image, making it a time-consuming process. Artists and designers often find themselves engaged in multiple generation iterations, manually inspecting and comparing image results to arrive at the desired output. This tedious process hinders the creative flow and slows down the art production cycle. The primary objective of this thesis is to address this challenge and develop an innovative approach that facilitates interactive image synthesis, offering the user to systematically explore the input parameter space.

As already touched on before, the tuning and exploration of the input parameters in the context of latent diffusion models represents a challenge especially when the user tries to understand the relationship between the input parameters and the model output. This work focuses on the field of art design with the help of generative artificial intelligence, which will be referred to as AI art design in the following chapters. Therefore, the core contribution is the implementation of a framework for image synthesis with latent diffusion models that incorporates the exploration of the input parameters as well as a 2D embedding of the generated images using projections. The resulting framework will be integrated into the open-source painting program Krita via a plugin, to enable the user to perform image composition and editing techniques on the generated images. After the elaboration of the requirements and the implementation phase, a user study serves as the foundation of the evaluation of the DiffusionExplorer.

Starting with Chapter 2, this part aims to build a technical foundation for the topics of latent diffusion models and projection using similarity metrics as well as a methodology for the design process of the tool. The background chapter is followed by Chapter 3, which summarizes existing implementations and scientific work related to topics such as parameter space exploration and frameworks for latent diffusion models. Before implementing the software solution, the process of requirements gathering and design prototyping is covered in Chapter 4, while Chapter 5 gives a brief overview of the painting program Krita and the underlying implementation details of the tool. Covering the evaluation and the subsequent discussion of the results, Chapter 6 and Chapter 7 highlight a user study followed by the discussion of the initial requirements of the tool and the results of the user study. Finally, Chapter 8 summarizes this work and gives an outlook on possible future iterations and improvements of the tool.

# 2 Background

The background chapter serves as the foundation for this thesis, providing a comprehensive overview of the key concepts and techniques that form the basis of the proposed DiffusionExplorer. It covers various topics such as *similarity metrics* in Section 2.1 and *machine learning* in Section 2.2 building the foundation for the embedding of images in a 2D plane covered by *dimensionality reduction* in Section 2.3. As the tool integrates image synthesis models, Section 2.4 introduces *generative adversarial networks (GANs)* and Section 2.5 gives an overview on *Latent Diffusion Models* that are used for the image generation in the tool. The *design study methodology* section introduces a framework to gather the requirements for the tool and to develop possible design solutions.

The following sections Section 2.1, Section 2.2 and Section 2.3 introduce basics and techniques that enable the visual presentation of sampled output images to the user. Techniques such as dimensionality reduction based on similarity metrics provide a framework to solve the problem of mapping high dimensional input data to lower dimensional data. In the context of this work this relates to the 2D arrangement of generated output images in an intuitive way.

## 2.1 Similarity Metrics for Vectors

Many machine learning models depend on the distance between two data points in order to predict an output. When working with vectors, similarity metrics provide a way to quantify the similarity or dissimilarity between two vectors. In our context, a similarity metric describes the distance between two vectors, where a small distance is associated with a high degree of similarity of the features. The vectors can represent different kinds of data, such as parameters, text documents and images. Some commonly used similarity metrics are the Euclidean distance, the Manhattan distance and the cosine similarity, while the advantages and disadvantages of the metrics depend on the use case. This will be addressed at the end of this section. Figure 2.1 shows the visualization of the previously mentioned three distance metrics for the two dimensional vectors $a$ and $b$, while both vectors might for example represent the values of two parameters. The Euclidean and the Manhattan distance can be derived from the *p-norms* $\|.\|_p$ which is a generalized distance metric. Let the vector $x = (x_1, x_2, \ldots, x_n)$ then the p-norms is defined as follows:

$$\|x\|_p := \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}, 1 \leq p < \infty \tag{2.1}$$

For a second vector $y = (y_1, y_2, \ldots, y_n)$ and by setting $p = 1$ we derive the Manhattan norm which is also referred to as the L1-norm $\|.\|_1$ [AHK01]. Thus, the L1-norm of the difference $x - y$ is defined as:

$$d_{\text{manhattan}} = \|x - y\|_1 = \sum_{i=1}^{n} |x_i - y_i| \tag{2.2}$$

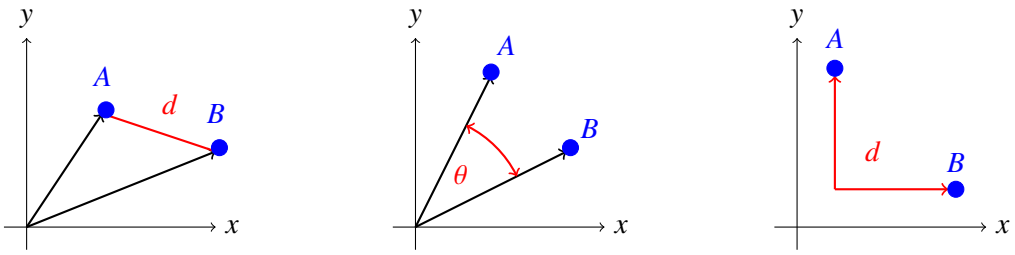
Similarly for the the Euclidean distance we fix $p = 2$ which is the L2-norm $\|.\|_2$ and for the L2-norm of the difference $x - y$ we get:

$$d_{\text{euclid}} = \|x - y\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.3}$$

The cosine similarity is computed as:

$$\text{cosine similarity} = cos(\theta) = \frac{x \cdot y}{|x||y|}, \tag{2.4}$$

for $x \cdot y$ being the dot product of the two vectors $x$ and $y$ [SPTS19]. As already touched upon in the introduction of this section, some similarity metrics may perform better than others in specific scenarios.




**(a) Euclidean Distance:** The length of the line segment between the two points A and B.

**(b) Cosine Distance:** The angle $\theta$ between the two vectors $a$ and $b$.

**(c) Manhattan Distance:** The distance between $A$ and $B$ measured along the axes.

**Figure 2.1: Distance Metrics:** Three different distance metrics to measure the distance between two vectors.

Aggrarwal et al. [AHK01] present the behaviour of distance metrics in high dimensional space. They compared the performance of distance metrics based on the p-norm for varying values of p applied to high dimensional input data and discovered a major trend of performance loss with increasing values of p. Looking at the example of the *K-Means* clustering algorithm [HW79], the authors state a significant decrease for the classification rate when moving from $p = 0.5$ to $p = 1$ [AHK01]. Besides, for two similar document vectors with a lot of entries, where the Euclidean distance indicates a low similarity due to the size of the vectors, the cosine similarity can yield better results as the angle $\theta$ between the vectors is still small. In this research, the cosine similarity metric will be applied to create projections based on parameters and image vectors in the context of image synthesis with latent diffusion models (Section 2.5). Libraries such as *scikit-learn* [PVG+11] offer implementations to compute the pairwise metrics for a given set of vectors, which can later be used as an input for dimensionality reduction algorithms. This topic will be covered in Section 2.3.

## 2.2 Machine Learning

Machine learning is a subset of artificial intelligence that involves the development of algorithms and models to make predictions for data that has not been part of the training data. The intention is to achieve a desired behavior of the model, such as making appropriate decisions in a given

application environment which is not directly programmed but learned based on input data [Die03]. The machine learning workflow can be split into two main processes: Training and inference. In some settings the latter is also called prediction, while this thesis will make use of the term inference. In the training phase, the model is fed with data which is then used to learn parameters that influence the decision making of the model. A trained machine learning model is usually deployed in an application environment where it used for inference. In the inference phase the model is used to make predictions on new, unseen data. The input data is passed through the model and an output is generated as the result of the inference. In general, machine learning tasks can be classified as one of the following three types: supervised, reinforced and unsupervised [Die03] and of special interest for this work is unsupervised learning. Unsupervised learning, which is also known as unsupervised machine learning, aims to discover structures and patterns in data sets without the necessity for labelled training data. Common applications of unsupervised learning are clustering and dimensionality reduction algorithms. Clustering describes a technique to group unlabeled data based on given metrics such as similarity. On the other hand, dimensionality reduction is used in situations where it promises new insights into data sets, which would be difficult to gain in the high dimensional space [MPH07]. Section 2.3 highlights some of the core points of dimensionality reduction, as it is utilized in the approach of this work.

### Deep Neural Networks

*Deep neural networks (DNNs)* are a specific type of *multilayer neural networks* which contain at least one hidden layer. The term deep in DNN refers to the depth of the neural network which is characterized by the number of layers, while DNNs consist of an input layer, one or multiple hidden layers and an output layer. The intermediate layers between the input and output layer are referred to as hidden layers as they are not visible to the user [Agg18]. Figure 2.2 shows an example of a fully connected DNN with two hidden layers. This network architecture is also called *feed-forward network*, as the input is fed subsequently through the layers from left to right until reaching the output layer. Another type of deep neural networks are so called *convolutional neural networks (CNNs)*, which are a special type of feed-forward networks and are used in computer vision for image classification and object detection. For each layer in a CNN a *convolution operation* is defined, which is a filter that gets the activations of the current layer as input and maps them to the next next layer. Layers in CNNs can be used to extract simple as well as complex features from the input data such as lines and other primitive or complex shapes [Agg18]. Feed-forward networks find their application in Generative Adversarial Networks (GANs) where the generator usually is a feed-forward network (see Section 2.4).

## 2.3 Dimensionality Reduction

The problem that dimensionality reduction (DR) algorithms aim to solve is the high dimensionality of data, such as parameter spaces and digital photographs, which is complex or even impossible to visualize and to interpret in a meaningful way. Dimensionality reduction describes the transformation of high-dimensional data into low-dimensional data which is usually two or three-dimensional [MPH07], [MH08]. For the purpose of this work, we will focus on two kinds of high-dimensional input data: Parameters, which are used to guide the image generation process and image samples, that

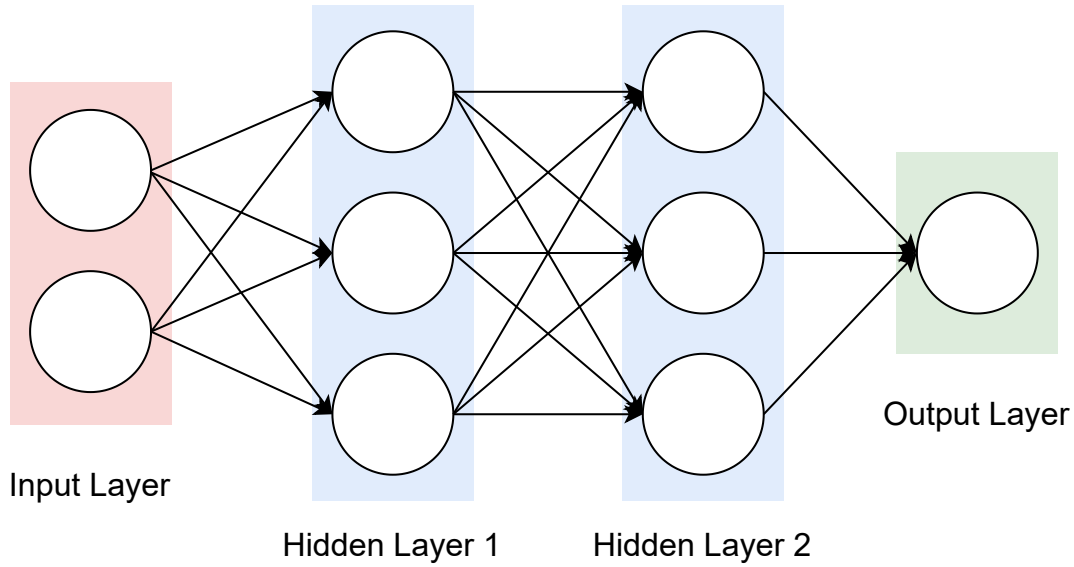



**Figure 2.2: Deep Neural Networks**: The visualization of a simple deep neural network consisting of an input layer, two hidden layers and an output layer. The layers are fully connected such that each neuron is connected to all neurons of the subsequent layer.

are generated based on these parameters. Let $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ be the high-dimensional input data set. The parameter or image vectors are not directly used as input for the DR algorithm, as we are interested in the similarity of the vectors. The similarities build up the high-dimensional dataset $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ which is transformed into a interpretable representation $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$ while preserving as much of the similarity properties as possible [MH08]. In our case $\mathcal{X}$ is transformed into a 2D representation and visualized in a 2D plane, where an increase of the distance of the elements is interpreted as a decrease of their similarity. For the application in this work, we can divide the whole process into two steps: The computation of the similarity of the input vectors and the projection of these similarities into a 2D plane using dimensionality reduction. Figure 2.3 shows a simplified visualization of these two steps. As already covered in Section 2.1, there exist several distance metrics that can be used to compute the similarity between vectors. The similarity metric of choice is then computed pairwise for all input vectors $a_i \in \mathcal{A}, 1 \leq i \leq n$ and fed into the DR algorithm. An example is shown in Figure 2.3, where the vector entry $x_{1,2}$ denotes the similarity value of the two input vectors $a_1, a_2 \in \mathcal{A}$. Following the work of Kurzhals [Kur21], DR techniques such as t-distributed Stochastic Neighbor Embedding (t-SNE) [MH08] and Uniform Manifold Approximation and Projection (UMAP) [MHM20] perform well for the purpose of creating a 2D embedding of image samples with visible clusters. Both techniques favor the preservation of local distances over global distances [MHM20]. In the following, the core properties of these two techniques will be highlighted.

### T-distributed Stochastic Neighbor Embedding

T-distributed stochastic neighbor embedding is a nonlinear dimensionality reduction technique which has been proposed by Maaten and Hinton in [MH08]. The technique is based on stochastic neighbor embedding (SNE) [HR02] and can be used to visualize similarity data. For two data

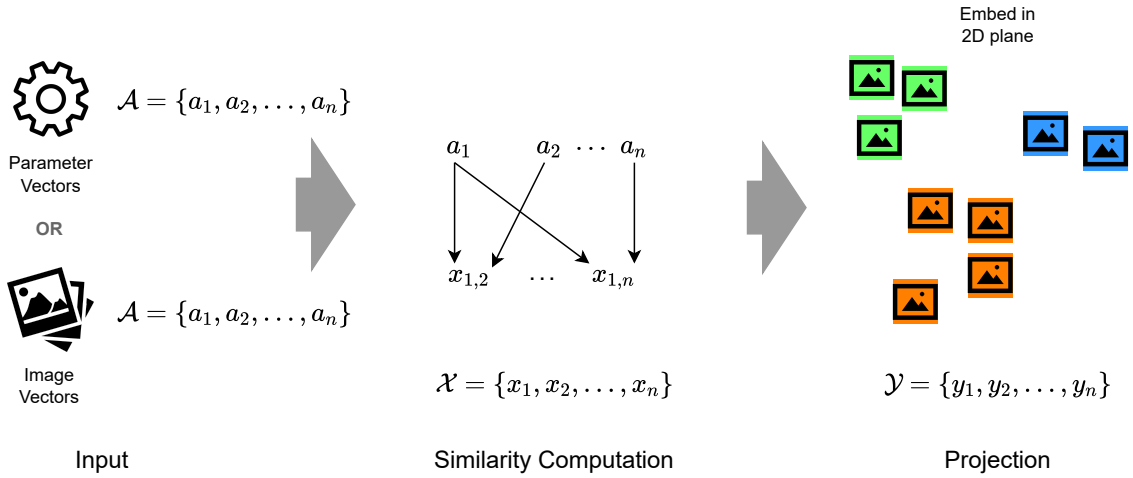



**Figure 2.3:** The process of preparing the input vectors for the application of dimensionality reduction in order to embed the feature vectors in a 2D plane. In the first step, the pairwise similarities of the input vectors are computed and then fed into the dimensionality reduction algorithm to create a 2D embedding. The embedding is used in the *Projection View* of the DiffusionExplorer where the image samples are projected onto the 2D plane.

points $x_i, x_j \in \mathcal{X}$ in the high-dimensional space, SNE assigns the high-dimensional distance to a conditional probability $p_{j|i}$ which represents the similarity between the $x_i$ and $x_j$. The value of $p_{j|i}$ describes the probability that $x_i$ picks $x_j$ as its neighbor, while $p_{j|i}$ is relatively high for data points that are close to each other. For the two corresponding data points $y_i, y_j \in \mathcal{Y}$ in the low-dimensional space a similar conditional probability $q_{j|i}$ can be formulated and analogously this conditional probability should be relatively high for similar data points $y_i$ and $y_j$. Intuitively, if the data points $y_i$ and $y_j$ correctly represent the similarity between the high-dimensional data points $x_i$ and $x_j$ we get $p_{j|i} = q_{j|i}$. SNE aims to minimize the difference between the two conditional probabilities by minimizing the sum of the *Kullback-Leibler divergences* over all data points with the gradient descent approach [MH08]. As presented by the authors, some problems arise with the usage of the SNE technique which are the difficulty to optimize the cost function and what the authors refer to as the "crowding problem". The latter basically describes the tendency of SNE to crowd data points together in the center of the map, which makes it more difficult to identify structures in the data. t-SNE provides a solution, as it uses a different cost function compared to SNE and a Student-t Distribution instead of a Gaussian to compute the similarity between two low-dimensional data points. These two modifications address both the optimization problem of the cost function and the crowding problem of the SNE technique [MH08].

### Uniform Manifold Approximation and Projection

Another dimensionality reduction technique used in this work, that is competitive with t-SNE in regard to the quality of the visualization, is Uniform Manifold Approximation and Projection (UMAP). UMAP has been proposed by Healy and Melville in [MHM20] and is characterized by the ability to preserve the global structure of the high-dimensional input data [MHM20] and the

advantage of a better run time performance compared to t-SNE [Kur21]. The UMAP algorithm starts with the creation of a high-dimensional graph representation of the input data, where each data point is connected to its nearest neighbors. It then aims to reduce the discrepancy between the pairwise relationships in the high-dimensional and low-dimensional embedding by applying stochastic gradient descent. Due to its flexibility in handling different types of data sets and in choosing the dimensions of the low-dimensional representation, UMAP can be applied to a wide range of problems, such as the visualization of high-dimensional data in two or three dimensions [MHM20].

The last sections covered several metrics for the similarity computation of vectors as well as two popular dimensionality reduction techniques that can be used to create a low-dimensional embedding for the high-dimensional similarity data. These topics are especially important for this work, as they allow the visualization of image samples on a 2D plane which makes it easier for a user to interpret large numbers of image samples and to identify similar samples based on different metrics. Going further, we will now take a closer look at techniques for the generation of these image samples that we are embedding in the low-dimensional space. Before addressing machine learning models such as latent diffusion models, the following section aims to build a foundation for the topic of generative machine learning models.

## 2.4 Generative Adversarial Networks

Generative Adversarial Networks, in short GANs are an unsupervised machine learning task that makes use of deep learning methods such as convolutional neural networks (CNNs). They have been one of the most successful models for image generation in the last years and build a foundation for several computer vision tasks. Figure 2.4 shows the framework of GANs, which consist of two networks: The generator and the discriminator. The generator, which is a feed forward network, gets noise as input and aims to reproduce the original images without noise. The discriminator however is a classifier that aims to differentiate between the original and the reproduced images and thus both networks are trained adversarilly. This means that the generator and discriminator are in a constant battle, as the generator tries to produce data that is realistic enough to fool the discriminator, while the discriminator tries to get better at distinguishing real from fake data. As a consequence, the generator can improve the generation of fake images when the discriminator correctly identified a fake image from the generator. Summarizing this, the objective of the discriminator is minimization of the classification error, while the generator aims to maximize the classification error of the discriminator. As a result of this training process, the generator learns to generate realistic, high-resolution images [ML21]. One usually wants to generate a wide variety of output samples when applying generative models such as GANs. A well-known problem of GANs is the failure mode called "mode collapse", which results in the problem that the generator only rotates through a small set of output types. This is caused by the generator, which might only learn to produce a certain output if that output is plausible. By construction, the discriminator aims to reject outputs that get generated over and over again, but if the discriminator gets stuck in a local minimum, the generator of the next iteration might find a way to generate a plausible input for the current discriminator. As a result, the discriminator never manages to overcome this cycle and the GAN fails [TTV18]. Another problem of GANs concerns the stability of the training when applied in large scale GANs and Brock et al. [BDS18] describe such instabilities for both the generator and discriminator.

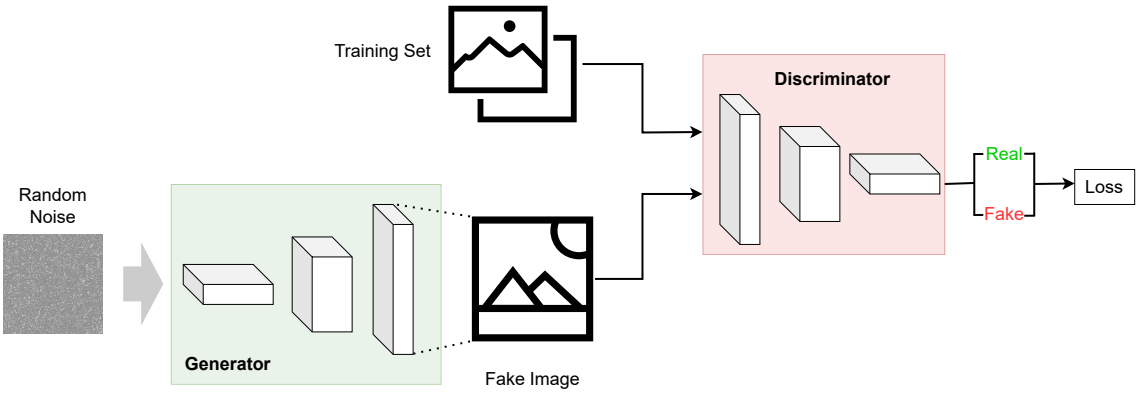



**Figure 2.4: Framework of GANs:** GANs are divided into two networks, the generator and the discriminator. The generator creates fake samples based on random noise input. Both networks are trained adversarially, such that the discriminator aims to differentiate between real samples from the training set and fake samples generated by the generator. Through the training process, the generator learns to synthesize photo-realistic and high-definition images.

Having impressive recent developments, the synthesis of high-definition images is a popular application in the computer vision field. One of the main problems is the high computational demand of the training and inference process of the machine learning models that are used for the synthesis. Especially when synthesizing highly detailed images, such as nature scenes with a high pixel density, the great computational effort leads to comparatively long inference times.
One of the latest advancements called latent diffusion models are an approach to solve this problem.

## 2.5 Latent Diffusion Models

Latent diffusion models are deep-learning models that can be used to generate high-definition images based on textual descriptions as well as for image inpainting and several other tasks. The model has been developed as a collaboration of the CompVis group, Stability AI and Runway and has been released in summer 2022. In contrast to GANs, the more recent diffusion probabilistic models (DMs) [SWMG15] don't exhibit mode collapse or training instabilities and they are capable of modeling highly complex distributions of detailed natural images without the use of billions of parameters [RBL+22]. However, DMs are still computationally demanding as the training and evaluation is taking place in the high-dimensional space of RGB images and thus a high number of function evaluations as well as gradient computations need to be done. In their work [RBL+22], Rombach et al. introduce the "Latent Space", which serves as the training ground for Diffusion Models (DMs) and also enables efficient image generation. Figure 2.5 shows a high-level overview of how Latent Diffusion Models (LDMs) work. In the first step, a compact representation in the low-dimensional latent space of the image is extracted using the encoder $\mathcal{E}$ as shown in the top left in Figure 2.5. In the next step described be the diffusion process, Gaussian noise is added to the image and the resulting $z_T$ representation is then passed through a U-Net which is a convolutional neural network. Finally, after the cross-attention based conditioning, the output is transformed back

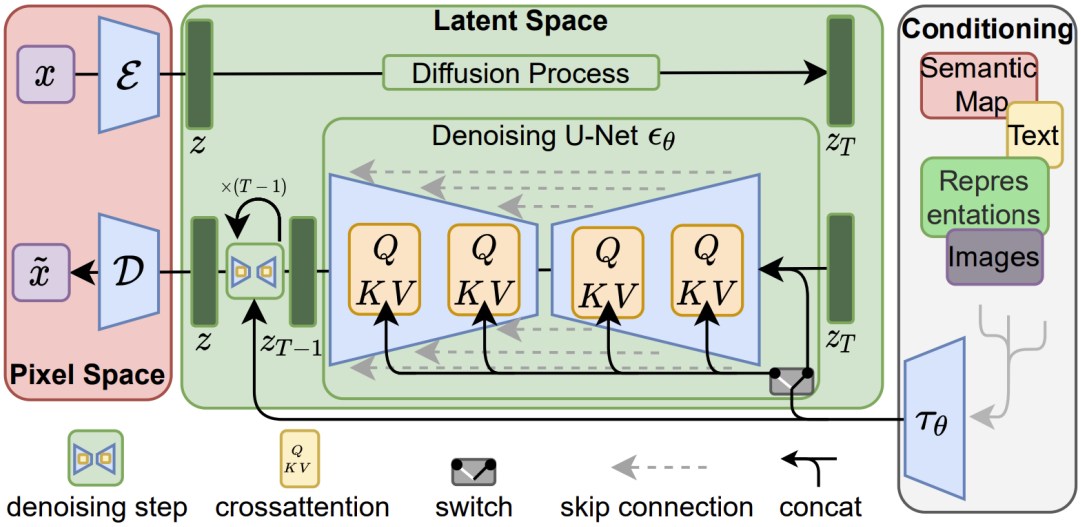



**Figure 2.5: Latent Diffusion Models:** A high-level overview on the synthesis process of latent diffusion models [RBL+22]. The diffusion process takes place in the "Latent Space", which is low-dimensional and allows a much more efficient generation of images as if the diffusion process would take place in the high-dimensional pixel space. *Note: Reprinted from [RBL+22].*

into the pixel space using the decoder $\mathcal{D}$. To guide the image synthesis process, several parameters can be adjusted including the cfg scale, the denoising strength and the seed. The cfg scale can be semantically interpreted as the text prompt guidance, controlling to what extent the text prompt should be considered for the image generation, while in the context of image-to-image synthesis, the denoising strength controls the influence of the input image. Finally, the seed can be used to get variations for the output samples. As already highlighted previously, these parameters will be used for parameter sampling in order to allow the user to explore the input parameter space of the latent diffusion models.

After covering some necessary technical background including similarity metrics and machine learning models for dimensionality reduction and image synthesis, the final section of this chapter describes the *Design Study Methodology* which guides the requirements for the proposed tool, as well as the visual solution finding process of this tool.

## 2.6 Design Study Methodology

This section addresses the first phase of the creation process of this tool, which consists of collecting the requirements and constraints as well as solution ideas for the problems that the proposed approach of this work aims to solve. As the core contribution of the tool is a visualization solution for the application of latent diffusion models in the process of AI art design, we will have a closer look at how to conduct design studies.

### About Design Studies

Design studies are a form of problem-driven visualization research, aiming to analyze problems that occur in the daily work of domain experts. Based on this analysis, design solutions are derived in collaboration with the domain experts and finally, the design is validated [SMM12]. Figure 2.6 shows one possible overview of how a design study might be carried out, where two main building blocks describe the process of the design study. The first block consists of the *Empathize* and *Define* phases, which are crucial for gathering the necessary information and insights that will guide the design process. During the empathize phase, the designer seeks to understand the domain experts' needs, desires and pain points with the current state of their daily work. In the define phase, the designer synthesizes the information collected during the empathize phase to create a clear problem statement that guides the subsequent phases. The second block is built from the *Ideate*, *Prototype* and *Test* phases. During the ideate phase, the designer generates a wide range of ideas and solutions that address the problem statement defined in the first block. In the prototype phase, the designer creates low-fidelity versions of the most promising ideas, allowing for rapid iteration and refinement. These low-fidelity prototypes might be represented by simple sketches, that can also be drawn together with the domain experts. Finally, during the test phase, the prototypes are tested with real users to gather feedback and insights that inform further iteration. This block emphasizes creativity, experimentation, and user feedback, and provides a structured approach to quickly developing and testing potential solutions [S23]. The figure originates from a method called *Design Thinking*, while the creation flow of the DiffusionExplorer will be strongly based on it. To complement this chapter about design studies the following subsection will briefly highlight important aspects of the single phases, whereby [SMM12] offers a great summary of this topic.

### The Learnings from Design Studies

In their work [SMM12] Sedlmair et al. investigated twenty-one different design studies and proposed a methodological framework for design studies. Although the framework that has been described there is not the same as in Figure 2.6, the concepts still apply. The first building block containing *Empathize* and *Define* is reflected by what the authors call the *discovery stage*, where the problem is characterized and abstracted. One key learning that is stated in this stage is the importance of not only focusing on the problem itself but also on the successful aspects of existing solutions in the workflow of the domain experts. Another main point that is mentioned by Sedlmair and colleagues is to control the initial discussions with the domain experts in such a way, that they don't focus on their own visions of possible visualization solutions and explain the problems that they are facing. Considering the second building block, the three phases *Ideate*, *Prototype* and *Test* can be associated with the so-called *design stage* in [SMM12]. The authors found out that making use of dual representations (swapping nodes and edges) when sketching ideas led to an efficient way for domain experts to model the outputs of algorithms. In addition, the importance of having a broad consideration space of possible solutions is mentioned for the design stage. This consideration space is then filtered down to what the authors call a narrow proposal space, which should be discussed with the domain experts. The final results of the discussion might be final design solutions that will be implemented [SMM12].

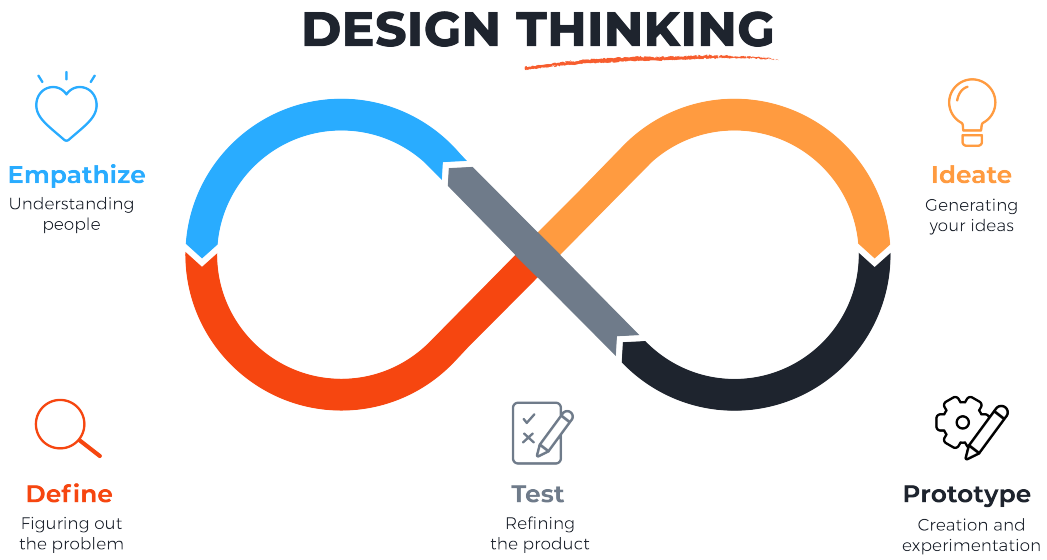



**Figure 2.6: Design Thinking:** The method of Design Thinking is one possible way to conduct a design study. It aims to solve real-world problems of domain experts with a visualization approach. As shown above, the process can be divided into five phases while *Empathize* and *Define* build the foundation for the identification of the problem statement. The *Ideate*, *Prototype* and *Test* phases aim to generate ideas based on the problem statement, to create prototypes and to test these prototypes in a feedback loop that connects again to the *Empathize* phase. *Note: Reprinted from [S23].*

# 3 Related Work

This chapter aims to give an overview of existing publications and implementations related to the topics of this work. This consists of a popular user interface for the usage of stable diffusion, work on visual parameter space analysis and further topics related to generative image models such as prompt engineering. While existing publications in the field of diffusion models and generative image models primarily emphasize prompt engineering for text-to-image generation in order to optimize the output of the models, this work takes a distinct direction by shifting the focus towards the inclusion of visual parameter space exploration in the topic of latent diffusion models. While Section 3.1 covers an existing user interface for stable diffusion, the following Section 3.2 presents the idea of visual parameter space analysis, which aims to enable the user the exploration and the analysis of the model parameters using visual interaction techniques. Finally, Section 3.3 summarizes publications related to the topics of prompt engineering and design space exploration.

## 3.1 Stable Diffusion Web UI

Stable Diffusion Web UI [AUT22] also referred to as AUTOMATIC1111 is a quite powerful web-based user interface for stable diffusion that provides the functionality to run both text-to-image and image-to-image generations. Figure 3.1 shows the text-to-image view of the Web UI which has been used to generate an image based on the given text prompt. It is also possible to provide a negative prompt as well as a set of parameters including the cfg scale and the random seed. Additionally, the tool enables the user to follow a certain workflow by providing portability of the images generated in the text-to-image tab. After generating an initial image in this tab it is possible to further modify this image in the image-to-image tab for multiple iterations until a desired output is generated.

Although the tool offers a variety of features and parameter options, this might be overwhelming at the beginning and it is difficult to understand the relations between the input parameters and the generated output images. Especially for new users, there is no functionality provided that allows the exploration of the input parameter space and the visual analysis of how different parameter constellations relate to output samples. While the tool offers sampling functionalities, the user still has to define scripts in order to make use of sampling which makes it quite complex for an inexperienced user.

## 3.2 Visual Parameter Space Analysis

Many machine learning models are guided by a set of input parameters where it is often challenging to find values for the parameters that influence the outputs of the model in the intended way. Although some parameters have a semantic interpretation it is often difficult to understand how

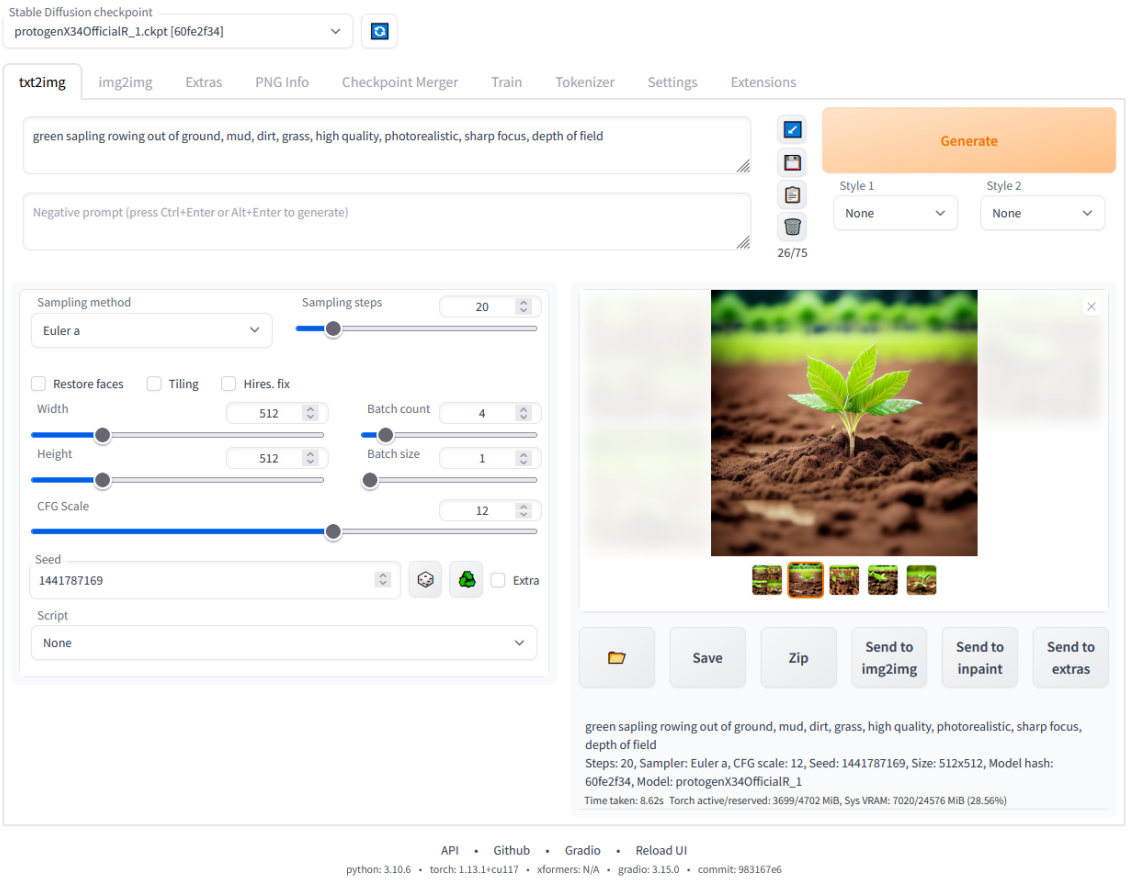



**Figure 3.1: Stable Diffusion Web UI:** The browser interface of the Stable Diffusion Web UI offers functionalities such as text-to-image and image-to-image generation using stable diffusion models. The user can control various model parameters such as the cfg scale and the random seed and the generated images can be transferred between the different tabs of the user interface. These tabs offer additional features including inpainting, which can be used to fix defects in images.

a parameter constellation interacts in regard to the output of the model. In their work, Sedlmair et al. describe a framework for *visual parameter space analysis (vPSA)* [SHB+14], which is a visual analytics technique that can support the validation and the usage of (simulation) models by applying both visual and automatic methods [PBM]. The models that are analyzed are represented by *input-output models* that map a set of input parameters to a set of outputs. Input and output parameters can be classified as either *multivariate/multidimensional* or *complex*, where the first category can contain semantically meaningful input parameters and the latter might be an image that is seen as a single complex input parameter. Figure 3.2 shows an example of an input-output model consisting of four input parameters, one of them being a complex input parameter. Following the definitions given by M. Sedlmair et al. we can then formulate the term *parameter space analysis (PSA)* as the systematic variation of model input parameters and the generation of the corresponding outputs in order to understand the relations between the input parameters and the outputs. In the context of this work, PSA is supported by interactive visualization which is referred to as the already-mentioned concept of vPSA. Although the tool that is implemented with this work does

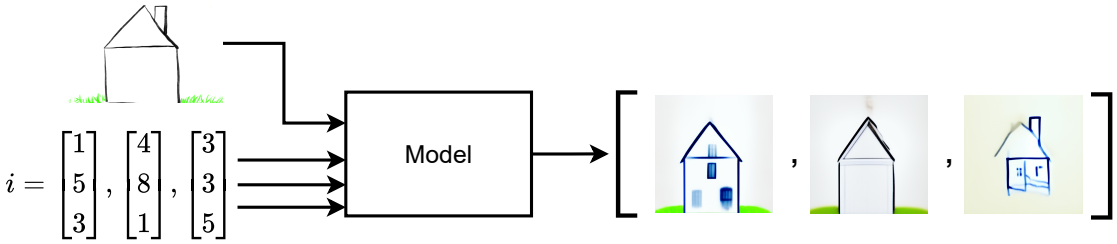



**Figure 3.2: Sampling a model:** The figure shows a model that has a complex input which is an image and three additional input parameters. The model is sampled three times for different input parameter constellations and the same input image, generating an output image for each parameter constellation.

not cover the whole process of vPSA some concepts will be applied in order to better understand the relations between the input parameters and the outputs of the latent diffusion models, which is one of the core contributions of this work. This enables the user to explore the parameter space consisting of parameters such as the cfg scale and the seed and to understand how these parameters influence the output of the latent diffusion models. While Section 2.5 was dedicated to the details of latent diffusion models, the model will be treated as a black box *input-output model* for the purpose of this topic. The process can be divided into two steps: The variation of the input parameters which will be referred to as parameter sampling and the interactive visualization of the outputs that are generated by the model for the sampled input parameters, which will be briefly covered in the following two sections.

### Parameter Sampling

The systematic sampling of the input parameter space is a crucial part of the vPSA process. Coming back to Figure 3.2 one can see that the model is sampled for three different parameter constellations which generate three corresponding output samples. As described in [SHB+14], either regular or stochastic sampling strategies are mostly used to generate parameter constellations. A commonly used regular sampling strategy is *Regular Cartesian sampling*, which is also used in the context of experimental design. Supposing a model has two different parameters $A$ and $B$, which are also denoted as factors, cartesian sampling would generate all possible combinations of parameter pairs. $A$ and $B$ with possible values $\{A_1, A_2\}$ and $\{B_1, B_2\}$ would lead to the samples $\{(A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2)\}$ [SHB+14]. A popular example of a stochastic sampling strategy is the so-called *uniform random sampling*, where the parameter values are randomly picked from a given range of possible values. Unlike cartesian sampling, which systematically covers the whole parameter space, uniform random sampling randomly selects parameter values without considering the specific combinations of those. In this work, uniform random sampling will be used for the generation of image samples.

### Navigation Strategy

After the sampling of the input parameters and the generation of the corresponding outputs, these outputs must be presented to the user in a manner that allows exploration and analysis of the parameter space. Sedlmair et al. classify four strategies that can be used for the navigation through the input parameters and the generated outputs, while we will focus on the so-called *local-to-global* navigation strategy. This strategy relies on the pre-computation of samples before starting with the parameter space analysis in order to prevent long waiting times of the user [SHB+14]. Depending on the model and the use case, the number of pre-computed samples may vary and is also limited by the chosen visualization of the output samples. The idea of this navigation strategy will be applied to this work, as it enables the visual presentation of various output samples to the user, from which he can then select the most appropriate one.

## 3.3 Prompt Engineering and Design Space Exploration

Prompt engineering refers to the process of strategically designing text prompts to generate desired outcomes from language models, specifically in the context of text-to-image models. In their work [LC21], V. Liu and L. Chilton conducted a study in order to identify text prompt components and model parameters that yield the generation of coherent model outputs. The study consisted of several experiments, including the permutation of prompts, the variation of the random seed and the investigation of the length of the optimization. In the first experiment, they started by comparing the outcomes of text prompts with only small differences such as the usage of different words with the same semantic meaning. As a result of the experiment they stated, that there is no significant variation in the results when rephrasing the text prompt. Thus when trying to optimize the output of the text-to-image generation the focus should lie on the keywords rather than on the rephrasing of the text prompt. In the second experiment, which covered the variation of the random seed that is used for the generation they started with the hypothesis that the variation of the random seed would have no significant impact on the quality of the generation output. However, in their experiment they used 10 different randomly generated seeds and they found out, that there exist significant quality differences between the outcomes of the generation when varying the random seed for the same text prompt. The authors conclude that in the process of prompt engineering before adapting the text prompt one should consider to vary the random seed first. The third experiment that is of interest for this work and that has been executed in the scope of the study is about the length of the optimization which is described by the number of iterations for which the model is executed. The question that the authors encountered is whether a higher number of iterations yields better-evaluated generations. As a result, they stated that in the range between 100 and 1000 iterations, a higher number of iterations did not significantly increase the quality of the generations. They conclude, that for the purpose of faster iteration, it would be valid to choose a lower number of iterations such as 400. To summarize the results of this work, varying the seed as well as choosing a number of iterations in the magnitude of 400 are good approaches for prompt engineering whereas rephrasing the prompt with similar semantic meanings does not significantly optimize the generation results. By applying the results from the just presented work on prompt engineering to this thesis, the variation of the random seed and the iterations represent feasible approaches for the optimization of the generation results. While the work by V. Liu and L. Chilton includes the variation of parameters such as the

random seed, it does not provide an approach for the systematic exploration of such parameters. This is where this work comes into place, which aims to provide a tool for the visual parameter space exploration as described in Section 3.2.

Another work, that also focuses on the improvement of text prompts in the context of text-to-image synthesis has been presented by Feng et al. [FWW+23] with the so called PromptMagician. Among other things, the PromptMagician helps the user to refine his text prompts by initially sampling images for a given text prompt of which the user can select an image with the desired style. The tool then proposed to the user additional keywords that match the selected style in order to improve the generation results. In contrast to the PromptMagician, this work focuses on the exploration of the model parameters of latent diffusion models rather than on the improvement of the text prompts. However, both the PromptMagician and this work share similar approaches by including sampling and projection for the visual presentation of image samples.

With their tool called generative.fashion [DWJ+23], Davis et al. present a GAN-based fashion design tool that is based on principles of design space exploration and aims to improve the support for creativity in the design process. For the purpose of exploring the latent space of a deep generative model, the implementation of generative.fashion offers a panel where the users can move generated images along meaningful directions in the latent space in order to intentionally explore the latent space of the GAN. Related to the idea of the presented work, this thesis focuses on the parameter space exploration of latent diffusion models using sampling and projection functionalities.

# 4 Design

This chapter describes the creation process of the design concept as well as the final design that is implemented by the DiffusionExplorer. The creation process consists of multiple steps, consisting of two interview phases that aim to elicit the fundamental requirements of the software solution as well as to develop visual solutions. As the tool is created in the context of this bachelor thesis, the main input for the interviews and the derived requirements originates from the two supervisors and two employees of the same institute of visualization and interactive systems (VISUS) of the University of Stuttgart. Both supervisors are visualization experts and already worked with latent diffusion models. This chapter is structured as follows: Section 4.1 will give an overview of the creation of the design concept including interviews that have been conducted with the two supervisors and the two employees of the institute. The section also addresses the extracted problem statement and the current workflow of the participants of the interviews and summarizes the requirements for the software solution that aims to solve the problem statement gathered from the interviews. Finally, the last part of the section highlights the solution phase, which includes the creation of possible visualization solutions that aim to meet the requirements and solve the problem statement. The course of Section 4.1 strongly reflects the *Design Study Methodology* that has been presented in Section 2.6, while each section can be associated with a specific step of the design thinking process. The second part of this chapter is built from Section 4.2 and presents the final visualization design of the tool as a result of the whole implementation process of the tool. In this section the final design decisions are described and to what extent the designs from the design concept in Section 4.1 have been adapted or modified.

## 4.1 Design Concept

As already touched upon in the introduction of this chapter, this section goes hand in hand with the *Design Study Methodology* and explains the process of deriving the requirements and possible design solutions step by step. The first step consists of the two interview phases which will be covered in the following.

### Interviews

To understand the current workflow and the requirements of the software solution, two interview phases have been conducted. The first interview phase refers to the *Empathize* and *Define* phases of the design thinking method and consists of fundamental questions that help to better understand the current workflow of the users, the problems they currently face and the requirements for a solution that addresses these problems. The first block of this interview phase consisted of the following questions:

- What tasks are you currently completing using Krita and Stable Diffusion?
- What is your workflow for these tasks?
- What problems are you currently facing regarding the solutions that you are currently using to complete the tasks?
- Which features currently work great for the completion of the tasks?

The question set of this first block consisting of the four above questions aims to capture the current state of the problem and solution space and they reflect the *Empathize* phase. Section 4.1 summarizes the results gathered from these questions such as the problem statement and the current workflow of the interview participants. The second interview block of the first interview phase which is associated with the *Define* phase consisted of the following questions:

- What are fundamental functionalities and properties that a solution should definitely offer?
- Are there functionalities and properties that are not fundamental but still important for you?
- Are there functionalities and properties that would excite you?

These questions are formulated with the goal of differentiating between the functionalities and properties in regard to their importance. The second interview phase goes hand in hand with the *Ideate* and *Prototype* phases of the design thinking method.

### Understanding the Problem and the Workflow

Based on the questions of the first block of the first interview phase, the current workflow as well as several problems with existing tools could be identified. Figure 4.1 shows the visualization of the workflow which combines the stated workflows of the interview participants and shows how they solve the task of art design using latent diffusion models. The workflow can be divided into three steps, while each step may be repeated multiple times. As art design is generally performed layer-wise, the three steps are executed for each layer which are later composed in tools such as Photoshop or Krita.

Step 1 starts with the creation of an initial text prompt as well as the selection of initial values for parameters such as the cfg scale, the denoising strength and the seed. Optionally, the participants make use of databases like *PromptHero* that offer a wide range of prompts for latent diffusion models with the corresponding output images. In this step, the prompt and the initial parameters are repeatedly updated until the generated image matches the desired style. The image resulting from the first step is then used as input for the sampling step.

The goal of step 2 is to generate a variety of output samples as an image grid using different sampling methods and seeds in order to explore parameter constellations and find the appropriate configuration for the desired output. In the context of this step, parameters such as the cfg scale and the denoising strength are less important and the focus lies on the usage of different sampling methods. Optionally, negative text prompts are used to intentionally remove elements from the image.

The last step which is shown in Figure 4.1 on the right consists only of repeated iterations of image-to-image in order to refine the output of step 2. This step usually starts with an upscaling of the output image from the previous step e.g. from 512x512 to 1024x1024 which is especially important when dealing with faces as 512x512 has a too low resolution for this purpose. By experimenting with the cfg scale and the denoising strength a couple of image-to-image samples are generated and images that match the requirements are exported to Photoshop or Krita and inserted into the specific layers.

As described by the participants they mainly used the Stable Diffusion Web UI (see Section 3.1) for the described workflow and they stated some core problems that they encountered, which are summarized in the following:

- **Problem 1:** Missing image history
- **Problem 2:** Complexity of the image sampling functionality
- **Problem 3:** No grouping of samples based on similarity metrics
- **Problem 4:** General complexity of the parameter configuration interface

The first problem is, that the Web UI does not offer the functionality to access an image history such that the user can apply image-to-image multiple times on the same image and go back to old versions of the image. Currently, the participants manually save images that they want to keep in order to prevent the output of the next image-to-image iteration from being worse than the current version and thus to lose this progress. Another problem is the complexity of the image sampling process in the Stable Diffusion Web UI, as a separate script needs to be created where the configuration of the sampling is defined. Especially for inexperienced users, this creates further friction to make use of sampling techniques. Also related to image samples is the problem, that these samples are not grouped based on any similarity metrics which makes it less intuitive to understand the relationships between the parameter constellations and the output samples. The last core problem of the existing tooling consists of the high complexity of the parameter configuration interface. As shown in Section 3.1, several parameters can be adjusted using sliders and other configuration methods which overwhelmed the participants especially when first starting to experiment with the tool.

### Identifying the Requirements

Based on the existing workflow and the problems that the interview participants came across with their current solution, the requirements of the new software solution can be formulated. In order to better understand the importance of certain requirements, the so-called *Kano Model* [OC12] is applied. The Kano Model describes a theory to classify requirements in the context of product development as one of several categories. Depending on the source, the number of categories varies and for this work, three categories will be considered. Figure 4.2 shows the visualization of these three categories consisting of basic, performance and excitement. The figure shows the relation between the fulfillment of requirements originating from the respective category and customer satisfaction. Basic requirements refer to the fundamental features or attributes that customers expect from the product. These are essential requirements that customers consider as the bare minimum for satisfaction and their absence will lead to a dissatisfaction of the customer. Performance requirements are related to the satisfaction of the customer and their improvement can lead to a proportional increase in satisfaction on the customer side. Finally, excitement requirements are

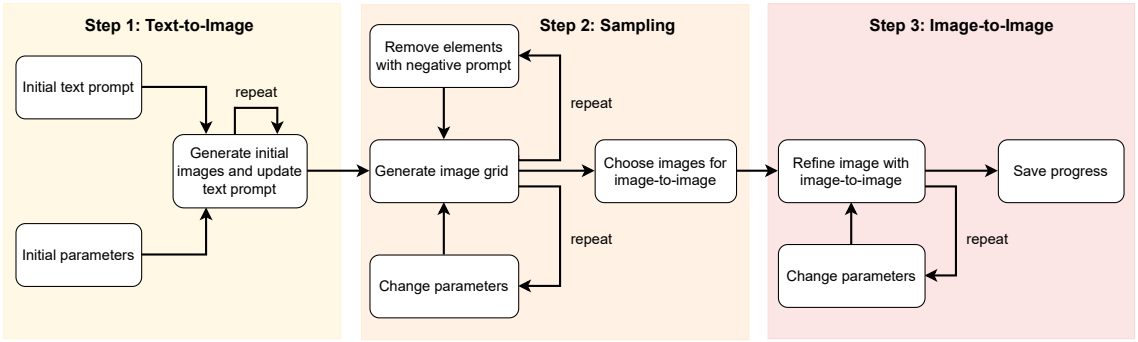



**Figure 4.1: Current workflow:** The workflow of the interview participants that they apply in their daily art design routine using diffusion models. The investigation of the procedure aims to support the formulation of the problem statement and how the participants interact with the technologies. The workflow consists of three steps, while the first step generates initial images using text-to-image, the second step uses sampling methods to cover different parameter configurations and the last step refines the output of step 2 with image-to-image.

known as unexpected features that have not been explicitly demanded by the customer. However, fulfilling these requirements can significantly increase the satisfaction of the customer and usually represents unique and innovative features [OC12]. The following question set describes the second block of the first interview phase and aims to classify the requirements as one of the three categories:

- What are fundamental functions and properties that the software solution should definitely have? (Basic)
- Are there functions and properties that you don't see as fundamental, but that are still important for you? (Performance)
- Which functions and properties are not expected, but would lead to excitement when they are fulfilled? (Excitement)

The result of the question set of this second block is shown in Table 4.1. The final requirements of the software solution consist of five basic requirements, two performance requirements and one excitement requirement. *R-B1* describes a history functionality that should be present when dealing with updates of images by iteratively applying latent diffusion models. The user should be able to go back to old versions of the image in order to revert updates to the image. The next requirement *R-B2* is related to the general concept of modifying composed images layer-wise. As already highlighted before, the participants mainly use tools such as Photoshop and Krita where they compose the results of the latent diffusion model generation using layers. To fluently integrate the DiffusionExplorer in this workflow, the solution should make it possible to import the layers of a document in Krita, modify them individually and export the modified layers back into Krita. As each layer might consist of different styles of images and different generation approaches, an individual parameter configuration for each layer must be provisioned which is described by the requirement *R-B3*. The final two basic requirements *R-B4* and *R-B5* are closely related to each other as they cover the topics of parameter exploration and the embedding of images in the 2D plane. *R-B4* summarizes the functionality to define parameter ranges that should be sampled and for those a set

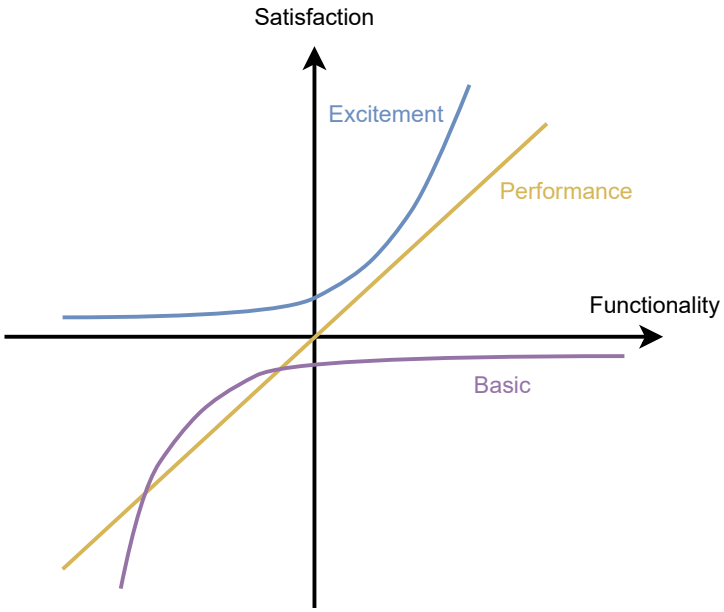



**Figure 4.2: Kano Model:** The Kano Model [OC12] describes the relation between the fulfillment of requirements and the satisfaction of the client in the context of software development. Requirements are divided into three categories: Basic, performance and excitement. While basic requirements are expected to be present in the software, performance requirements have a linear correlation with client satisfaction and the presence of excitement requirements results in an exponential increase in client satisfaction.

of output samples is generated. In order to better understand the relationship between the parameter constellations and the output samples, *R-B5* describes that the images should be embedded in a 2D plane based on either the similarity of the parameter or the image vectors. Moving forward to the performance requirements, two of these namely *R-P1* and *R-P2* have been stated by the interview participants. *R-P1* is about the visualization of the current context of the displayed image, such as providing the input image, which the image has been generated from. The second requirement *R-P2* can be seen as an advanced sampling functionality, where the user can choose between different sampling methods, whereas the basic requirements would only cover the provision of parameter sampling, in general, involving at least one parameter sampling method. As these two requirements are classified as performance requirements, their fulfillment would indeed enhance the workflow of the user but have not been seen as fundamental properties of the software solution. The last category of requirements consists of a single excitement requirement. The interview participants came up with *R-E1* which summarizes the functionality to not only have a linear history of the images but to provide the user with the opportunity to branch into multiple directions originating from a single image. This involves, that the user could perform multiple image-to-image generations starting from the same input image, which would be visually presented to the user as branches originating from the same image state. As all the requirements have now been classified as one of the three categories, this can be used to lead the implementation process by prioritizing the requirements. For the implementation phase, the basic requirements will have the highest priority as they represent fundamental properties of the software solution. The second highest priority will be assigned to the

**Table 4.1:** The requirements of the software solution classified as one of the three categories: Basic, performance and excitement.

| Basic Requirements | |
|---|---|
| ID | Description |
| R-B1 | The provision of a linear history that enables to go back to older versions of the image. |
| R-B2 | The functionality to import individual layers from Krita, modify them and export them back into Krita. |
| R-B3 | Configuration of a parameter set for each individual layer. |
| R-B4 | The provision of a systematic exploration of the parameter space using parameter sampling for specified ranges. |
| R-B5 | The possibility to compare output samples using a projection view that embeds the samples in a 2D plane. |
| Performance Requirements | |
| ID | Description |
| R-P1 | Visualization in what context a series of images has been generated e.g. what the the generation input was. |
| R-P2 | Selection of the parameter sampling method. |
| Excitement Requirements | |
| ID | Description |
| R-E1 | The provision of a history which is not only linear but also offers several branches from a specific state. |

performance requirements, as they further enhance the workflow of the user and the excitement requirements will be implemented with the lowest priority. After having collected the requirements of the software solution, the next paragraph will cover the *Prototype* phase of the design thinking method. This involves the creation of visual prototypes together with the domain experts which are the interview participants in order to steer the final design into the desired direction.

### Design Prototype

The *Prototype* phase which corresponds to the second interview phase aims to collect ideas and possible solutions for the visual realization of the software solution. Being one of the most exciting phases as all the collected requirements are brought into a prototype of the tool, this phase is also the least structured one. For this interview phase, screenshots of the skeleton of the tool

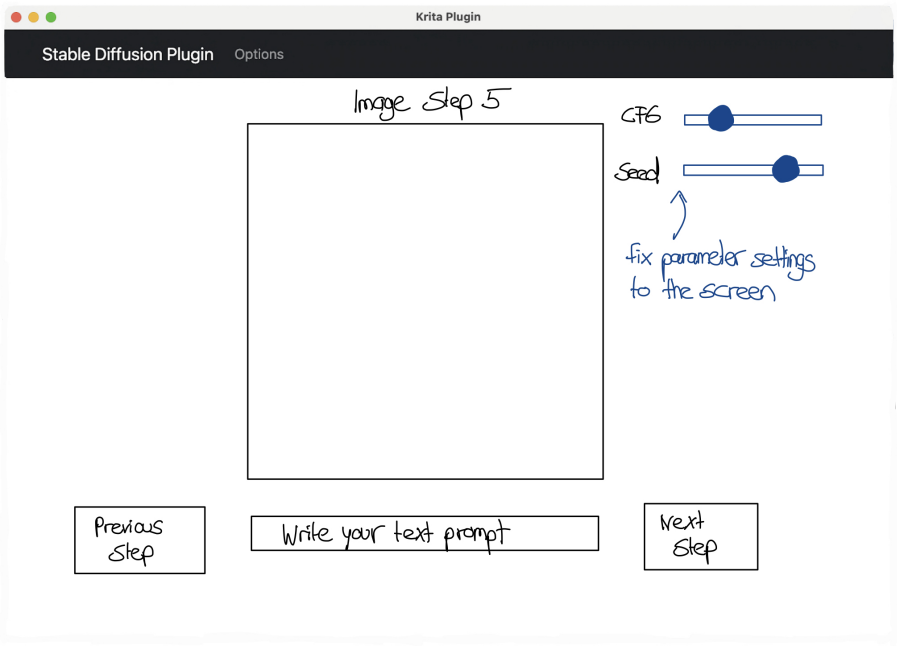



**Figure 4.3: Iterative View Prototype:** The *Iterative View* of the prototype which is the default view when applying image-to-image generations in the DiffusionExplorer. The view offers the functionality to configure the parameters and to go back and forth in the image history. As the model parameters in this view are fixed to a certain value, only a single image output is generated.

have been prepared in order to allow an interaction with the tool which should be as realistic as possible. Together with the interview participants, multiple views of the visual solution have been discussed, which can be broken down into three different views: The *Iterative View*, the *Projection View* and the *Pipeline View*. Figure 4.3 shows the so called *Iterative View* which is shown when image-to-image generations without sampling take place in the tool. The view provides the result of the image generation, a text field to write the text prompt and configuration possibilities for the model parameters placed at the upper right. The user can fix a specified set of parameters to the screen, such that only the desired parameters are continuously shown. As expressed in the basic requirement *R-B1*, the DiffusionExplorer should support a linear history which is accessible through the two buttons called *Previous Step* and *Next Step* enabling the user to return to older or newer versions of the current image. The options menu, which is located in the navigation bar is supposed to offer the configuration of the parameters that are shown on the screen as well as other parameters such as the image size. Proceeding to the next view of the prototype, the *Projection View* implements the visualization of the parameter sampling output. As shown in Figure 4.4 the input image, as well as the generated output samples are presented on the screen. Related to *R-B4* and *R-B5*, the output samples are embedded in the 2D plane based on either the similarity of the parameter or the image vectors. As it is sketched in the screenshot, the user can access the individual parameter configurations for the output samples by hovering over an image and thus better understand the relationship between the samples and the corresponding parameter configurations. By clicking on a specific output sample, the user can select the sample he wants to continue with. The final view created in the second interview phase is the *Pipeline View*. As

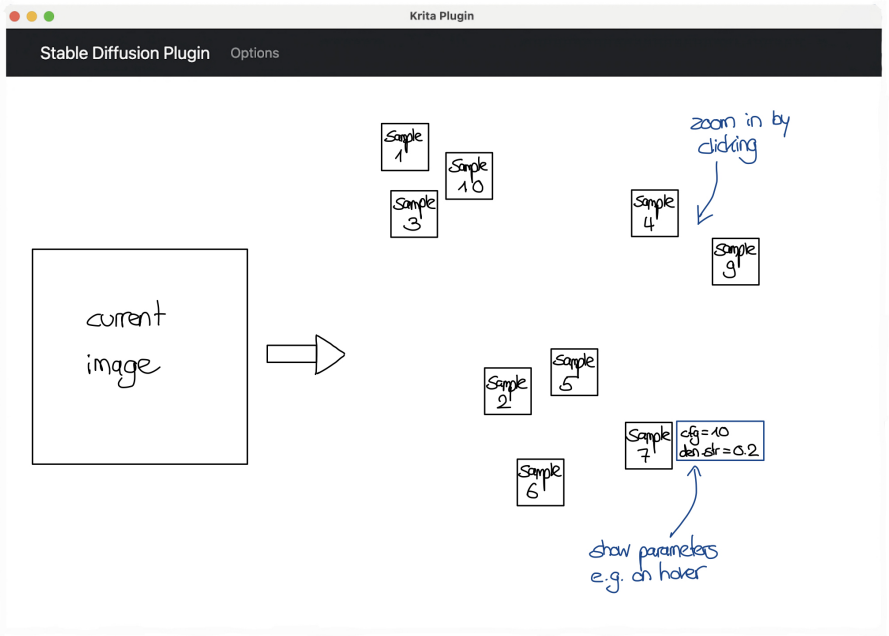



**Figure 4.4: Projection View Prototype:** The *Projection View* of the prototype is shown after applying parameter sampling to a given input image. The view shows the 2D embedding of the output samples based on similarity metrics and the coordinates of the images can be calculated using either t-SNE or UMAP.

the DiffusionExplorer allows the iterative modification of an input image using image-to-image generations, the subsequent application generates a pipeline of images that represents the image history. The *Pipeline View* shown in Figure 4.5 is dedicated to giving the user an overview of multiple generation steps by presenting the images as nodes connected with edges that are labeled with the parameter configuration that led from one image to another. The user can click on a node in the pipeline and the screen jumps into the *Iterative View* of the selected image allowing the user to run further image-to-image generations and replace the image history coming after the selected image with the new generation results. Something that is currently represented by the blue box filled with the parameter values in the *Pipeline View*, is the visual encoding of those parameters using so-called *Glyphs* which can be used to visually describe a single parameter as well as a whole parameter constellation.

## 4.2 Visualization Design

After having covered the requirements gathering process as well as the development of prototypes of the different views of the DiffusionExplorer, this section covers the final design of the tool. Each of the views will be presented in a dedicated paragraph and the original prototypes of the views will be compared to their final designs.

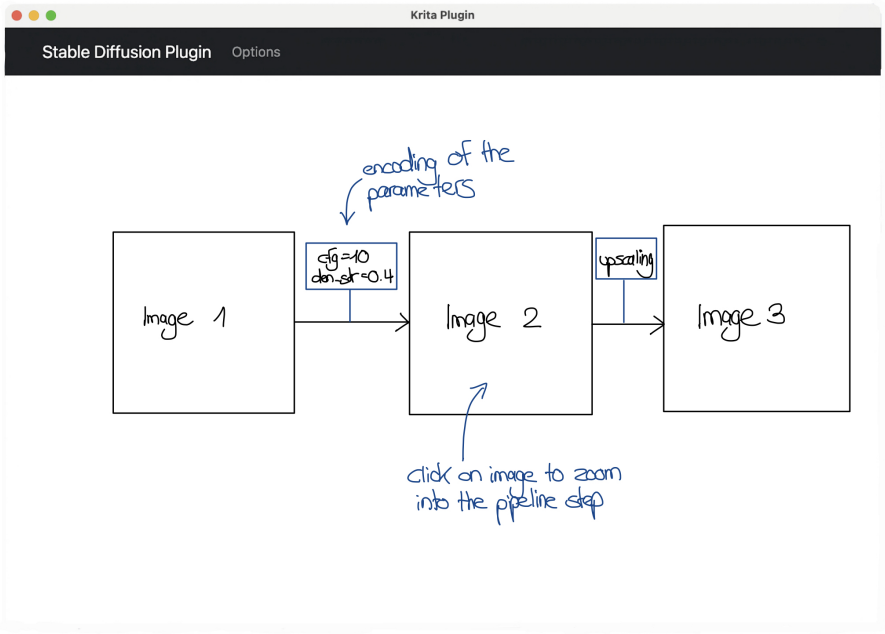



**Figure 4.5: Pipeline View Prototype:** The *Pipeline View* of the prototype visualizes the image generation pipeline of the DiffusionExplorer and allows the user to jump into a specific step of the pipeline by clicking on an image node. Above the edges that connect the nodes with each other, the model parameters that have been configured for the respective generation are visualized.

### Iterative View

As already touched upon in the solution phase, the so-called *Iterative View* is shown to the user when performing image-to-image generations with the sampling option disabled. Figure 4.6 shows the final and implemented design of the *Iterative View*. By comparing it to the prototype in Figure 4.3 one can see that both views have a resembling structure but several elements have been changed. While implementing this view, the idea of a bottom integration of the *Pipeline View* came up which enables the user to directly keep track of the history and switch to older versions of the image simply by clicking on a node in the pipeline which is shown at the bottom of the screen. This functionality makes the usage of buttons for the back-and-forth navigation in the image history obsolete which is why these buttons have been removed in the final design. Similar to the prototype, at the right side of the current image, this view offers several parameter configurations for the cfg scale, the denoising strength and the seed. The two buttons above the configuration sliders allow the user to enable the sampling option as well as to zoom out in order to show the *Pipeline View* which will be covered in a following paragraph. One crucial element of the design that has been introduced in the prototype of the *Pipeline View* is the glyph which is used to visually encode several generation parameters. In the implementation phase, the decision has been made to not only include this glyph in the *Pipeline View* but to also add it to the *Iterative View* such that the user is provided with a visual overview of the parameters that have been used to generate the currently shown image. This glyph is shown on the left in Figure 4.6 and consists of a radial radial bar chart that visually encodes both the cfg scale and the denoising strength as well as the sampler that has been applied for the

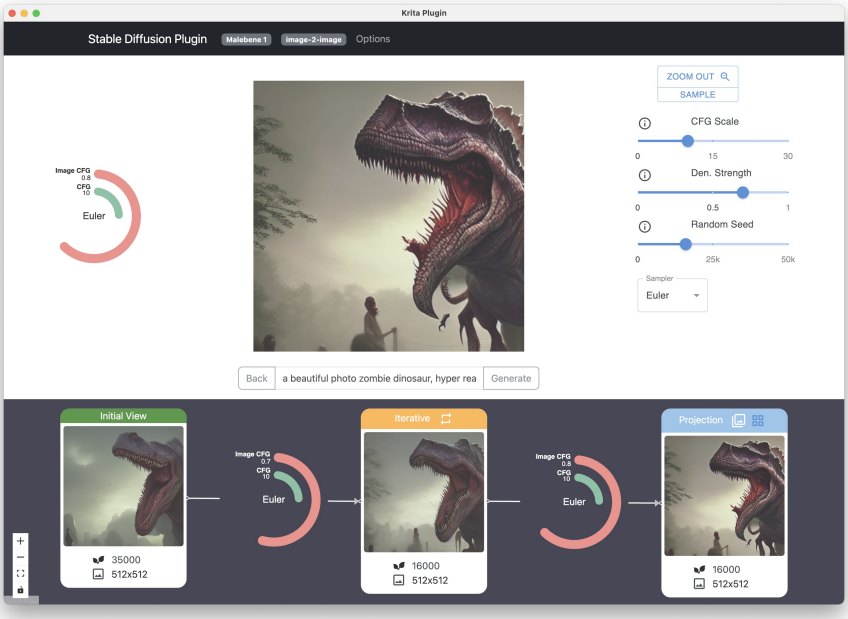



**Figure 4.6: Iterative View Design:** The final design of the *Iterative View* shows the current image in the center and the image history at the bottom. On the right side, the user can customize the parameters for the image-to-image generation, enable the sampling functionality and zoom out to the full view of the image history. The glyph on the left side of the image visually encodes the model parameters that have been configured for the generation that led to the image in the center of the view.

image generation. The angle of the arcs is dependent on the value of the respective parameters in relation to the maximum allowed values of the parameters. Finally, the appearance of this screen is customizable in the options menu which will be covered later. By using the options menu, the user can remove parameter configurations from the screen and substitute the sliders with input fields for a more precise parameter configuration.

### Projection View

Switching to the so-called *Projection View*, this view is presented to the user when the sampling option is enabled and after the user executed the image generation. The *Projection View* is strongly based on the prototype and Figure 4.7 shows the final implementation. This view can be divided into the left and the right side while the left side consists of the current image, a glyph and a slider to change the size of the image samples that are shown on the right. On the right side, the generated image samples are embedded using 2D coordinates that are computed based on either the similarity of the parameter or the image vectors. Whenever the user hovers over an image sample, the glyph on the left side shows the generation parameters of the specific image sample. As the images on the right might overlap especially when sampling for a larger amount of parameter constellations, the slider can be helpful to reduce the overlapping of the images. For the 2D embedding in the *Projection View* both t-SNE and UMAP are offered as dimensionality reduction algorithms which

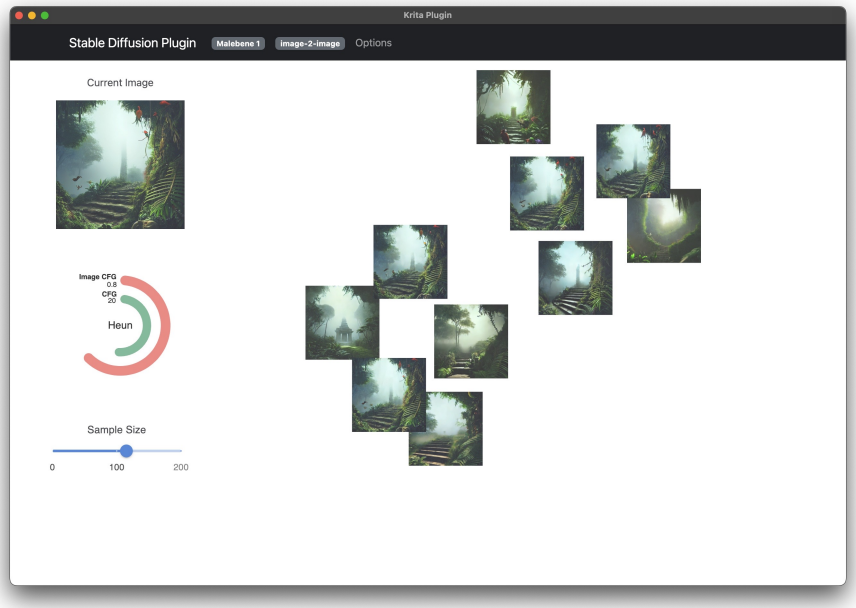



**Figure 4.7: Projection View Design:** The final design of the *Projection View* which shows an example of 10 image samples that are embedded in the 2D space based on the similarity of the parameter vectors using the UMAP dimensionality reduction algorithm. When the user hovers over one of the image samples, the glyph on the left shows the model parameter configuration of the hovered image. Above the glyph, the input image of the generation step is shown.

the user can choose from in the options menu. In order to continue with the workflow, the user can choose one of the image samples by clicking on it and the view changes back to the *Iterative View* where the user can proceed with further image-to-images generations. As a result, the sampling step is then added to the image history.

### Pipeline View

The third major view of the DiffusionExplorer is the *Pipeline View* which visually represents the image history of the currently selected layer. The view can be accessed by clicking the *Zoom Out* button in the navigation bar of the tool when being in the *Iterative View* or the *Projection View*. Figure 4.8 shows the final design of the *Pipeline View* for an example where two individual generation steps have been performed. In this view, each generation step is represented by a node and connected to the subsequent step via an edge. As shown in the example, the nodes are colored differently depending on the type of image generation that has been performed. In the tool, there exist different types of generations that lead to three different node types: *Initial View*, *Iterative* and *Projection*. The first type appears when the input image is completely ignored and the image generation is only based on the given text prompt. This might be the case when the user imports a Krita document with empty layers into the tool and wants to perform an initial text-to-image generation. The second node type *Iterative* features regular image-to-image generations where the sampling option is disabled and the user only adapts the text prompt as well as other generation

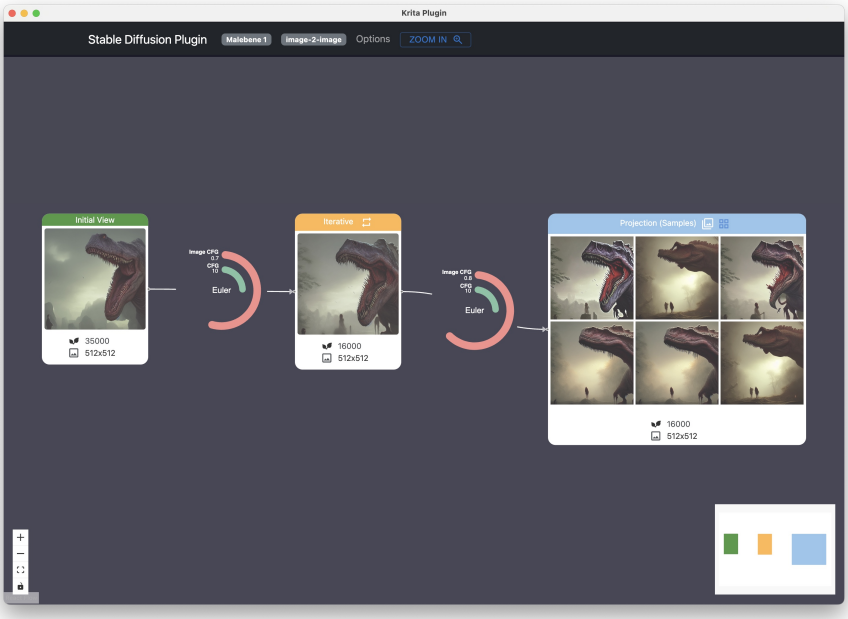



**Figure 4.8: Pipeline View Design:** The final design of the *Pipeline View*. In this example, an initial text-to-image generation has been performed, followed by one image-to-image generation and one sampling iteration. Each iteration type is represented by a different node color and on-demand, all the image samples of the sampling step can be shown inside the blue node. The edges, that connect the nodes with each other consist of a glyph that visually encodes the model parameter configuration that has been applied for the respective generation step.

parameters. Finally, the last node type describes generation steps with the sampling option enabled which leads to the *Projection View*. After the user selects an image sample in the *Projection View* the image is added as a node of type *Projection* to the pipeline. A node of the Projection type offers the functionality to give a quick overview of the samples that have been generated in the sampling step. As shown in Figure 4.8 six samples have been generated in the second step, which can be displayed on demand in the node. Each node additionally contains the value of the random seed as well as the size of the generated image. Besides the nodes themselves, the *Pipeline View* also features glyphs which are displayed between nodes. Each glyph visually encodes several generation parameters that lead from the node on its left to the node on its right. As a result, the user has an overview of the image history as well as of all major generation parameters that have been applied in the generation steps.

### Options Menu

The last visual element of the tool is the options menu which can be accessed through the navigation bar on the top of the tool. Figure 4.9 shows the implemented options menu which offers a wide variety of customizations and parameters. The upper half of the menu is related to the visual appearance of the parameter settings that are shown on the right in the *Iterative View*. By clicking

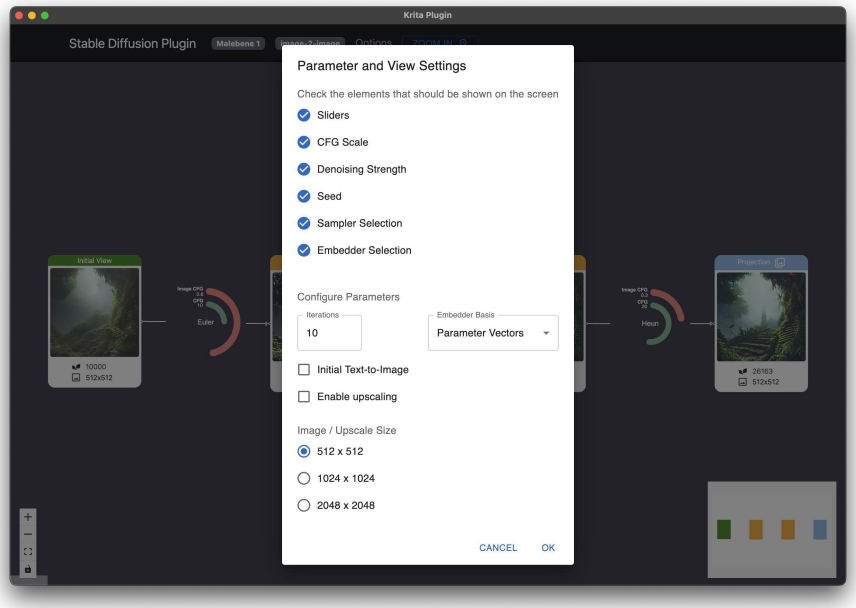



**Figure 4.9: Options Menu Design:** The final design of the *Options Menu* offers several functionalities including the customization of the screen as well as the number of iterations and the selection of the dimensionality reduction algorithm. By setting the *Initial Text-to-Image* flag, the user can control whether the current image or solely the text prompt should be used for the generation step.

on these checkboxes, the user can decide which of the parameters should be shown in the parameter settings. The lower half of the options menu offers further model settings that are not shown in the parameter settings including the choice of the dimensionality reduction algorithm and the size of the images when performing upscaling. The *Initial Text-to-Image* check box provides the functionality to ignore the content of the current layer that has been imported from Krita and to perform an initial text-to-image generation based on the given text prompt.

This chapter covered the application of the design thinking process to the development of the software solution as well as the final resulting visualization design. As this completely left out the underlying technical implementation details, the following chapter aims to give an overview of these.

# 5 Developing a Tool for Visual Parameter Exploration

After having covered the visualization aspects of the software solution, this chapter aims to address the technical implementation details such as the frameworks and the architecture that has been used to implement the DiffusionExplorer which might be especially interesting for future extensions of the tool. The first Section 5.1 is all about the open-source painting program Krita and how to develop plugins for it. The next Section 5.2 covers the technical implementation of the tool consisting of the frameworks and other technologies, as well as how multiple components of the tool interact with each other in order to offer the desired functionalities.

## 5.1 Krita

As already introduced at the beginning of this chapter, Krita is a free open-source painting program that offers extensibility through plugins. In general, paintings in Krita are usually decomposed into several layers that are then laid on top of each other.

### Krita Python API

The Krita Python API provides a comprehensive set of tools and functions that allow developers to extend and customize the capabilities of Krita using the Python programming language. With the API, programmers can automate repetitive tasks, create custom brushes, modify the user interface and interact with Krita's document structure, layers and image data. In the context of this work, the interaction with the current active document as well as the layers contained in the document is important, as these functionalities allow the import and export of the layer data in order to manipulate it in the tool. For this purpose, the Krita Python API offers functions to access all layers, change their properties such as their name and their content. To make use of this API, Krita offers the possibility to write plugins using API functions and that can be visually embedded into the user interface of the painting program. The following paragraph describes how plugins can be defined and included in Krita.

### Plugin Architecture

Krita provides a flexible plugin architecture that allows users to extend its functionality through custom plugins. Plugins can be written in Python and integrated into Krita. In this paragraph, the file structure of a Krita plugin will be presented and it will be explained how it can be included in the application. Figure Figure 5.1 illustrates an example file structure for a Krita plugin called

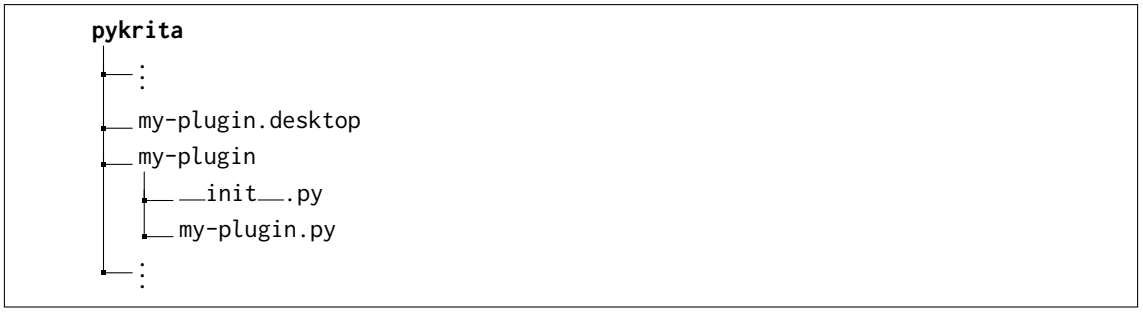



**Figure 5.1: Plugin File Structure:** The file structure how the sample plugin *my-plugin* is included into Krita. The plugin can then be enabled in the settings of the painting program.

*my-plugin*. The **pykrita** directory serves as the main directory where Krita looks for Python plugins and acts as the parent directory for all Python-based plugins. Within this directory, the *my-plugin.desktop* file is a *desktop entry* file that provides metadata about the plugin including information such as the plugin's name, description and the path to the main Python script file. The *my-plugin/* directory represents the plugin itself. It contains the plugin's Python files and any additional resources or assets required by the plugin. The *_init_.py* file within this directory acts as the initialization script for the plugin. It is executed when the plugin is loaded and allows you to perform any necessary setup or initialization steps. The *my-plugin.py* file is the main Python script file for the plugin containing the code that defines the functionality of the plugin. This can also include the user interface of the plugin which is then shown in the running Krita application.

## 5.2 Tool Development

This section aims to give an overview of the technical background of the DiffusionExplorer, which is split into multiple components. The first paragraph of this section presents how these components of the tool interact with each other and the second paragraph addresses the technology decisions that have been made for these components.

### Tool Architecture

Figure 5.2 visualizes the component view of the DiffusionExplorer consisting of four major components. The first component is a Krita plugin, that enables the import and export of the layer data of the current active document. The plugin acts basically as an interface between Krita and the other components, enabling the exchange of the layer data back and forth having direct access to the data of the current document which is opened in Krita. Label *B* in Figure 6.1 shows the minimalist user interface of the Krita plugin, which consists of only two buttons that provide the functionality to import and export layer data. While the first button exports every single layer consisting of a layer name and the content of the layer and sends it to the electron application, the second button can be used to update the content of the selected layer in Krita with the modified layer content from the electron application. The second and fundamental component of the tool is the electron application which represents the main interaction with the user. As shown in Section 4.2 the electron application offers the user interface the user mainly interacts with and provides all the

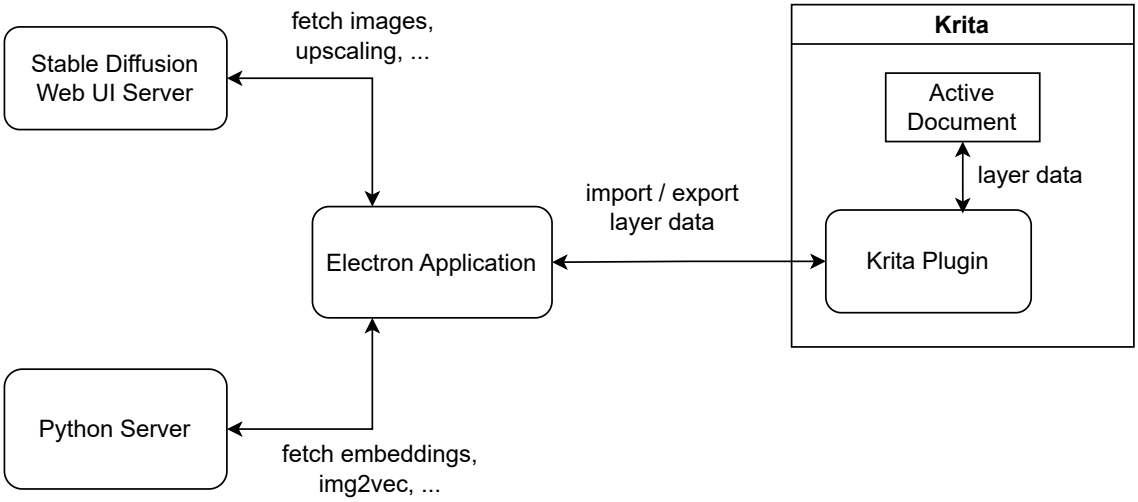



**Figure 5.2: Component View of the DiffusionExplorer:** This figure shows the four main components of the tool, consisting of the Krita application and the Krita plugin, the main electron application and two helper components for the image generation and the dimensionality reduction. The electron app is directly connected to Krita via the plugin, enabling the user the transfer of layer data between the two instances.

functionalities that have been described in the requirements. This component communicates with two additional components as shown in Figure 5.2 on the left, while one component is the Stable Diffusion Web UI running in API-only mode and the other component is a Python server that offers endpoints to compute the 2D embedding of given vectors as well as *image2vec* in order to reduce the dimension of images to much smaller vectors.

### Technology Decisions

One of the biggest technology decisions has been the usage of *Electron* in combination with the library *React* for the frontend implementation. *Electron* is a framework to build cross-platform applications that can be used with frontend frameworks such as *React* and *Angular*. The first approach for the implementation of a plugin for Krita would have been to fully implement the user interface inside the Krita plugin. However, the usage of an external application window for the plugin offers many freedoms and the possibility to develop an independent desktop application. The communication between the Krita plugin and the electron application takes place using sockets. This technology has been chosen to enable fast and real-time communication between the two components as they might have frequent exchanges of layer data. Finally, the Stable Diffusion Web UI API has been chosen as an endpoint for all image generations including the upscaling of images as the API offers a wide range of features and reliable implementations.

After having covered the technical background of the DiffusionExplorer, the following chapter covers the evaluation of the tool which represents one of the core chapters of this work as it presents the theoretical and practical interaction with the tool.

# 6 Evaluation of the Tool

This chapter presents the evaluation of the developed software tool, which aims to address a specific use case within the process of art design using latent diffusion models. The evaluation process consists of two main parts: A case study with the proposed workflow and a user evaluation where the tool is tested by the same users who have been interviewed for the requirements of the tool. The goal of this evaluation is to assess the effectiveness, usability and alignment of the DiffusionExplorer with the intended use case, as well as to gather valuable user feedback for further improvements. The case study in Section 6.1 focuses on the initial conceptualization and design of the tool. It entails presenting the workflow that was devised to guide users through the different steps and functionalities of the tool. By demonstrating the predefined use case, it is illustrated how the tool was originally envisioned to be used and how it behaves in a given scenario. In contrast, Section 6.2 covers the user evaluation and assesses the practical usability of the tool through direct engagement with the target users. As the requirements and expectations were gathered during the interview phases, in the user testing phase, the participants were given access to the software and encouraged to explore its features. The goal is the execution of tasks related to the use case and the provision of feedback on their experience. This evaluation not only aims to compare the proposed usage with the actual user interactions but also aims to gather insights into user satisfaction and identify strengths and weaknesses, as well as limitations of the tool. Before jumping into the evaluations, the following paragraph describes the task that will be used throughout this chapter.

### Task and Scenario: Art Design

This task consists of the design of two layers in Krita, one being the background of the image and the other layer being a paint layer that is laid on top of the background layer. The user wants to create a background for a specific topic, which is, in this case, a *beautiful, photo-realistic landscape with mountains*. In addition to this background, the user wants to add specific elements such as a *wooden house* to the image, which are added to the paint layer and laid on top of the background. This should be achieved as combined work using both Krita and the DiffusionExplorer which enables the user to use latent diffusion models. Section 6.1 presents a theoretical workflow of a user that solves this task using the tool based on the developer and background knowledge of the tool. In Section 6.2 this workflow will then be compared with the workflow of actual users that try to execute the scenario and solve the task. To summarize, the task consists of the following points:

- A document in Krita consisting of two layers: A background and a paint layer
- The background layer should be topic-specific for the given topic: *beautiful, photo-realistic landscape with mountains*
- The paint layer consists of a specific element, namely a *wooden house*

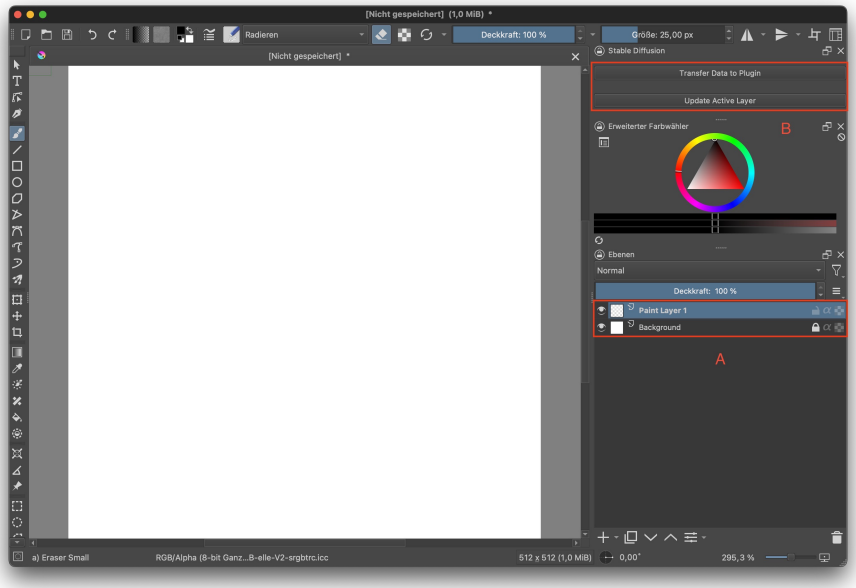



**Figure 6.1: Step 1:** The user starts with a new document in Krita consisting of two layers: A background layer and a paint layer. Label *A* in the screenshot shows the two layers of the newly created document and label *B* shows the UI of the Krita plugin consisting of two buttons.

## 6.1 Case Study

The previously described task may be solved in different ways, while this case study presents a particular solution for the given task. The solution can be broken down into the following steps:

- **Step 1:** Create an empty document in Krita consisting of two layers (background and paint layer)
- **Step 2:** Export the layers from Krita into the tool
- **Step 3:** Run initial text-to-image generations for both the background and the paint layer until a desired state is achieved
- **Step 4:** Iterative application of image-to-image generations for both layers including the usage of features such as sampling
- **Step 5:** Import the results from the tool back into Krita and integrate the paint layer into the background. Finally, run an image-to-image generation.

Starting with *Step 1* the user creates a new document in Krita and does not perform any drawing operations in the newly created document. The document contains two empty layers, the first layer being a white background layer and the second layer being an empty paint layer. Figure 6.1 shows the resulting state of the Krita program of the user after executing the first step. The two layers of the new Krita document are highlighted and labeled with *A*. As visible in the screenshot and labeled with *B* are the two buttons of the Krita plugin that are connected to the tool. By clicking on the button called *Transfer Data to Plugin* the user executes the second step of the workflow and exports

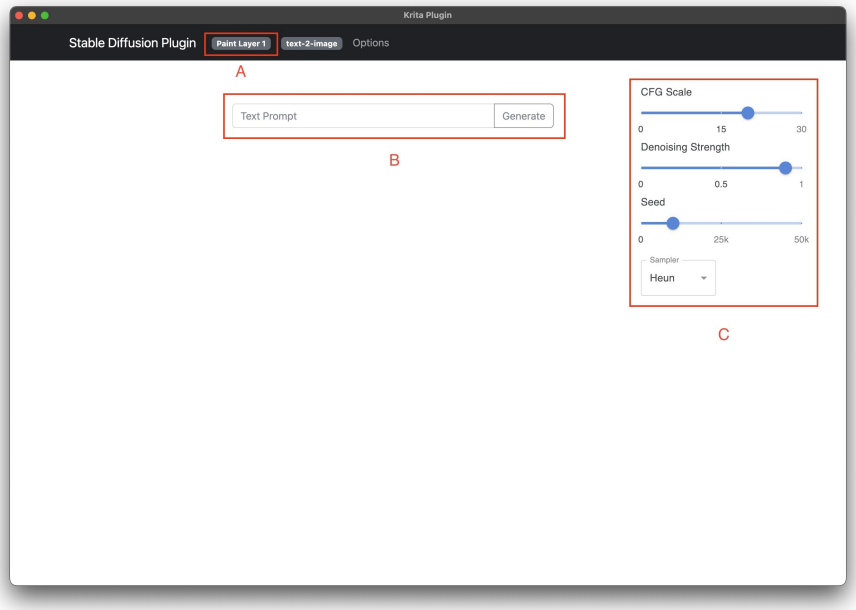



**Figure 6.2: Step 2:** The user interface of the DiffusionExplorer after the export of the layers of the Krita document to the tool. Label *A* shows the currently selected layer and label *B* highlights the prompt field for the initial text-to-image generation. The parameters for the first generation such as the cfg scale and the seed can be found on the right highlighted by label *C*.

all layers of the Krita document into the tool. At this point, the focus of the workflow switches from Krita to the DiffusionExplorer and Figure 6.2 shows the initial state of the tool after the export of the layers from Krita into the tool. By default, the content of the layers is ignored and the tool offers an initial text-to-image generation for all layers. As labeled by *A* in the screenshot, the last layer called *Paint Layer 1* is selected and the user can start a text-to-image generation based on a text prompt in *B* and configure the generation parameters in *C*.

Transitioning to *Step 3*, both Figure 6.3 and Figure 6.4 show the generated images for the background and the paint layer after the initial text-to-image generation. Additionally, the labels *A* and *B* in Figure 6.3 highlight important elements of the shown view. Spotlighted by label *A* is the glyph that summarizes some of the parameters that have been used for the generation of the image that is shown next to the glyph. Also highlighted in Figure 6.3 by label *B* is the first element of the pipeline that shows the image generation history. Assuming the user is satisfied with the background image, he now wants to further modify the *wooden house* in the paint layer which brings us to *Step 4*.

As the user would like the house to be more photo-realistic and to have a door, with *Step 4* he adapts the text prompt and wants to create multiple samples for different values of the cfg scale and the seed and uses the sample functionality to sample 15 images for the full ranges of both parameters. The result of the sampling process is shown in Figure 6.5, where hovering an image sample shows a glyph with the corresponding generation parameters of the image (label *A*). Label *B* represents the area where all the image samples are embedded, in this case, based on the similarity of the parameter vectors.

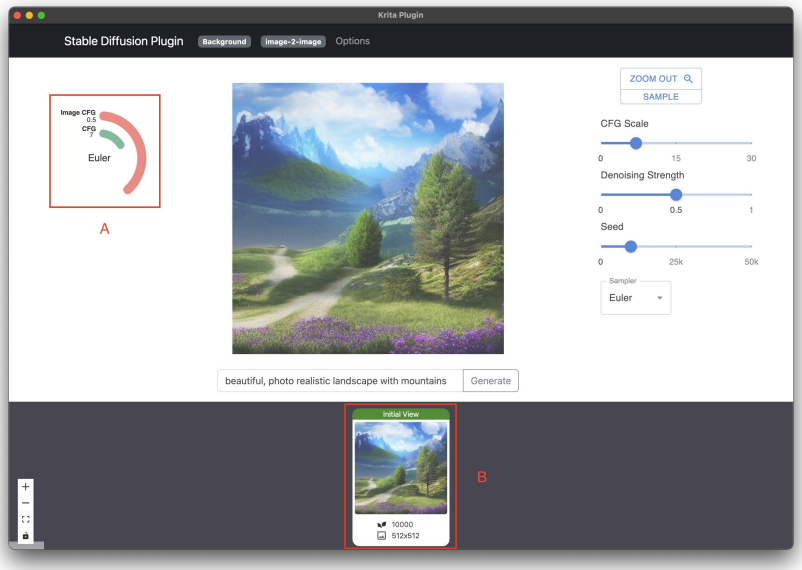



**Figure 6.3: Step 3:** The background layer in the tool after the initial text-to-image generation which shows a *beautiful, photo realistic landscape with mountains*. The first label *A* shows the glyph that represents the generation parameters and the second label *B* highlights the first element of the image history which has just been generated.

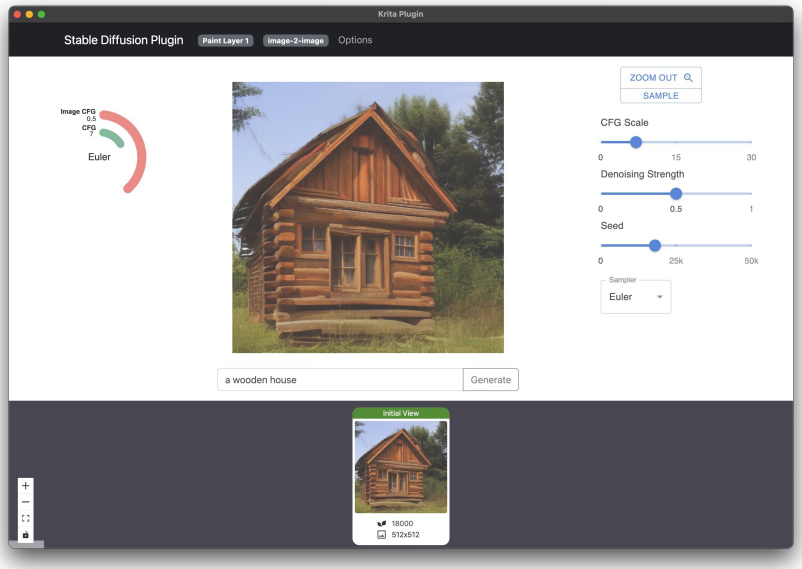



**Figure 6.4: Step 3:** The paint layer in the DiffusionExplorer after the initial text-to-image generation which shows the generation result for a *wooden house*.

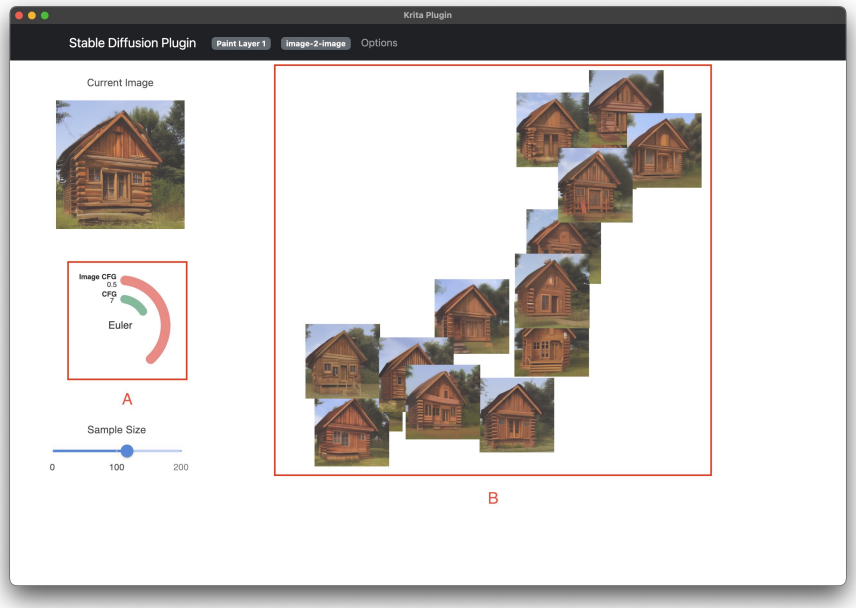



**Figure 6.5: Step 4:** The result of the parameter sampling for the paint layer for the full ranges of the cfg scale and the seed. Label *A* highlights the glyph that summarizes the generation parameters of the hovered image sample and label *B* marks the 2D embedding of the image samples based on the similarity of the parameter vectors.

The user then selects one of the samples that matches his requirements and continues with *Step 5*, where the focus of the workflow switches back to Krita and the user can use the second button of the plugin user interface to import the modified layers back into Krita. After switching back to Krita, the user cuts out the wooden house from the paint layer and inserts it into the background layer. As it is now clearly visible that the wooden house has been manually inserted into the background, the user finishes the last step by running a final image-to-image generation in the tool using the combined prompt *beautiful, photo-realistic landscape with mountains and a wooden house* and exports the result. Figure 6.6 shows the final output of the whole task, while the last image-to-image generation enabled a smooth integration of the wooden house into the background layer.

To summarize this case study, the presented task shows how the DiffusionExplorer can be used to solve a basic art design task by applying the intended workflow of the developer of the tool. However, this is more of an iterative presentation of the features using a concrete task than a study with real users, which is done in the next section.

## 6.2 User Study

This section of the tool evaluation chapter covers the execution of a user study with the same participants who have been interviewed in the requirements gathering phase. The goal of the study is to collect some valuable insights of the user interaction with the tool, in order to evaluate the DiffusionExplorer and its capabilities to solve the problem statement from the interview phases.

**Figure 6.6: Step 5:** The result of the integration of the paint layer into the background layer in Krita. After the integration, a final image-to-image generation is executed to improve the smoothness of the integration.

Each participant is equipped with a computer that has all the necessary software installed, including Krita and a running instance of the tool. The study can be divided into three steps, while in the first step, each participant is provided with a short demo of the tool where all the core features as well as the views are explained. Additionally, the interface between Krita and the tool, as well as some basic editing functionalities in Krita are covered in order to prepare the participants for the study task. The second step consists of a task that is supposed to be solved in the scope of the study which is scheduled for around 30 minutes and can be summarized as follows:

- Creation of an image containing *Bob Ross* painted in the style of Bob Ross.
- Start with an empty document in Krita (512x512) consisting of two layers.
- Proposed workflow:
  - Import the empty Krita document into the tool.
  - Generate e.g. a picture frame in the background layer and a portrait of Bob Ross in the paint layer.
  - Experiment with multiple image-to-image generations, the configuration of the parameters and sampling.
  - Import both layers back into Krita and make use of masking e.g. in order to put the portrait of Bob Ross inside the picture frame until a desired final image is achieved.

Bob Ross was an iconic American painter and TV host known for his soothing voice and "Joy of Painting" series, teaching art with a calming demeanor and happy landscapes. The goal of the task is a basic interaction of the participants with the tool, by providing them with a proposed workflow that aims to structure the task. Depending on the level of expertise of the study participants, the task can be solved in various ways and there is no clear restriction given on how the task should be solved. Thus, the task execution might start with some basic drawing in Krita, but the participant could also directly export the empty layers into the DiffusionExplorer and start with text-to-image generations. In the third and final step after solving the task, each participant of the user study is asked to fill out a survey that addresses each view of the tool individually. Each statement of the survey can be evaluated using a likert scale and for each view, the participant can leave further comments that have not been covered by the provided statements. As already presented in Section 4.2, the tool consists of the *Iterative View*, the *Projection View*, the *Pipeline View* and the *Options Menu*. For each of these views, the survey covers three statements, while the following two are stated for each view:

- **S1:** This view is easy to navigate and the offered functionalities in this view are easy to use.
- **S2:** This view enables me to solve the problem, that I want to address with it.

The first statement addresses the intuitiveness of the respective view, including the navigation inside it as well as how easy/difficult it is to make use of the offered functionalities. The second one covers another core evaluation of the views, which is to what extent a view actually helps to solve the problem that it is supposed to address. In addition to these two statements, the survey includes a third statement individually for each view:

- **S3.1:** The visual parameter encoding (Glyph, Sliders) is understandable and it is clear what the values refer to *(Iterative View)*.
- **S3.2:** This view offers comparability between the generated samples and their parameters *(Projection View)*.
- **S3.3:** This view offers an overview of the image history with a suitable density of information *(Pipeline View)*.
- **S3.4:** The options in the menu are structured in an intuitive way *(Options Menu)*.

Each of the additional statements aims to evaluate the respective view based on the core properties that have been collected in the interview phases before the implementation. The following paragraphs summarize the results of the user study including the resulting images of the task, the evaluation of the survey and further findings.

### Image Results

Before starting the study, each participant could choose to publish the resulting image either anonymously or with a self-chosen pseudonym. In the following, the images are referred to as the pseudonyms of the respective participants. A solution strategy, that is of special interest for the comparison of the image results is the usage of painting in Krita in order to add elements to the image and support their generation with latent diffusion models. In an example scenario of an image showing a lake, a user might want to perform an image-to-image generation to add a boat to the lake. Therefore the user could only add the keyword to the text prompt or additionally sketch a boat into the input image to support the generation process. As the users of the study applied

**Figure 6.7:** The resulting image of the user study executed by the participant with the pseudonym *J*. The participant did not make use of any initial sketching and achieved the result solely through text prompts and the combination of two layers.

different strategies for the inclusion of specific elements, the resulting images can be compared based on these different approaches. Figure 6.7 shows the resulting image of the participant with the pseudonym *J*. In the execution of the study, the participant transferred the empty document from Krita into the tool and both the portrait of Bob Ross and the picture frame have been generated in separate layers. Interestingly, the participant did not make use of any supportive painting in Krita and the results of the layers have been achieved by applying initial sampling followed by multiple iterations of image-to-image generations. Finally, the participant transferred the layers back into Krita and inserted the layer containing Bob Ross into the layer with the picture frame. A major obstacle that occurred in the generation process of the picture frame was the inclusion of a white background surrounding the picture frame instead of the dark background as shown in Figure 6.7. Despite multiple attempts to include the white background in the prompt and different parameter configurations, the participant could not achieve the desired white background. This shows the limitation of only using prompt adaptations to include specific details in the provided image.

The approaches of the other three participants *jokkurt*, *Moe* and *hageldave* have all been inspired by the idea to create a painting in the style of Bob Ross in which Bob Ross is painting with a brush on a canvas. In contrast to the first participant, *jokkurt*, *Moe* and *hageldave* approached the task with supportive painting in Krita, while each participant used painting to a different extent. Starting with *jokkurt*, the participant started initially by roughly sketching both the landscape background and the silhouette of Bob Ross in two separate layers in Krita. After transferring both layers into the tool, the participant executed an initial image-to-image generation with each layer, that transformed the sketched layers together with a similar text prompt into images that already matched the desired style quite well. At this point, it is remarkable how accurately the results of the image-to-image generations match the desired style just by providing rough sketches as input.

**Figure 6.8:** The resulting image of the user study executed by the participant with the pseudonym *jokkurt*. Starting with quite detailed sketching of Bob Ross and other elements such as the style of the landscape, the participant managed to generate accurate results matching the desired outcome.

After multiple applications of sampling over the seed and the denoising strength and some minor painting in between, the participant inserted the layer containing Bob Ross into the landscape and executed additional image-to-image generations in order to make the embedding of Bob Ross in the landscape smoother. The resulting image of this process is shown in Figure 6.8. The resulting images of the user study executed by the last two participants *hageldave* and *Moe* are shown in Figure 6.10 and Figure 6.9. Both participants took similar approaches as *jokkurt*, with the difference that they used initial text-to-image generations with text prompts that described the landscape and Bob Ross before using supportive sketching to include specific elements in the images.

To summarize the image results of the user study, all participants could solve the task and match the desired outcome using Krita in combination with the tool. However the application of initial sketching or sketching in between made it much easier for the participants to include specific elements into the images, which would have been difficult or not even achievable with plain text prompts. These results disclose the advantages of the tool, especially offering a seamless integration into Krita that allows the user to quickly transfer the layers between Krita and the tool to add sketches that support the image generation process. During the user study, the sampling functionality was used very often, as the majority of the participants did not have advanced experience using latent diffusion models as well as the parameters related to the models. Some of the participants also used the *Projection View* to compare the parameters of the generated samples. The participants could identify the relationship between the denoising strength and the intensity of visual adaptions in the generated image. For higher values of the denoising strength, the output samples differed more from the input image than other samples where the denoising strength had lower values. This fact

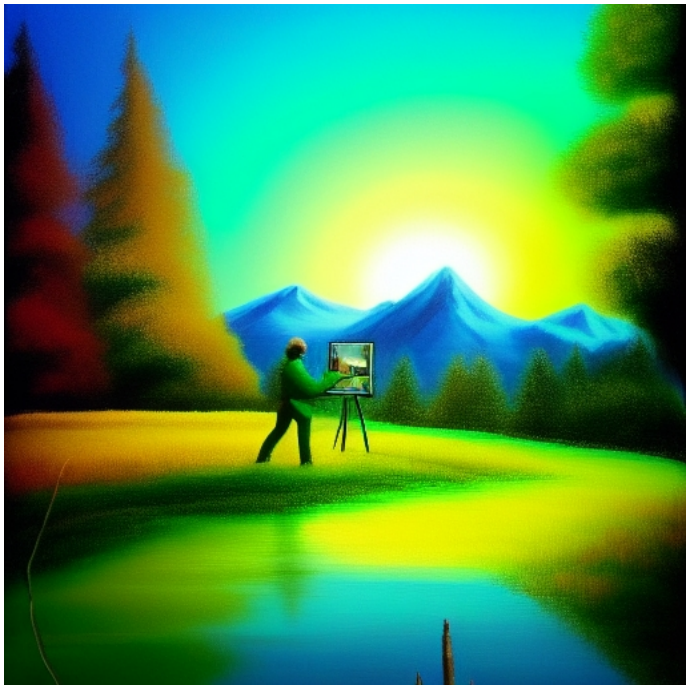


**Figure 6.9:** The resulting image of the user study executed by the participant with the pseudonym *Moe*. In this study execution, a lot of sketching was applied between the generation steps and the participant tried to insert the content of the whole image as the content of the canvas that Bob Ross paints on.

**Figure 6.10:** The resulting image of the user study executed by the participant with the pseudonym *hageldave*. The participant made use of supportive sketching to generate specific elements in the image.

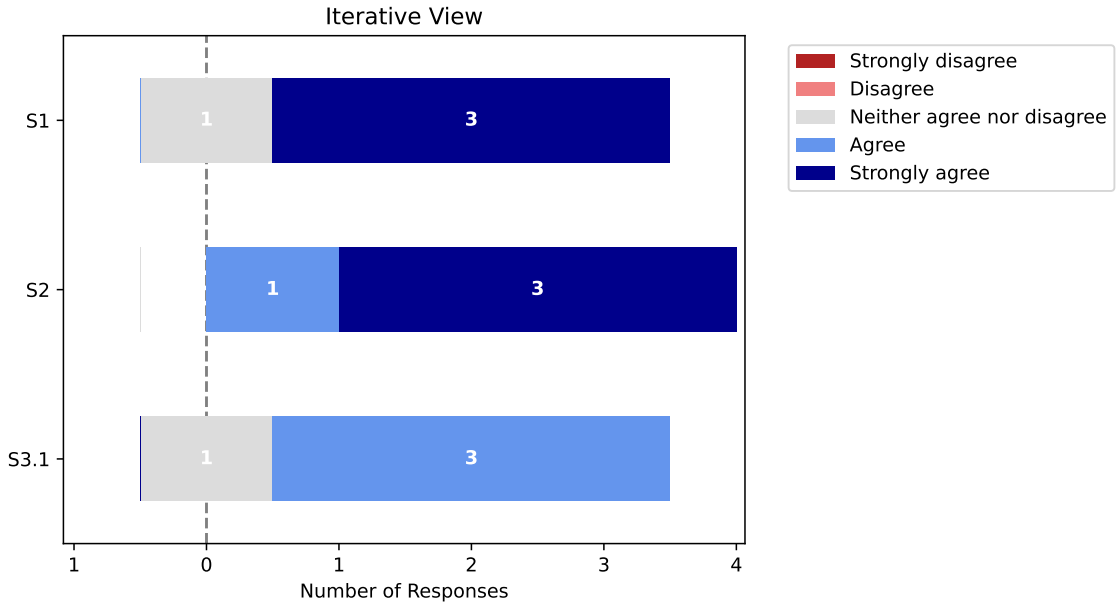



**Figure 6.11:** The Likert plot of the survey results for the *Iterative View*.

was then used by the participants to control the strength of the visual changes in the input image. In addition to these observations, the participants with less experience related to latent diffusion models generally applied more sampling than iterative generations with fixed parameter values. In contrast to this behavior, the participants with prior knowledge of the semantic interpretation of the model parameters such as the denoising strength performed selective adaptations of the parameters using iterative generations.

### Survey Results

After having analyzed the resulting images of the user study, this paragraph covers the evaluation survey which has been filled out by each participant after solving the given task. As already touched upon earlier, the survey consisted of three statements for each view where the participants could communicate their level of agreement using a likert scale with five response categories, as they are shown in the legend of Figure 6.11. The given figure shows the answers of the four participants for each of the three statements addressing the *Iterative View* (see design in Figure 4.6) of the DiffusionExplorer. In general, the feedback for the view in regard to the given statements turned out to be positive, as the level of agreement was high. While the statements *S1* and *S2* covering the navigation, the ease of use and the capability to solve problems had a strong level of agreement, the statement *S3.1* was mainly rated with an agreement regarding the understandability of the visual parameter encoding. Depending on the level of expertise of the respective participant, the textual comments on the *Iterative View* took completely different directions. While the participant with further expertise proposed to offer a bar chart instead of a radial encoding of the parameter values, another participant with less experience stated not having used the glyphs at all as mainly the sampling option has been applied and the true values of the parameters did not matter. Proceeding to the survey results of the *Projection View* (see design in Figure 4.7), this view resulted in the

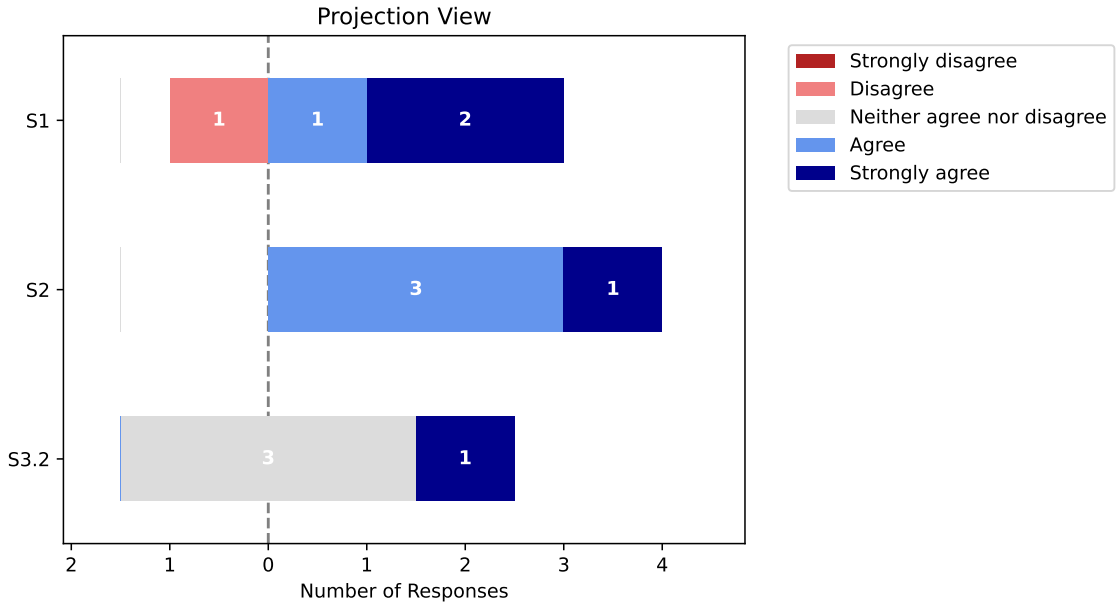



**Figure 6.12:** The Likert plot of the survey results for the *Projection View*.

lowest levels of agreement compared to the other views. While the comparability of the generated samples based on their parameter values in statement *S2* was rated with an agreeing attitude, the answers for the statements *S1* and *S3.2* had a greater variance in the level of agreement as shown in Figure 6.12. Similar to the comments for the *Iterative View* some of the participants stated that they did not focus on the parameter configurations of the individual samples and simply chose the sample that they liked best. Another comment was related to the visual layout based on the number of samples, such that the samples should be displayed as a list for lower numbers of samples and that dimensionality reduction is only applied when exceeding a certain threshold in the number of samples. Alternatively to a 2D embedding using dimensionality reduction, one of the participants proposed embedding the samples in a 2D coordinate system where the two axes represent two input parameters such as the cfg scale and the denoising strength. With no further modifications to the computed coordinates of the dimensionality reduction algorithm, often the problem occurs that images overlap in the *Projection View* as their coordinates are close to each other. This observation has also been stated by a participant, who proposed an alternative list view of the image samples to overcome this problem. The third view that has been covered by the survey is the *Pipeline View* (see design in Figure 4.8). Figure 6.13 shows the survey results for the given view, where all statements generally have been rated with a (strong) agreement in regard to the ease of navigation, the capability to solve the problem and the suitable density of the presented information. Unlike to the other views, half of the participants did not leave any further comments, while the given comments mainly address the presentation of the generation parameters and the image samples when sampling is enabled. One of the two participants stated, that the parameters that are shown in the glyph as well as in the nodes should be better grouped together. In addition to the grouping, the participant noted to use more of the available space in the *Pipeline View*. As shown in Figure 4.8, the projection node only shows the selected image from the image samples that have been generated in the specific step. Related to the projection node, the second participant proposed a modification that enables the user to show the image samples from which the presented image has been chosen. The last

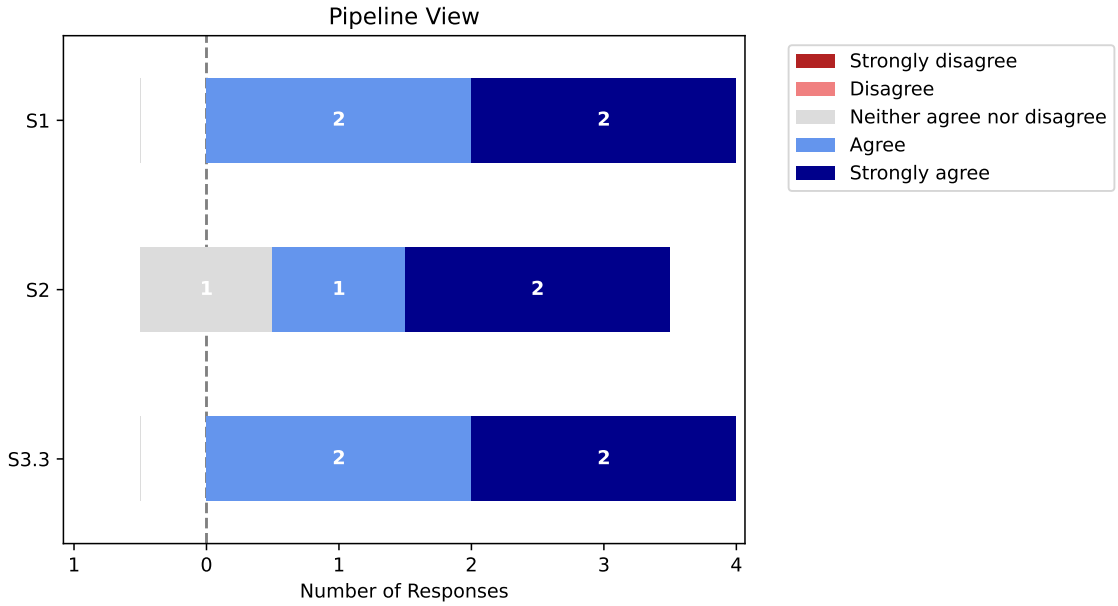



**Figure 6.13:** The Likert plot of the survey results for the *Pipeline View*.

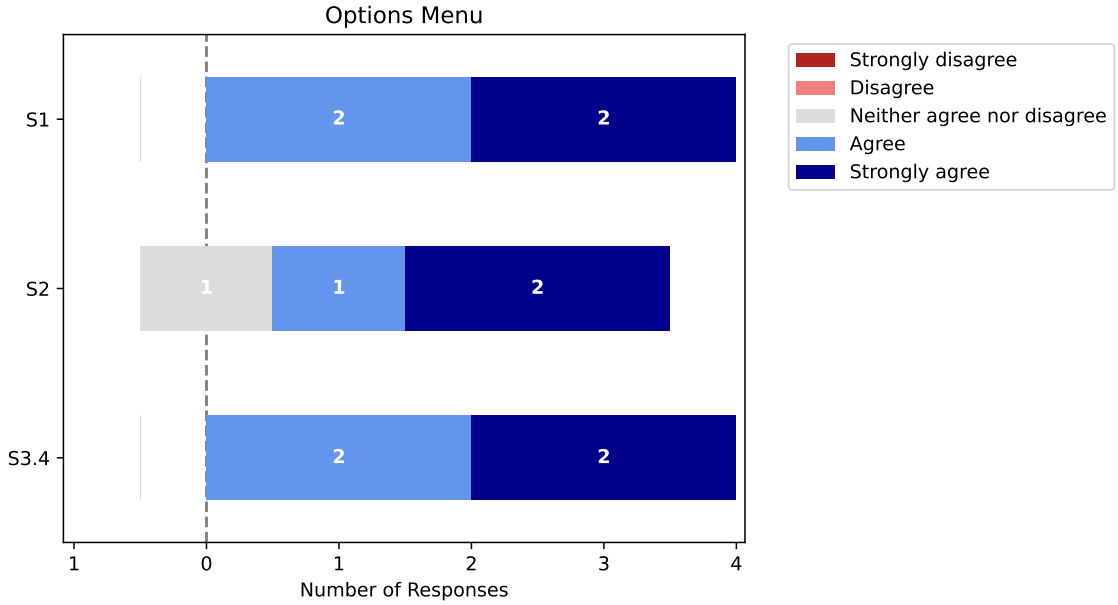



**Figure 6.14:** The Likert plot of the survey results for the *Options Menu*.

view that has been part of the survey is the *Options Menu* (see design in Figure 4.9). The results of the survey strongly resemble those from the *Pipeline View* and are shown in Figure 6.14. All statements that cover the ease of navigation, the capability to solve the problem and the intuitiveness of the structure of the menu have been rated with a (strong) agreement on average. There was only one comment, that addresses the *Initial Text-to-Image* flag which is shown in Figure 4.9. When the user initially transfers the layers from Krita into the tool, the view in the tool is dependent on whether the flag is set or not. When the flag is set, the content of the selected layer is ignored and the user can run a text-to-image generation only based on the provided text prompt. In the other case, where the flag is not set the content of the layer is used as input for the image generation and an image-to-image generation is performed. The participant proposed that this flag is automatically set based on the content of the given layer such that when a blank layer is transferred to the tool the flag is automatically set, as the user likely wants to ignore the content of the empty layer.

To summarize the evaluation chapter, the case study presented in Section 6.1 covered the general workflow of the DiffusionExplorer based on a given use case. In the second part of the evaluation chapter in Section 6.2 the user study aimed to test the tool with real users in order to observe their interaction and workflow with the tool as well as to detect possible improvements. The following chapter discusses the results of the user study together with the individual visual components of the tool as well as the initial requirements gathered in the interview phase.

# 7 Discussion

After performing the user study and collecting valuable feedback from the participants, this discussion chapter aims to address the initial requirements and the workflow of the tool as well as to integrate the results of the user study from Chapter 6. While Section 7.1 compares the requirements and the workflow produced in the interview phase with those reflected by the final state of the DiffusionExplorer, Section 7.2 of this chapter summarizes the limitations based on the presented results.

## 7.1 Requirements and Workflow

Recalling the results of the interview phase presented in Section 4.1, there were eight requirements in total divided into the three categories *basic*, *performance* and *excitement*. In the final state of the tool, all of the five basic requirements have been fully implemented, addressing fundamental functionalities such as a revertible image history, the seamless integration into the painting program Krita as well as sampling and the 2D projection of the resulting images. The image history that is shown in the *Pipeline View* successfully enabled the study participants to revert changes to the image simply by clicking on a previous node. This also supported the willingness to try new parameter constellations, as bad results could be reverted easily. Another basic requirement that worked perfectly fine for the participants was the plugin integration into Krita. With a nearly instant transfer of the layer data, the focus could be effortlessly switched between Krita and the DiffusionExplorer, allowing the fast application of drawing and editing. Reconsidering the results of the survey in the user study related to these basic functionalities, the sampling, as well as the projection functionalities, got the most suggestions for improvement. Especially for inexperienced users, rather the resulting images than the actual model parameters are relevant, which could be addressed by introducing a mode that hides these parameters that are currently visually communicated using range sliders in the *Iterative View* (see Figure 4.6) and glyphs that visually encode the core parameters of the image generation in all major views of the tool.

Another discussion point with a great presence in the user study is the visual presentation of the images in the *Projection View*. As already stated in Section 6.2, a fundamental problem of the current 2D projection in the DiffusionExplorer is the overlap of images that have a high degree of similarity. One possible way to approach this problem is to create an image grid based on the 2D coordinates resulting from the dimensionality reduction. This can be achieved with the tool *Hagrid* that makes use of Hilbert and Gosper curves to handle overlaps in data and creates a grid-like structure [CMC+22]. Another approach to improve the readability of the *Projection View* could also be to focus on two model parameters such as the *cfg scale* and the *denoising strength* and to use them as axis values for a 2D coordinate system. On one hand, this would decrease the complexity of the view for inexperienced users and could improve the capability for direct comparisons of images.

On the other hand, the information on the other generation parameters or the image vectors would get lost, however, this could be integrated as part of the already mentioned mode for inexperienced users in order to reduce the complexity of the sampling and projection process.

The last component of the DiffusionExplorer that is discussed in the context of the basic requirements is the glyph that visually encodes the model parameters that led to the generated images. In the tool, the glyphs are present in the *Iterative View* at the middle-left (see Figure 4.6), in the *Projection View* on hover on an image sample and in the *Pipeline View* on top of the edges that connect the nodes with each other. Being part of multiple comments in the user study and several discussions in the development process, the glyph offers great potential for parameter encoding. As of right now, the *cfg scale*, the *denoising strength* and the selected sampler of the image generation are visually encoded in the glyph. The reason why the scope of the glyph is limited to these parameters is to keep a balance between the density of information communicated by the glyph and the visual readability without overwhelming the user with too much information. There are several ways to modify the current implementation of the glyph including different types of visualizations such as a standard bar chart instead of a radial representation of the data. The amount of parameters that are encoded by the glyph can also be increased by including additional values namely the *random seed* and the text prompt. In the case where the sampling option is enabled, the glyph could also be used to communicate the ranges that have been selected in the generation step with markings of the actual values of the selected image sample. For the purpose of the discussed modifications to the glyph, the library *D3.js* [Bos12] that has been used for the implementation of the glyph offers the tools to apply them.

Proceeding the *performance* category of the set of requirements, the category consists of two requirements that cover the visualization of the context of an image generation as well as the selection of the parameter sampling method. The first requirement is reflected in the *Pipeline View* (see Figure 4.8) where the user can see which input image as well as which parameters led to the result of the following step. The second performance requirement has not been implemented in the DiffusionExplorer and the current sampling method is the uniform random choice from the given parameter ranges. Concerning this topic, the inclusion of additional sampling methods such as cartesian sampling (see Section 3.2) or grid sampling could be considered.

The last category of requirements which only consists of a single requirement is the *excitement* category. The requirement describes the extension of a linear history to enable multiple branches, such that the user can go into multiple directions starting from an image and visually compare the different results. Due to its high complexity, this requirement could also not be implemented in the scope of this work, however, this topic has also been quite popular in discussions with the participants of the user study. This functionality could be visually integrated into the *Pipeline View* which currently shows the linear history of the image generations of the selected layer. The current visualization could be extended such that nodes that represent images could have arbitrary numbers of outgoing edges associated with the different paths that a user took starting from the respective node. Being a technical as well a visual challenge, this topic could be interesting for further iterations of the tool.

After having looked at the requirements of the tool, the next topic that is of interest for this discussion is the workflow in the AI art design. As presented earlier in Figure 4.1, the workflow of the consulted experts in the interviews could be broken down into three main steps, starting with initial text-to-image generations, continuing with sampling and finishing up with image-to-image generations in order to refine the result. After having evaluated the DiffusionExplorer with the same

experts and additional participants in the user study, the workflow the participants applied in the tool strongly differed from the one determined in the interview phase. As already highlighted in the evaluation chapter (see Chapter 6) the majority of the participants included drawing/sketching in their design workflow, resulting in good outcomes in regard to the images they created. As the user study showed, the current state of the tool enables the user to easily switch between Krita and the tool, allowing the inclusion of painting and other operations performed in Krita. However, sometimes the user might only want to modify certain areas in the image. This might be useful in several contexts, e.g. when the user needs to restore a damaged area in the image. To approach this problem, inpainting in combination with diffusion models can be used, as it is already offered by the Stable Diffusion Web UI. This functionality has been discussed multiple times in the development process of the tool and could also be considered when developing future iterations of this tool.

## 7.2 Limitations

After having discussed the initial requirements and the workflow of the DiffusionExplorer combined with the results from the user study, this section aims to present and discuss the limitations of the tool. The limitations can be divided into the technical ones and those that are related to the visualizations themselves.

From a technical standpoint, in each generation step a single image or in the case of sampling all the image samples are stored in the state of the application as they are needed for the image history. For increasing numbers of generation steps, samples and resolutions, this might cause performance issues. However, during the development and the user study, sample sizes of 50 and resolutions of 512x512 did not cause any problems.

Concerning the visual limitations, mainly the *Projection View* and the *Pipeline View* are affected. As all the image samples are embedded in the 2D plane in the *Projection View* the readability of the current visualization strongly depends on the number of samples as well as their degree of similarity. With higher degrees of similarity and samples, the probability of overlaps is increasing which can be coped with techniques as described earlier. However, if larger sample sizes e.g. 100 are needed in certain use cases, techniques such as clustering improve the readability of the visualization. Switching to the *Pipeline View*, the visual limitations in this view are dependent on the number of total generation steps in the image history. One approach to address this scalability problem of the *Pipeline View* could be to visually summarize multiple generation steps into a single node that is displayed in the pipeline. A sample strategy for this could be to summarize multiple occurrences of the same node type that appear in a row. If a user performed multiple image-to-image generations without sampling, he might not be interested in getting all of them displayed after each other in the pipeline, as he is only interested in the result of the last one. Thus, they could be summarized into a single node and be displayed on demand.

# 8 Conclusion

Coming back to the initial motivation and problem statement of this work, the goal was to create a framework that incorporates an interactive way to explore the parameter space of latent diffusion models. After implementing and evaluating the DiffusionExplorer, one can conclude that the current state of the software solution meets the requirements of such a framework. As the user study proved, the participants with different levels of expertise in the field managed to use the tool in the AI art design process and successfully created images that satisfied the requirements of the given task in the user study. The images the participants synthesized with the DiffusionExplorer, including the embedding of individual components could not be generated directly in one generation step with a classical approach. However, through the integration of Krita in the generation process, the participants generated more complex scenes giving them more flexibility and precision regarding the resulting images. The users could transfer the image data between Krita and the tool, giving them the flexibility to draw and apply other editing techniques to the generated images. Core functionalities such as the revertible image history and the parameter sampling have been fundamental features that the participants in the user study made use of when designing their images.

In the greater context of latent diffusion models, tools that address the exploration of the parameter spaces play an important role in improving the understanding of the relationships between the input parameters and the generated images. As already discussed in the previous chapter, there are several starting points concerning future work and research on the topic of parameter space exploration. As the visualizations in the tool are usually limited in the size of data they can communicate without overwhelming the user, the exploration of the parameter space is limited to a subregion of the whole space. Incorporating other forms of visualization in the DiffusionExplorer could enlarge the covered parameter space and further improve the completeness of the exploration. Regarding the integration of the tool into Krita via a plugin, there are also other possibilities to integrate painting and other editing techniques into the image synthesis process. An image editor, that is specifically tailored towards supporting image synthesis could be directly integrated into the DiffusionExplorer, making it a standalone tool independent of additional software such as Krita.

# Bibliography


[Agg18] C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer, 2018 (cit. on p. 7).

[AHK01] C. C. Aggarwal, A. Hinneburg, D. A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Spaces". In: *Proceedings of the 8th International Conference on Database Theory*. ICDT '01. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 420–434. ISBN: 3540414568 (cit. on pp. 5, 6).

[AUT22] AUTOMATIC1111. *Stable Diffusion Web UI*. https://github.com/AUTOMATIC1111/stable-diffusion-webui. Accessed: 2023-10-14. 2022 (cit. on p. 15).

[BDS18] A. Brock, J. Donahue, K. Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: *CoRR* abs/1809.11096 (2018). arXiv: 1809.11096. URL: http://arxiv.org/abs/1809.11096 (cit. on p. 10).

[Bos12] M. Bostock. *D3.js - Data-Driven Documents*. 2012. URL: http://d3js.org/ (cit. on p. 51).

[CMC+22] R. Cutura, C. Morariu, Z. Cheng, Y. Wang, D. Weiskopf, M. Sedlmair. "Hagrid: using Hilbert and Gosper curves to gridify scatterplots". In: *Journal of Visualization* 25 (July 2022). DOI: 10.1007/s12650-022-00854-7 (cit. on p. 50).

[Die03] T. G. Dietterich. "Machine Learning". In: *Encyclopedia of Computer Science*. GBR: John Wiley and Sons Ltd., 2003, pp. 1056–1059. ISBN: 0470864125 (cit. on p. 7).

[DWJ+23] R. L. Davis, T. Wambsganss, W. Jiang, K. G. Kim, T. Käser, P. Dillenbourg. "Fashioning the Future: Unlocking the Creative Potential of Deep Generative Models for Design Space Exploration". In: *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI EA '23. Hamburg, Germany: Association for Computing Machinery, 2023. ISBN: 9781450394222. DOI: 10.1145/3544549.3585644. URL: https://doi.org/10.1145/3544549.3585644 (cit. on p. 19).

[FWW+23] Y. Feng, X. Wang, K. K. Wong, S. Wang, Y. Lu, M. Zhu, B. Wang, W. Chen. *PromptMagician: Interactive Prompt Engineering for Text-to-Image Creation*. 2023. arXiv: 2307.09036 [cs.AI] (cit. on p. 19).

[HR02] G. E. Hinton, S. Roweis. "Stochastic Neighbor Embedding". In: *Advances in Neural Information Processing Systems*. Ed. by S. Becker, S. Thrun, K. Obermayer. Vol. 15. MIT Press, 2002. URL: https://proceedings.neurips.cc/paper_files/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf (cit. on p. 8).

[HW79] J. A. Hartigan, M. A. Wong. "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 100–108. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2346830 (visited on 10/21/2023) (cit. on p. 6).

[Kur21] K. Kurzhals. “Image-Based Projection Labeling for Mobile Eye Tracking”. In: *ACM Symposium on Eye Tracking Research and Applications*. ETRA ’21 Full Papers. Virtual Event, Germany: Association for Computing Machinery, 2021. ISBN: 9781450383448. DOI: 10.1145/3448017.3457382. URL: https://doi.org/10.1145/3448017.3457382 (cit. on pp. 8, 10).

[LC21] V. Liu, L. B. Chilton. “Design Guidelines for Prompt Engineering Text-to-Image Generative Models”. In: *CoRR* abs/2109.06977 (2021). arXiv: 2109.06977. URL: https://arxiv.org/abs/2109.06977 (cit. on p. 18).

[MH08] L. van der Maaten, G. Hinton. “Viualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605 (cit. on pp. 7–9).

[MHM20] L. McInnes, J. Healy, J. Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML] (cit. on pp. 8–10).

[ML21] X. Mao, Q. Li. *Generative adversarial networks for Image Generation*. Springer, 2021 (cit. on p. 10).

[MPH07] L. van der Maaten, E. Postma, H. Herik. “Dimensionality Reduction: A Comparative Review”. In: *Journal of Machine Learning Research - JMLR* 10 (Jan. 2007) (cit. on p. 7).

[OC12] A. Octavian, A. Cotîrlet. “Kano Model”. In: Issue 2/2012 (2012). URL: https://www.ugb.ro/etc/etc2012no2/18_Paraschivescu_final.pdf (cit. on pp. 22–24).

[PBM] N. Piccolotto, M. Bögl, S. Miksch. “Visual Parameter Space Exploration in Time and Space”. In: *Computer Graphics Forum* n/a.n/a (). DOI: https://doi.org/10.1111/cgf.14785. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14785. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14785 (cit. on p. 16).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 6).

[RBL+21] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer. “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *CoRR* abs/2112.10752 (2021). arXiv: 2112.10752. URL: https://arxiv.org/abs/2112.10752 (cit. on p. 4).

[RBL+22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV] (cit. on pp. 4, 11, 12).

[S23] K. S. *The Design Thinking Process - How does it work? - MAQE - Insights*. https://www.maqe.com/insight/the-design-thinking-process-how-does-it-work/. (Accessed on 05/07/2023). 2023 (cit. on pp. 13, 14).

[SHB+14] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, T. Möller. “Visual Parameter Space Analysis: A Conceptual Framework”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2161–2170. DOI: 10.1109/TVCG.2014.2346321 (cit. on pp. 16–18).

[SMM12] M. Sedlmair, M. Meyer, T. Munzner. “Design Study Methodology: Reflections from the Trenches and the Stacks”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (2012), pp. 2431–2440. DOI: 10.1109/TVCG.2012.213 (cit. on p. 13).

[SPTS19] P. Sitikhu, K. Pahi, P. Thapa, S. Shakya. *A Comparison of Semantic Similarity Methods for Maximum Human Interpretability*. 2019. arXiv: 1910.09129 [cs.IR] (cit. on p. 6).

[SWMG15] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, S. Ganguli. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG] (cit. on p. 11).

[TTV18] H. Thanh-Tung, T. Tran, S. Venkatesh. “On catastrophic forgetting and mode collapse in Generative Adversarial Networks”. In: *CoRR* abs/1807.04015 (2018). arXiv: 1807.04015. URL: http://arxiv.org/abs/1807.04015 (cit. on p. 10).

**Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature