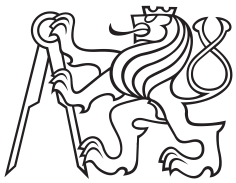
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Text-driven Real-time Video Stylization using Diffusion Models

Bc. David Kunz

Supervisor: prof. Ing. Daniel Sýkora, Ph.D.
Field of study: Artificial Intelligence
January 2024

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kunz David** Personal ID number: **467838**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master's thesis details

Master's thesis title in English:

Text-driven Real-time Video Stylization using Diffusion Models

Master's thesis title in Czech:

Textem řízená stylizace videa v reálném čase s využitím difuzních modelů

Guidelines:

Study methods for transferring an artistic style to a video using a set of hand-drawn keyframes [1, 2, 3]. Explore also techniques for image synthesis [4, 5] based on diffusion models [6] that can be used to prepare stylized keyframes automatically. Implement a tool that allow to stylize a live video in real time [2] using keyframes generated by text-driven diffusion process. Verify the effectiveness of the proposed solution on a simple video conferencing application that enables the user to change his or her visual appearance in the live video on the fly using interactively by typing text prompts.

Bibliography / sources:

[1] Jamriška et al.: Stylizing Video by Example, ACM Transactions on Graphics 38(4):107, 2019.
[2] Texler et al.: Interactive Video Stylization Using Few-Shot Patch-Based Training, ACM Transactions on Graphics 39(4):73, 2020.
[3] Futschik et al.: STALP: Style Transfer With Auxiliary Limited Pairing, Computer Graphics Forum 40(2):563–573, 2021.
[4] Rombach et al.: High-Resolution Image Synthesis With Latent Diffusion Models, Proceedings of Conference on Computer Vision and Pattern Recognition, pp. 10684-10695, 2022.
[5] Brooks et al.: InstructPix2Pix: Learning to Follow Image Editing Instructions, CVPR 2023.
[6] Ho, et al.: Cascaded Diffusion Models for High Fidelity Image Generation, Journal of Machine Learning Research 23(47):1-33, 2022.

Name and workplace of master's thesis supervisor:

prof. Ing. Daniel Sýkora, Ph.D. Department of Computer Graphics and Interaction

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **18.09.2023** Deadline for master's thesis submission: _____

Assignment valid until: **16.02.2025**

prof. Ing. Daniel Sýkora, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. | _____
Date of assignment receipt | Student's signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

I would like to thank my tutor, Prof. Daniel Sýkora, for his assistance and valuable advice during this project. His positivity and motivation significantly contributed to the completion of this work. Additionally, I would like to acknowledge the foundational work of the authors whose research has been instrumental in shaping this project. Their pioneering efforts laid the groundwork for this project. I also wish to extend my gratitude to my partner and family for their ongoing support, not just during my thesis work but in all my academic pursuits. Finally, I appreciate all the subjects who generously agreed to participate by allowing their videos to be included in this thesis.

Declaration

I declare that all the work presented in this project is my own. No part of this work has been plagiarized or copied from another source without proper attribution. I have upheld and respected the principles of academic honesty and integrity in completing this project.

Prague, January 9, 2024

Prohlašuji, že jsem předloženou práci vypracoval samostatntě a že jsem uvedl veškeré použité zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 9. ledna, 2024

Abstract

This thesis presents a new approach to text-driven video stylization. The primary objective was to enable users to transform a live video stream using simple text prompts while ensuring real-time performance and maintaining high visual quality. Our approach combines the complementary capabilities of InstructPix2Pix and StyleVid to synthesize stylized keyframes and efficiently propagate them across video frames. This hybrid pipeline achieves over 30 fps performance with minimal latency and style delay on typical video conferencing footage. The system not only showcases fast performance and interactive text prompt capabilities, but also excels at producing diverse and visually compelling stylizations in diverse video scenarios. Despite its strengths, the method also reveals limitations in keyframe consistency and coverage, especially for motions beyond standard video call scenarios. User feedback from public demonstrations confirms the system's intuitive use and expressive potential. Future work aims to enhance keyframe consistency and automate frame selection to further refine this novel avenue in text-guided real-time video stylization.

Keywords: style transfer, video stylization, image synthesis, latent diffusion models, real-time processing

Supervisor: prof. Ing. Daniel Sýkora, Ph.D.
FEE, Department of Computer Graphics and Interaction

Abstrakt

Tato práce zpracovává nový přístup k textově řízené stylizaci videa. Hlavním cílem je interaktivní transformace živého video streamu pomocí jednoduchých textových příkazů zadaných uživatelem při zachování vysoké vizuální kvality videa. Naše metoda kombinuje komplementární schopnost syntézy stylizovaných klíčových snímků a jejich efektivního šíření u modelů InstructPix2Pix a StyleVid. Toto hybridní řešení dosahuje výkonu přes 30 snímků za vteřinu s minimální latencí a prodlevou stylů na typických video konferenčních záběrech. Systém také vyniká v produkci rozmanitých a vizuálně přesvědčivých stylizací videí ve vnitřním i vnějším prostředí. Omezení spočívá v konzistenci a pokrytí klíčových snímků, zvláště při komplexních pohybech uživatele během videohovoru. Zpětná vazba uživatelů při veřejných prezentacích potvrzuje intuitivní použití systému a jeho expresivní potenciál. Možnosti dalšího vývoje metody pro textově řízenou stylizaci videa v reálném čase jsou především zlepšení konzistence klíčových snímků a v automatizaci výběru snímků.

Klíčová slova: přenos stylu, stylizace videa, syntéza obrazu, latentní difuzní modely, zpracování v reálném čase

Překlad názvu: Textem řízená stylizace videa v reálném čase s využitím difuzních modelů

Contents

Figures

Tables

Chapter 1
Introduction

The field of video stylization has emerged as a captivating intersection of art and technology, enabling the transformation of conventional videos into visually engaging artistic expressions. Although existing methods have made strides toward achieving compelling stylizations, real-time processing remains an elusive goal. These methods often rely on precomputed styles, which limit user interactivity or entail computationally expensive (pre)training, thereby constraining their applicability in interactive real-time scenarios such as video conferencing or live streaming.

In recent years, video stylization has garnered substantial attention due to its ability to transform ordinary videos into appealing artistic creations. Despite the advances, real-time video stylization remains a challenging endeavor, particularly when aiming to achieve a high diversity of possible outputs and at the same time speed of execution.

The advent of diffusion models has opened new avenues for image and video synthesis. These models, especially when driven by text inputs, empower ordinary, less artistic, people to create and stylize their own images without the need for a paintbrush or experience in Photoshop. They could enable more natural and interactive user engagement, allowing for customization of styles on the fly based on simple text prompts.

1.1 Problem Statement

Video stylization is a transformation in which a video is artistically altered, changing the video content while preserving its structure. Structure refers to the characteristics describing its geometry and dynamics, e.g., shapes and locations of subjects, as well as their temporal changes. And content refers to the appearance and semantics of the video, such as the colors and styles of objects and the lighting of the scene. An example of style transfer can be seen in Figure 1.1.

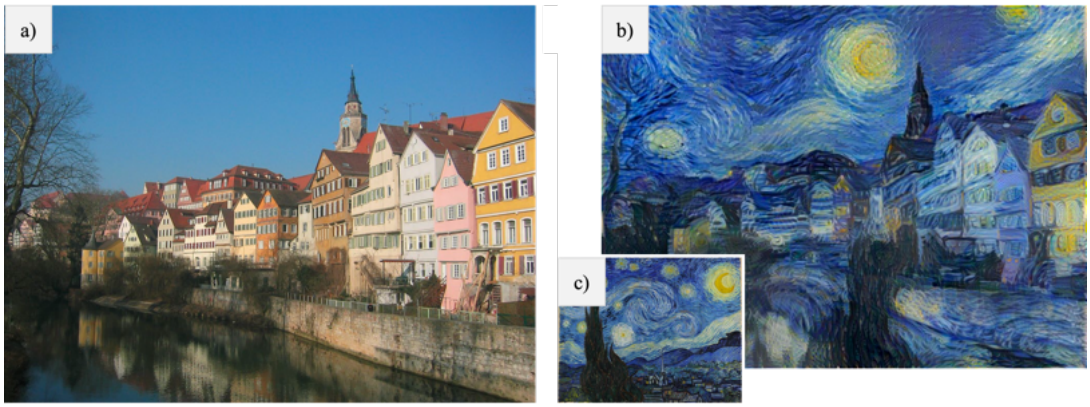


Figure 1.1: An example of a neural network based style transfer technique [13]. The stylized image b) depicts the same semantic content as the source photograph a) styled by the style exemplar, Starry Night by Vincent Van Gogh [1].

This thesis focuses on real-time video stylization employing text-driven diffusion models for style creation and propagation of this created style to the video. The inputs are a continuous live video stream and a text prompt that specifies the desired artistic style. The goal is to apply this style to keyframes taken from the stream and then propagate the style to subsequent frames, ensuring interactive, coherent, and visually consistent stylization. This can be seen in Figure 1.2.

The difficulty of this problem lies in ensuring real-time performance, user interactivity, and high visual quality. Specifically, real-time performance involves minimizing the delay between frame capture and display of the stylized output, while maintaining high frame rates for smooth video playback. Interactivity means minimizing the stylization delay; the time between the user providing a text prompt and observing its stylization effects in the video. At the same time, the stylization must preserve video content, remain temporally coherent, and appear visually compelling throughout the sequence. Achieving these goals simultaneously poses a significant challenge.

1.2 Motivation and Goals

In an era where video content is prevalent and the demand for personalization is high, the capacity to adapt and personalize video streams in real time is invaluable. This thesis dives into this space with a specific goal: to create a real-time live-stream application that utilizes diffusion models to stylize video in a user-interactive manner. Users can alter the visual aesthetic of their video output using simple text prompts, which are converted to styles via a text-to-image process enabled by diffusion models. An example of how a style can be changed using a text prompt can be seen in Figure 1.3. These styles can then be applied to the video frames. This setup brings video stylization directly to users, allowing them to engage with and shape their video experience in real time.

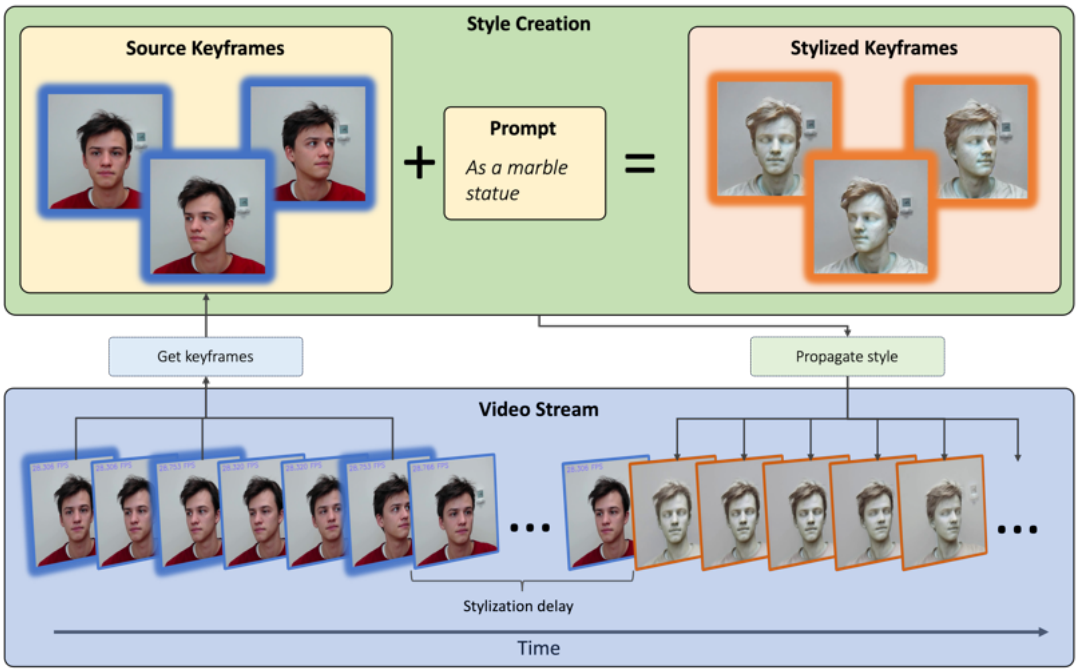


Figure 1.2: Real-time Video Stylization with Text-Driven Diffusion Models. The process begins with a video stream of unstylized input frames (blue border). Keyframes (blue glow), are extracted from these input frames. A text prompt - *"As a marble statue"* - describes the desired transformation to stylize input keyframes, creating stylized keyframes (orange glow). The created style is then propagated as soon as possible to the rest of the video stream's frames (orange borders).

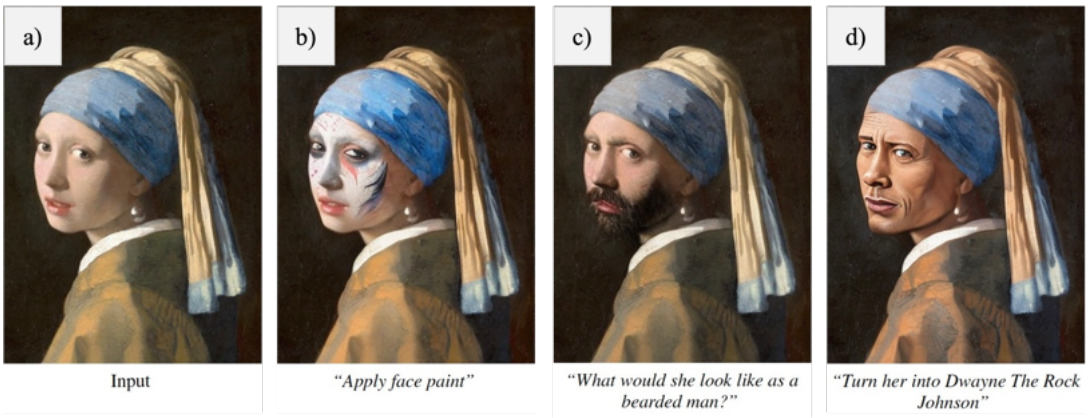


Figure 1.3: Images edited using the InstructPix2Pix [4] model. Given an input image a) and an instruction to edit the image, the model performs the appropriate edit b)-d).

1.3 Outline

This thesis is structured as follows.

- Chapter 2 provides an introduction to neural networks and diffusion models, discussing key concepts like convolutional and residual networks needed to understand

modern video stylization techniques.

- Chapter 3 presents an overview of style transfer methods, categorizing them, and delving into prominent neural network and diffusion model approaches.
- Chapter 4 outlines the design and objectives of the proposed real-time video stylization technique.
- Chapter 5 details the technical aspects of the implementation.
- Chapter 6 evaluates both performance in terms of frame rate and latency and qualitative style coherence of stylizations and discusses limitations and possible future work.
- Chapter 7 summarizes the key contributions of the thesis and suggests avenues for future research.
- Appendices B and C contain additional stylization results and descriptions of attached files.

Chapter 2

Neural Networks and Diffusion Models

In computer vision and image processing, the evolution of neural networks has been at the forefront of numerous breakthroughs, with stylization being no exception. Recent advances in diffusion models represent a significant leap in capabilities for image creation and transformation. As this marks the future direction of stylization, we will first build a strong foundation by introducing neural networks and focusing on their role in the development of diffusion models. We will start with a short historical overview, then discuss convolutional neural networks, and finally diffusion models.

2.1 Historical Context

The concept of neural networks (NNs) was originally inspired by the biological neural system inside our brains in the 1940s, later, in 1957 the perceptron algorithm [40] was introduced, but since it could not solve nonlinear problems (like XOR), it was not successful straight away. In the 1980s, the introduction of backpropagation reignited interest in neural networks by enabling effective training of multilayer structures.

The next important moment was the introduction of LeNet-5 [29] in 1998, one of the first successful applications of Convolutional Neural Networks (CNNs) for document recognition. This was further propelled by the introduction of AlexNet [28] in 2012, a deep CNN that proved the architecture's remarkable performance in image classification, signaling the onset of the deep learning revolution.

In 2014, Generative Adversarial Networks (GAN) [15] were introduced, becoming the state of the art for generative uses of NN, such as image synthesis or style transfer. The following year Residual Networks (ResNets) were introduced [18], helping to solve the problem of vanishing gradients and allowing NNs to have more layers than before.

In 2017, Attention Is All You Need [45] by Vaswani et al. introduced the Transformer model, primarily designed for tasks in Natural Language Processing (NLP). The key innovation of the Transformer was the use of attention mechanisms, which allow the model to focus on different parts of the input data dynamically. This was a key ingredient in image generation as it allowed the models to globally route information between parts of the entire image.

In 2020, OpenAI introduced the CLIP (Contrastive Language-Image Pretraining) model [36], a system designed to understand the relationship between visual and textual information by predicting which images and texts pair together. This understanding was achieved by developing encoders for both modalities and mapping them into a

shared embedding space. The model then assesses the similarity and dissimilarity between these embeddings to identify matching and non-matching pairs of images and text descriptions.

In 2021, diffusion models [43, 22] gained prominence, particularly for their prowess in synthesizing diverse high-quality images, surpassing earlier methods such as GANs and Variational Autoencoders (VAEs) in both complexity and image fidelity.

In 2022, the use of lantent space was adopted in diffusion models by [38]. This proved to be a key feature for flexible conditioning mechanisms, which enabled a wide range of applications including image-to-image and text-to-image synthesis, representing a significant advancement in the field of generative modeling.

2.2 Neural Networks Foundation

In neural network architectures, neurons are fundamental computational units. Layers, composed of these neurons, are stacked to form the network. A neural network comprises of an input layer that receives initial data, followed by one or more hidden layers, and concludes with an output layer. The most basic type of layer is the fully connected layer, characterized by each neuron being connected to every neuron in the preceding layer. In a fully connected layer, the computation involves taking a weighted sum of its inputs, adding a bias term, and then applying a non-linear activation function, such as the Rectified Linear Unit (ReLU). The ReLU function is defined as ReLU(x) = max(0, x), providing a simple yet effective mechanism for introducing nonlinearity into the network.

Mathematically, the transformation of a fully connected layer l can be expressed as:

$$h^{(l)} = f\left(W^{(l)}h^{(l-1)} + b^{(l)}\right)$$

where $h^{(l-1)}$ is the output of the previous layer–the input data for the layer, $W^{(l)}$ represents the weights, $b^{(l)}$ the bias and f the activation function. This ensures that information processed by one layer propagates forward and influences the subsequent layers' calculations.

Different NN architectures replace some of these fully connected layers with specialized layers. For example, a CNN can also have convolutional and pooling layers. A Fully Convolutional Network (FCN) [30] has only convolutional and pooling layers. And a U-Net [39] extends a FCN by also including upsampling layers, such as transposed convolutions, nearest-neighbor upsampling, or pixel shuffle.

Different NN architectures incorporate specialized layers or blocks in place of some fully connected layers to optimize performance for specific tasks, such as image processing. For instance, a Convolutional Neural Network (CNN) adds convolutional and pooling layers. A Fully Convolutional Network (FCN) [30], designed to work on any image size, is composed exclusively of convolutional and pooling layers, entirely omitting fully connected layers. A U-Net [39] builds upon the FCN architecture by integrating upsampling layers, enhancing its capabilities in image reconstruction. Residual Networks (ResNets) [18] utilize ResNet blocks, which are sets of layers featuring skip connections. These connections allow for the direct flow of information across layers, addressing the vanishing gradient problem and improving learning in deep networks.

In the following paragraphs, these layers and concepts will be discussed in detail.

2.2.1 Convolutional Layer

A convolutional layer is the core layer of a CNN. This layer allows the network to gather information from more than just one pixel at a time, also taking into account the values of the surrounding pixels. This is done by a kernel (also known as a filter) that traverses the image grid, computing a dot product at each position, and capturing local information into a feature map. An example can be seen in Figure 2.1. The weights in this layer are stored per kernel and not per input pixel; this allows the layer to process inputs of any size.

The area from which the network gathers information is known as the **effective receptive field** (or field of view). For a single layer, this field is influenced by the hyperparameters of the layer: kernel size, dilation, and stride. **Kernel size** determines the area of the filter; larger sizes capture larger features, but increase the number of weights. **Dilation** introduces gaps between the weights in the filter, enlarging its coverage area and improving its ability to detect larger patterns without an increase in the number of weights. **Stride** specifies the step size the filter takes when traversing the image grid. A higher stride value results in the skipping of more pixels during the filter application, which reduces the spatial dimension of the output feature map and does not introduce any new parameters.

Downsampling, coupled with stacking convolutional layers, increases the effective receptive field as each subsequent layer aggregates features over an expanding area of the input, due to the cumulative effect of convolution operations. This enables deeper layers to capture a wider spatial context.

As we can see in Figure 2.1, convolution cannot be performed along the edges of the image due to the absence of neighboring values. As a result, the output dimensions are reduced by $\lfloor \frac{k}{2} \rfloor$ at each edge for a kernel size k. To address this, the **padding** hyperparameter is used, which either adds zeros or replicates the image values around the edges to extend the input image size. This compensates for the reduction in size and maintains the output dimensions closer to those of the original input.

Real images usually have multiple channels; the kernel is adapted to have a corresponding number of layers. To extract a variety of features, multiple filters can be employed within a single convolutional layer, leading to an output with as many channels as the number of filters used. Figure 2.2 demonstrates the operation of a complete convolutional layer with 2 filters.

2.2.2 Pooling Layer

Another important layer is the pooling layer; it typically follows convolutional layers and does not have any learnable parameters. This layer serves to downscale the input. By doing so it; contributes to robustness, decreases computational requirements and increases the effective receptive field of the network–since the following convolutional layers would now have a larger relative reach.

[1] https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

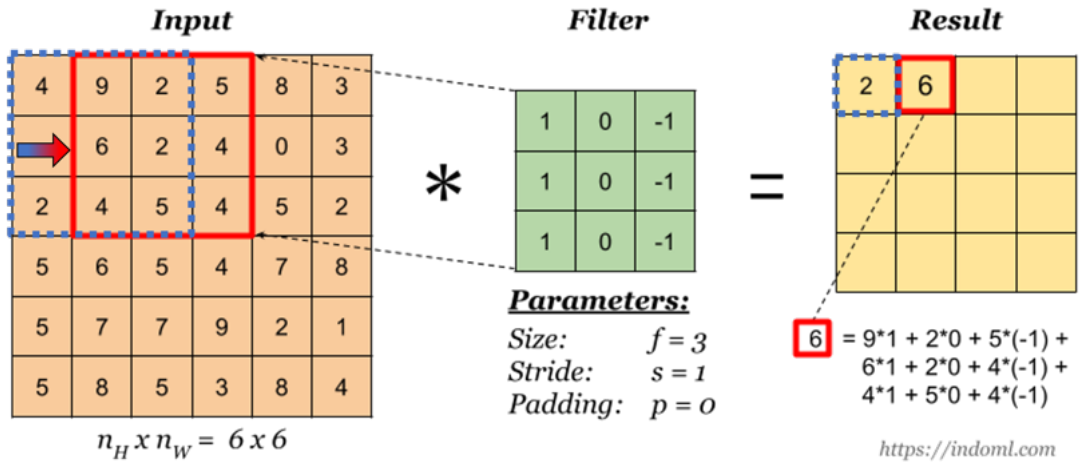


Figure 2.1: Visual representation of a convolution operation. A 3x3 kernel with predefined weights slides across the input image grid, applying a dot product at each position to create a feature map.

The most common form is max-pooling; in this process, a window slides over each feature map, selecting the maximum value that is retained in the output. A typical window size is 2x2 with a stride of 2–this reduces the feature map by half.

2.2.3 Upsampling Layer

In neural network architectures, particularly those applied in image segmentation, super-resolution, and stylization, there is a need to match the spatial resolution of the output with that of the input. To achieve this, upsampling layers are employed to reverse the resolution reduction caused by previous layers. Two prominent types of upsampling are Transposed Convolution and Nearest-Neighbor Upsampling.

Transposed Convolution, also known as fractionally-strided convolution, is a learnable upsampling method. The process involves adding zeros among the elements of the low-resolution feature map to increase its size, then a standard convolutional operation is applied. This not only increases the spatial dimensions, but also allows the network to learn the optimal upscaling patterns during training, thus helping in the reconstruction of lost spatial details.

Nearest-Neighbor Upsampling is a nonlearnable method where the value of the nearest pixel is duplicated to increase the feature map's size. This approach is straightforward and efficient as it does not involve learning; however, it may result in less refined output compared to Transposed Convolution.

See Figure 2.3 for a visual explanation of the upsampling methods.

2.2.4 Residual Networks (ResNets)

Residual Networks (ResNets) are a type of deep neural network architecture that introduced the concept known as *skip connections* (also called residual connections). **Skip connections** allow the activation of one layer to bypass one or more layers and be added directly to the activation of a subsequent layer. This design counters the

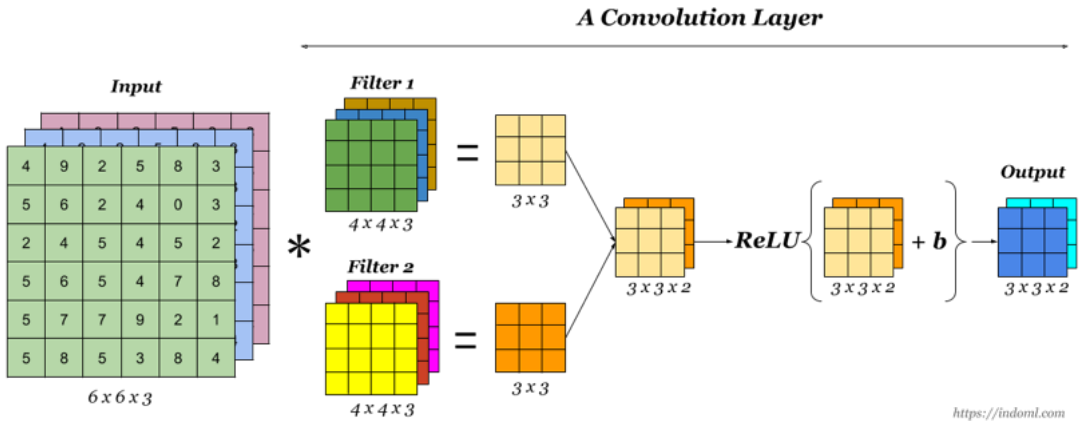


Figure 2.2: A full convolutional layer: An input image with three channels (left) undergoes convolution with two distinct 4x4x3 filters, resulting in two separate feature maps (beige and orange). These feature maps, each with an added bias, are then subjected to a ReLU activation function, yielding activated feature maps (teal and blue). These are then ready to be processed by subsequent convolutional layers, now with two channels. Image taken from indoml.com[1].

vanishing gradient problem by allowing the flow of gradients during backpropagation, thereby supporting the successful training of very deep networks.

The core of ResNets are **Residual Blocks**, which consist of a set of layers followed by a skip connection that adds the block's input directly to its output. This process, known as identity mapping, ensures that the network can at least maintain the performance of the shallower architecture, with deeper layers poised to improve it if they can learn useful features. An example of this can be seen in Figure 2.4.

Batch normalization is another technique used within ResNets that helps stabilize the learning process. By normalizing the input of each layer, batch normalization ensures that the values do not get too high or too low, thus helping in faster and more stable convergence during training.

ResNets also excel in feature reuse, as skip connections allow features from initial layers to be reintroduced into later layers, promoting efficient reuse of features and reducing the need to relearn redundant feature representations.

2.2.5 U-Networks (U-Nets)

U-Nets are characterized by their symmetric structure–in the shape of the letter U. They consist of a contracting path that reduces spatial dimensions and an expanding path that restores spatial dimensions to the size of the input.

The contracting path is composed of repeated modules, each containing convolutions followed by ReLU activation and pooling for spatial reduction. This downsampling process captures the broader context within the image. Then, the expanding path uses upsampling methods to enlarge the feature maps. This part of the network also incorporates skip connections from the contracting path, which reintegrate high-resolution information lost during downsampling.

The U-Net architecture was first used in medical image segmentation, the original architecture can be seen in Figure 2.5.

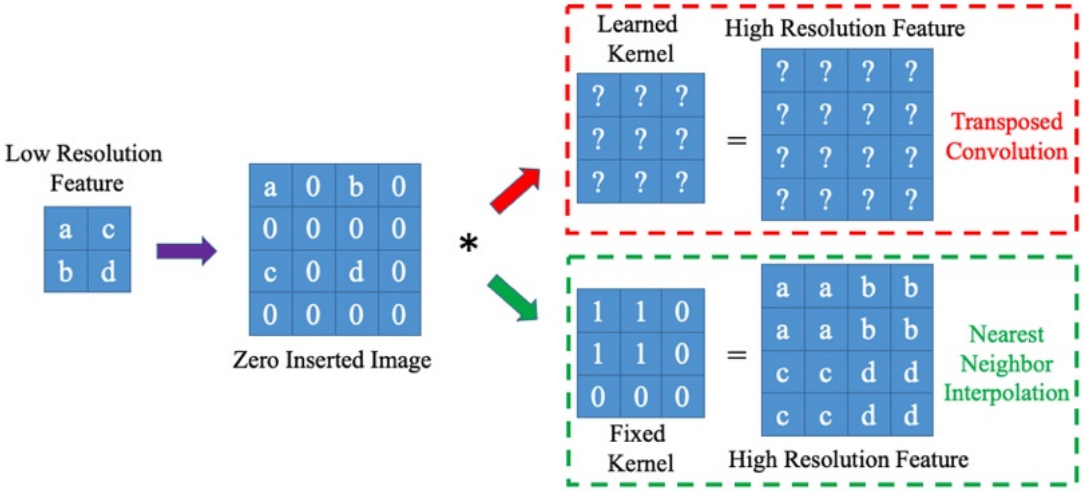


Figure 2.3: A comparison of upsampling methods: Transposed Convolution with zero insertion (top) versus Nearest-Neighbor Interpolation (bottom). Image from [49].

2.3 Diffusion Models Foundation

Denoising Diffusion Probabilistic Models (DDPMs), or diffusion models for short, are a class of generative models distinguished by their two-phase process: adding noise to the data and then, using a UNet, learning to reverse this addition [22]. Unlike traditional generative models that directly generate data, diffusion models iteratively transform data into a noisy state and then back to its original form or a new sample. A visual explanation of this is shown in Figure 2.6.

In the forward phase, the diffusion model systematically adds Gaussian noise to the data through a series of steps, leading to a state similar to random noise. This progression, known as the **diffusion process**, is mathematically modeled as a Markov chain and consists of T distinct timesteps. It starts with $\mathbf{x}_0$, the original image $\mathbf{x}$. Each timestep, labeled $\mathbf{x}_t$, gradually increases the level of noise in the image. By the time it reaches the final timestep T, the data is indistinguishable from random noise.

The reverse phase involves a neural network trained to predict and subtract the noise added at each step of the forward phase. Sequentially applying this denoising process reconstructs the original data or generates new samples. The training objective is to minimize the difference between the actual and predicted noise at each step, optimizing the model's denoising capability.

Mathematically, this loss can be expressed as

$$L_{DM} = \mathbb{E}_{\mathbf{x},\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|_2^2\right],$$

where L_{DM} is the loss function of the diffusion model, calculated as the expected value of the square difference between the real noise ϵ and the noise predicted by the model ϵ_θ for the noised data $\mathbf{x}_t$ at each time step t. This metric assesses the model's ability to accurately restore the original data $\mathbf{x}$ and create new instances from the learned distribution.

This iterative approach of diffusion models, while powerful in handling complex data

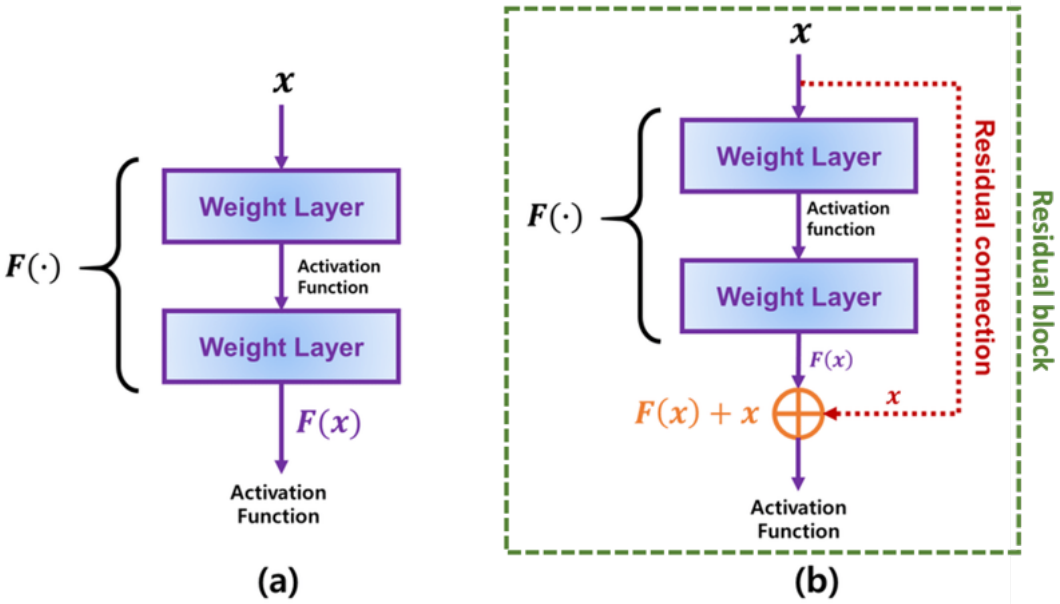


Figure 2.4: Comparison of traditional neural network layers versus a residual block in ResNet architectures. (a) illustrates a typical sequence of weighted layers and activation functions, denoted as $F(x)$. (b) shows the residual block with a skip connection, allowing the input x to bypass the two layers by adding it directly to their output, denoted as $F(x) + x$, before the activation function. Image adapted from [27] under CC BY 4.0 license.

distributions and producing high-quality, diverse images, can also be a limitation in terms of speed of sampling. Specifically, this means that the image passes through the UNet once for each timestep. For larger images (or larger T), this quickly becomes a problem.

To address this limitation in sampling speed, **cascaded diffusion models** [23] have been proposed. They divide the problem into multiple generative tasks. Each task is handled by a different submodel that performs the image generation for the specific scale. For example, one model could generate a 32x32 image, and the next uses this image as a conditional input, creating a 64x64 version, etc. This is repeated to cascade up to the full target resolution. Although conditioning is a key enabler of cascading, it will be revisited more thoroughly later in the text.

An alternative approach to accelerate diffusion models is through latent space processing–**Latent Diffusion Models (LDM)** [38]. This technique significantly improves computational efficiency by reducing the dimensionality of the data, thus abstracting imperceptible high-frequency details, focusing on the most semantically significant aspects.

The model first compresses (encodes) the input image into a lower-dimensional latent representation using a trained perceptual compression model, consisting of an encoder E and a decoder D. This encoding eliminates the need to manipulate full-resolution images directly, but maintains their image-like 2D structure. Once an image is encoded, the noising and denoising steps are performed as before. The decoder D is then used to revert the data back to the image space.

An essential aspect of model generation is the ability to exert control over the output.

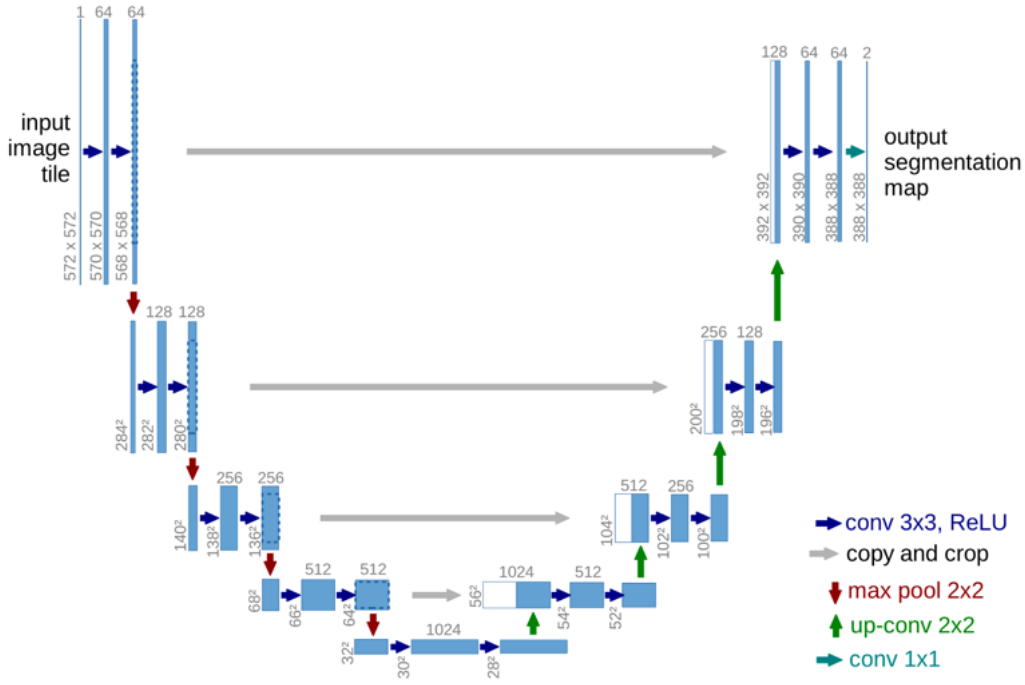


Figure 2.5: Original U-Net architecture of [39].

To achieve this, conditioning is introduced as a means of guiding the generative process.

2.3.1 Conditioning

Conditioning is a way to make diffusion models create specific types of images or follow certain themes. Instead of just copying what they learned, they use special signals or inputs such as ImageNet [7] image classes, captions, segmentation information, or even other images (as mentioned earlier with cascaded diffusion models) to direct the model toward the exact picture you want. An example can be seen in Figure 2.7.

Conditioning inputs are integrated into the diffusion model's architecture and training process. This can be achieved by concatenating the conditioning data with the input at each timestep of the diffusion process or by modifying the neural network architecture to include additional pathways or layers specifically for processing the conditioning information–such as cross-attention mechanisms [6]. The key is to ensure that the conditioning data influences the generation process at each step of the noise addition and removal.

These architectural adjustments affect both the training and inference phases of the model at each time step. During training, the model learns to align its output with the conditioning signals, effectively learning a conditional distribution. During inference, the conditioning signals guide the model to generate output that follows the specified characteristics.

Conditioning inputs, such as text prompts and spatial layouts, can provide useful guidance during sampling from diffusion models. For example, **GLIDE** [34] feeds text embeddings into the attention layers of its denoising model. **DALL-E 2** [37] intead

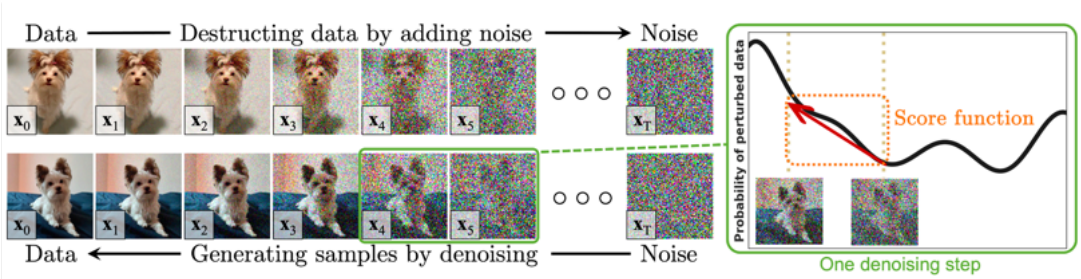


Figure 2.6: Diffusion process in generative modeling. The top row represents the forward diffusion process, where structured data is progressively noised until it becomes indistinguishable from random noise. The bottom row illustrates the reverse process, where noise is removed to reconstruct the input data or synthesize new data instances, guided by the score function as shown on the right, in which the gradient points to the direction of data with higher likelihood and less noise. Image adapted from [46].

encodes the text into a shared text-image embedding using CLIP, which guides its variational autoencoder diffusion model. Meanwhile, latent diffusion models [38]–on which **Stable Diffusion** is based on–allow manipulation in lower-dimensional latent spaces, where text or other signals can be fused to enable conditional generation from compressed representations. **ControlNet** [48] has another approach; it uses specialized neural networks to output modulation signals conditioned on target guidance inputs. The outputs are feature-wise offsets, which are vectors that indicate how much each intermediate feature map in the diffusion model should be modified to reflect the conditioning.

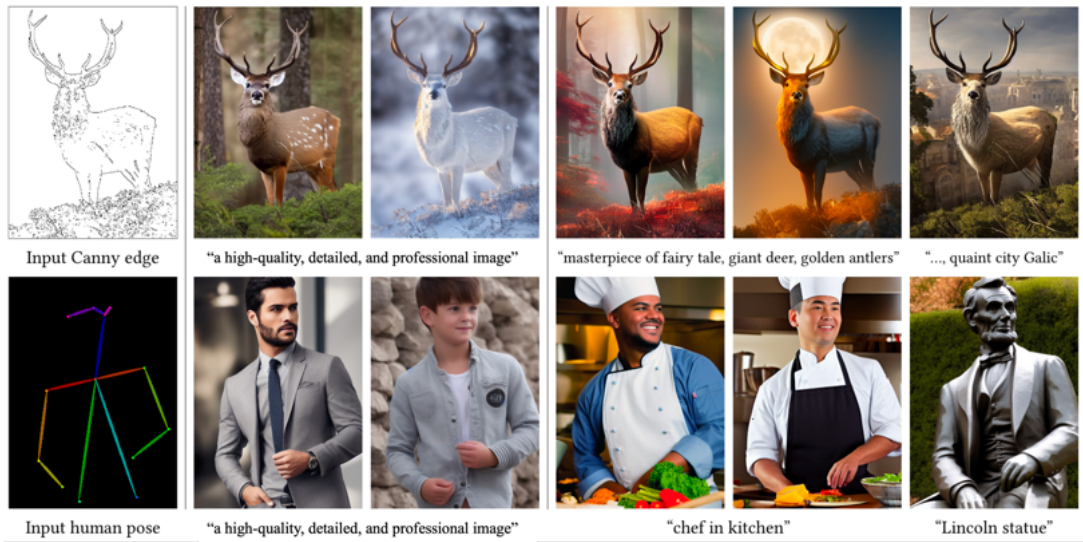


Figure 2.7: Example of conditioning adapted from [48]. ControlNet is used to add conditions; Canny edges (top) and human pose (bottom) to control the generation of image using Stable Diffusion. Stable Diffusion is further conditioned on input text.

Chapter 3

Style Transfer Methods

The concept of stylization lacks a precise definition. It involves altering aspects such as the colors used, the texture patterns left by different artistic mediums, and even changing the shapes of people or objects in an image. The ultimate goal is to give the impression that the image belongs to a particular artistic style. A variety of techniques can achieve this goal through different approaches. In this chapter, we will first categorize stylization methods. Then we will discuss the major works in detail, with a special focus on the papers that gave form to this thesis.

As described by Futschik in [10], style transfer techniques can be broadly categorized into three approaches: procedural methods, non-parametric methods, and parametric/learning-based methods. This classification is based on the fundamental principles of the methods, but it is evident that each one has its own characteristics, advantages, and disadvantages, which leads to distinct and recognizable visual results.

Procedural Methods. Procedural methods work by algorithmically manipulating images to achieve artistic effects. They rely on hand-crafted rules and heuristics to guide the manipulation of content images. For example, Hertzmann's Painterly Rendering algorithm, which uses brush stroke placement rules to give a painted appearance [20].

Non-parametric Methods. Non-parametric methods copy small patches of pixels from style images and paste them onto the content image. The key aspect is selecting which patches to copy from where, in order to produce a coherent and appealing result, this is done by additional guidance mechanisms.

Parametric Methods. Finally, in parametric or learning-based methods, the image data is used to train a model, which is then used for content synthesis. We can further divide these methods into neural and diffusion-based methods. Neural methods rely on convolutional neural networks to encode the content and style information of images in their feature representations. Their loss functions are generally designed to match content while reconstructing style textures. Diffusion-based methods use learned data distributions to guide the creation of images by iterative a process that gradually refines images from a noisy initial state. Conditioning is used to guide this generation to a specific style.

Although procedural methods marked early progress, continued advances have primarily emerged in non-parametric and parametric techniques. We now delve deeper

Figure 3.1: An example of the stylization setup from Image Analogies [21]. A content image A and its styled counterpart A' form the desired transformation. The goal is to perform the same transformation on image B creating image B'.

into the latter two families, which represent the current state-of-the-art and have greater relevance to the methods explored in this thesis.

3.1 Non-parametric Methods

Non-parametric, patch-based style transfer methods work by synthesizing small texture patches from the style image, and copying those patches over to corresponding regions in the content image. This keeps the style and content somewhat separate–the content outlines and structure remain intact, while style is added on top. This offers a significant advantage for artists, allowing them to focus on style creation and confidently fine-tune their work, with the assurance that any adjustments will result in predictable and coherent changes in the stylization.

A pioneering work in this area is Image Analogies by Hertzmann et al. [21]. Nothing specific about the stylization is assumed. Rather, it requires an example consisting of an image pair that demonstrates the desired artistic stylization. New input images can then be stylized in the same manner as the example stylized output. This setup can be seen in Figure 3.1.

Image Analogies, however, have some shortcomings due to its reliance on only local pixel neighborhoods to infer the stylization transform. This has two main issues; first, the perfect alignment between the original and the stylized image becomes crucial for effective style transfer. Second, the method is restricted to making only local changes in the image, rendering high-level analogies impossible. Still, Image Analogies were an influential milestone in example-based style transfer.

Jamriška et al. adapted this approach with great success, managing to apply the concept to coherent video stylization [25] and creating a tool called EbSynth[1]. This method specifically aims to provide artists with more direct control over the stylization process. Prior methods like those developed by Gatys et al. [13] and Ruder et al. [41] mainly addressed the global appearance of the target to approximate a given visual style, but lacked mechanisms for explicit local artist control.

To ensure a semantically meaningful and time-coherent transfer, several guidance channels are calculated from the input video. The guides are;

[1] https://ebsynth.com

- **Color Guide:** Maintain the original color of the video frames, capturing the visual changes in the scene.
- **Mask Guide:** Useful in layered styling and managing occlusions, like in green-screen scenarios.
- **Positional Guide:** This involves creating a map based on optical flow to resolve ambiguity between distinct features with similar appearance that should be stylized differently.
- **Edge Guide:** Focuses on highlighting edges, a critical element in many art styles.
- **Temporal Guide:** Works to minimize changes in stylization from one frame to the next, keeping the video's flow consistent.

These guidance channels are incorporated into a patch-based error metric that is optimized in the StyLit algorithm [9] to balance the adherence to the example stylization with the various guidance constraints.

When the scene undergoes changes (such as a face turning) revealing previously unseen features, the artist may specify additional styled keyframes to style these areas. To handle multiple keyframes, the video is first stylized by each keyframe separately, then the resulting frames are blended together. Rather than using a linear blend, which would diminish the high-frequency contrast of the artistic medium and cause a ghosting effect, the pixels are selected from either stylized sequence. Using the fact that the patch matching error for each pixel and for each stylized sequence is known, the authors chose to prefer the pixel with a lower match error. This is further controlled by a temporal coherence pixel selection mask, which helps to avoid frequent alternations between choosing the contents from the stylized sequences. Finally, global color histogram blending is used to ensure smooth variation over time.

3.2 Parametric Methods

Parametric leverage the representational power of either a neural network or a diffusion model to directly learn a desired image manipulation task, without the need for extensive feature engineering or specific algorithm design. One approach is to design and train a model from scratch specifically for the artistic stylization task, using example style images to guide the training just as in Image Analogies. However, recent breakthroughs in deep learning allow us to leverage large pre-trained models. These models have already learned powerful general-purpose representations of images by training on massive diverse datasets. Either they can be fine-tuned on specific stylization data. Or alternatively, the pre-trained models can be used directly as fixed feature extractors, transferring their learned knowledge about images to the style transfer application without extra training. Additionally, some diffusion models can be used as-is to directly stylize images and videos by manipulating their generative capabilities by augmenting them with other controlling mechanisms such as attention or feature-wise offsets, without the need for any fine-tuning or feature extraction.

Compared to non-parametric or procedural methods, parametric methods can represent highly complex mappings between domains, allowing them to create more

flexible or complex transformations. The learned end-to-end mapping fuses style and content together, making them able to do more global transformations, which can be difficult to achieve using non-parametric patch-based methods. However, this makes the stylization more of a "black box"; with more entagled style and content, it can be hard for an artist to directly and predictably modify the style (procedural methods expose more parameters, and non-parametric approaches let the artist edit the style exemplar). In some regards, especially in diffusion model image generation and stylization, this can also be advantageous, as it offers a straightforward and stress-free experience for the user, eliminating the need for any parameter adjustments and ensuring an automatic stylization process. In other scenarios, particularly in video stylization using diffusion models, the lack of controllability emerges as a major challenge. When the stylization is done independently frame-by-frame, each output frame can come out looking starkly different from its neighbors, even in videos with small frame-to-frame differences in the input. These inconsistencies result in a flickering effect that disrupts the visual continuity and are a central focus of every video stylization and generation paper employing diffusion models.

3.2.1 Neural Methods

Gatys et al. [13] pioneered artistic style transfer using convolutional neural networks (CNN). Their key insight was that CNNs trained for image recognition capture both style and content information in different layers of the network. Style is encoded in early convolutional layers that capture textures, while content is preserved in deeper layers.

Their algorithm works by optimizing a target image to match the content features of one image and the style features of another. Specifically, they define a content loss between activations of a content image and the target image in deeper CNN layers. They also define a style loss between early layer activations and Gram matrices. Minimizing both losses with gradient descent synthesizes an artistic hybrid image that reflects content and style.

A limitation is that the optimization process is slow, taking minutes per image. But the work inspired many follow-ups to accelerate the method and opened the door to using pre-trained CNNs as powerful feature extractors for style transfer and other image manipulations.

Ruder et al. [42] addressed this by formulating feedforward networks to perform stylization in real-time by learning the image transformations. Specifically, they train CNNs that directly map an input image/video frame to a stylized output in a single forward pass. This is achieved by using perceptual losses that match the output style and content statistics to the reference style image.

To deal with temporal coherence in videos, their feedforward architecture uses diluted convolutions and multi-frame training introducing temporal consistency losses between frames by using optical flow warping to achieve smooth stylized videos. They also develop a multi-pass optimization strategy that processes the video bidirectionally, blending stylized frames to prevent quality degradation along occlusions.

Compared to optimization-based approaches, this achieves a 3 orders of magnitude speedup (~400 ms per frame), allowing for almost real-time editing effects. A limitation

is that their method requires training a new network per style rather than the flexible optimization of Gatys et al.

Building on previous work by Futschik et al. in [11], Texler et al. developed StyleVid [44], an approach to fast and flexible video stylization using neural networks. Their goal was to achieve the semantic faithfulness of patch-based methods like Stylizing Video by Example [25], while enabling real-time performance and the random access of feedforward techniques.

StyleVid's key insight is a patch-based training scheme that acts as an implicit regularizer. Specifically, they train a small UNet on random crops from just a few stylized keyframes, compared to the much larger dataset required by Futschik et al. [11]. The cropped patches combined with proper batch size and network capacity tuning prevent overfitting to this limited training data. This allows the network to better generalize to new frames of a similar video while still reproducing the artistic style.

The training loss function combines adversarial, perceptual VGG, and color reconstruction terms. Once trained, arbitrary frames from the same video can then be stylized in a non-sequential fully convolutional pass. This is because the network learns a set of translation filters that can be applied to input of any resolution, not just the fixed patch size used during training. So there is no need to explicitly blend overlapping patches.

For temporal coherence, StyleVid first relies on the network's implicit learning, since the training set contains frames from a coherent video. However, with temporal flicker still apparent, the authors identify two main sources: (1) temporal noise in the original video and (2) visual ambiguity of the stylized content. They address input noise with motion-compensated bilateral filtering as a pre-process which loads surrounding frames but can still run in parallel. They address ambiguity with an auxiliary input layer of sparse, registered Gaussian noise that disambiguates local regions. Another, faster, domain-specific method could assume that the target object we want to stylize doesn't have any ambiguous regions and use a mask to train the method only on this target area (e.g. think a face infront of a white wall). So, while some coherence does depend on seeing nearby frames, the stylization pass itself remains fully independent and parallelizable, maintaining lightweight processing compared to sequential approaches that pass intermediate stylized results between frames.

Building on the idea of fast (real-time) frame-independent stylization, Futschik et al. introduce STALP (Style Transfer with Auxiliary Limited Pairing) [12]. The key insight in STALP is to utilize information from both the stylized keyframes as well as the unstylized target video frames during network training. This enables them to achieve temporal stability without explicit guidance and better preserves style transfer to frames with a larger deviation from the original keyframe.

To ensure that keyframe stylization is as close as possible, a L_1 loss is calculated between network output and ground truth stylized frames. As a complementary objective, to act as a regularizer to enforce consistency on unstylized target frames, the authors use a stylization loss similar to that of Gatys et al. [13] between network output and style exemplars computed on VGG features. By considering both sources of data in calculating loss, the network generalizes better to new frames compared to only seeing the sparse stylized keyframes. The VGG style loss allows the network to implicitly learn coherence without needing specialized losses tuned for video data.

Compared to StyleVid, a limitation of STALP is significantly longer training times due to the costly optimization for similarity of Gram matrices. However, the benefit is improved temporal stability and preservation of style details even with greater appearance changes in the target video. This increased robustness enables applications beyond video including autopainting of panorama images, stylization of 3D renders, or of different portraits captured under similar illumination conditions.

3.2.2 Diffusion Methods

Diffusion models present a generative approach to neural style transfer based on iterative refinement. They are trained to gradually perturb and noise a content image over hundreds of repeated denoising steps. The conditioning provided at each step steers this noisy diffusion process to reveal a final stylized output.

A major advantage over feedforward methods is the ability to train on large diverse datasets of artistic images, capturing complex styles. The iterative procedure also provides more control, as intermediate outputs can be observed and the conditioning adjusted. However, repeated denoising incurs a computational cost, taking orders of magnitude longer to stylize an image. There is also a trade-off between quality and coherence when applying diffusion models to video frames.

Specifically, diffusion models can synthesize artistic images guided by a reference style–usually in the form of a text description, but fine-tuning, specialized attention, or other guidance mechanisms are required to ensure temporal consistency across video frames.

The most straightforward approach to stylizing a video with diffusion models is to simply treat each frame independently as an image. We take an off-the-shelf text-to-image diffusion model and iteratively denoise the input frames that have been noised with Gaussian noise, just as in SDEdit [32]. The text prompt guides the emergence of details as each noised frame is denoised, steering the artistic stylization.

The level of noise added an then removed controls how much the original structure is retained - higher noise allows larger changes, but risks incoherence between frames and vice versa. This trade-off between faithfulness to the desired style and alignment to the original frame is shown in Figure 3.2.

Other naive approaches, such as independently applying InstructPix2Pix (IP2P) [4] to each frame, also exhibit flickering without explicit temporal context. IP2P parameters controlling adherence to text and to the image can partially mitigate this, but changes are made blindly without considering video structure. We will go into more depth about IP2P in a later section.

While single-image diffusion models enable powerful synthesis abilities, applying them to video input directly overlooks crucial dependencies in time. Depending on the strategy these methods use to overcome this, we can categorize techniques for video stylization using diffusion models into two main approaches:

1. **Pure Diffusion Methods**; Methods that create each frame independently using a diffusion model, but incorporate additional constraints or guidance to maintain temporal coherence across frames. Most of these methods treat each frame separately during stylization but consider relationships between frames. We discuss such architectures in Section 3.2.2.

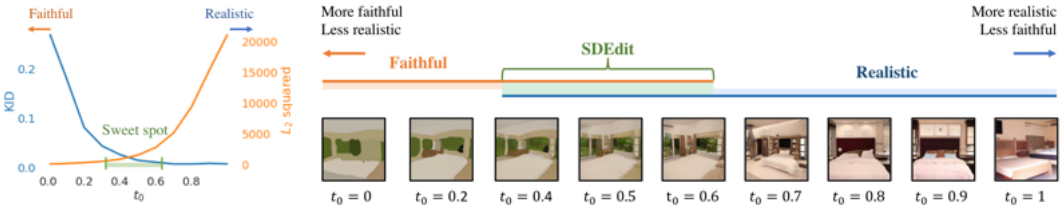


Figure 3.2: This diagram illustrates the balance between adherence to the guide image and model distribution fidelity in the SDEdit process [32], modulated by the noise level variable t_0. The graph to the left plots two metrics: L_2 (similarity to the guide image) and Kernel Inception Distance (KID) (similarity to the model's image distribution). As t_0 increases from 0 to 1, the L_2 score increases, indicating that the image is diverging from the original guide. Conversely, a decreasing KID score denotes a closer alignment with the model's distribution. The 'Sweet spot' is where an ideal mix of originality and creativity is achieved. The images to the right display this effect visually, with $t_0 = 0$ presenting the unchanged guide image and $t_0 = 1$ offering a completely new image informed by the model's data, with intermediate values depicting the gradual shift between these two extremes.

2. **Hybrid Diffusion Methods**; Methods that use diffusion to create one or more keyframes, then explicitly propagate the changes onto other frames. These methods are inspired by example-based approaches and often use similar methods of propagation, using diffusion only as a style generator. We explore these methods in Section 3.2.2.

Pure Diffusion Methods

Training high-quality generative diffusion models requires substantial computing resources. As such, creating dedicated video models is prohibitively expensive for most researchers and is primarily done by large tech companies. Even when training video models, given the lack of large-scale diverse video datasets, pre-training on images has been shown to improve sample quality and prompt-following abilities.

While training purely on video can better capture motion and temporal patterns, it risks losing the expressive power of image models. Many works balance video training with image training to retain these capabilities.

Rather than training specialized video models, most methods instead start with pre-trained text-to-image models and augment them to handle video generation. The two main strategies are: zero-shot application, directly using the model to stylize video frames, and fine-tuning the model on the target video to improve coherence.

To ensure smooth transitions between frames, these approaches commonly modify self-attention layers to attend to nearby frames. Some also add positional encodings or auxiliary losses to further encourage temporal consistency.

Next, we survey prominent examples of techniques following these different strategies.

One approach is to add structural representations extracted from the input video to guide the stylization process. Methods such as **Gen1** [8] use depth maps to encourage consistency in geometric layouts between frames. These inputs act as scaffolds, helping the model to preserve intrinsic motion and features. Gen1 incorporates CFG scale parameters to control how precisely the outputs must align to this depth guidance,

with blurrier depth enforcing looser consistency. Another CFG parameter balances the adherence to frame coherence versus text conditioning, as Gen1 is trained on both images and videos. Spatio-temporal resnet blocks and attention allow information to propagate across both space and time.

Rather than directly stylizing each frame, some methods first edit an anchor frame and then propagate changes to neighbors. For example, **Pix2Video** [5] performs DDIM inversion on input video frames to obtain partial noise estimates x_T that retain structural information for each frame. It first performs text- and depth-guided diffusion on an anchor frame to achieve the target edit. It then propagates changes to the other frames by injecting the anchor's self-attention features, and at each diffusion step updates the latent code of the current frame guided by the latent of the anchor.

The very new **EMU Video** [14] model from Meta also demonstrates the usefulness of the factorized approach of Pix2Video, this time in text-to-video generation. It consists of three diffusion models: one generates an anchor frame from the text prompt, another synthesizes a low frame rate video conditioned on the anchor, and a final interpolation model increases the frame rate. By explicitly generating an initial frame, it provides a strong conditioning signal to aid in coherent video rendering over time.

FateZero [35] takes a more comprehensive approach than Pix2Video to take advantage of attention to maintain coherence when editing video frames. Specifically, FateZero captures attention maps across all frames during the initial DDIM inversion process under a specific source text prompt describing the original input video (e.g., *"Jeep driving on the road."*). At each inversion timestep under this source prompt, it stores intermediate self-attention maps s^t_{src} encoding spatial relationships and cross-attention maps c^t_{src} relating visual features to text semantics.

Later, during iterative denoising under the target prompt (*"Porsche driving on the road."*), FateZero cleverly fuses these *source* attention maps with the *target* ones. This makes the model focus only on the differences between the source and target prompts. In particular, FateZero uses c^t_{src} to retain the original cross-attention maps for any background regions that are unchanged (*"...driving on the road"*), while allowing cross-attention for the edited parts of the target prompt to be updated based on the changes (*"Porsche..."*). A similar idea is also used to create a spatial blending mask for self-attention. To maintain better temporal consistency, FateZero also adds inter-frame attention.

AnimateDiff [17] introduces a motion module to add animation capabilities to existing personalized text-to-image diffusion models without the need for fine-tuning. The lightweight motion module handles coherence across frames via temporal convolutions and attention and is trained on video data while appended to a base frozen text-to-image model. Once trained, the module can be plugged into models fine-tuned with techniques like LoRA [24], which updates only certain weights to adapt the model to a target artistic domain (e.g. Pokémon). To work correctly, the motion module is designed to be orthogonal to LoRA, ControlNet [48], and other conditioning mechanisms, which means that it does not alter these tuned artistic weights, allowing stylization to be preserved while adding animation capabilities. With the addition of ControlNet, it becomes even more powerful, effectively extending the use of the

method to editing videos, as we can easily use depth or other conditioning extracted from a source video to keep the video's form while making stylization edits.

Similarly to AnimateDiff, **Text2Video-Zero** [26] uses a text-to-image model without any additional training. They create a text-to-video method compatible with ControlNet [48], but instead of training a motion model on video data, they enrich the latent codes (from which frames are later generated) with motion dynamics. This is done by sampling the noise for the first frame, doing Δt steps of the backward diffusion process, then warping it along a translation vector to simulate motion for subsequent frames. After warping, noise is reintroduced to allow flexibility in object motions. The frames created from these latent codes share a certain appearance, making the video more consistent.

Also, as in Pix2Video, it introduces a cross-frame self-attention where the features in each frame attend only those of the first frame.

The authors also experiment with InstructPix2Pix, exchanging self-attention mechanisms with cross-frame attention to create Video InstructPix2Pix which greatly improves the method's per-frame consistency.

The recently proposed **Dreamix** method [33] adapts a cascaded pixel-space conditional diffusion model architecture for the task of video editing. In contrast to previous LDM approaches, it builds on top of the Imagen Video model, which consists of hierarchical text-conditional diffusion models. Before editing a video, Dreamix first fine-tunes this model on the specific input sequence using a mixed objective function - one component focuses on full video reconstruction, learning overall motion patterns, while the other separately reconstructs individual frames to enhance detail. At inference time, it creates a degraded low-resolution version of the input and leverages the fine-tuned model to iteratively upsample this sequence to full resolution guided by the edit instructions.

While **Align Your Latents** [2] focuses on text-to-video generation, it provides useful techniques for adapting image diffusion models for video generation. A text-to-image model is fine-tuned to the video data by feeding in frames in batches (illustrated in Figure 3.3), incorporating temporal convolution and attention. This alignment helps to reduce flickering.

The method also uses masking to enable longer video generation. A model is trained to predict future frames given a starting context, using a mask to differentiate given vs. predicted frames in the batch timeline.

Similarly, using the same masking scheme, a separate interpolation LDM fills gaps between sparsely predicted keyframes. This split addresses memory constraints as we don't have to fit as many frames into one batch.

While Pure Diffusion Methods have advanced video stylization, they are not without limitations. Generating frames individually is computationally demanding and often restricted to short video segments. Moreover, maintaining temporal coherence between frames is still a significant challenge. Despite rapid growth and potential in this field, real-time applications remain out of reach for now.

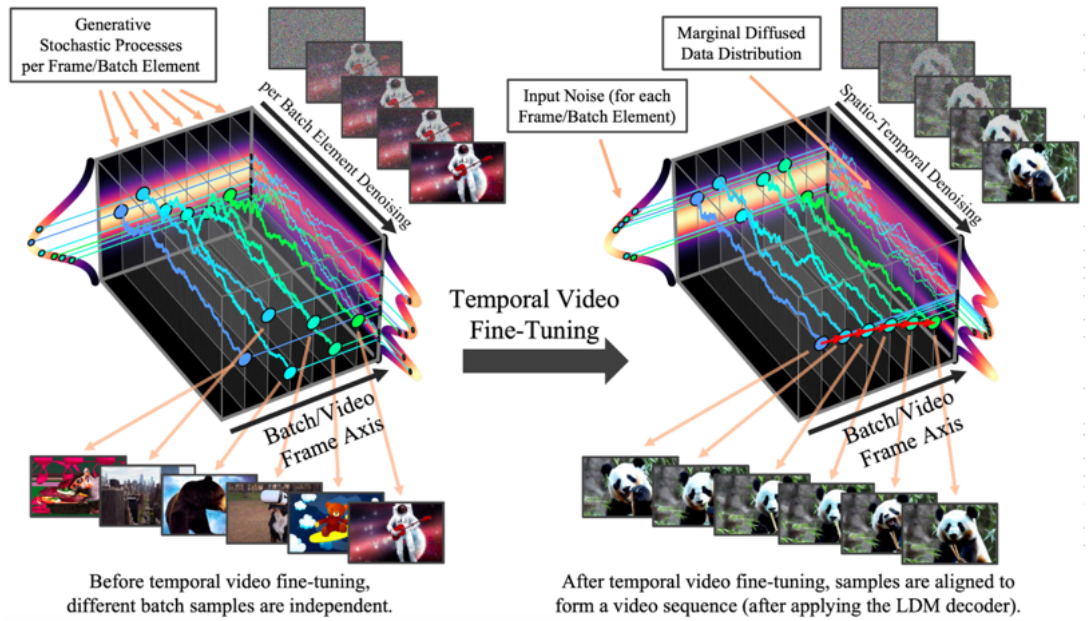


Figure 3.3: An illustration of fine-tuning text-to-image models on video data from [2]. Initially, the samples synthesised from one batch are different, then the fine-tuning takes advantage of batches to feed in video frames as training images, allowing the generation of consistent videos.

Hybrid Diffusion Methods

Hybrid Diffusion Methods combine the strengths of diffusion-based frame generation with example-based propagation techniques. They typically generate keyframes using diffusion models and then extend these styles or changes to the entire video sequence. This approach balances the generative power of diffusion models with efficient propagation, ensuring temporal coherence with less computational demand. A notable strength of hybrid methods is their speed advantage. Although generating a single frame using a diffusion model can take up to several seconds, propagation of these styles or changes across frames, depending on the method, can be much faster. This efficiency in propagation is especially advantageous for video applications, where processing speed is a critical factor, especially for real-time applications.

The recent work **Rerender a Video** by Yang et al. [47] demonstrates impressive results in adapting image diffusion for temporally coherent video stylization. Their key insight is introducing hierarchical cross-frame constraints across both global style and low-level textures.

Specifically, Rerender a Video operates in two phases, stylizing select keyframes using an augmented diffusion model and then propagating these to neighboring frames. A diagram of this strategy can be seen in Figure 3.4. For keyframe stylization, constraints are applied at different diffusion steps to align the shape, texture, and color properties between the current keyframe, the previous keyframe, and the anchor keyframe (the first frame of the video). This coherence also relies on the proposed fidelity-oriented encoding to limit the accumulation of artifacts during repeated latency projections.

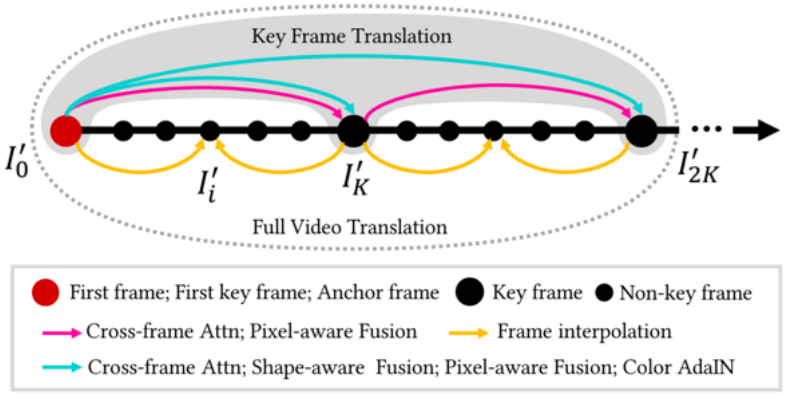


Figure 3.4: Schematic from Rerender A Video [47], showing the two-phase process: the generation of coherent keyframes at every K-th frame of the original video and the propagation to produce a temporally consistent video sequence. First the keyframes are created with attention according to the blue and pink arrows. Then, the frames between these keyframes are stylized using EbSynth [25], the style propagated from the keyframes on both sides, then blended.

The second phase extends the stylization from keyframes to other frames using EbSynth [25] as temporal-aware patch matching and blending.

An obvious limitation of these hybrid approaches is shared by many example-based methods, when a frame contains new content not captured in a keyframe, the stylization can fail to propagate. When using these methods, we have to be careful to choose keyframes that capture the scene.

Another method that can be used to create keyframes is **InstructPix2Pix** (IP2P) [4]. IP2P pioneers a compelling approach for learning text-conditional image manipulation without human-labeled data. It instead creates an entirely synthetic training dataset by chaining the outputs of two powerful pre-trained models: the GPT-3 language model and the Stable Diffusion image generator.

Specifically, GPT-3 is first fine-tuned on a small dataset of image captions and editing instructions to generate text edits. This fine-tuned model produces more than 450,000 text triplets of an input caption, an instruction to edit the image, and an output caption after editing. Then, Stable Diffusion equipped with Prompt-to-Prompt [19] and CLIP generates multiple candidate image pairs per text triplet. For each triplet, 100 pairs are generated with different similarity hyperparameters. A CLIP-based metric then selects pairs where text and image changes align best.

The resulting synthetic dataset of aligned image and text pairs is used to train InstructPix2Pix, a diffusion model conditioned on the input image and text instruction that performs the desired image edit. Though trained purely on synthetic data, InstructPix2Pix generalizes to real images and human-written edits at test time.

We can provide the IP2P model with text prompts describing the desired artistic keyframe style, along with the frame images themselves. Configuration parameters allow controlling the balance between adhering to the text versus adhering to the image–preserving frame layouts and content. IP2P then outputs stylized keyframes reflecting the specified style, which can propagate across video sequences.

Chapter 4

Method

In this chapter, we discuss the approach taken to enable real-time stylization of a live video stream using text prompts. We first examine the overall approach at a conceptual level, with details of specific components elaborated on in later sections.

4.1 Overall Approach

The objective is to stylize a streaming video in real-time based on a textual description of the desired artistic style supplied by the user.

In practical terms, a user would be sitting in front of a camera and monitor to see himself in a live video stream at 30 frames per second (fps). A configuration window with a text box (and other settings) allows the user to enter a text prompt describing the change he/she would apply to the video. With minimal latency after entering a prompt, the user would see himself/herself stylized in real time in the same smooth stream at around 30 fps. The configuration window remains on the screen and lets the user further interact with the stream by entering another prompt or tuning the settings for the previous one. An example of this setup taken during a user demonstration can be seen in Figure 4.1.

To achieve this, we opt for a hybrid technique that takes advantage of the complementary strengths of diffusion models and neural style transfer. This approach takes keyframes from the target video, stylizes them, and then, using these keyframe pairs, propagates the style to the rest of the video using an example-based method. An overview of this pipeline can be seen in Figure 4.2.

Specifically, we use the InstructPix2Pix (IP2P) text-to-image diffusion model [4] to synthesize stylized keyframes from keyframes taken from the video stream. IP2P was chosen for its ability to maintain the identity of the original image better than other generative models when applying edits. As a model that is conditioned on both an image and a text prompt, IP2P takes the original image as input along with the text prompt, allowing it to selectively edit parts of the image described in the prompt while preserving the rest of the content. This makes it well-suited for our use case of stylizing a person in a video while retaining their core visual identity. Additional considerations are discussed in Section 4.2.

The style is then propagated to all other frames of the video using the StyleVid [44] neural network. StyleVid is a good fit due to its ability to train an image stylization model in tens of seconds and run at more than 30 fps for real-time video stylization.

Figure 4.1: Participant using the real-time style transfer application during the Uroboros: Creative AI meet-up.

To further improve the training speed, we apply a mask to select the foreground of each keyframe and select patches only from these regions. Even though we cannot use the same pre-processing methods to deal with temporal consistency as the authors suggest (as their approach needs access to the whole sequence before stylization), the implicit temporal consistency with some additional post-processing on-the-fly shows to be sufficient. Additional considerations are discussed in Section 4.3.

For this to work in a real-time setting, we split the work between keyframe generation (IP2P), style propagation (StyleVid), user interaction for entering prompts, and displaying the live stylized video stream.

4.2 Keyframe Selection and Stylization

To propagate the style throughout the video, we first need to generate stylized keyframes that will serve as examples of the style. The first step is to select the appropriate keyframes from the video stream. The second step is to transform them using the InstructPix2Pix [4] (IP2P) diffusion model.

A good set of keyframes would be those that capture the subject from angles that are likely to be facing the camera at some point in the video stream. In a video conference-like setup, the most important keyframe would be the face captured from

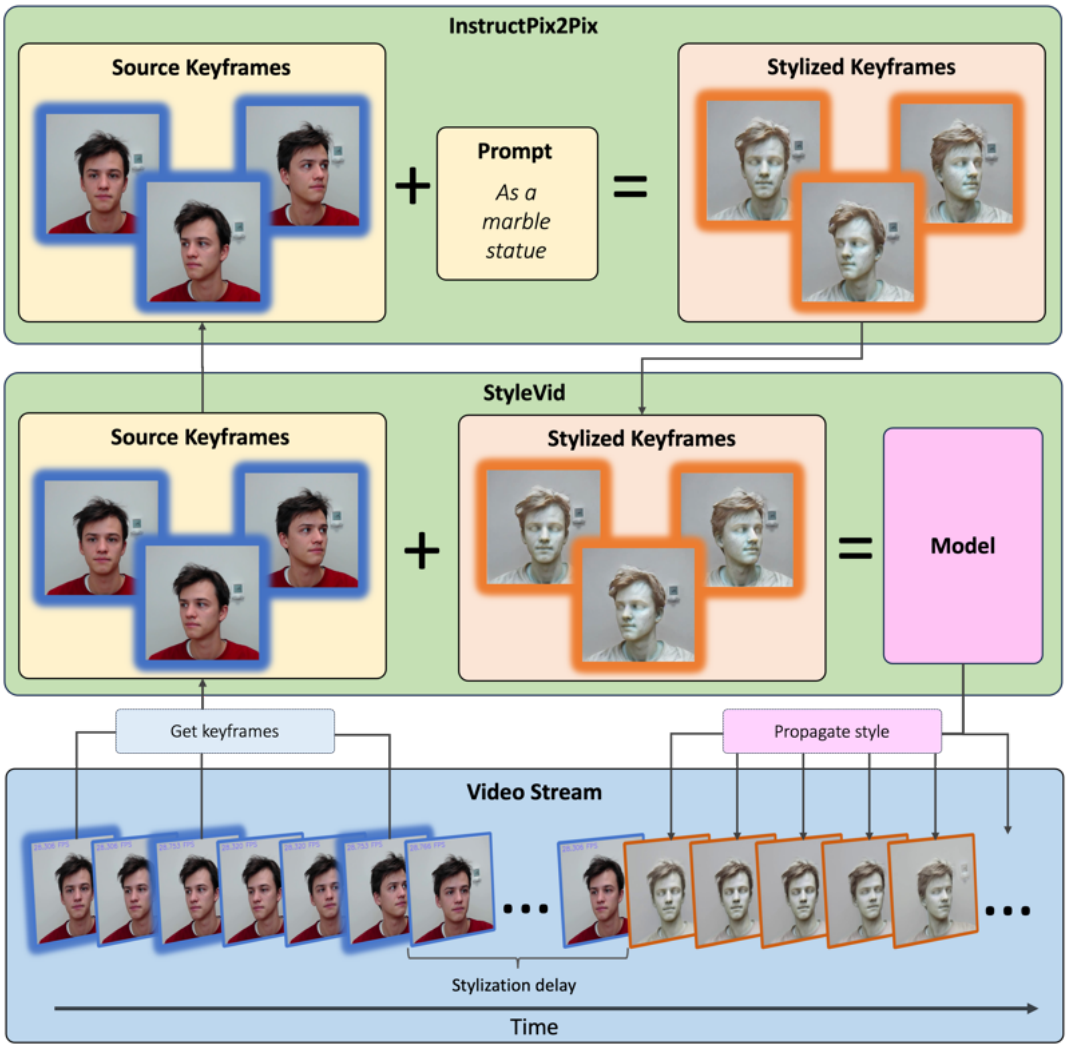


Figure 4.2: Stylizing a video using StyleVid in conjunction with InstructPix2Pix.

the front. This single keyframe can suffice in certain situations, especially when the desired style preserves the low-level features of the face. For example, changing the color tones requires only one keyframe to keep the stylization consistent even when the subject turns, revealing their face from the side.

However, if the stylization consists of a more dramatic change that reveals new occluded parts, a single keyframe may not stylize the revealed parts satisfactorily. In such cases, it is helpful to include multiple keyframes that capture the subject from different angles. When using multiple keyframes, we need to ensure that they are stylized in a consistent manner. This means that if we stylize a front-facing face and then rotate to stylize the face from the side, we should end up with something close to the rotated version of the stylized front-facing face.

When running InstructPix2Pix conditioned on an image, we have no guarantee that a small change in the input image translates into a corresponding change in the generated output when using the same text prompt. To solve this potential incoherence

between multiple stylized keyframes, we concatenate the input keyframes into one joint image and run a single inference. This enforces consistency in the stylization, as can be seen in Figure 4.3.

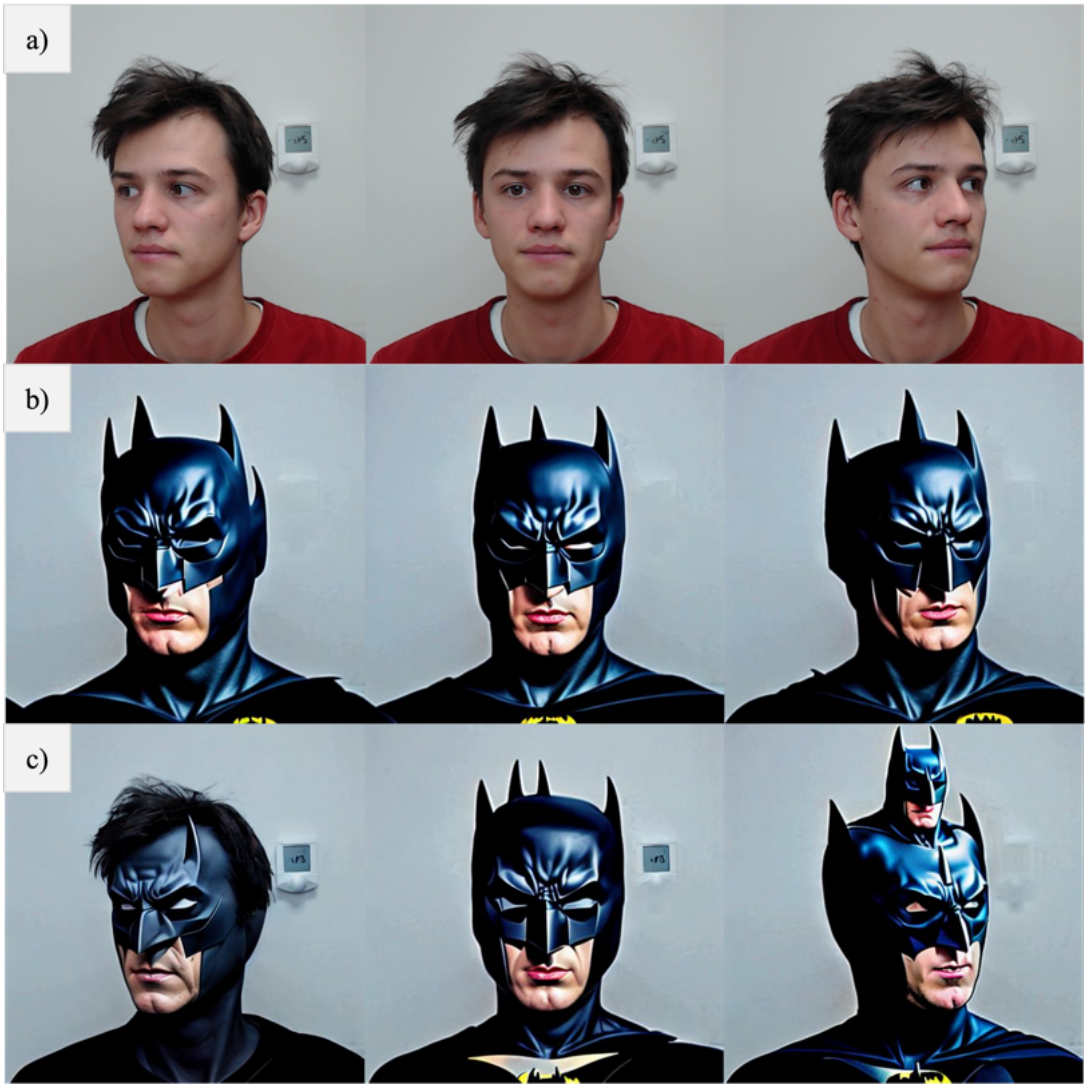


Figure 4.3: Comparison of stylized keyframes generated using IP2P. a) contains the input keyframes and b) and c) contain the respective keyframes stylized with the prompt *"make him look like batman"* with CFG image 13 and CFG text 2. Keyframes in b) were created all at once in one inference run and the keyframes in c) were generated one by one in 3 distinct inference runs.

Generating the images all at once has a downside; the inference has to be done on one GPU, and the images have to fit into the memory. In our setup we can fit up to 3 images at 448x448 pixels each. For stylizing a face in a video-conferencing application, 3 keyframes prove to be enough, with one keyframe facing the camera and the other two facing either side.

Another downside of multiple keyframes and one inference run is that it takes slightly longer to generate the images. It is about 1.6x faster to generate 3 images separately.

IP2P was selected over other generative models due to its ability to edit the input image as described by the text prompt while preserving identity and content not related to the edit. This method avoids complete identity loss compared to other techniques when using text prompts with targeted changes, such as *"Give him a moustache"*. IP2P does this by conditioning the original input image (along with the text prompt), rather than only providing an extracted feature map, such as an edge image, depth, or normal map, which is the strategy of ControlNet. This identity-preserving quality can be seen in Figure 4.4. On the other hand, when we want a complete transformation of the style of the input image, the guidance features of ControlNet can be enough and sometimes even better than IP2P.

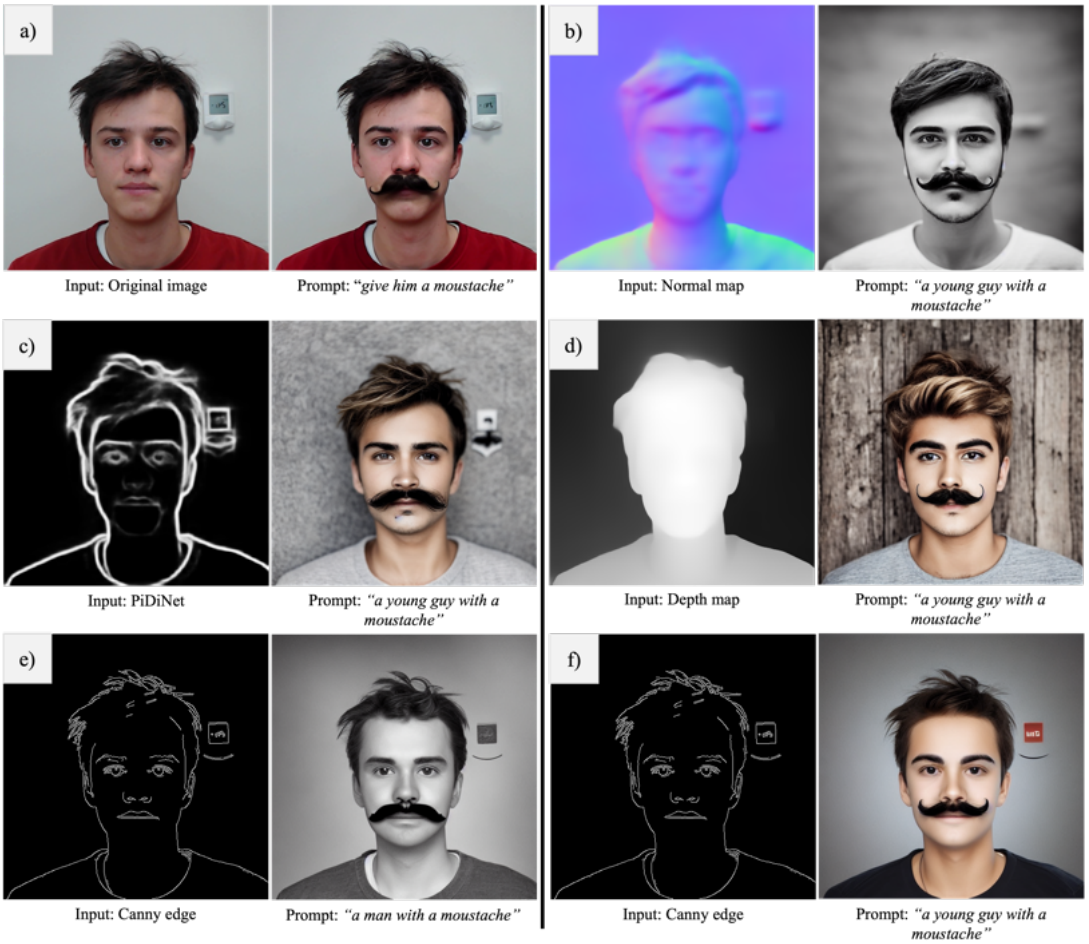


Figure 4.4: Comparison of images generated using IP2P and ControlNet with Stable Diffusion v1.5 as a base model. The image pairs a)-f) contain an input image that the model is conditioned on, shown on the left. On the right is the generated output paired with the text prompt used. Pair a) is generated by IP2P, while pairs b)-f) are outputs using ControlNet.

IP2P allows adjusting the guidance strength to configure how closely the output adheres to the text prompt versus preserving the original input image. This classifier-free guidance mechanism gives intuitive control ranging from fully preserving the input image to strongly steering the output based on the text prompt. If the stylization

does not match the user's expectations, it is easy to tune the guidance to either give more strength to the text or better preserve the identity of the input image. A sensible default usually gives a satisfactory result, but it really depends on the specific edit prompt.

Another small aspect to consider is the formulation of the prompt; in the case of IP2P, the prompt text is an edit instruction. The alternative, which most other methods use, is a text description of the output image. So instead of *"give him a moustache"* we would have to do something like *"a young man with a moustache"*. An example of these prompts can be seen in Figure 4.4. This formulation of what should change rather than what should be the output is very intuitive to humans.

4.3 Style Propagation

After generating the stylized keyframes, the next step is propagating the artistic style to the rest of the video sequence. This is done in an example-based manner using the StyleVid [44] neural network.

StyleVid aims to achieve the semantic faithfulness of patch-based methods such as Stylizing Video by Example [25], while enabling real-time performance and the random access of feedforward techniques. Importantly, StyleVid runs inference at more than 30 fps on a sufficiently fast GPU. It also trains the image stylization model in just a few seconds, allowing quick adaptation of new styles.

The authors identify two main sources of temporal flickering in their method:

1. temporal noise in the original video and
2. visual ambiguity of the stylized content.

While we cannot leverage the same pre-processing as the original method (since we need to stylize each frame as it is created to keep the application interactive), the implicit temporal consistency of StyleVid combined with additional post-processing shows to be sufficient.

To address temporal noise in the input video, we blend the current frame with the previously processed frame. This approach combines the information from the immediate frame with a weighted average of the past frames, effectively reducing temporal noise. Mathematically, this process is described as follows:

$$\texttt{blended\_frame}_{(t)} = \alpha \cdot \texttt{current\_frame}_{(t)} + (1-\alpha) \cdot \texttt{blended\_frame}_{(t-1)}$$

where α is the weight given to the `current_frame`, and $1-\alpha$ is the weight of the previously `blended_frame`. The contribution of each frame to the final image can then be described as an exponentially weighted moving average:

$$\texttt{blended\_frame}_{(t)} = \alpha \cdot \sum_{k=0}^{t} (1-\alpha)^k \cdot \texttt{current\_frame}_{(t-k)}.$$

A visualization of this contribution can be seen in Figure 4.5.

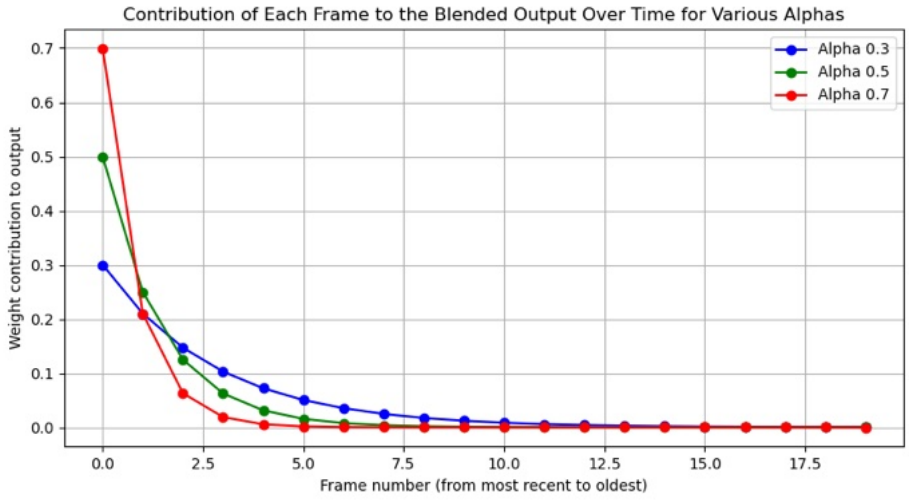


Figure 4.5: Weight contributions of frames in temporal blending: The graph displays the declining influence of each preceding frame on the blended output over time for multiple values of α.

To further fight temporal inconsistencies, due to visual ambiguity, we introduce foreground masking to focus the stylization on the main region of interest; the head and shoulders in a video conferencing setup. Compared to the background, the foreground does not contain as many visually ambiguous areas. Training only on the foreground significantly improves the convergence time, since we are learning to stylize a smaller region with less visual ambiguity. The foreground mask is obtained by segmenting the input keyframes using the Google MediaPipe [31] selfie segmenter[1], selected for its real-time performance. An example of the masks can be seen in Figure 4.6.

With the overall approach established, the next chapter will focus on implementation details of integrating IP2P and StyleVid into an interactive application.

[1] https://developers.google.com/mediapipe/solutions/vision/image_segmenter

Figure 4.6: Foreground masks created using the MediaPipe selfie segmenter. a) contains the input keyframes from which the training masks b) are created.

Chapter 5

Implementation

This section provides the technical details and practical realization of the real-time video stylization video-conferencing application created as part of this thesis. To meet real-time performance goals and enable user interactivity, we separate the workload across key asynchronous components: camera capture, stylization inference, video display, the client-interaction application, keyframe stylization, and style transfer model training. These components run on two separate machines; a client machine and a server machine.

First, we discuss the hardware and software technologies used, including the specifics of the camera, GPUs, operating system, and libraries used. We then explain the application design and discuss the workings and data-flow of individual components.

To achieve real-time stylization, it is crucial to run these components concurrently. Multithreading techniques and inter-process communication mechanisms that allow different parts of the pipeline to operate in parallel and on separate computers. We also detail the integration challenges faced and strategies to address them.

The final section of the text will focus on the user interface for entering artistic stylization prompts, settings to direct the process, and the display mechanisms to show the user a live stream of the stylized video output.

The next chapter Results and Experiments 6 demonstrates the real-time performance and qualitatively evaluates visual coherence through demos.

5.1 Hardware and Software Configuration

To split the work as much as possible, we use two machines–a client machine and a server machine–with separate hardware setups tailored to their distinct roles. The client handles communication, camera capture, and display of outputs and interfaces. The server runs diffusion model inference and trains the neural network.

The client machine exists to enable user interaction. It has the camera and displays the camera view and stylization along with all the interactive components. The client runs on a Windows 11 desktop equipped with an NVIDIA GeForce RTX 4080 GPU[1], an Intel Core i9-10900X CPU, and a Logitech HD Pro Webcam C920 camera[2].

[1] https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4080/

[2] https://www.logitech.com/en-us/products/webcams/c920s-pro-hd-webcam.960-001257.html

The server is there to handle most of the heavy lifting. It runs the InstructPix2Pix [4] diffusion model inference and trains the StyleVid [44] stylization network. The server uses an NVIDIA DGX platform[3] with multiple NVIDIA A100 GPUs.

The server or back-end is built on top of the official implementations of InstructPix2Pix [4] and StyleVid[44]. IP2P forms the foundation for keyframe stylization, the code is taken from the official open source repository[4]. StyleVid enables efficient style propagation, with code adapted from its official open source repository[5].

A Flask server [16] wraps the IP2P diffusion model to handle stylization requests through a RESTful API. Communication with the client uses SocketIO for real-time interactivity. Another separate Python program adapts the StyleVid implementation to perpetually train stylization models.

The client or front-end of the video conferencing application is built using Python, with PyQt (Python binding for the Qt application framework) for the graphical interface and OpenCV [3] for the capture and display of real-time video. It also uses the StyleVid implementation[6] to run stylization model inference.

SSHFS (Secure SHell FileSystem) is used to establish a connection between the client and the server, enabling the seamless and efficient exchange of files by mapping a directory on the server to the client.

5.2 Application Design

In a real-time interactive setting, we need to take special care to handle every user interaction without any visible delay and to have the stylization run smoothly without any timing inconsistencies. This can be practically achieved by separating work into parallel execution flows where possible. However, concurrency in Python faces some challenges. Since Python's memory management is not thread-safe (specifically, the CPython implementation), it uses a Global Interpreter Lock (GIL), which prevents multiple native threads from executing Python bytecode concurrently. Therefore, in a multithreaded Python program, only one thread can execute Python code at once. For I/O-bound operations, such as file and network operations, Python can still achieve concurrency as the GIL is released to allow other threads to run during these times. However, for CPU-bound computations, the GIL limits parallelism. Although multiprocessing avoids this issue, the lack of shared memory between processes makes this impractical for our needs.

Importantly, Python can bypass the GIL when executing tasks in external libraries, particularly those written in C/C++ or other languages. For instance, Graphical User Interface (GUI) tasks such as rendering the UI in PyQt, many NumPy operations, or operations on data using libraries like TensorFlow or PyTorch can run in parallel on multiple cores or on a GPU because they are not executed directly by the CPython interpreter. We take advantage of this capability to incorporate concurrency by wrapping key parallelizable components in separate threads when feasible.

[3] https://www.nvidia.com/en-us/data-center/dgx-platform/
[4] https://github.com/timothybrooks/instruct-pix2pix
[5] https://github.com/OndrejTexler/Few-Shot-Patch-Based-Training
[6] https://github.com/OndrejTexler/Few-Shot-Patch-Based-Training

To make the stylization faster, we also distribute functions across client and server machines, interacting via file sharing and a REST API. The client built with PyQt consists of camera capture, stylized video display, stylization inference, and a GUI to enter text prompts and interact with various video and other settings. The server handles keyframe generation using InstructPix2Pix and perpetual training of the style transfer network as new stylized keyframes are created. A schema of the entire application design can be seen in Figure 5.1.

During the development of this project, SSH was used for communication between the client and the server machines. To enable a flexible and efficient workflow, the client was connected to the server using SSHFS by mapping the development directory on the server to a local directory on the client. This way, the files in the shared directory could be accessed on the client machine, even though they physically reside on the server. Since the drawbacks proved insignificant, this setup was kept for seamless file sharing between machines.

5.2.1 Client Components

Since access to the camera can be given to one process at a time, we have two options; Fit all of the client components (such as the camera capture, stylized video display, stylization inference, and the GUI) into one Python program or separate this work further into two or more programs. At first, I separated the GUI from the rest of the client application. Although this solution worked well, it introduced new unnecessary complexity in communication and was not very user-friendly. To have a cleaner and a more future-proof product, I decided to integrate the solution into one.

The following text describes the components of this client application and how they come together as a whole.

Camera Capture

Capturing camera frames is practically similar to reading from the hard drive in that it releases the GIL and is thus non-blocking. It can be easily partitioned into a separate thread. The camera capture module retrieves frames from the webcam at its maximum sampling speed–30 fps–and at a resolution of 800x600, which is then cropped to a square. These frames are saved in a double-ended queue for the rest of the application to access. In addition to providing all camera needs for the application, this module can optionally blend incoming frames to reduce temporal noise.

Stylization Inference

To stylize the frames captured by the camera module and hold the GIL as little as possible, we use three complementary threads. First, to keep the stylization delay as short as possible, we have a thread periodically checking for the newest model. When a new model is found, its path is passed to another thread, which then loads this model into the GPU. The third and final thread then performs the actual inference on the GPU by passing the input frame through the trained StyleVid [44] model.

The first two threads access a different external non-blocking resource and thus can be separated out. The third thread, similarly to the second, accesses the GPU, but

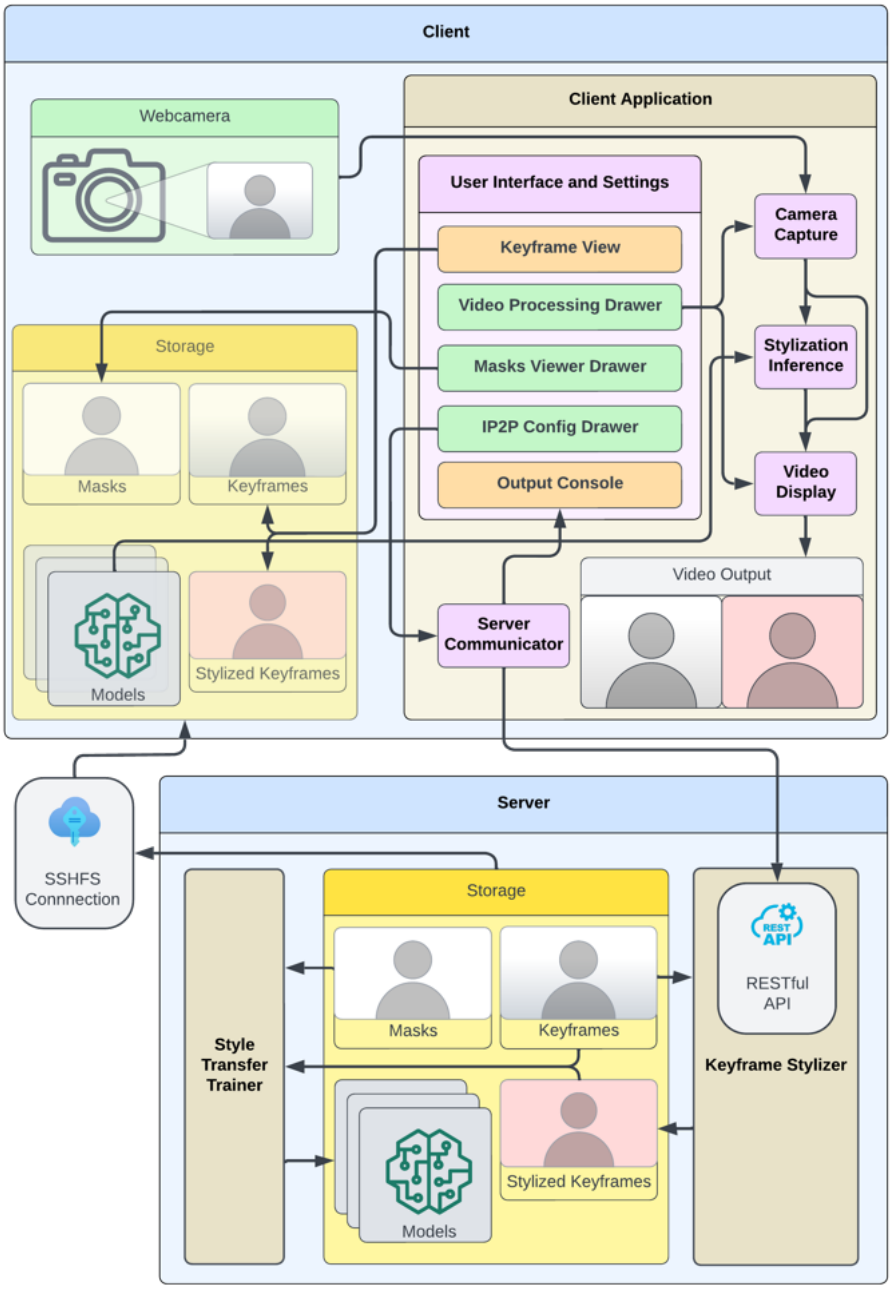


Figure 5.1: Diagram of the application design. Shows the inner workings and connection of two machines a client and a server (in blue). Separately running programs are depicted in brown, yellow signifies storage and the distinct client components are shown in pink. The arrows show the flow of information between the machines, components and other resources. The storage on the client is shown as opaque to indicate that the server is the actual location of the data.

since we need the stylization to run without any noticeable hiccups, it is useful to run separately.

Video Display

An important component is displaying the camera frames along with the stylized frames. The concurrency of this component is handled by PyQt itself, since updating a frame in an open window is done outside of CPython itself.

To help smooth the stream and avoid possible stuttering, the video display also enforces a cap on the frame update frequency. If a cycle of the display loop takes less than $\frac{1}{32}$th of a second, the thread waits until this time has elapsed. This, sleep time gives other threads space to run and helps prevent minor timing inconsistencies among the threads, in decreasing in the maximum achievable 30 frames per second (fps).

This component can also optionally blend the output frames to reduce noise caused by stylization ambiguity. Another functionality is to replace the background of the stylized frames with the background of the input frames to eliminate disturbing color changes sometimes caused by updating models stylization models.

Server Communicator

To direct keyframe stylization on the server machine, the client application employs two communication mechanisms.

Real-time messaging leverages SocketIO to enable bidirectional status updates and notifications between the client and the server. This allows dynamic feedback about actions performed, errors encountered, and style transfer progress.

Sending stylization instructions relies on a REST API exposed by the Flask server that wraps the InstructPix2Pix model. The API exposes an endpoint that accepts HTTP POST requests containing the configuration parameters for the text prompt, diffusion settings, image guidance strength, etc. The server queues these requests and handles stylization one by one, loading the specified input frames, generating the artistic keyframe transformations, and saving the output.

User Interface and Settings

The GUI is built using PyQt and provides an intuitive way for users to interact with the video stylization application. A screenshot can be seen in Figure 5.2. The app contains several key components:

Keyframe View Section. Provides a side-by-side view of the input and output folders showing the input keyframes and the corresponding stylized output keyframes. The images can be opened for closer inspection, deleted individually or all at once using the *Clear* button. The folder view updates automatically based on directory changes, but can be manually refreshed with the *Reload* button. This allows reviewing and clearing out the generated stylized frames.

Video Processing Drawer. Includes options to control real-time video stylization video display:

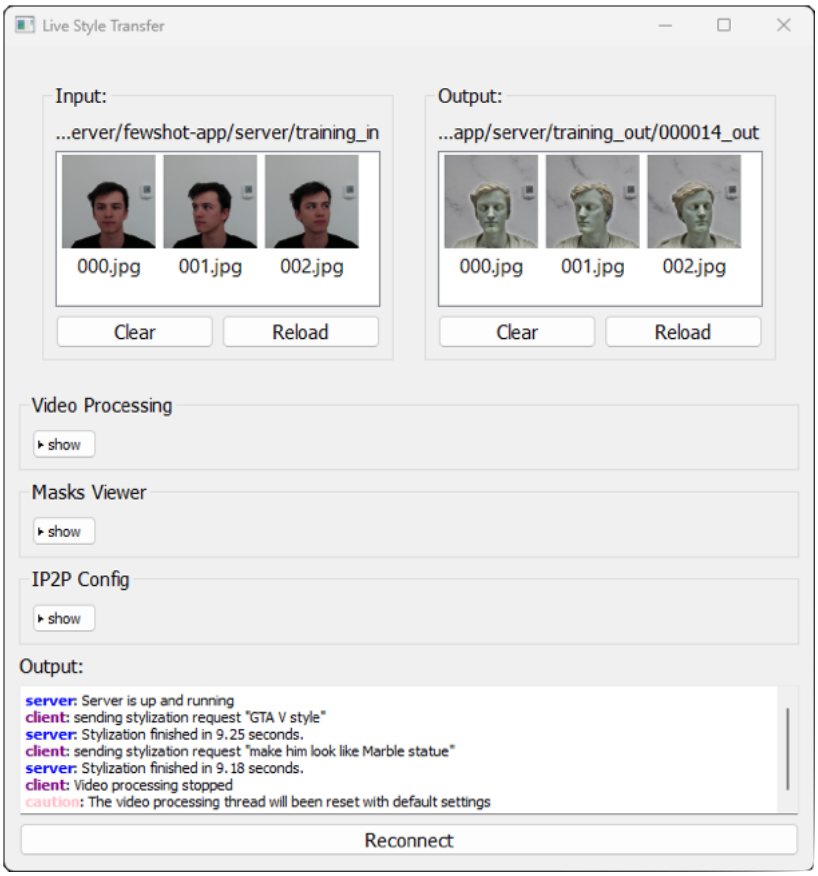


Figure 5.2: A screenshot of the entire GUI app with collapsed drawers. Shows the Keyframe View Section and the Output Console.

- *Save Frame*: Save a new frame to the input folder.
- *Keyframe Options*: Switch between *Multiple Keyframes* and *One Keyframe* options. The choice determines whether a newly saved frame will be added or will replace the previous frames.
- *Blend In Settings*: Options to blend input video frames with an adjustable *alpha* value.
- *Blend Out Settings*: Options to blend output video frames with an adjustable *alpha* value.
- *Output Mask*: Toggle ON/OFF the background replacement of the stylized frames by the backgrounds of the input frames.
- *Composer View*: Select the layout of input/output views of the video display window. Has three options *Side by Side*, *Input Only* and *Output Only*.
- *Input Source*: Choose between a live camera or a prerecorded sequence.
- *Start Video Processing*: Open the video display window to see your live stylization.

A screenshot of this component can be seen in Figure 5.3.

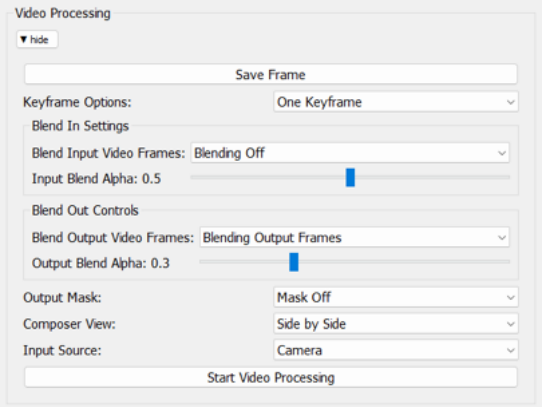


Figure 5.3: A screenshot of the expanded Video Processing Drawer.

Masks Viewer Drawer. Displays input keyframe masks and provides options to generate masks manually or automatically when sending a new stylization instruction. Masks control which parts of the image are used to train the stylization. Contains an option to switch to an "*All White mask*" to effectively train on the entire image.

The viewer otherwise functions similarly to the input and output image viewers. A screenshot of this component can be seen in Figure 5.4.

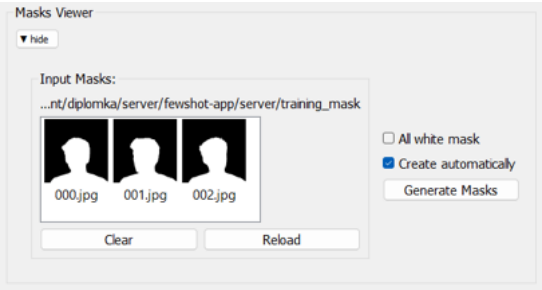


Figure 5.4: A screenshot of the expanded Mask Viewer Drawer.

IP2P Config Drawer. Provides configuration settings and controls for the IP2P keyframe stylization:

- Input fields to set parameters like seed, CUDA device, image resolution and the number of diffusion steps.
- Sliders to adjust textual and image-based guidance strengths.
- Text field to enter the edit prompt.
- Save or load the stylization configurations to and from a YAML file with the *Load* and *Save* buttons.
- *Go* button to send the stylization command to the server with the current settings.

A screenshot of this component can be seen in Figure 5.5.

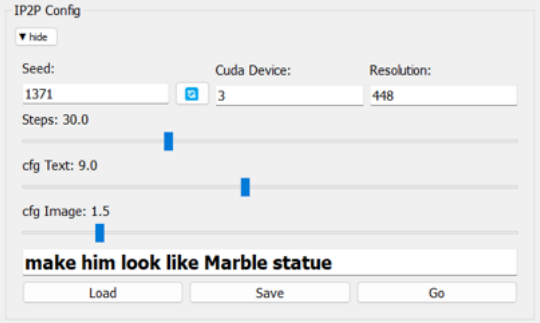


Figure 5.5: A screenshot of the expanded IP2P Config Drawer.

Output Console. Logs messages from the server and the client, indicating status, actions taken, and errors. Also contains a *Reconnect* button to check or update the server connection.

5.2.2 Server Components

The server handles the heavy-weight tasks of keyframe stylization using IP2P [4] and perpetual training of the neural style transfer models through StyleVid [44]. It runs on a system equipped with multiple GPUs to enable parallel execution. The server workload is handled by two separate Python programs:

Keyframe Stylizer

Integrates the official implementation of InstructPix2Pix[7] [4] into a RESTful API using the Flask framework. This module stylizes keyframes based on configuration instructions received from the client. At launch, it loads the ~34 GB IP2P model into the GPU (20 seconds). The API then waits for incoming HTTP POST requests containing parameters such as the number of diffusion steps, seed, textual/visual guidance weights, edit prompt, CUDA device, and input/output paths.

Requests are queued and handled one by one. For multiple inputs, keyframes are concatenated, stylized jointly, and then split back up. The stylized output keyframes are saved to the designated path, which is accessible to the Style Transfer Trainer.

Style Transfer Trainer

Adapts the official implementation of StyleVid[8] [44] to perpetually train style transfer models on new stylized keyframes and periodically saves these updated models.

A single dedicated thread handles the entire model training, checking for new stylized keyframes every 100 batches. If available, these are loaded and swapped with the previous training data (taking ~130 ms). Whenever there are new stylized keyframes available, we also check for new input keyframes and update those as well. Every 150 batches, the current model is saved to disk (~80 ms). Saved models are accessible to the client via SSHFS.

[7] https://github.com/timothybrooks/instruct-pix2pix
[8] https://github.com/OndrejTexler/Few-Shot-Patch-Based-Training

Chapter 6

Results and Experiments

In this chapter, we present the results and experiments conducted to evaluate our real-time neural style transfer method. We begin with a quantitative analysis, focusing on key performance metrics such as frame rate and latency. This is followed by qualitative coherence analysis, focusing on several key aspects of the style transfer, including the effect of additional keyframes, style convergence speed, and the role of masking in enhancing stylization. Then, in Section 6.3, we present and discuss the stylized results, showing the achievable style diversity. We also include observations from public demonstrations of our application, offering insight into user interactions.

6.1 Performance Analysis

To quantitatively evaluate the real-time and interactive capabilities of the system, three metrics were analyzed: frame rate, latency, and style delay.

Frame rate indicates the smoothness of the output video stream. Processing a single frame captured at 800x600 and stylized at 448x448 takes approximately 25 milliseconds. Given the 30 fps limit of our webcam, the frames are being stylized faster than they are captured. Meaning that we reach the 30 fps stylization output. To further test the limits of the system, we artificially increase the input framerate and achieve stylization at about 39 fps.

End-to-end latency combines the camera capture delay and time needed for stylization and frame display. With the typical webcam lag of 100-150 ms, and image processing adding roughly 30 ms, the total lag stays under 200 ms - an imperceptible level of delay during live streaming. When running at 30 fps, the stylized video is about one frame behind the input video.

Style delay measures the waiting time from entering a text prompt to initial visible changes in the stylized output stream. Using the typical 3 keyframes, this delay can be up to 13.4 seconds. With only 1 keyframe, it drops to 6.2 seconds. These totals come from several steps:

Generating 3 keyframes with InstructPix2Pix requires approximately 9.2 seconds at 30 diffusion steps. The keyframe outputs are then loaded into the style transfer trainer (~130 ms), which checks for new images every 20 batches or ~560 ms seconds before starting to train on the fresh data. The trainer then releases a new model every 100 batches or ~2.8 seconds. On the client-side, a check for the latest model occurs every 0.5 seconds. Once found, the client loads this new model in ~270 ms to start

using it for inference.

Summing these pipeline timings accounts for the total style delay from prompt to early observed stylization effects. However, the style transfer continues to improve with the release of new models until training has fully converged. The convergence time strongly depends on the difficulty of the style.

6.2 Style Coherence Analysis

To assess the coherence of the style, we examine two key aspects related to the quality of real-time video stylization. The first is stylization coverage; the ability of the propagated stylization to fully takes hold across all video frames given factors like the number of keyframes or the complexity of the desired transformation or sequence. The second is the convergence speed, the rate at which the stylization improves over continual training iterations, influenced by elements such as background masking and style difficulty.

Evaluating these elements of quality and speed provides important practical insights. The effectiveness across different style settings validates the flexibility of the approach. By observing the convergence, we gain insight into the expected delay for observing a completed stylization.

6.2.1 Style Coverage

Style coverage is largely determined by two factors, the complexity of a given style and how well the input video sequence is covered by keyframes.

The complexity of a style can be seen by comparing the input and the stylized keyframes. It is largely dictated by how the style changes the structure of the keyframe. Simple styles can change colors or slightly decrease low-level structure, such as smoothing skin, etc. Complex styles change the high-level structure, such as making someone fat (or otherwise changing the shape of the face) or adding low-level structure to regions that lack it, such as giving the frames a painterly appearance (Figure 6.10).

Simple Styles. Simple styles typically generalize well, so a single frontal keyframe can be sufficient to capture the desired change. This can be seen in Figure 6.1, where the style is well generalized from one keyframe. Additional keyframes can still help improve the style, but can also cause problems if they aren't coherent with each other. Another factor to consider is that more keyframes can also increase the chance of introducing common diffusion artifacts, such as malformed eyes. Figure 6.2 shows the negligible improvement from using multiple keyframes on an easy target style.

Complex Styles. Complex styles typically need multiple keyframes to generalize to the whole output sequence. This can be seen in Figure 6.3 where the model struggles to stylize the frames convincingly. Figure 6.4 shows how multiple keyframes that capture the subjects' turned head help stylize the heads' motion.

The style transfer can still struggle if the input sequence contains more complex head movements or faces not captured by the keyframes. This can be seen in Figure 6.5.

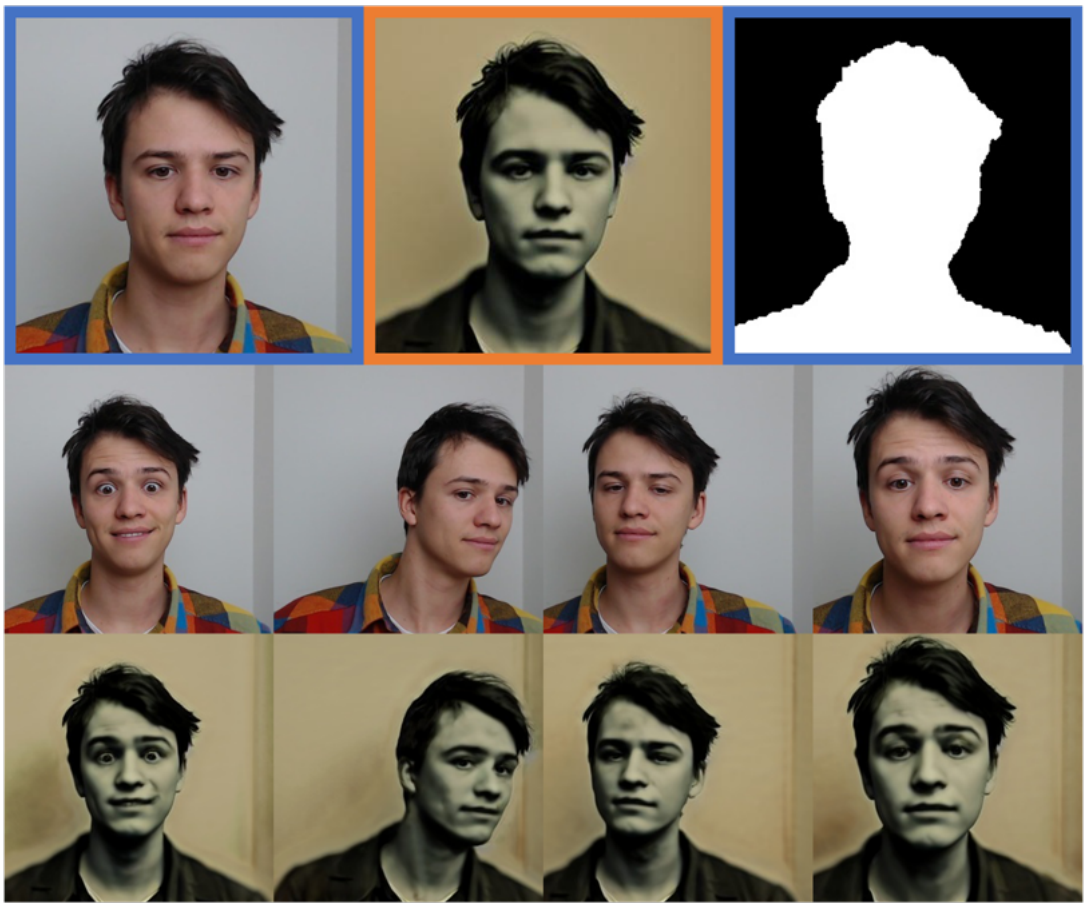


Figure 6.1: Simple style stylization using **one keyframe**: input keyframe and mask (blue border), stylized keyframe (orange border), input and stylized frames (bottom two rows). Used parameters; `prompt: "as a daguerreotype", cfg_text: 8, cfg_image: 2`.

6.2.2 Convergence Speed

Style convergence is mainly influenced by three factors; style complexity, number of keyframes, and foreground masking.

Style complexity. Simple styles converge considerably faster than complex styles. For simple style, the stylization result is easily recognizable and pleasing even on the first loaded model. On the other hand, complex styles take several, sometimes even tens of seconds, to become recognizable as the desired style and take even longer to fully converge. Figure 6.6 shows this effect starting with an easy style–saturating the colors and adding a fine structure, continuing with a harder style–slightly changing the geometry of the face and adding hard lines, and finally displaying a hard style–changing the geometry of the face and adding a lot of small sharp details.

Multiple Keyframes. Employing multiple keyframes generally improves the model's ability to handle complex styles and significant head movements. However, this benefit comes at the cost of increased convergence time, particularly for complex

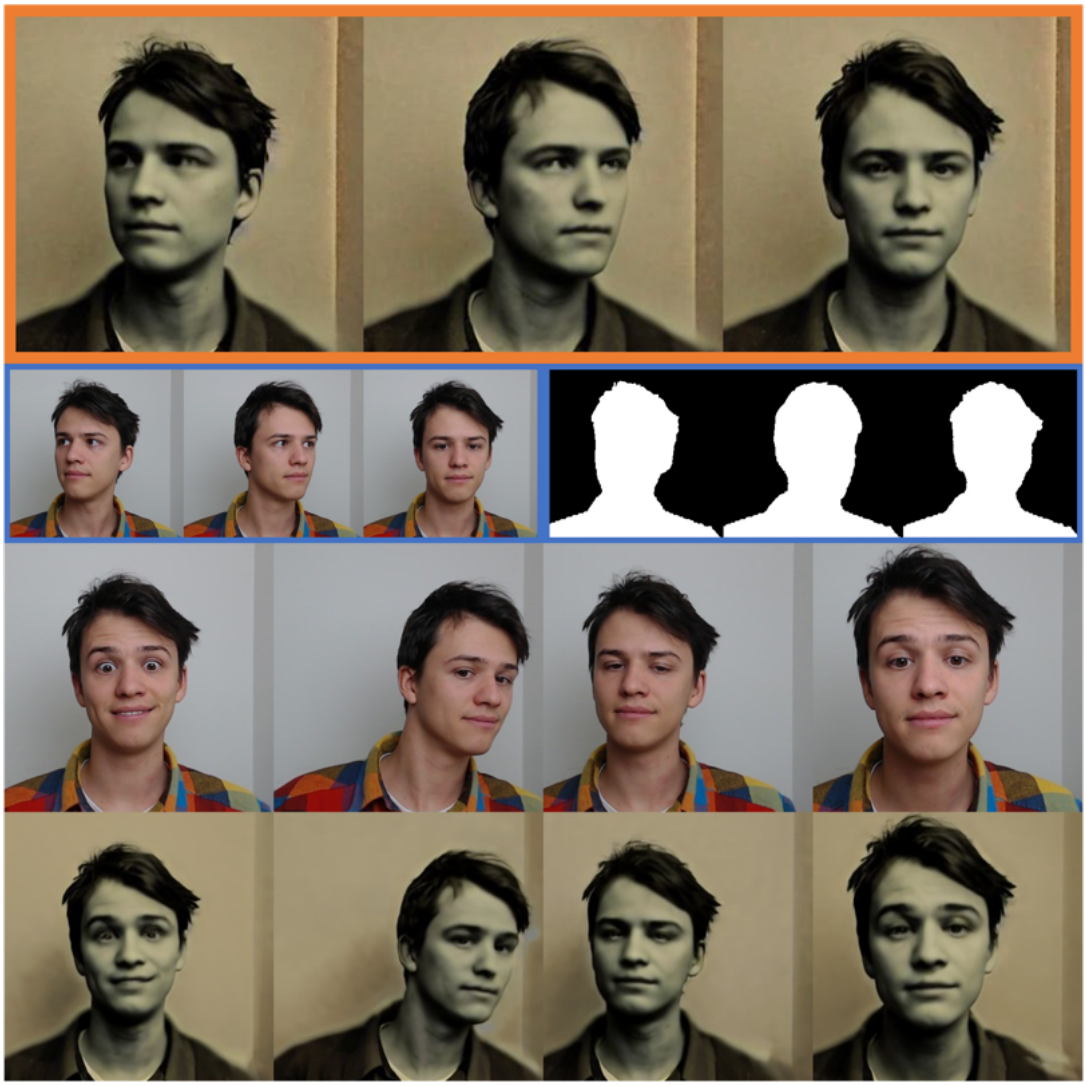


Figure 6.2: Simple style stylization using **multiple keyframes**: input keyframes and masks (blue border), stylized keyframes (orange border), input and stylized frames (bottom two rows). Used parameters; `prompt: "as a daguerreotype", cfg_text: 8, cfg_image: 2`.

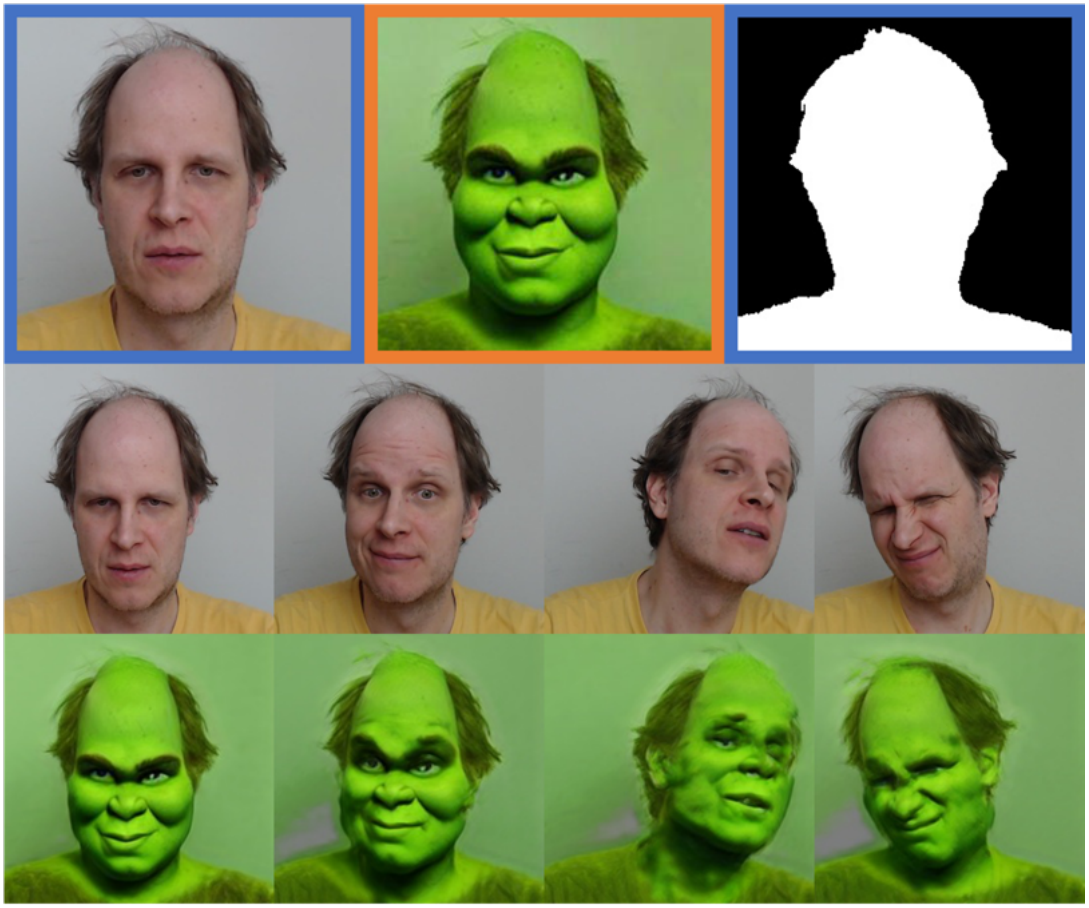


Figure 6.3: Hard style stylization using **one keyframe**: input keyframe and mask (blue border), stylized keyframe (orange border), input and stylized frames (bottom two rows). Used parameters; `prompt: "turn him into Shrek", cfg_text: 7.5, cfg_image: 2`.

styles. Figures 6.7, 6.8, and 6.9 demonstrate the convergence times for the same simple, medium, and hard styles as prior single-keyframe examples (Figure 6.6), now trained on multiple keyframes. The simple style converged as fast as before, while medium and hard styles take almost twice the time to converge. (Note that the CFG parameters were adjusted to better match the stylized keyframes of the previous figure, in the case of the hard–yeti–example a closer match was not achieved.)

Foreground Masking. Applying a foreground mask during training focuses the style transfer network exclusively on the key facial and torso regions of the subject rather than the entire frame. We are primarily interested in coherently stylizing the person themselves, and not the ambivalent background. Constraining the area speeds convergence for two main reasons:

First, reducing the output resolution lessens the risk of overfitting to noise or nonexistent structures in background regions of stylized keyframes. Without masking, the network tries to reconstruct fictional details in areas like walls that lack clear correspondences between the input and target frames.

Figure 6.4: Hard style stylization using **multiple keyframes**: input keyframes and masks (blue border), stylized keyframes (orange border), input and stylized frames (bottom two rows). Used parameters; `prompt: "turn him into Shrek", cfg_text: 7.5, cfg_image: 2`.

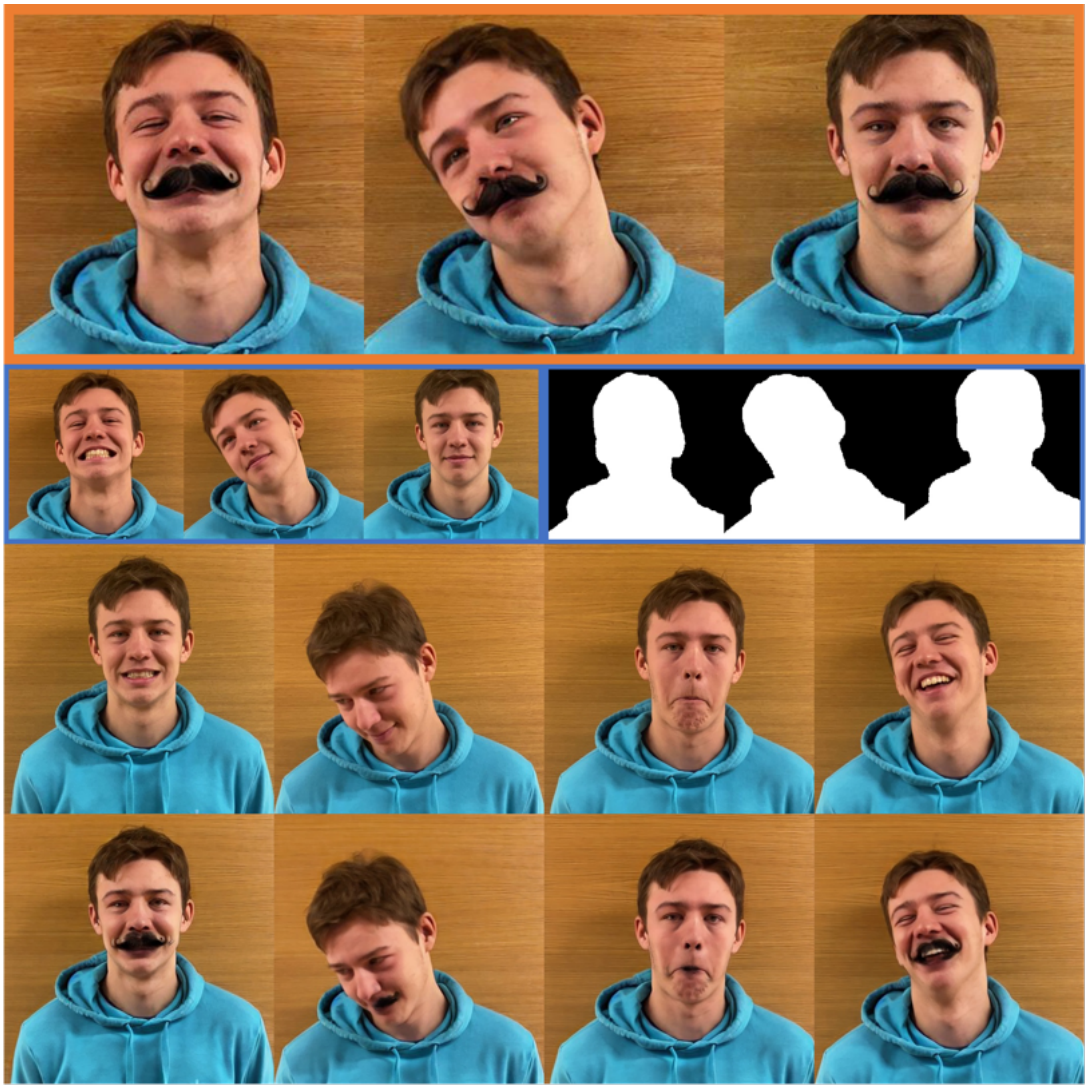


Figure 6.5: Hard sequence stylization using **multiple keyframes**: input keyframes and masks (blue border), stylized keyframes (orange border), input and stylized frames (bottom two rows). Used parameters; `prompt: "give him a moustache", cfg_text: 8, cfg_image: 2`.

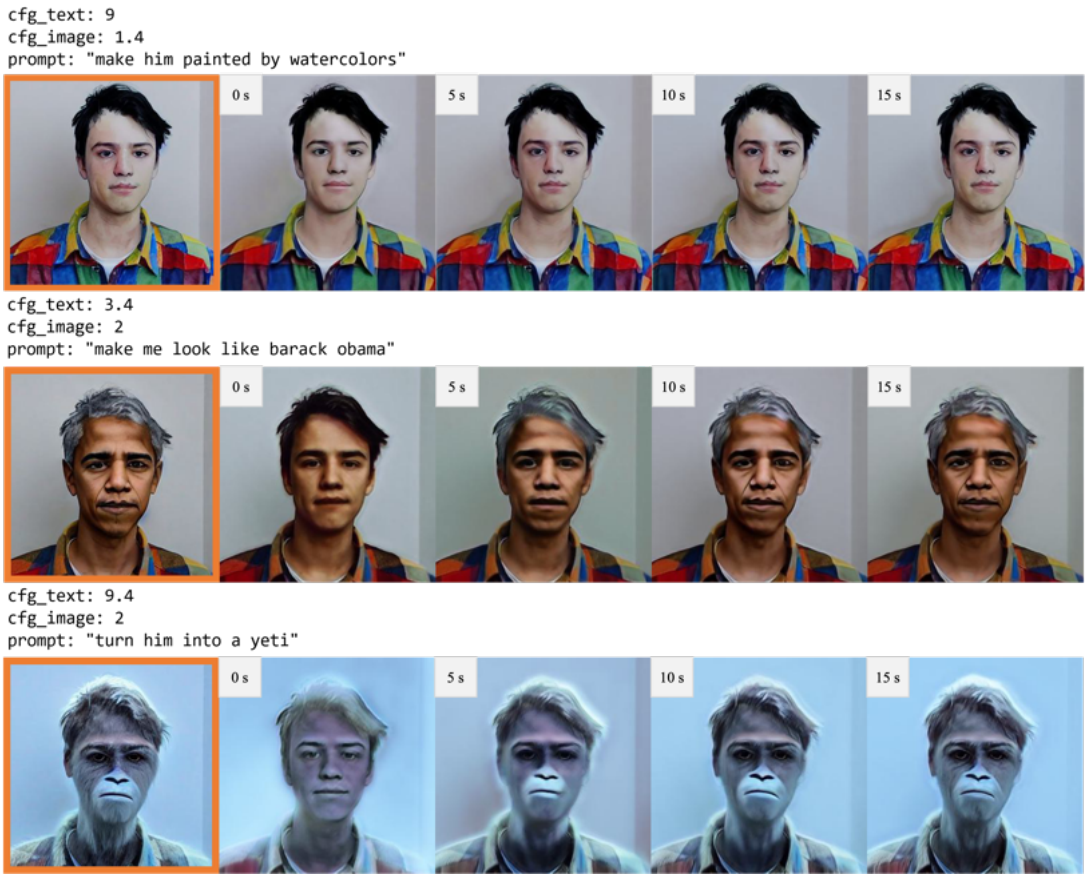


Figure 6.6: Convergence times of three styles; an simple style (top), medium style (middle), hard style (bottom). Each row contains the style parameters on top and the stylized keyframe (orange border) and stylized frames taken every 5 s starting at 0 s with the first model.

Second, even with valid background details, narrowing down the target area of the style transfer effectively reduces the number of pixels the network needs to process, leading to an easier learning task. The network can more rapidly latch onto and reproduce consistent textures and patterns.

Figures 6.10 and 6.11 illustrate the impact of foreground masking on convergence speed, comparing the results with and without mask use. In these figures, the target style presents a particularly challenging scenario for transfer without masking, as the stylized keyframe background contains brush textures, whereas the corresponding input area is a plain white wall.

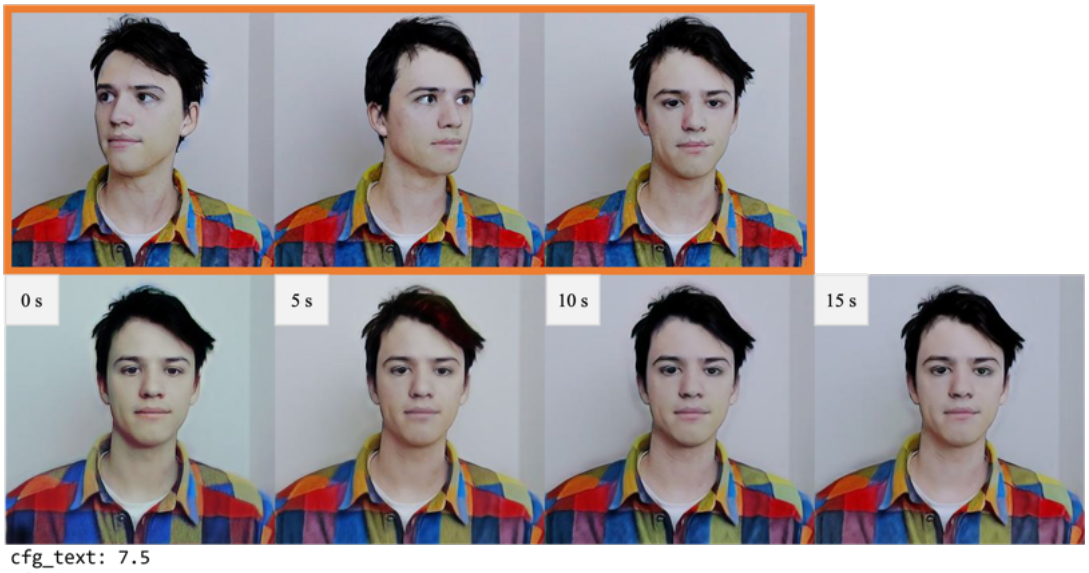


Figure 6.7: Convergence times of a simple style trained on multiple keyframes; stylized keyframes (orange border, top), stylized frames taken every 5 s starting at 0 s with the first model (bottom).


```
cfg_text: 3.4
cfg_image: 2
prompt: "make him look like barack obama"
```

Figure 6.8: Convergence times of a medium style trained on multiple keyframes; stylized keyframes (orange border, top), stylized frames taken every 5 s starting at 0 s with the first model (bottom).

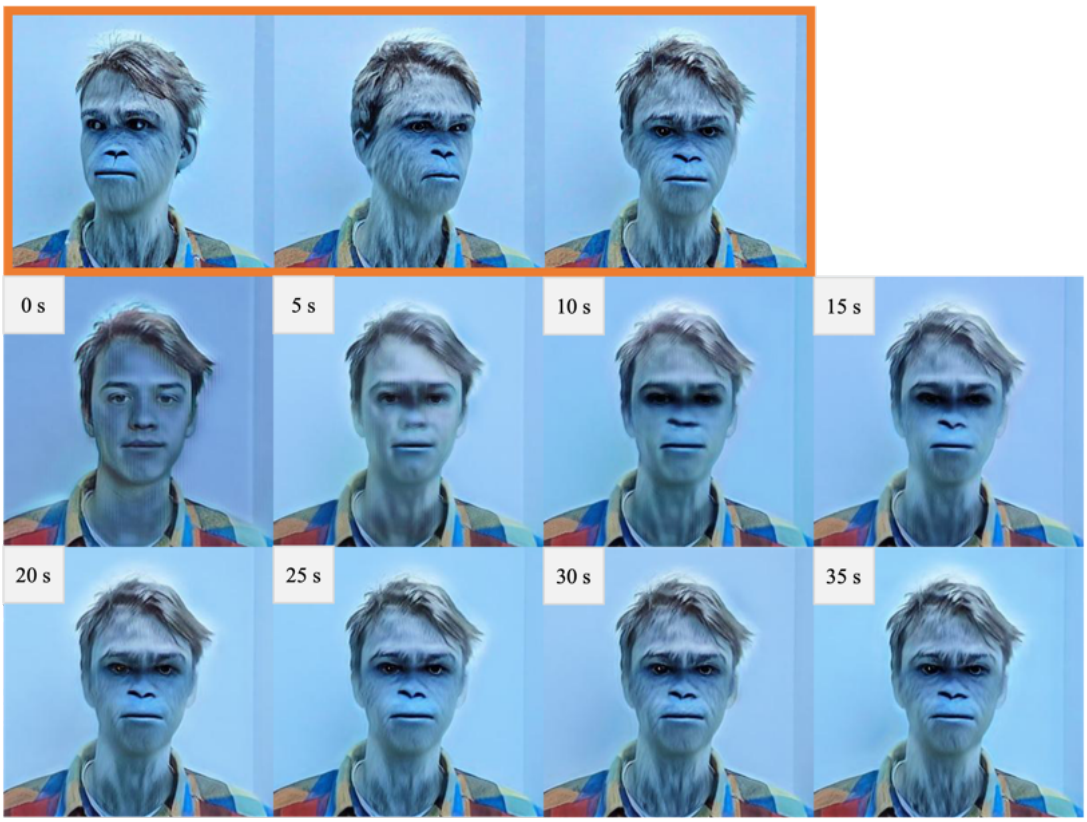


Figure 6.9: Convergence times of a hard style trained on multiple keyframes; stylized keyframes (orange border, top), stylized frames taken every 5 s starting at 0 s with the first model (bottom).

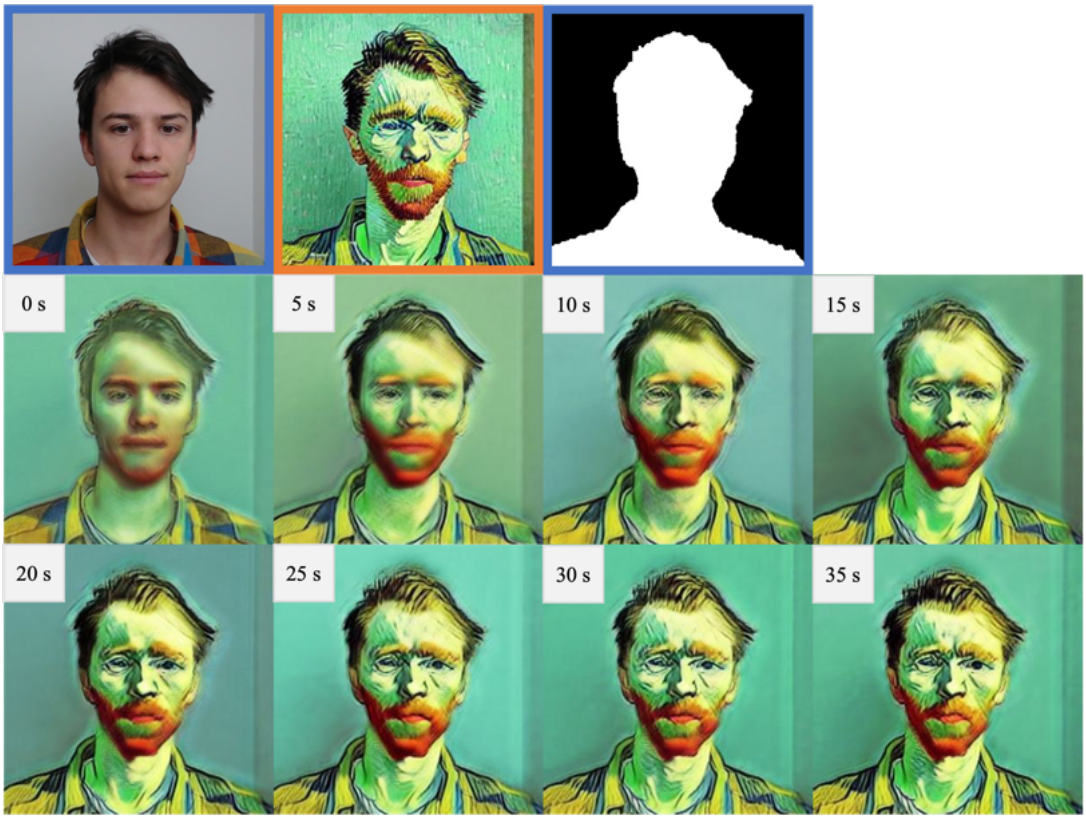


Figure 6.10: Convergence times of a hard style using a mask; input keyframe (blue border, top) stylized keyframes (orange border, top), training mask (blue border, top), stylized frames taken every 5 s starting at 0 s with the first model (bottom 2 rows).

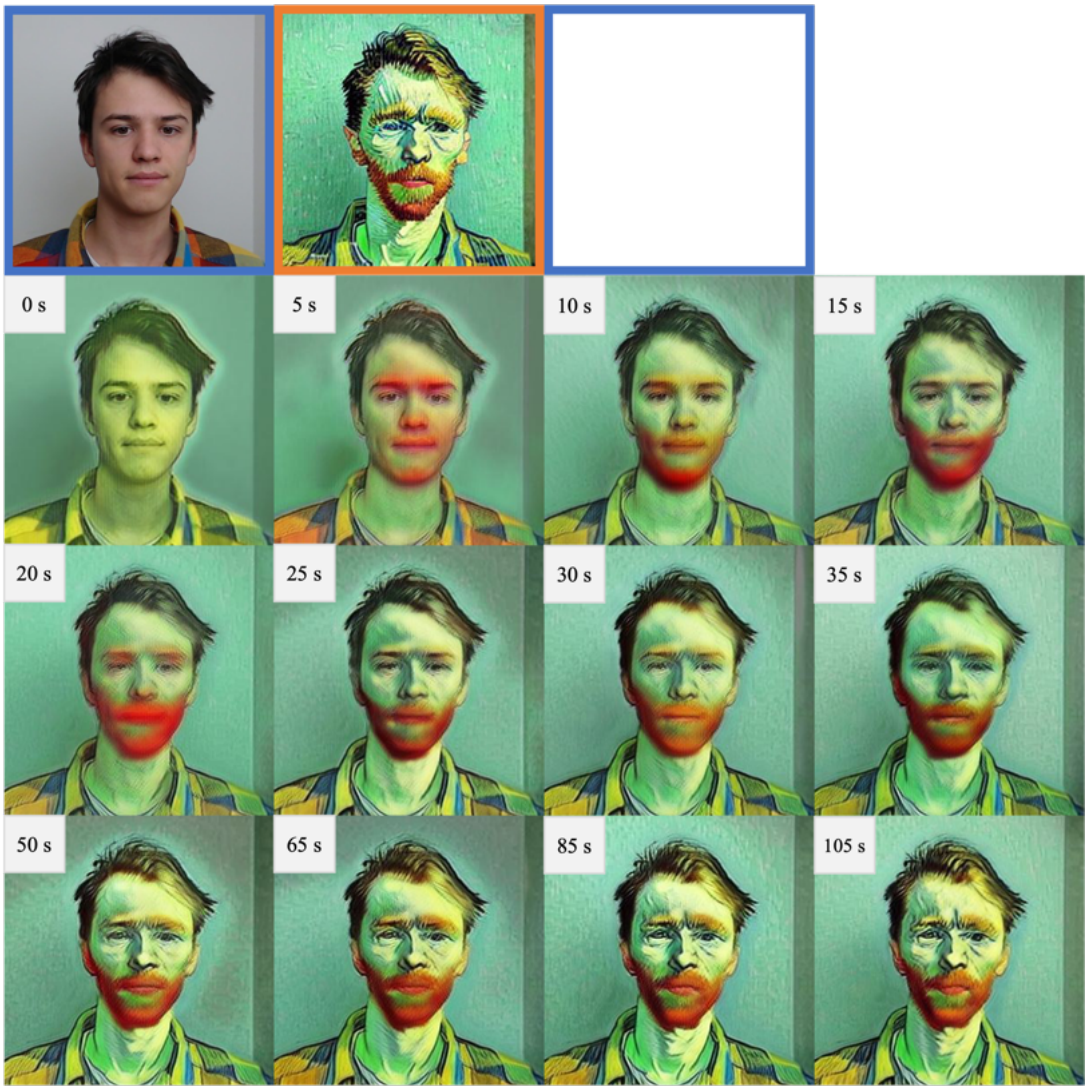


Figure 6.11: Convergence times of a hard style without using a mask; input keyframe (blue border, top) stylized keyframes (orange border, top), training mask (blue border, top), stylized frames starting at 0 s with the first model (bottom 3 rows).

6.3 Results

Five videos featuring five different subjects were used to evaluate the real-time stylization system. These videos show typical movements seen in video calls, with subjects primarily facing forward or sideways and engaging in head tilts. To further test the system's robustness, two videos also showcase more dynamic actions, including exaggerated head rotations and facial expressions with hands covering the subjects' faces.

The videos were filmed on 3 different cameras; Logitech C920 webcam, and iPhone 11 and iPhone 15 cameras.

Multiple different styles were applied to each video to evaluate the stylization coherence. Styles include simple color and texture changes as well as more complex shape distortions. For the purposes of a fair evaluation, we made an effort not to cherry-pick only great looking styles. However, we do present only the styles where the stylized keyframes well overlayed the input keyframes, since without this condition our method cannot work properly. Even so, some of the stylized keyframes better match the face geometry, are thus easier to learn, and provide a more convincing style transfer.

All styles were generated using 3 keyframes, with 30 diffusion steps, the other relevant parameters CFG text and CFG image were chosen to best fit each specific prompt and are stated along with the exact prompt wording next to each style.

The criteria for selecting input keyframes were to capture angles with expected visibility given typical video-conferencing motions facing the camera. For most subjects, we used keyframes; with forward-, left-, and right-facing poses. For subjects making faces in front of the camera, we replace a side-facing keyframe with a facial expression keyframe.

All styles were trained using only the input keyframe foreground. Styles were trained until perceived convergence. Depending on the keyframe match, convergence times ranged from tens of seconds to a minute.

The following is a description of results for all subjects.

Subject 1. Figures 6.12 and 6.13 present results for Subject 1. The video contains challenging frames with hands occluding the face and casting slight shadows. Besides exaggerated head tilts, most of the footage features natural head movements facing forward.

The three keyframes include; hands covering the face, a slightly tilted frontal perspective, and a smiling forward shot. The stylized versions of these keyframes remain visually consistent with each other.

All three styles applied to this subject's video adapt well to the varying poses. However, there are specific challenges: The extreme head tilt poses a difficulty for the *"give him sunglasses"* style, resulting in slight misstylization. Additionally, the *"digital art style"* encounters artifacts due to shadows created by the hands in front of the face, though the rest of the sequence maintains a high-quality stylization.

Subject 2. Figures 6.14 and 6.15 present results for Subject 2. The footage includes challenging frames with exaggerated head tilts, interspersed with minor motions, and

natural expressions of smiling, laughing, and making faces.

The selected keyframes capture a head tilt, a straightforward frontal shot, and a frontal shot with teeth showing, as an example of making a face. The stylized versions remain reasonably consistent in preserving high-level facial structure, with only small deviations in the more distorting *"as a pig"* style and low-level inconsistencies in the *"make him a zombie"* style.

The applied styles adapt well to simple head positions, creating a coherent stylization throughout most of the sequence. However, they encounter difficulties with the exaggerated head tilts, where details are sometimes lost or the intended effect is not fully achieved. Facial expressions are mostly well stylized, but the *"as pig"* and *"make him into a zombie"* styles show a loss of detail, particularly in complex facial movements.

Subject 3. Figures 6.16 and 6.17 show the stylization results for Subject 3. The video predominantly features simple, front-facing frames with various facial expressions. The sequence was recorded outdoors against the backdrop of a lush garden.

The three keyframes capture the forward view and slight head rotations and effectively represent the range of movements and expressions in the video. Their stylized versions remain visually consistent.

All three styles applied to this subject's video adapt well to the test frames, maintaining a coherent stylization. However, in addition to stylizing in the intended way, the style *"make him look like a clown"* colors the sky orange. This undesirable effect is caused by training only on the foreground, resulting in non-sensical or disruptive background stylization not present in the stylized keyframes. To address this, we can replace the background of the stylized frames with the background of the input frames, effectively correcting these anomalies. Figures illustrate stylization both before and after this background replacement, demonstrating the effectiveness of this technique.

Subject 4. Figures 6.18 and 6.19 present the results for Subject 4. This subject's video features a variety of head tilts, turns, and facial expressions captured indoors. The motion remains limited in range with no extreme rotated perspectives.

The chosen keyframes include a frontal view and one for each side of the face. These keyframes capture the range of head movements and expressions exhibited in the video. Although the stylized keyframes for the *"give him a carnival mask"* style appear consistent at first glance, a closer inspection reveals misalignments. Changes in the mouth and eyes do not align with the input keyframes, and the mask's position varies slightly around the nose and forehead between the stylized keyframes. Other stylized keyframes are consistent.

These inconsistencies in the stylized keyframes of the *"give him a carnival mask"* style result in a broken and non-generalizing stylization. The style effectively transfers only to test frames nearly identical to the initial input keyframes. The other styles applied to this subject's video perform quite well, demonstrating better coherence and adaptability to the various head movements and expressions.

Subject 5. Figure 6.20 shows the results for Subject 5. The video features very minor head movements.

The chosen keyframes include a frontal view and one for each side of the face, effectively covering the range of limited motion presented in the video.

The flat *"make him into a ghibli studio character"* style, applied to this subject's video, performs well for the non-moving parts of the subject's face. However, it faces challenges in more expressive areas such as the eyes or mouth, especially when the test frame deviates significantly from the keyframes.

More stylization results can be found in Appendix B.

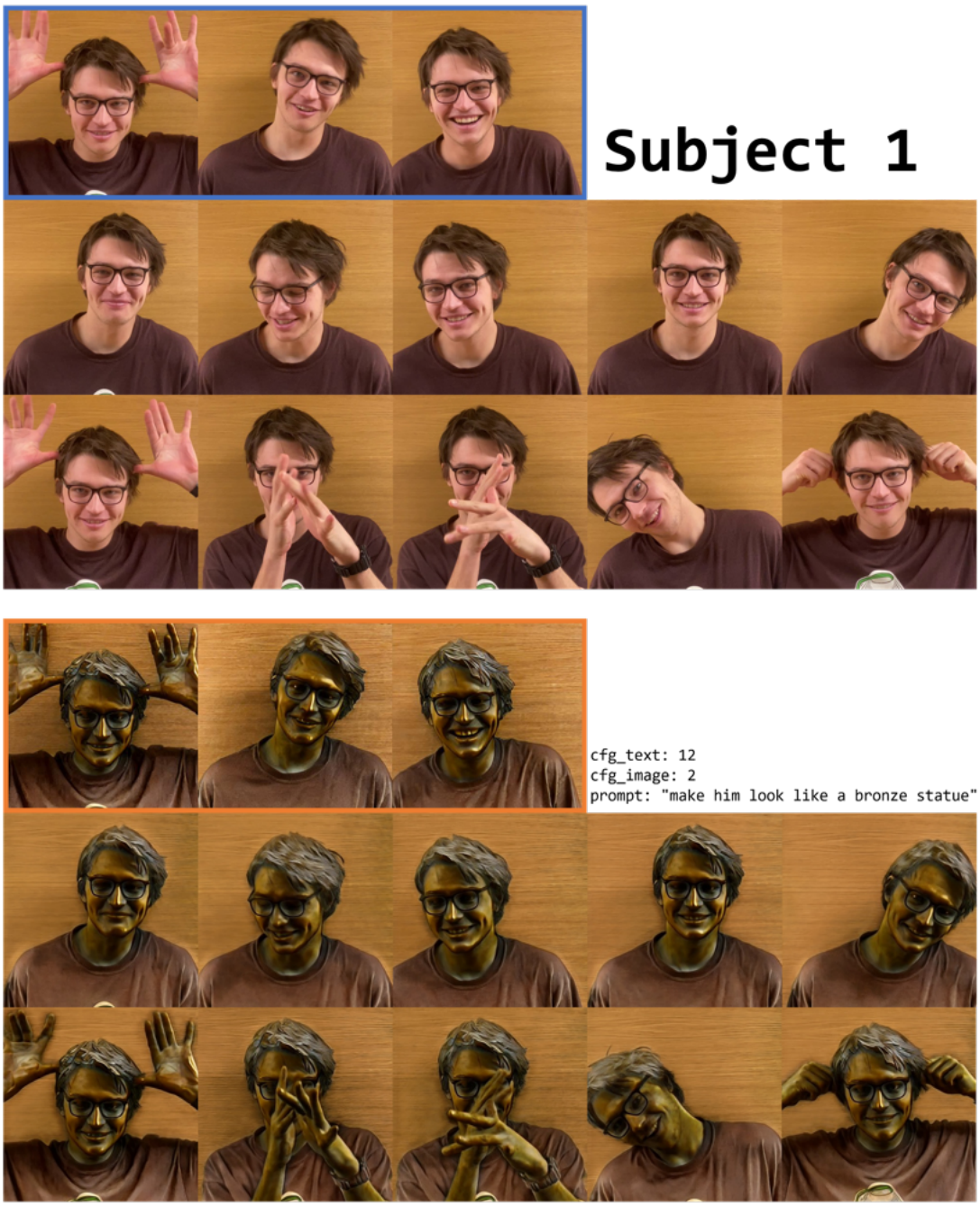


Figure 6.12: Subject 1:
Top: Input keyframes (blue border) and input video test frames.
Bottom: Stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

Figure 6.13: Subject 1:
Two styles (top and bottom); each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

Figure 6.14: Subject 2:
Top: Input keyframes (blue border) and input video test frames.
Bottom: Stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

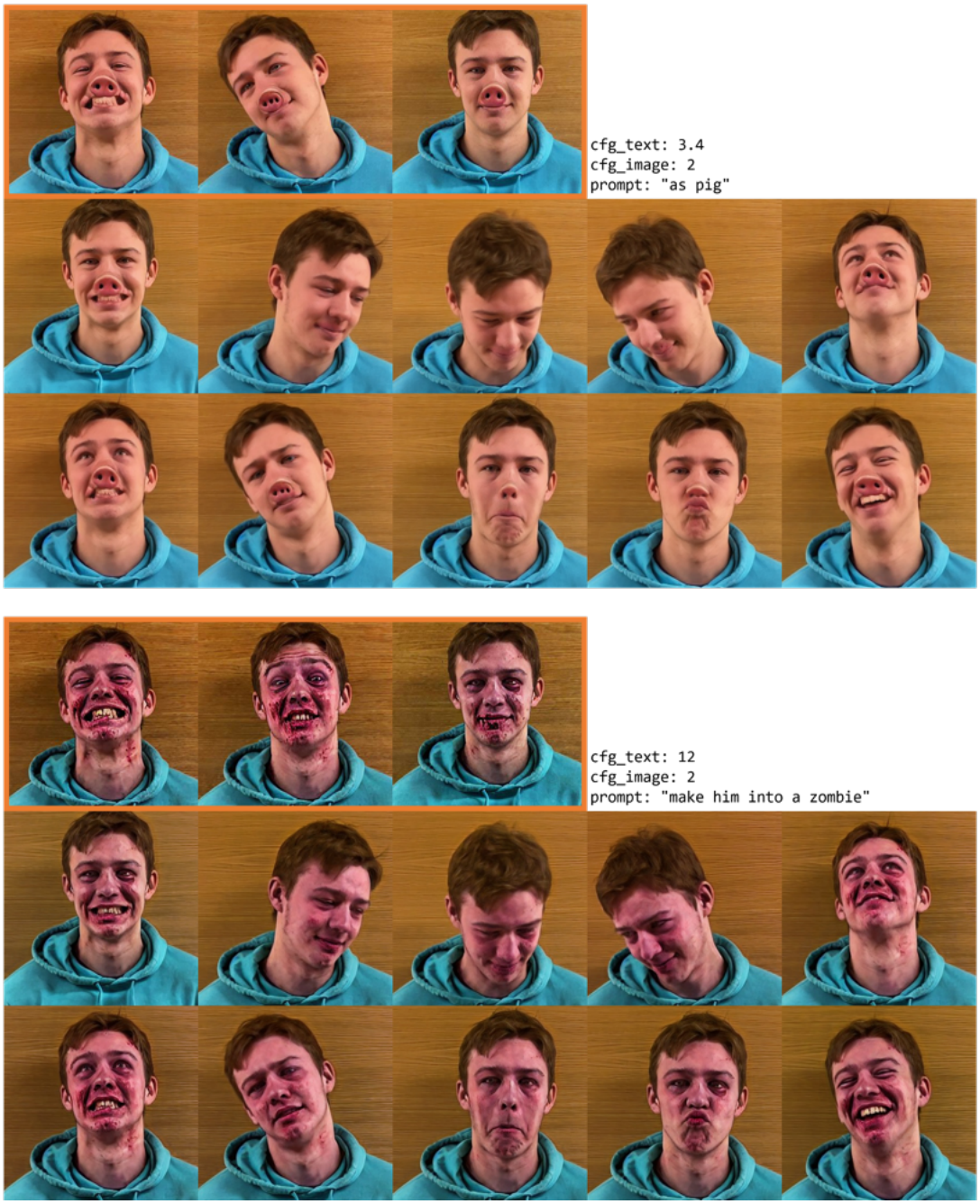


Figure 6.15: Subject 2:
Two styles (top and bottom); each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

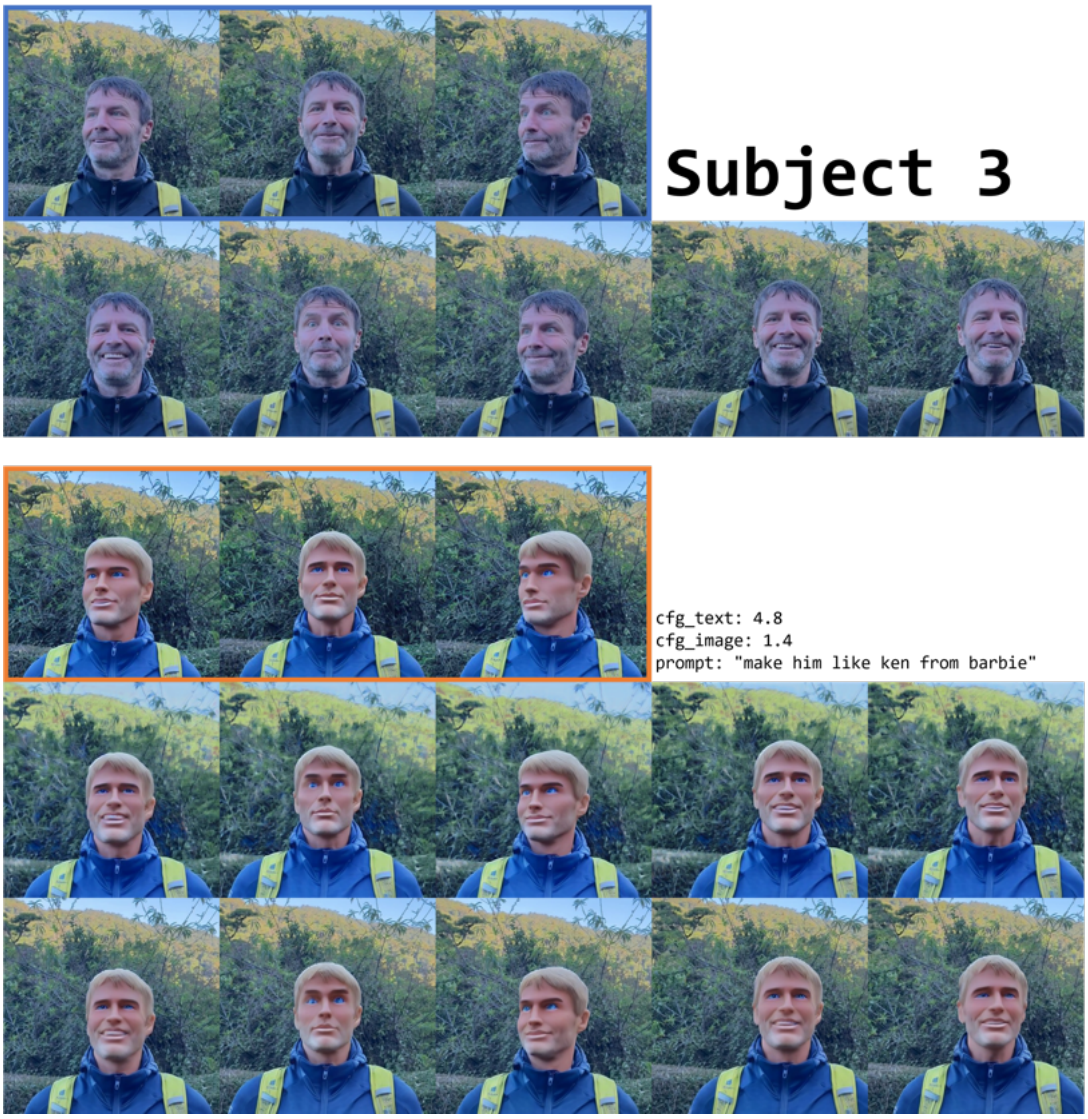


Figure 6.16: Subject 3:
Top: Input keyframes (blue border) and input video test frames.
Bottom: Stylized keyframes (orange border), target style configuration parameters, and stylized test frames before and after background replacement.

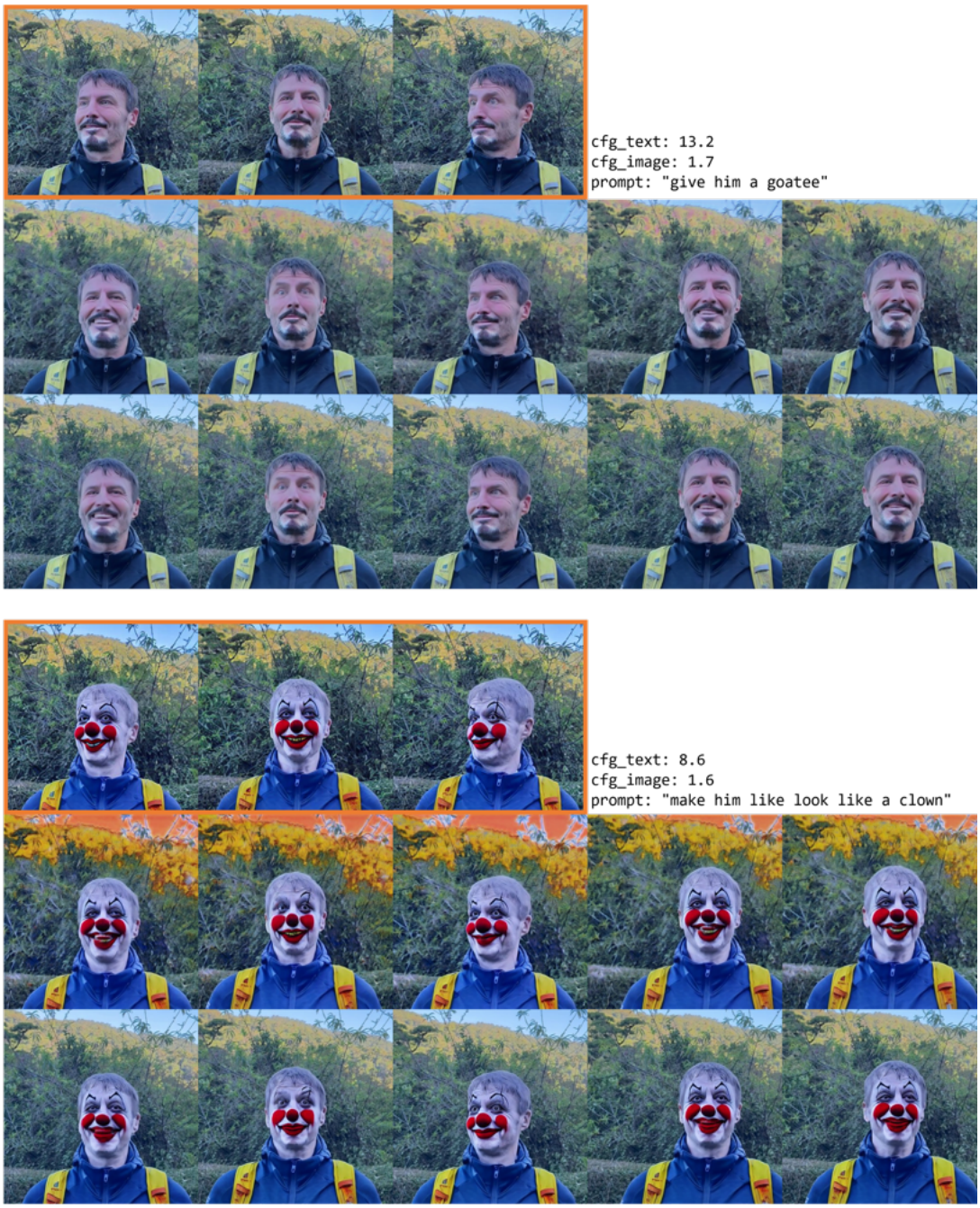


Figure 6.17: Subject 3:
Two styles (top and bottom); each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames before and after background replacement.

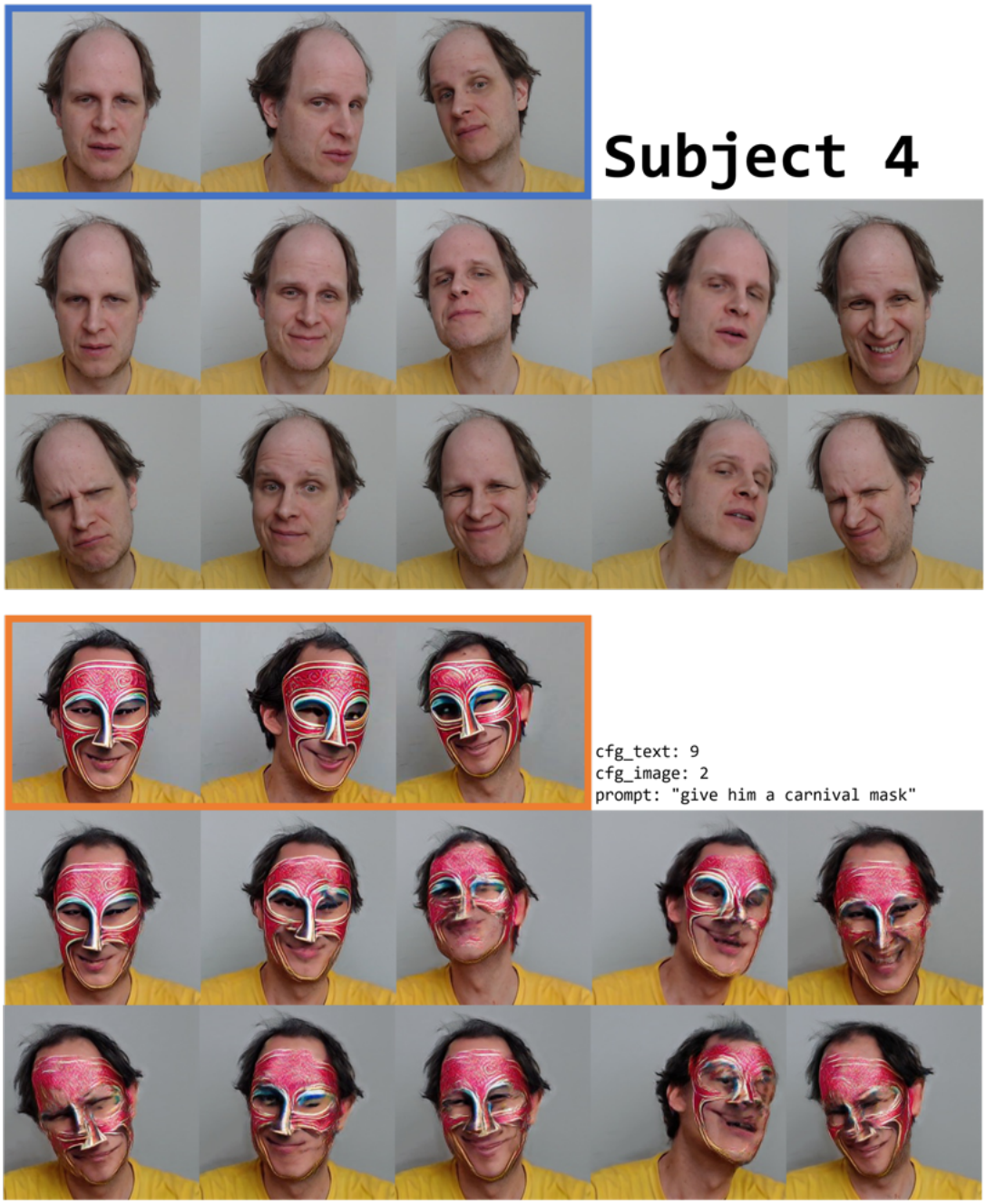


Figure 6.18: Subject 4:
Top: Input keyframes (blue border) and input video test frames.
Bottom: Stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

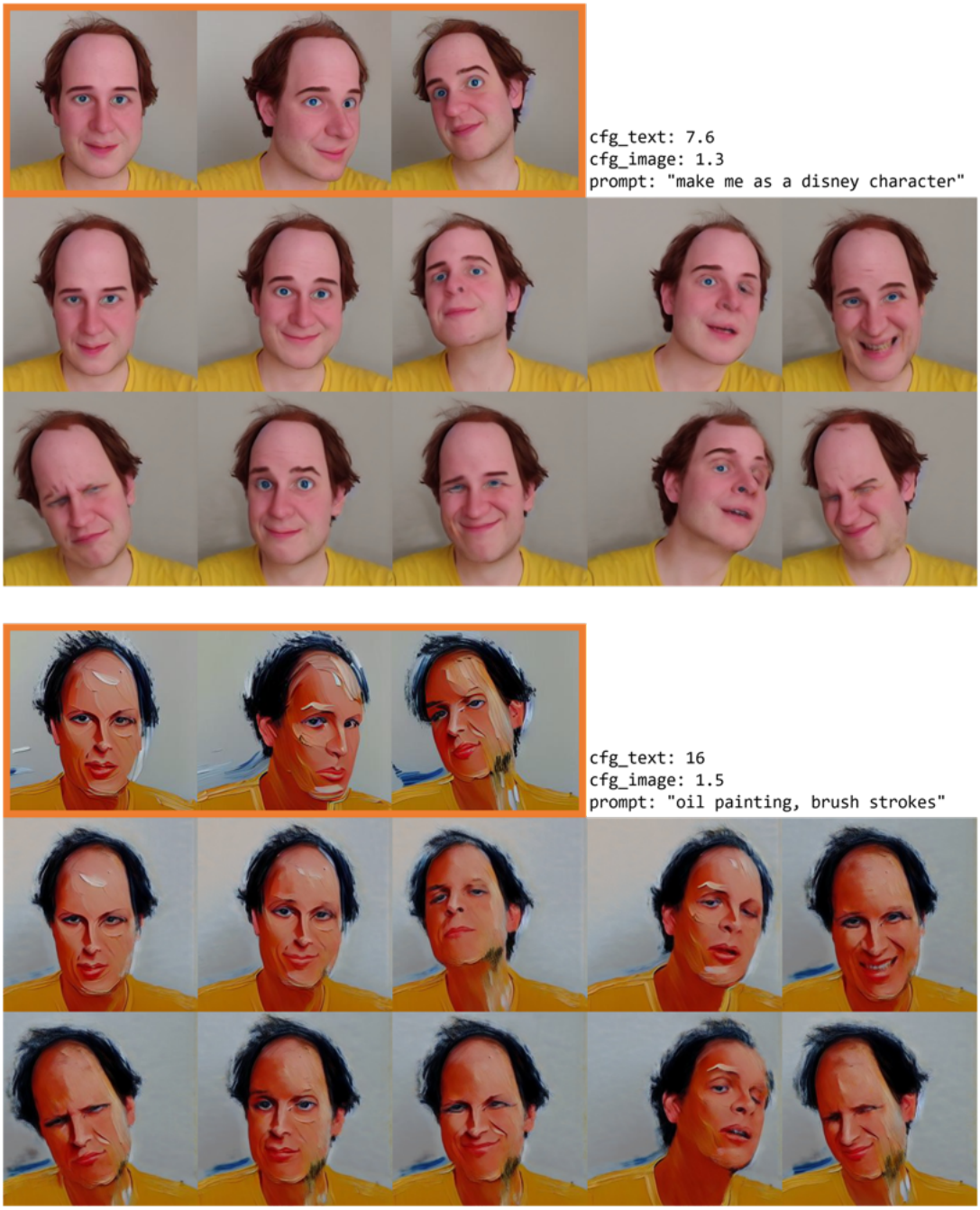


Figure 6.19: Subject 4:
Two styles (top and bottom); each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

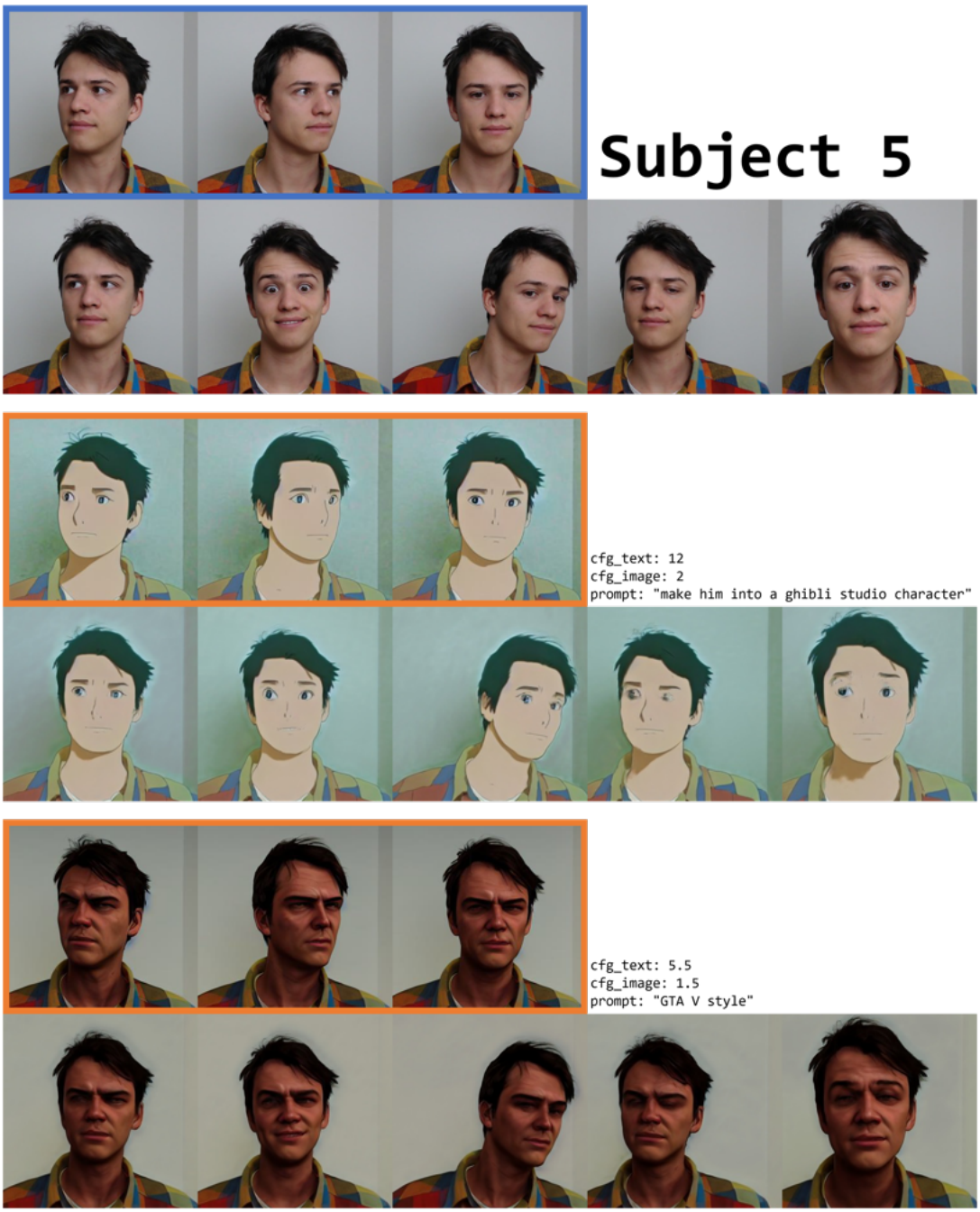


Figure 6.20: Subject 5:
Top: Input keyframes (blue border) and input video test frames.
Middle and bottom: Two styles, each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

6.3.1 Limitations and Failure Cases

This section examines the boundaries of our method's capabilities and identifies scenarios where it fails to deliver the intended results. Understanding these limitations is crucial for further refinement of the system.

Keyframe Consistency. The primary limitation arises from the capabilities of IP2P. The efficacy of our style transfer relies on the diffusion model's ability to create a consistent style. If the diffusion model does not create consistent keyframes, the subsequent video stylization is compromised. In this context, consistency means one of two things:

One is the consistency between an input keyframe and the corresponding stylized output keyframe. This means the alignment of features between these two key frames, such as the mouth, eyes, etc. For inexpressive or mostly non-moving features (such as nose, ears or hair), this alignment isn't necessary for a successful stylization. The *"as pig"* style of Subject 2 (Figure 6.15) is an example of this form of inconsistency that produces the desired result. Nevertheless, when the facial features align, the subsequent movement of the given feature in the stylized video will be well defined–it will correspond to the movement of the feature in the input video. For example, if the mouth of the stylized keyframe aligns with the mouth of the input keyframe, opening the mouth will cause the mouth to open in the stylized video.

The other is consistency between stylized keyframes. This means that the feature mapped by one stylized keyframe to a specific location will be mapped to the same location by other stylized keyframes. For example, this means that if one stylized keyframe has ears slightly lower on the face compared to the input keyframe, other stylized keyframes should also have ears slightly lower compared to their input keyframe. If the other keyframes have ears in a different location, the keyframes are inconsistent.

Even minor inconsistencies between the stylized keyframes can have a pronounced impact on the stylization. IP2P lacks a mechanism to address this issue directly. Our current method attempts to mitigate this by generating all stylized keyframes in a single inference run, introducing a form of regularization. However, this strategy does not inherently guarantee consistency, and discrepancies between keyframes still arise. An example of this issue can be seen in the style *"give him a carnival mask"* of Subject 4 in Figure 6.18. In this example, the produced style completely fails to generalize to unseen angles and the inconsistently stylized keyframes compete against each other, resulting in a broken mess.

Keyframe Coverage. Another limitation is observed when keyframes do not encapsulate the full spectrum of the subject's movements or expressions. In such cases, the style fails to generalize well to new positions or expressions, resulting in unsatisfactory stylization for previously unseen facets of the head or exaggerated expressions. This issue is particularly noticeable in the video of Subject 2, where the wide array of head movements and facial expressions extend beyond the perspectives captured by the limited set of keyframes.

Style Boundary. We also encounter a limitation inherited from StyleVid. The scope of style possibilities is confined to those that do not extend beyond the subject's outline.

Any stylistic elements that exceed these boundaries are associated with the background during the training phase. This limitation is exemplified in Figure 6.21, where the added elements that exceed the subject's outline are lost due to being trained to latch onto the featureless wall in the background.

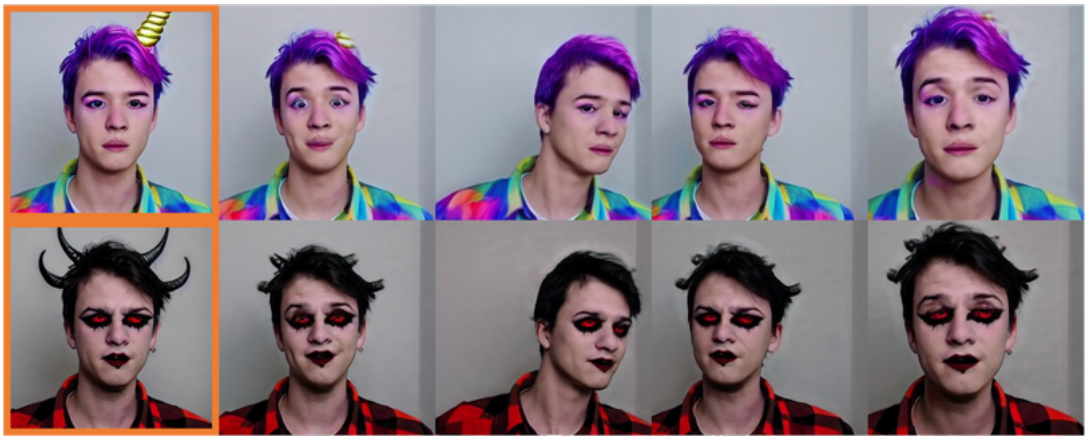


Figure 6.21: Limitation of styles extending beyond the subject's silhouette.

Manual Keyframe Choice. Finally, our implementation is also slightly limited in terms of interactivity. The system relies on the user to provide keyframes manually. Users must actively choose frames from the video feed that they anticipate will best represent the range of motions, camera angles, and facial expressions they will make during interaction. This extra burden placed on the user takes away from the effortlessness of the interaction, forcing the user to retake keyframes in case of change in perspective.

6.3.2 Future Work

One of the main limitations identified is the challenge of ensuring consistent stylization across multiple keyframes generated by InstructPix2Pix. Addressing the consistency issue is crucial to enable robust propagation of the style to the full video.

There are several promising directions to consistency across keyframes. We could use one of the many diffusion model video stylization techniques: One approach is to leverage Video InstructPix2Pix, an extension of the InstructPix2Pix framework designed for temporally consistent video generation [26]. Another possibility is to use the approach from Rerender A Video [47] and let the user capture and fine-tune the style on an anchor keyframe, then propagate the style consistently to the rest. This approach would also improve interactivity, as tuning parameters on one image could be significantly faster.

Another way to improve user experience and style coverage is to automate keyframe selection. This extension would be compatible with the anchor keyframe approach; after setting an anchor, other keyframes would be identified automatically based on the current set of keyframes and the user's face position.

6.4 User Demonstration

Throughout the development of this project, we had the opportunity to present the application to the public during two distinct events, providing invaluable feedback on user interaction with the system.

The first demonstration took place at the Uroboros: Creative AI meet-up, hosted at Petrohradská kolektiv, as a part of Dny AI. The second opportunity was at the Open Day event at CTU, where we represented the Department of Computer Graphics and Interaction (DCGI).

During the Uroboros event, approximately 15 individuals directly interacted with the application (Figures 6.22 and 6.23), while the Open Day at CTU saw more than 100 people come into contact with the application, with around 20 engaging hands-on. These demonstrations, though not formal user studies, offered crucial insights into user engagement and interaction with the application.

Figure 6.22: Participants at the Uroboros: Creative AI meet-up.

The intuitive nature of the text prompt interface was mostly confirmed by users, although observations led to ideas for further enhancements to streamline the experience. At this stage, the creation and propagation of the style to the video was slower, taking about 25 s, which limited the users' willingness to experiment with multiple styles. However, they still found the experience fun and interactive.

The real-time stylization effects were generally perceived as compelling. Users who managed to create well-matching, coherent stylized keyframes showed enjoyment

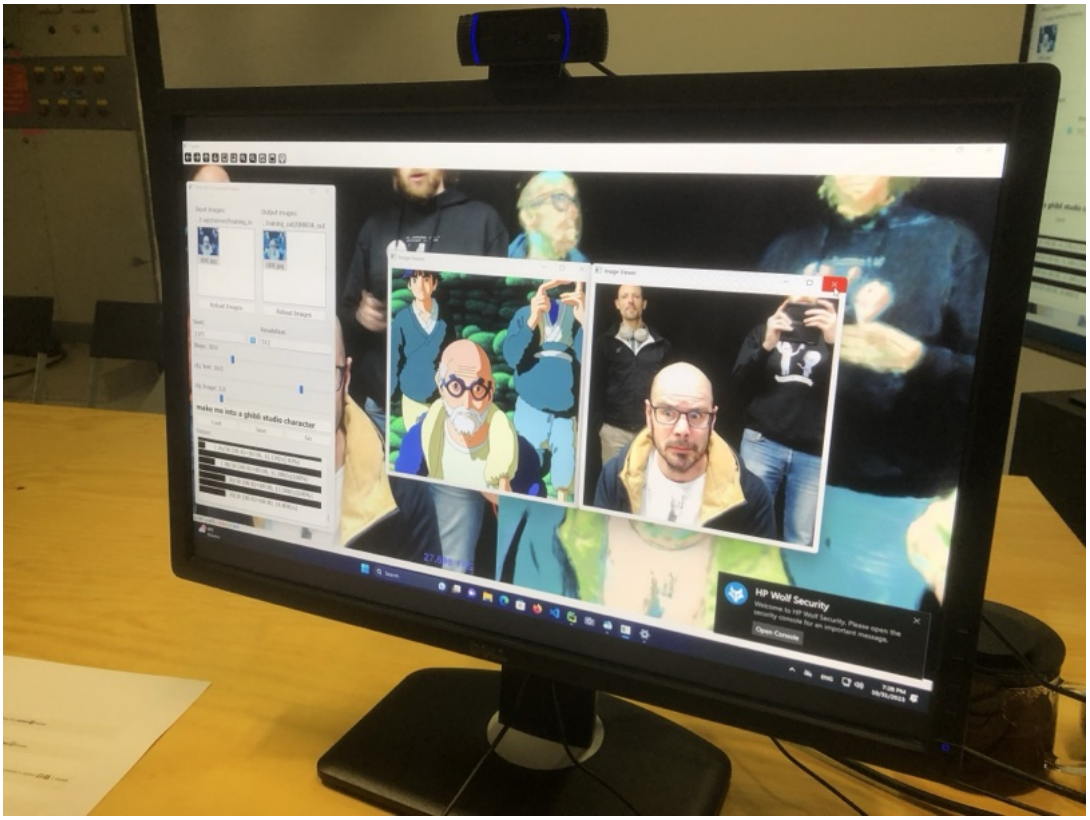


Figure 6.23: Participant at the Uroboros: Creative AI meet-up examining the created style.

in experimenting with movements in front of the camera. Some were intrigued by testing the limits of stylization, exploring how far they could move before breaking the stylization.

One notable limitation observed was users' desire to add elements such as hats to their faces. However, our method could not accommodate this use case, as the structure of added items like hats would attach to the background during training and not move with the user's head in the video. This limitation was recognized, but it remains unaddressed within the current scope of our method.

The ability to produce a wide diversity of styles was established. However, first-time users found it challenging to create certain styles because adjusting CFG parameters requires experience to develop a sense of intuition.

Users also noted limitations, particularly regarding the interface setup. Initially, the interface was divided into two client applications, one for video display and the other containing UI elements to direct IP2P keyframe stylization. This setup was cumbersome because, due to technical limitations of camera access, the user was required to switch focus between these windows when capturing new keyframes. This feedback helped to streamline the interface for a more cohesive user experience.

Although these demonstrations were only indicative, the feedback received was important in validating the application's goals like interactivity, speed, coherence, and the range of artistic expression.

Chapter 7

Conclusion

This thesis introduced a novel approach to real-time artistic stylization of video using text prompts and diffusion models. The key challenge was to enable users to interactively transform their live video stream through simple text prompts while maintaining high visual quality and real-time performance.

Our technique combined the complementary strengths of InstructPix2Pix and StyleVid. InstructPix2Pix leveraged recent advances in text-guided diffusion models to synthesize stylized keyframes representing the desired artistic edits. StyleVid then efficiently propagated these styles to all video frames using a fast patch-based neural stylization network. By balancing the generative capabilities of diffusion models and the speed of neural propagation, this hybrid pipeline delivered real-time streaming at 30 fps with imperceptible lag between input capture and stylized output.

Beyond fast performance, our method enables intuitive user interaction through text prompts with reasonable style delays between entering prompts and observing their video effects. Tailored to typical video conferencing footage containing a subject's head and torso, it achieved visually compelling and diverse stylizations tuned by the diffusion model configuration parameters. The flexibility was validated across videos featuring different subjects, head motions, facial expressions, cameras, and background settings. However, limitations regarding consistency of stylized keyframes and coverage of exaggerated motions outside the scope of conventional video calls were identified to be addressed in future work.

Compared to existing video stylization literature, our method enables previously unattained creative self-expression in a real-time interactive setting, albeit with a more restricted range of styles than full video diffusion models. Still, user feedback from public demonstrations affirmed the system's intuitive interaction and showed a sufficiently expressive range of styles validating its real-world application.

Future work should focus on ensuring consistent keyframe stylization and automatically selecting optimal frames to maximize coverage. Addressing these limitations can progress text-guided video stylization toward flexible real-time applications like visually customizable video conferencing and streaming.

Appendix A

Bibliography

[1] Vincent van Gogh. The Starry Night. Saint Rémy, June 1889 | MoMA.

[2] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your Latents: High-Resolution Video Synthesis with Latent Diffusion Models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 22563–22575, 2023.

[3] Gary Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. InstructPix2Pix: Learning to Follow Image Editing Instructions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 18392–18402, 2023.

[5] Duygu Ceylan, Chun-Hao Huang, and Niloy J. Mitra. Pix2Video: Video Editing using Image Diffusion. In *Proceedings of IEEE International Conference on Computer Vision*, pages 23206–23217, 2023.

[6] Cheng-Kang Ted Chao and Yotam Gingold. Text-guided Image-and-Shape Editing and Generation: A Short Survey, 2023. arXiv:2304.09244.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[8] Patrick Esser, Johnathan Chiu, Parmida Atighehchian, Jonathan Granskog, and Anastasis Germanidis. Structure and Content-Guided Video Synthesis with Diffusion Models. In *Proceedings of IEEE International Conference on Computer Vision*, pages 7346–7356, 2023.

[9] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings. *ACM Transactions on Graphics*, 35(4):92, 2016.

[10] David Futschik. *Leveraging Machine Learning for Artistic Stylization*. PhD Thesis, Czech Technical University in Prague, 2023.

[11] David Futschik, Menglei Chai, Chen Cao, Chongyang Ma, Aleksei Stoliar, Sergey Korolev, Sergey Tulyakov, Michal Kučera, and Daniel Sýkora. Real-Time Patch-Based Stylization of Portraits Using Generative Adversarial Network. In *Proceedings of the ACM/EG Expressive Symposium*, pages 33–42, 2019.

[12] David Futschik, Michal Kučera, Michal Lukáč, Zhaowen Wang, Eli Shechtman, and Daniel Sýkora. STALP: Style Transfer with Auxiliary Limited Pairing. *Computer Graphics Forum*, 40(2):563–573, 2021.

[13] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[14] Rohit Girdhar, Mannat Singh, Andrew Brown, Quentin Duval, Samaneh Azadi, Sai Saketh Rambhatla, Akbar Shah, Xi Yin, Devi Parikh, and Ishan Misra. Emu Video: Factorizing Text-to-Video Generation by Explicit Image Conditioning, 2023. arXiv:2311.10709.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[16] Miguel Grinberg. *Flask web development: developing web applications with python*. O'Reilly Media, Inc., 2018.

[17] Yuwei Guo, Ceyuan Yang, Anyi Rao, Yaohui Wang, Yu Qiao, Dahua Lin, and Bo Dai. AnimateDiff: Animate Your Personalized Text-to-Image Diffusion Models without Specific Tuning. In *Proceedings of International Conference on Learning Representations*, 2024.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[19] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-Prompt Image Editing with Cross Attention Control. In *Proceedings of International Conference on Learning Representations*, 2023.

[20] Aaron Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *SIGGRAPH Conference Proceedings*, pages 453–460, 1998.

[21] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image Analogies. In *SIGGRAPH Conference Proceedings*, pages 327–340, 2001.

[22] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.

[23] Jonathan Ho, Chitwan Saharia, William Chan, David Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded Diffusion Models for High Fidelity Image Generation. *The Journal of Machine Learning Research*, 23(47):2249–2281, 2021.

[24] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *Proceedings of International Conference on Learning Representations*, 2022.

[25] Ondřej Jamriška, Šárka Sochorová, Ondřej Texler, Michal Lukáč, Jakub Fišer, Jingwan Lu, Eli Shechtman, and Daniel Sýkora. Stylizing Video by Example. *ACM Transactions on Graphics*, 38(4):107, 2019.

[26] Levon Khachatryan, Andranik Movsisyan, Vahram Tadevosyan, Roberto Henschel, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. Text2Video-Zero: Text-to-Image Diffusion Models are Zero-Shot Video Generators. In *Proceedings of IEEE International Conference on Computer Vision*, pages 15954–15964, 2023.

[27] Kyung-Soo Kim and Yong-Suk Choi. HyAdamC: A New Adam-Based Hybrid Optimization Algorithm for Convolution Neural Networks. *Sensors*, 21(12):4054, 2021.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, 2012.

[29] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[31] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. MediaPipe: A Framework for Building Perception Pipelines, 2019. arXiv:1906.08172.

[32] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations. In *Proceedings of International Conference on Learning Representations*, 2022.

[33] Eyal Molad, Eliahu Horwitz, Dani Valevski, Alex Rav Acha, Yossi Matias, Yael Pritch, Yaniv Leviathan, and Yedid Hoshen. Dreamix: Video Diffusion Models are General Video Editors. In *Proceedings of International Conference on Learning Representations*, 2024.

[34] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models. In *Proceedings of International Conference on Machine Learning*, pages 16784–16804, 2022.

[35] Chenyang Qi, Xiaodong Cun, Yong Zhang, Chenyang Lei, Xintao Wang, Ying Shan, and Qifeng Chen. FateZero: Fusing Attentions for Zero-shot Text-based Video Editing. In *Proceedings of IEEE International Conference on Computer Vision*, pages 15932–15942, 2023.

[36] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of International Conference on Machine Learning*, pages 8748–8763, 2021.

[37] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents, 2022. arXiv:2204.06125.

[38] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 10674–10685, 2022.

[39] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Proceedings of Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[40] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[41] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic Style Transfer for Videos. In *Proceedings of German Conference Pattern Recognition*, pages 26–36, 2016.

[42] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic Style Transfer for Videos and Spherical Images. *International Journal of Computer Vision*, 126(11):1199–1219, 2018.

[43] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning Using Nonequilibrium Thermodynamics. In *Proceedings of International Conference on Machine Learning*, pages 2256–2265, 2015.

[44] Ondřej Texler, David Futschik, Michal Kučera, Ondřej Jamriška, Šárka Sochorová, Menglei Chai, Sergey Tulyakov, and Daniel Sýkora. Interactive Video Stylization Using Few-Shot Patch-Based Training. *ACM Transactions on Graphics*, 39(4):73, 2020.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, 2017.

[46] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion Models: A Comprehensive Survey of Methods and Applications. *ACM Computing Surveys*, 56(4):105, 2023.

[47] Shuai Yang, Yifan Zhou, Ziwei Liu, , and Chen Change Loy. Rerender A Video: Zero-Shot Text-Guided Video-to-Video Translation. In *SIGGRAPH Asia Conference Papers*, page 95, 2023.

[48] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models. In *Proceedings of IEEE International Conference on Computer Vision*, pages 3836–3847, 2023.

[49] Xu Zhang, Svebor Karaman, and Shih-Fu Chang. Detecting and Simulating Artifacts in GAN Fake Images. In *Proceedings of IEEE International Workshop on Information Forensics and Security*, 2019.

Appendix B
Additional Results

Additional stylization results for Subjects 1, 2 and 4.

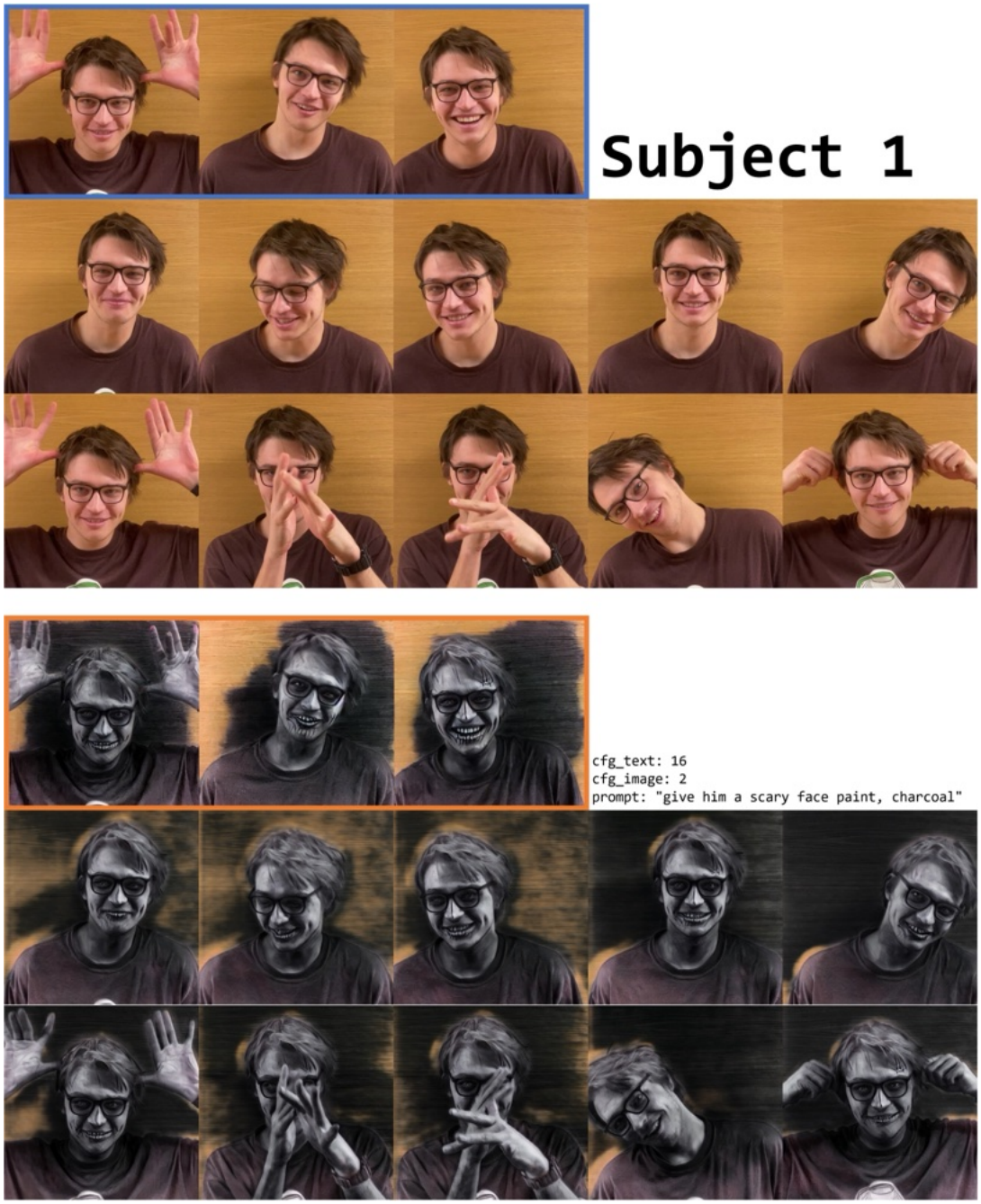


Figure B.1: Subject 1:
Top: Input keyframes (blue border) and input video test frames.
Bottom: Stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

Figure B.2: Subject 1:
Two styles (top and bottom); each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

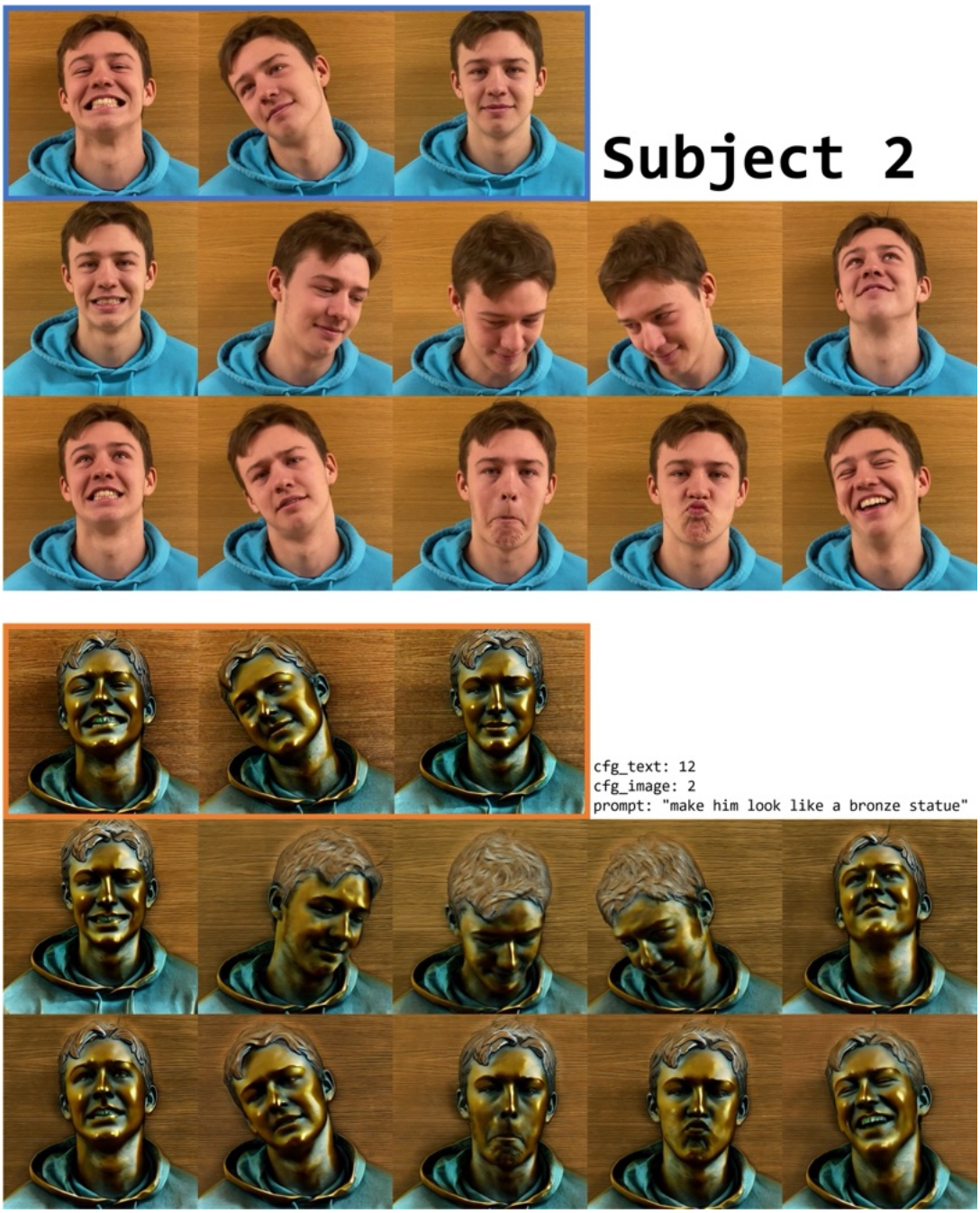


Figure B.3: Subject 2:
Top: Input keyframes (blue border) and input video test frames.
Bottom: Stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

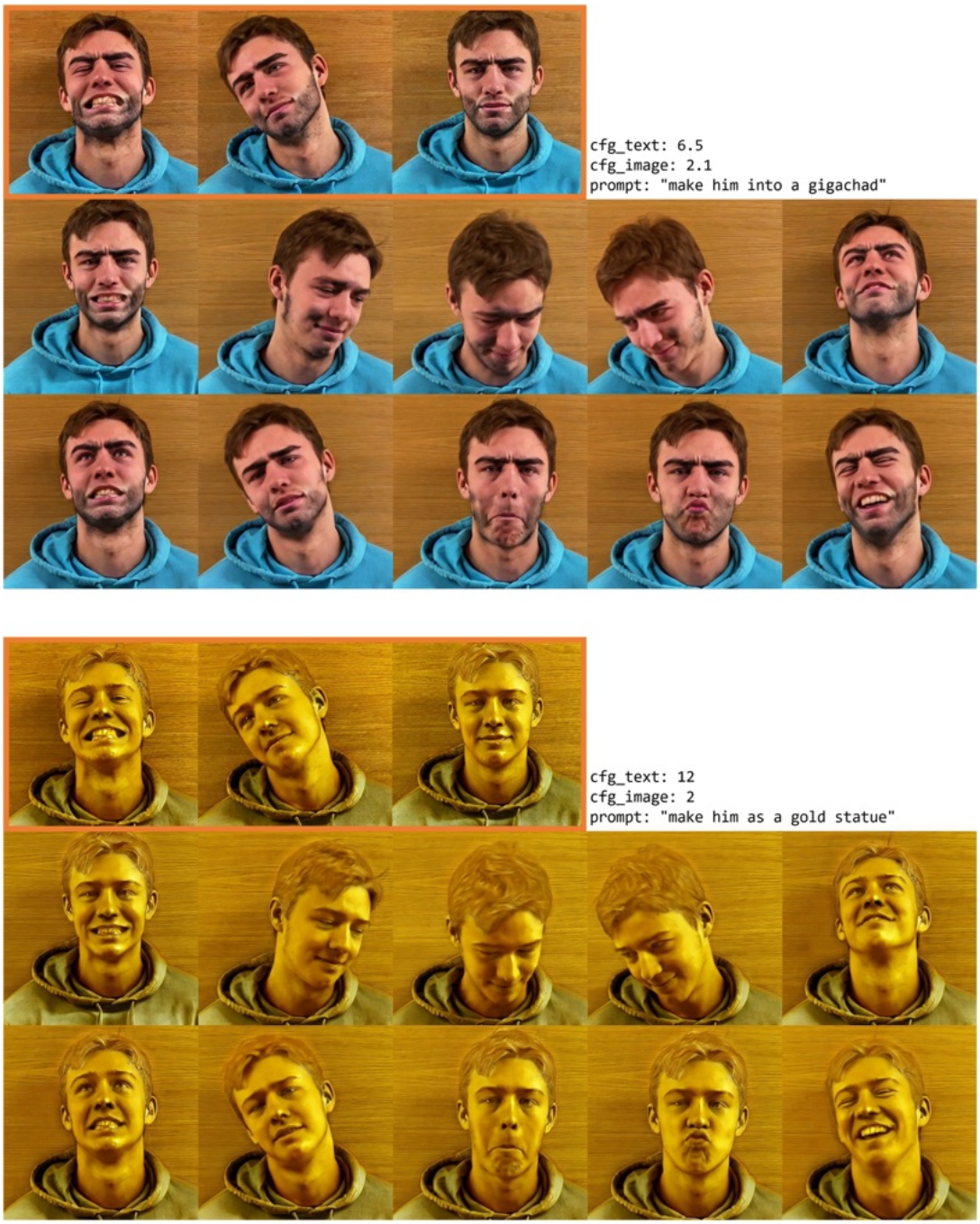


Figure B.4: Subject 2:
Two styles (top and bottom); each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

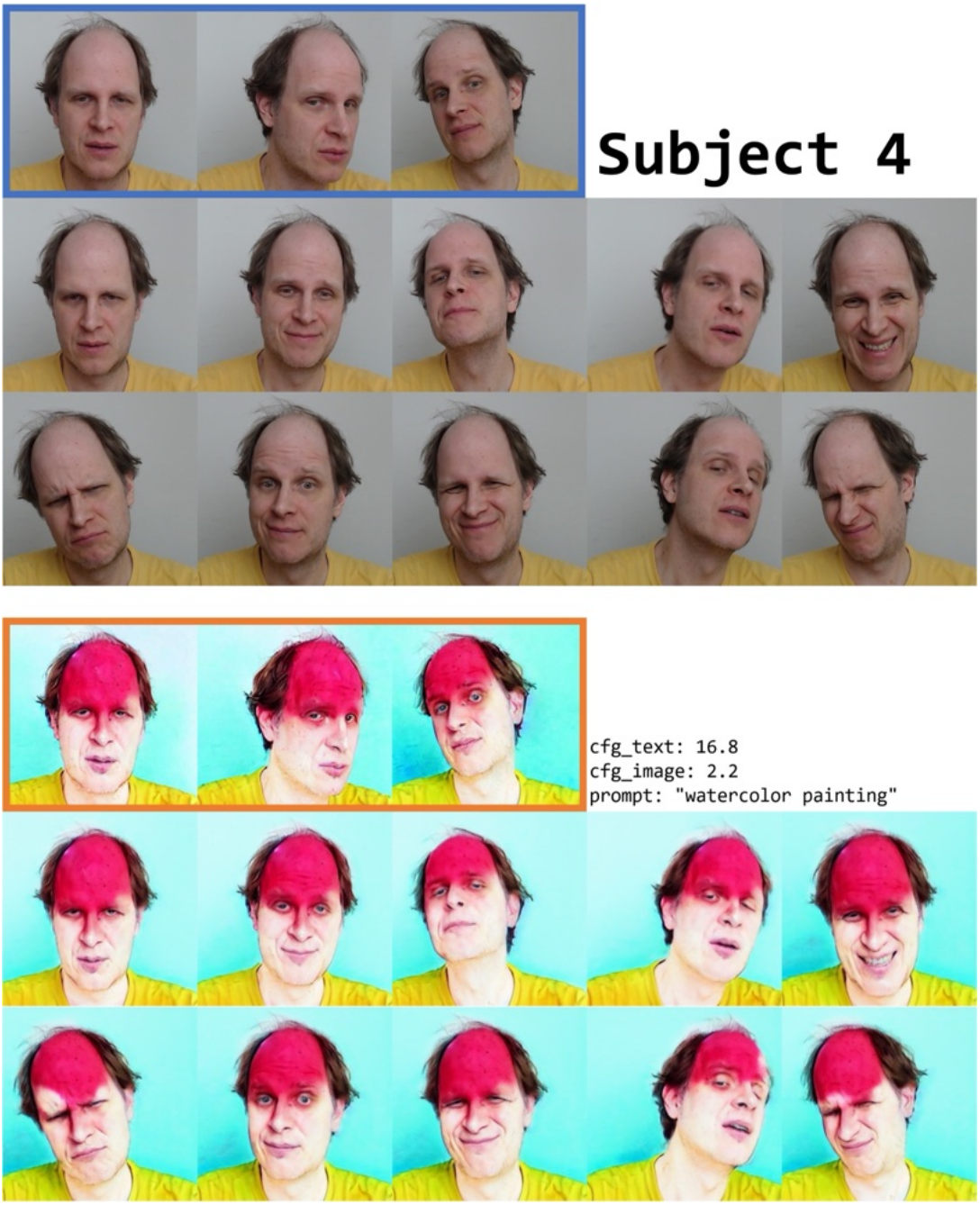


Figure B.5: Subject 4:
Top: Input keyframes (blue border) and input video test frames.
Bottom: Stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

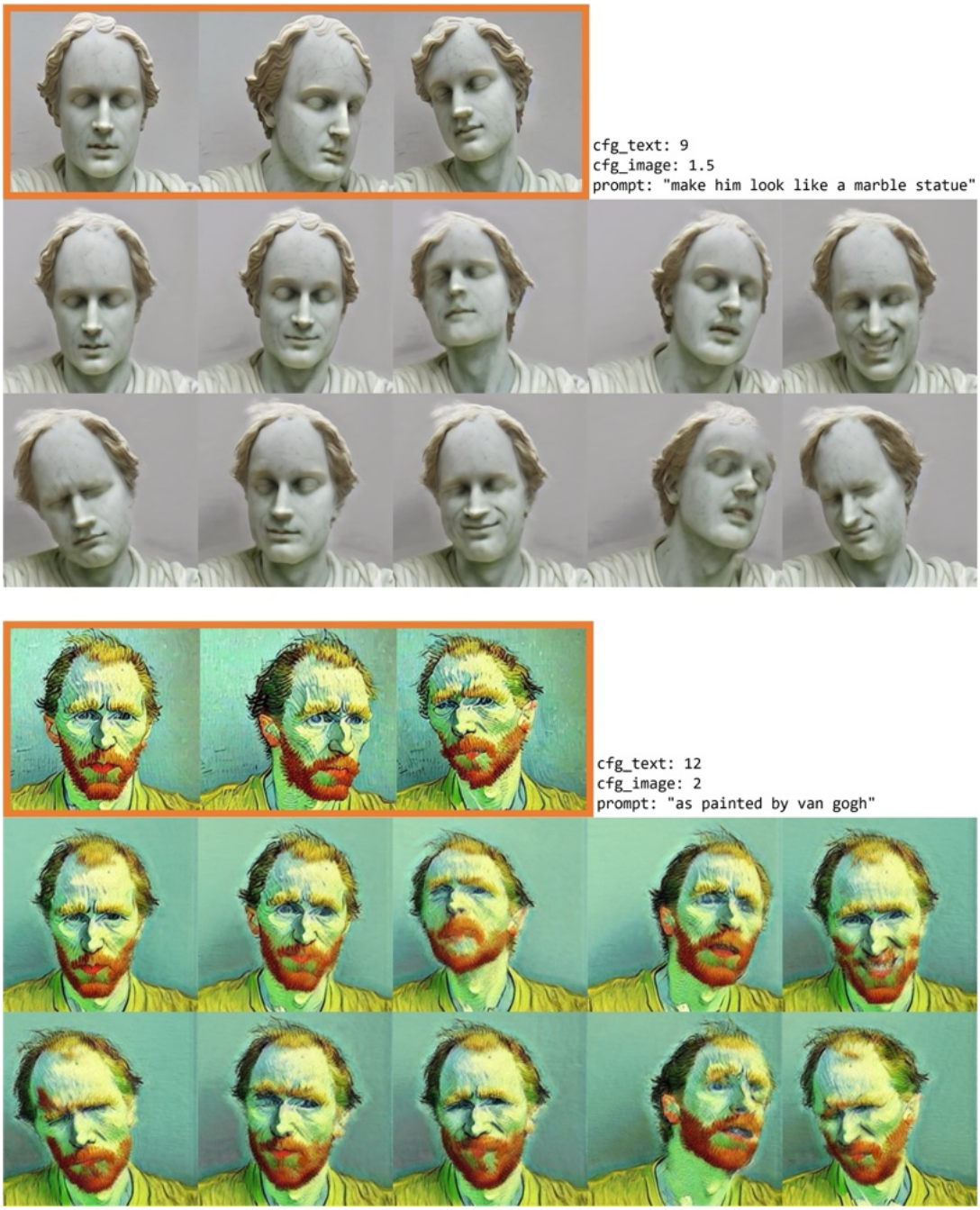


Figure B.6: Subject 4:
Two styles (top and bottom); each with stylized keyframes (orange border), target style configuration parameters, and stylized test frames.

Appendix C

Attachment Files

/
- `thesis.pdf` ..... pdf file with the thesis text
- `code` ..... code directory for the client and server
- `README.md` ..... GitHub links