

Learned Parametric Emotion Editing of Images

A Real-Time Android Application for
Emotionally Balanced
Social Media Consumption

Bachelor Thesis 2026

Eastern Switzerland University of Applied Sciences

Authors: Musa Rochi, Marcel Schubert

Advisor: Prof. Dr. Christoph Gebhardt

1 Abstract

Problematic internet use is increasing, particularly among younger populations. In Switzerland, the share of affected individuals aged 15 and older rose from 3.8% in 2017 to 6.8% in 2022. Since common interventions such as time limits can be perceived as coercive, this thesis investigates a less restrictive alternative: regulating the emotional intensity of visual social media content during consumption through image adaptation.

Prior optimization based emotion guided image editing has shown that visual affect can be steered, but its iterative nature limits practical use. This thesis therefore learns a substitute for such editing and integrates it into a proof of concept system (PoC) for real time video transformation on an Android device.

The presented model is trained and evaluated on COCO images and targets arousal as defined in the circumplex model of affect (CMA). Using a relative loss objective, it learns to shift each image's arousal, as estimated by a pre-trained regressor, by a given conditioning value. Instead of generating images, the model predicts parameters for differentiable global transformations, which are applied sequentially in a fixed order. Its architecture uses MobileNetV4 as an edge oriented feature extractor, while FiLM based modulation enables flexible conditioning and supports edit diversity. Additionally, CLIP based preservation supports image quality, while sigmoid-bounded outputs keep predictions anchored around default values.

The PoC embeds Instagram through Android WebView, detects video elements during playback, and applies the predicted parameters using shader based rendering. To handle inference times that can exceed the frame interval, asynchronous inference and fixed scheduling decouple prediction from rendering, while low pass filtering reduces distortions.

The resulting arousal changes follow the conditioning direction, with an OLS slope of $a = 0.317$, $b = 0.006$, $R^2 = 0.668$ and $MAE = 0.117$. This behavior is also reflected in arousal related low level image metrics, including saturation and colorfulness. Weaker, less linear effects are observed for valence, the second dimension of the CMA, with $a = 0.089$ and $R^2 = 0.091$. As conditioning strength increases, KID, FID, and L1 distance rise in both mean and variance. Mean inference latency decreases from 99.01 ms to 3.72 ms on the training hardware, corresponding to a 27-fold speedup over the reference optimization baseline, while the model reaches 40.48 ms on a Samsung Galaxy S23.

Overall, the findings suggest that learned parametric emotion editing provides a promising technical basis for efficient affective adaptation in practical social media environments.

Keywords: Problematic Internet Use, Digital Well-Being, Affective Computing, Learned Parametric Image Editing, Real-Time Video Transformation, Mobile Machine Learning

1	<u>ABSTRACT.....</u>	<u>2</u>
2	<u>INTRODUCTION.....</u>	<u>7</u>
2.1	GOAL.....	8
2.2	METHODOLOGY	9
2.3	SCOPE.....	9
3	<u>BACKGROUND AND RELATED WORK.....</u>	<u>10</u>
3.1	REGRESSOR-GUIDED IMAGE EDITING	11
3.1.1	AFFECT-SPACE REGRESSOR	12
3.1.2	CRITERIA	13
3.1.3	PARAMETRIC OPTIMIZATION METHOD	13
3.1.4	LATENT STYLE OPTIMIZATION METHOD	13
3.1.5	REGRESSOR-GUIDED DIFFUSION RESAMPLING (RGDR)	14
3.1.6	COMPUTATIONAL FEASIBILITY.....	14
3.2	AFFECTIVE IMAGE ADAPTATION	17
3.2.1	PARAMETRIC MODELS	17
3.2.2	LATENT SPACE MODELS	18
3.2.3	ADDITIONAL APPROACHES.....	19
3.3	AMORTIZED OPTIMIZATION	20
3.4	MODEL EFFICIENCY CONSIDERATIONS	21
3.4.1	QUANTIZATION	21
3.4.2	PRUNING	21
3.4.3	KNOWLEDGE DISTILLATION	21
3.4.4	DISTILLATION APPLICATION USING REAL-TIME FILTERS	22
3.5	READING AND TRANSFORMING FRAMES	22
3.5.1	WEBVIEW.....	23
3.5.2	SCREEN CAPTURING	25
3.5.3	CAPTURE AND RENDER	25
4	<u>REQUIREMENTS.....</u>	<u>27</u>
4.1	FUNCTIONAL REQUIREMENTS	28
4.2	NON-FUNCTIONAL REQUIREMENTS	31
4.3	DOMAIN MODEL	32
5	<u>DESIGN.....</u>	<u>33</u>

5.1	DESIGN CONSIDERATIONS	33
5.1.1	FRAME EXTRACTION APPROACH	34
5.1.2	MODEL RUNTIME	35
5.1.3	MEMORY LOCALITY	36
5.1.4	GLOBAL TRANSFORMATIONS	38
5.2	ARCHITECTURE.....	39
5.2.1	ARCHITECTURALLY RELEVANT DECISIONS (ARD)	41
6	<u>METHOD.....</u>	43
6.1	PARAMETRIC GENERATOR	44
6.1.1	IMPROVING TRANSFORMATION IDENTITIES	45
6.1.2	IMPROVING IMAGE PRESERVATION	46
6.1.3	IMPROVING EDIT DIVERSITY	46
6.1.4	TRAINING AND TUNING.....	50
6.2	CHOICE OF CONTROL SIGNAL	52
7	<u>IMPLEMENTATION</u>	53
7.1	WEBVIEW CORS.....	53
7.2	GLOBAL TRANSFORMATIONS: INDEPENDENCE FROM IMAGE SIZES	53
7.2.1	SHADER.....	54
7.2.2	RENDERING PROCESS.....	55
7.3	SCHEDULING AND SMOOTHING.....	57
7.4	END-TO-END PROCESS	59
7.5	USER INTERFACE.....	61
7.6	DEPLOYMENT	62
8	<u>EVALUATION.....</u>	63
8.1	MODEL	63
8.1.1	TRAINING	65
8.1.2	IMAGE PRESERVATION METRICS	68
8.1.3	IMAGE ATTRIBUTE METRICS.....	69
8.1.4	OLS.....	71
8.1.5	QUALITATIVE RESULTS	73
8.2	PERFORMANCE BENCHMARKING.....	76
8.3	APPLICATION	77

9	<u>CONCLUSION.....</u>	<u>81</u>
9.1	DISCUSSION	82
9.2	ETHICAL IMPLICATIONS.....	83
9.3	TECHNICAL LIMITATIONS.....	84
9.4	FUTURE WORK	84
10	<u>BIBLIOGRAPHY.....</u>	<u>86</u>
APX. A	<u>REGISTERS.....</u>	<u>96</u>
	GLOSSARY	96
	FIGURES.....	99
	TABLES	99
	FORMULAS	100
	TOOLS	101
APX. B	<u>IMAGE TRANSFORMATION FUNCTIONS.....</u>	<u>102</u>
APX. C	<u>ADDITIONAL MODELING CONSIDERATIONS</u>	<u>103</u>
	ABSOLUTE LOSS	103
	LATENT-STYLE	104
APX. D	<u>PARAMETER TRANSFORMATION FUNCTIONS</u>	<u>110</u>
APX. E	<u>CODEBASE.....</u>	<u>111</u>
	PIPELINE.....	111
APX. F	<u>DEPENDENCIES</u>	<u>112</u>
APX. G	<u>RESOURCE OWNERSHIP AND MEDIA PRESSURE HANDLING</u>	<u>114</u>
APX. H	<u>DOWNLOAD ONNX WEB DEPENDENCIES</u>	<u>116</u>

2 Introduction

How can we encourage people to spend less time online without forcing them to do so?

Problematic internet use (PIU) can be seen as an important concern in digital society. During the COVID-19 pandemic, one study found that 30.1% of children and adolescents and 21.3% of young adults met the criteria for PIU during the first lockdown [1]. Although these figures refer to an exceptional period, broader Swiss data also suggest that PIU has become increasingly relevant in recent years. A synthesis report by Fachverband Sucht and GREA, which examined developments in Switzerland, shows that PIU has increased in recent years [2]. According to Swiss Health Survey data cited in the report, the share of people aged 15 and older with PIU rose from 3.8% in 2017 to 6.8% in 2022. This corresponds to around 500'000 people in Switzerland. The increase was especially pronounced among younger age groups, with 22.2% of 15- to 24-year-olds and 11.7% of 25- to 34-year-olds affected in 2022. At the same time, adolescents' general online media use also continued to rise, with a median of around 3 hours and 14 minutes online on weekdays and almost 5 hours on weekend days.

Existing approaches, such as time limits, often intervene rather directly. Such mechanisms can be perceived as restrictive and may reduce user acceptance. This motivates approaches that support users in reducing online engagement without relying on direct enforcement. Literature suggests a link between elicited emotional responses and online engagement. Since visual properties such as color, brightness, contrast, or image content can likely affect emotional perception, emotional adaptation of images may be a possible direction for interventions.

Gebhardt et al. compile and synthesize several such findings in their work [3]. They contextualize their efforts within the broader goal of reducing engagement and counteracting PIU through non-coercive means. The authors develop three optimization methods using two key factors from psychology as guiding signals and evaluate whether they can alter perceived responses. Their results show that these edits can elicit distinct reactions. However, because these methods rely on iterative processes, their direct use in real-time or practical settings, such as actual social media feeds, remains limited.

This limitation motivates the present work. Through a proof of concept, we aim to show that translating the existing iterative methods to a learned model can provide a technical basis for more realistic application scenarios and future long-term analysis.

2.1 Goal

The objective of this work is to develop and evaluate a learning-based approach that builds upon optimization methods and demonstrates its applicability through proof of concept (PoC) on real social media feeds for the Android platform.

The overall intent is shown in Figure 1. The system starts from an original social media feed on an Android device, where image or video media elements are captured (1). These media elements are then processed by the proposed system, which extracts the relevant visual information and applies a learned emotionality-neutralizing filter in real time (2). The result is an adapted feed in which users are presented with visually less emotional content, while the general structure and usability of the feed remain in place (3). In this way, the approach aims to support digital well-being through non-coercive intervention.

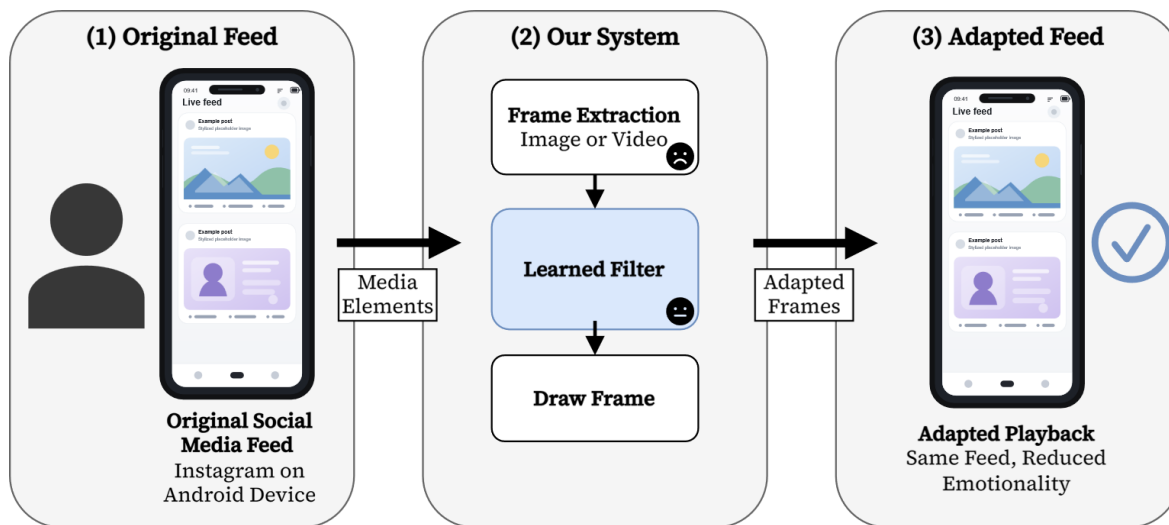


Figure 1: Goal Overview

2.2 Methodology

Initially, a literature review is conducted to identify related approaches and determine whether techniques can be adapted. Building on the findings, suitable strategies are applied to translate or extend the existing approaches. The model implementation proceeds iteratively, and the problem of manipulating frames is studied independently. The translated models are then compared and evaluated. Finally, a feasible system architecture for real-time manipulation on end user devices is designed and realized through the development of a functional prototype.

2.3 Scope

This project is carried out within the framework of applied sciences. Its primary objective is the application of engineering-based methods in the field of computer science. Accordingly, the work comprises both a theoretical component as well as a practical implementation part. Consequently, the focus of this thesis is not on fundamental research, but rather on the applied development of a practical solution.

The bachelor thesis is constrained to 12 ECTS, corresponding to approximately 720 hours of work. The work is to be evaluated according to the following topics: organization and execution, formal quality of the report, analysis, design, evaluation and technical implementation.

3 Background and Related Work

Our objective contains several subproblems to solve. These issues can be broken down into four key steps as shown in Figure 2. To manipulate media, the system must first be able to read and interpret the input data. If this is achieved, the data must be prepared and issued for inference. The model may then either predict some parameters or directly generate a new version of the frame. Finally, the newly transformed version must be overlayed on top of the original frame or replace it using alternative methods.

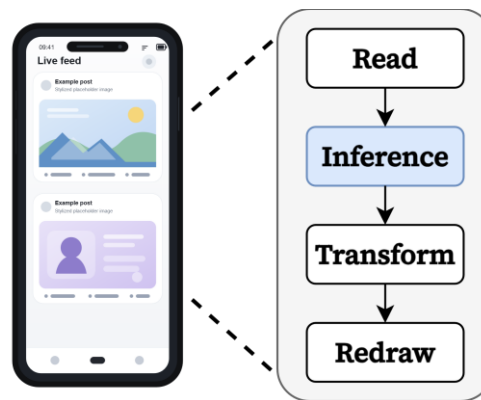


Figure 2: Problem Overview

This section aims to lay the theoretical foundation of each of these steps. First, the relevant background is introduced and the technical results of Gebhardt et al. [3] are summarized (Sec. 3.1). The focus then turns to inference, where current literature on manipulating low-level attributes to alter the emotionality of visual media with learned models is analyzed (Sec. 3.2, 3.3). In this context, situational methods for improving inference efficiency are also considered (Sec. 3.4). Finally, possible approaches for reading, transforming, and rendering frames are examined (Sec. 3.5).

3.1 Regressor-Guided Image Editing

Gebhardt et al. use two key concepts from psychology as the guiding optimization signals: *valence* and *arousal* [3]. These are defined within the *Circumplex Model of Affect* (CMA) [4], to describe emotionality. The first term describes the degree of pleasantness and the latter the level of intensity. The dynamics are summarized in Table 1. In general, higher arousal and polarized valence are linked to higher social media engagement behavior.

Gebhardt et al. alter image attributes across multiple levels to reach more neutral emotional points in this affect space [3]. These levels include *low-level attributes* such as color and light, *mid-level* structural and spatial properties as well as *high-level* semantic content. To this end, they developed three different regressor guided optimization methods including

- *the global optimization of emotion-related image attributes,*
- *the optimization in a latent style space and*
- *a diffusion-based method using classifier and classifier-free guidance.*

Furthermore, the authors conduct an image-rating and a social media study in an experimental setting. The images used in these are pre-generated, which makes it possible to compare the methods under controlled conditions. The results show that they can influence how users emotionally perceive images. This effect is especially visible for the diffusion-based method e.g. by decreasing nudity. In contrast, the global and the latent style-space optimization change the visual appearance but do not reliably alter the perceived emotionality.

To contextualize the motivation of this work, their methods are summarized in more detail. This helps illustrate why applying these in real-world scenarios remains difficult.

Dimension	Direction	Effect on Online Engagement Behavior	Sources
Valence	(+)	Higher participation and engagement	[5, 6, 7]
Valence	(-)	Drives the rapid spreading of false news	[8]
Arousal	(+)	Boosts engagement, virality, sharing	[6, 9, 10, 7]
		Promotes the spread of moral content	[11]
Arousal	(-)	Reduces spread of morally framed content	[11]
Valence			
Arousal	(+) arousal	Increases engagement levels	[3]
Valence	polarized valence		

Table 1: Effects of Valence and Arousal on Online Engagement [3]

3.1.1 Affect-Space Regressor

Gebhardt et al. use the two dimensions of the aforementioned CMA space to model an emotion regressor [3]:

$$R: [-1,1]^{C \times H \times W} \rightarrow [0,1]^2, \quad I \mapsto [v, a]^T$$

Equation 1:
 Emotion
 Regressor

The regressor, shown in Equation 1, maps RGB images $I \in [-1,1]^{C \times H \times W}$ to an emotion vector $e \in [0,1]^2$ where v and a denote valence and arousal, respectively. The regressor is a fine-tuned ResNet50 model pre-trained on ImageNet. They assemble a custom dataset containing 10 different emotional databases. For the diffusion approach the input interpretation for the regressor changes to u_t which is the middle layer output of a diffusion model. To enable more robust predictions for images of different sizes, the regressor includes the functionality to randomly crop an image and average the prediction.

3.1.2 Criteria

Their objectives are formulated around four key criteria:

- I. balancing valence
- II. reducing arousal
- III. preserving overall image quality
- IV. maintaining alignment with the original content

The methods pursue these criteria in different ways. The approaches all use the reference values $v' = 0.5$ and $a' = 0$ as guiding signals. The preservation and alignment objectives, however, are implemented differently depending on the approach. Within this work, these values are also referred to as the *emotional reference vector* e' .

3.1.3 Parametric Optimization Method

$$\arg \min_{p \in P} w_1 L_{CLIP}(I, T(I, p)) + w_2 \|[v', a']^T - R(T(I, p))\|$$

Equation 2: Parametric Method

The parametric approach shown in Equation 2 only edits low-level attributes. It optimizes the parameters p of a differentiable image transformation function T (Apx. B). Where R is the regressor from Equation 1, I is the image tensor and L_{CLIP} is defined as the cosine similarity between CLIP [12] embeddings as shown in Equation 3. The first two criteria are achieved by calculating the distance to the reference emotionality vector and the similarity in the embedding space is used to ensure the III and IV criteria.

$$L_{CLIP}(c, \hat{c}) = \frac{c \cdot \hat{c}}{\|c\| \|\hat{c}\|}, \text{ where } c = I \text{ and } \hat{c} = T(I, p)$$

Equation 3: CLIP Similarity

3.1.4 Latent Style Optimization Method

$$\arg \min_{s \in S} w_1 |c_0 - E_c(D(E_c(I), s))| + w_2 \|[v', a'] - R(D(E_c(I), s))\|$$

Equation 4: Latent Method

The latent approach shown in Equation 4 uses the MUNIT framework by Xun et al. [13], that demonstrates how images can be decomposed into a domain-invariant content and a domain-specific style code. A MUNIT model follows an encoder-decoder structure to learn these representations, with GAN-based training objective. Gebhardt et al. use NVIDIA's high-resolution implementation from the Imagenaire project¹ [3].

They optimize the latent style vector as shown in Equation 4. There E_c denotes the content encoder and D the decoder. The initial content and style vectors c_0, s_0 are given

¹ [NVlabs/imagenaire: NVIDIA's Deep Imagination Team's PyTorch Library](https://github.com/NVlabs/imagenaire)

by the encoders of the unedited image. To achieve criteria I and II they use the same method as in the previous approach in section 3.1.3, but for the III and IV criteria they use a content-consistency loss to the original image c_0 . Similar to the parametric approach, since only the style is optimized, the approach only modifies low-level attributes.

3.1.5 Regressor-Guided Diffusion Resampling (RGDR)

The last approach by Gebhardt et al. [3] uses a diffusion architecture to edit up to high-level image attributes. For sampling, they do the following:

1. Invert image into the noise vector
2. Apply null-text optimization
3. During denoising use two additional mechanisms:
 - a. Classifier guidance
 - b. Classifier free guidance (CFG)

In other terms, they retrieve the latent noise vector using an inverse deterministic scheduler. They then optimize only the null text conditional of the classifier-free guidance term. Then, a classifier guidance term, based on the distance between the reference emotion vector and the current emotion regression, is added. Finally, they incorporate a caption of the original image to preserve content alignment. This caption² is represented as the text embedding for the classifier free guidance term. They use the term dual-conditioned resampling for this.

In this approach the regressor R_u is defined on a different set of inputs. They extract new input data by running the forward diffusion process on the emotion dataset. There the noise vectors z at different timesteps t are run through the diffusion and then the middle layer outputs are extracted as inputs u_t for regressor.

3.1.6 Computational Feasibility

In this section, we discuss the computational feasibility of these approaches in the context of real-time social media adaptation. It is particularly relevant given that all three methods depend on a long-lasting iterative strategy. The first two use gradient-based optimization for finding stationary points, whereas the third approach builds on the diffusion denoising process.

Table 2 shows the inference characteristics of some of the components used. It is noticeable that this MUNIT implementation requires more computation despite having

² using GPT4 Vision 2024-05-01-Preview

a lower parameter count than the convolutional ResNet. This is likely related to its structure, in particular to the possibility that several convolutional operations are performed at relatively high spatial resolutions. In such cases, the same filters are applied across many output positions.

It should be noted that many factors can influence the final inference time. Factors may include model variants, resolutions, batch sizes, preprocessing, floating-point precision and even the runtime.

Model	Parameters (M)	GFLOP	Remarks	Sources
ResNet50	25.6	4	224×224	[14, 15]
MUNIT	15	77.3	256×256 , full forward translation, Gebhardt et al. use 1024×1024	[16]
Open-CLIP, ViT-B-32	151.28	8.82 (image) 5.96 (text)	224×224 , 77 tokens	[17]
SDXL Pipeline	6.6k	9.56k	parameter count for full pipeline with refiner, denoising 50 steps (only U-Net) measures 478k in total	[18, 19]

Table 2: Computational Inference Characteristics

For real-time usage, a latency of less than 16.667ms is assumed, corresponding to a frame rate of 60 frames per second. Using floating-point operations (FLOP) as proxy, a RTX4070 TI Super graphics card can produce about 44 TFLOPS [20] and a Samsung Galaxy s23 Ultra up to 2.1 TFLOPS [21]. This yields about 733.33 GFLOP per frame and 33.6 GFLOP per frame, respectively.

Using this rough estimate, it becomes clear that diffusion-based methods, such as SDXL, are currently likely not very practical. The approximate value reported in Table 2 only refers to a single denoising step and this would already exceed the estimated budget (9.56k/33.6 GFLOP). The true cost depends strongly on the number of steps, which is influenced heavily by the chosen sampling method, for example Euler or DPM++. Better samplers or models may reduce the iteration count. In addition, this GFLOP value does not reflect all operations of the full pipeline. Depending on the setup, further components are used, such as text encoding, decoding the latent vector into image space, or applying a refinement model.

Compared to diffusion, even the lighter MUNIT-based approach may face bottlenecks. While it requires fewer operations for full translation, about 77.3 GFLOP, the advantage decreases because of its embedding in an optimization procedure. In this latent-style approach, the initial encoding of the input can be reused, but each step still requires generating a candidate image and evaluating it through the network. The cost therefore is scaled with the number of iterations and with the resolution. Since convergence to a usable representation is not guaranteed, even lightweight architectures can become demanding.

Diffusion and GAN-based architectures may be additionally less attractive, since adaptations would be required for each subsequent video frame to preserve coherence. Even then, consistent temporal behavior may not be guaranteed [22].³ In contrast to the second and third approach, global image modifications may be more efficient, since they can be relatively cheap to compute depending on the architecture and may only need to be recalculated when relevant scene changes occur.⁴

As an initial proposition, we argue that the parametric approach could potentially be efficiently approximated through one-shot inference. This may make it more convenient for real applications, although likely at the cost of reduced output quality. In contrast, the diffusion process is inherently iterative and, at its current stage, remains probably too computationally demanding for the intended scenario. It is therefore not part of the objective of this work.

³As discussed later in 3.4.4, this limitation does not apply equally to all GAN-based approaches. Some demonstrate that localized edits can be performed efficiently.

⁴Alternative techniques or model adaptations could make more complex architectures computationally feasible, for example through quantized model variants, as briefly discussed later in 3.4.1.

3.2 Affective Image Adaptation

To the best of the conducted review, no studies were identified that explicitly aim to reduce arousal and neutralize the valence of images. Gebhardt et al. provide an extensive overview of relevant computational approaches [3]. Building on their work, this section summarizes and extends these findings, focusing in particular on learned models that manipulate low-level image attributes to change emotional attributes.

Most of the identified literature focuses on recent advances in deep learning [23, 24, 25, 26]. No optimization-based methods were identified in the sense defined by Gebhardt et al. [3]. A particularly notable development is the chronological shift in focus, from earlier approaches centered on emotional distribution classification and learning [27, 28], to GAN-based image editing and generation [23, 24, 25], and more recently to diffusion or multimodal transformer based image editing and generation frameworks [29, 30, 31].

The approaches are grouped conceptually according to their main mechanism, beginning with parametric models (Sec. 3.2.1), followed by latent space models (Sec. 3.2.2) and additional approaches (Sec. 3.2.3).

3.2.1 Parametric Models

Among the reviewed works, the only explicit parametric approach identified is presented by Afsheen et al. in *Automatic Image Transformation for Inducing Affect* [32]. Their method retrieves the k most similar images from an image-emotion database using a source image and target affect distributions as input. Based on the selected references, the source image is then modified through a predefined color transformation designed to induce the target distribution.

However, this approach appears less suitable for our scenario. It depends on a collection of reference data. This information could potentially be embedded within a model, although it is unclear how this would be achieved in practice. Otherwise, additional information stores as well as corresponding retrieval mechanisms would be required, which may introduce external dependencies. Moreover, the reference data would likely have to be adapted or extended to match the affect representation used in our setting. The method is also mainly based on color transfer, which limits the types of changes that can be expressed. In this regard, it does not fully match the implementation by Gebhardt et al. [3], which considers a wider range of parametric image transformations.

By considering a broader set of related problems, a more suitable approach can be identified. Mejjati et al. [33] use deep neural networks to predict parameters of differentiable global image transformations. Their goal is to guide attention toward or away from specific image regions by using a mask. They achieve this by presenting a *parametric generator*, which relies on an initial convolutional part, as well as a final dense part. The mask is concatenated before each convolutional block. They split the dense part into two branches to produce separate edits for the foreground and background. The parameters are used within a fixed sequence of non-trainable image transformations, while at last the system is trained adversarially. According to the authors, this enables inference at interactive rates while supporting arbitrary image sizes. They further state that the method can be applied to video data by estimating the parameters of the first frame and reusing them for the remaining ones. The parametric approach from Gebhardt et al. follows a similar principle [3]. This makes the approach highly relevant for our scenario. It separates the prediction from the actual transformation. This is attractive because the learned component may remain comparatively small. The architecture also appears easier to adapt due to its simple structure.

3.2.2 Latent Space Models

Turning to the latent style approaches, Goetschalckx et al. [23], in *GANalyze*, propose a generative approach. Here a latent noise vector z , an emotion class label y and a continuous control parameter α are used to synthesize images. By varying α , a specific image property can be varied in a controlled manner, enabling gradual and interpretable changes along the target attribute dimension. They achieve this by defining the relative loss shown in Equation 5.

$$\mathcal{L}(\theta) = \left(A(G(T_\theta(z, \alpha), y)) - (A(G(z, y)) + \alpha) \right)^2$$

Equation 5:
GANalyze Loss

In Equation 5, A is an assessor function, G is a generator, T is a *transformer function* that modifies a latent vector z by α and finally y is a class vector. It is important to note that the transformer function used here is unrelated to the transformer architecture commonly associated with attention mechanisms.

$$T_\theta(z, \alpha) = z + \alpha\theta$$

Equation 6:
GANalyze Transformer Function

The transformer function is shown in Equation 6. It moves by the scalar α in a learned direction in latent space. This raises the question of how the definition could be extended to a vector-valued α , which may be required in the present context. Overall, the

approach is flexible, since components from different GAN variants and regressors acting as assessors can be integrated into the framework relatively easily. In addition, the transformation can be learned in an unsupervised manner, since no paired examples are required.

Chanjong et al. [24], in *Emotional Landscaping*, condition a GAN on a latent noise vector together with continuous valence and arousal values to generate landscape images with varying emotional characteristics. Their model introduces an *Emotional Residual Unit* (ERU), a residual block inspired by LSTM and GRU mechanisms, and uses ERU-based feature differences in the loss computation. For our scenario, this approach is less relevant because it requires changes to a GAN's architecture and is tightly coupled to adversarial training with a discriminator and Wasserstein loss. This limits transferability to other architectures and may increase complexity, especially since training generally requires careful balancing of generator and discriminator.

Zhu et al. in *EGAN* [25], propose a GAN-based framework for image emotion transfer. The model disentangles images into a high-level content representation and a low-level emotion space. By extracting the emotional features from a target image and combining them with the content of a source image, the model enables emotion transfer while preserving the original content. This disentangled latent representation allows the modification of emotional characteristics independently of content. Gebhardt et al. adopt the idea and apply similar steps for disentanglement within the MUNIT framework [3].

3.2.3 Additional Approaches

Finally, Weng et al., in *Affective Image Filter* (AIF) [26], introduce a multimodal approach that takes as input an image together with an emotional text description. Their method relies on a transformer-based architecture to jointly encode visual and textual features, followed by a CNN-based decoder that generates the image. The resulting output is adjusted such that its emotional characteristics are aligned with the affective content expressed in the input text, while maintaining the semantics of the original image. While this approach provides a flexible mechanism, it introduces requirements that are not central to our problem setting. In particular, the method relies on additional textual input to specify the desired affective characteristics. Incorporating a multimodal pipeline may increase the complexity of the system. Moreover, the transformer-based architecture used for joint encoding may entail substantial computational and memory costs. For example, standard self-attention has quadratic time and memory complexity with respect to the input sequence length.

3.3 Amortized Optimization

When optimization-based approaches are transferred into a learned setting, amortized optimization provides a particularly relevant perspective. The term amortized optimization can be adopted to refer to the general class of methods that approximate traditional optimization procedures, as described by Amos [34]. In amortized optimization, the goal is not to solve each instance of an optimization problem from scratch but to learn a model that either predicts the final solution directly or accelerates the optimization process across similar problem instances. This concept appears across many domains, though the specific term amortized optimization is not used consistently in literature.

Fundamentally, amortized optimization shifts the bulk of computation into a training phase. During inference, a learned predictor produces either a full solution estimate or an improved initialization that a traditional optimizer can refine. In other words:

- a *fully amortized model* maps problem contexts directly to solutions without further access to the objective,
- whereas *semi-amortized models* still interact with the objective function during inference. This can be achieved, for example, by computing an approximate step into the solution's direction.

Within this framework there is also an important distinction in how the model is trained:

- One training approach minimizes *regression-style loss*, encouraging the predicted outputs to match known optimal solutions.
- The other uses an *objective-based loss*, in which the learned model is evaluated by directly substituting its predictions into the original objective function. This requires that the Jacobian of the objective can be computed.

Although amortized optimization offers a relevant conceptual direction, it was not pursued further in this work. As discussed in Apx. C, its application would require additional training overhead, likely involving the pre-generation of optimization-based results to serve as targets. This would introduce considerable uncertainty regarding the expected performance, the required amount of data, and the extent of data preparation needed to learn the mapping reliably.

3.4 Model Efficiency Considerations

Model efficiency can be optimized along several dimensions. This section introduces common architecture agnostic techniques for reducing model size and computational complexity.

3.4.1 Quantization

In quantization [35] the objective is to reduce computation and memory cost. This can be done by lowering the precision of weights within a network. The fundamental idea is to transform a floating-point representation of real value x into an integer q . During inference the network then uses only integer arithmetic to compute outputs. This has the benefits of reducing the model size e.g. up to four times when using 32-bit to 8-bit. The usage of integer arithmetic may be implemented more efficiently than floating-point arithmetic on suitable hardware. Jacob et al. show that integer-only inference can improve the trade-off between accuracy and on-device latency, particularly on mobile CPUs [36]. There are different approaches to quantization. Post-training static quantization (PTQ) converts an already trained floating-point model into a quantized model. Quantization-aware training (QAT), by contrast, incorporates the effects of quantization during training, allowing the model to adapt to the reduced numerical precision. Both PTQ and QAT are supported by PyTorch [37].

3.4.2 Pruning

Another method is pruning. In pruning, parameters are removed from a network [38]. This technique can be applied in an over-parametrized setting like commonly seen in LLMs, where redundant parameters can often be removed with limited impact on performance. Prior work has shown that pruning can substantially reduce the number of parameters or connections while maintaining comparable accuracy [39]. In PyTorch, similar methods can be implemented, for example, using utilities such as `torch.nn.utils.prune` [40].

3.4.3 Knowledge Distillation

A third well-known technique is knowledge distillation. The idea is to train a smaller student model to mimic a larger teacher model. For example, Hinton et al. [41] demonstrate that the soft probability outputs of a large network carry informative signals about class relationships, beyond the information contained in one-hot encoded labels alone. This can be used to compress large models into smaller ones or to transfer ensemble knowledge into a single model.

3.4.4 Distillation Application using Real-Time Filters

Google researchers [42] present a production-oriented application of this technique for real-time YouTube Shorts filters in resource-constrained mobile environments. They set the goal of building real-time face filters that can be inferred on mobile devices. For this purpose, they use a big teacher model, in their case StyleGAN2, that is pretrained for facial effects. For the student model they use a U-Net architecture with MobileNet models as backbone. They then distill knowledge from the teacher output pairs. They employ different loss terms to ensure the students' output is not simply numerically accurate but also visually consistent and finally use neural architecture search (NAS) to find a working model architecture.

3.5 Reading and Transforming Frames

As a first technical basis for this project, the Institute for Interactive Informatics I3 conducted an initial feasibility test. In this test, they demonstrated that frames of images and videos of a website can be altered using JavaScript. The validated approach is designed to operate on Google Chrome [43]. This section examines whether the same general idea can be transferred to an Android setting.

Browser	Plugins Possible	Source
Chrome	No	[44]
Mozilla Firefox	Partially (WebExtensions API, approved extensions are installable)	[45]
Microsoft Edge	No	
Chromium Based Alternative: Kiwi	Yes	[46]

Table 3: Browser Plugins on Android

As an initial step, it was assessed whether plugins can be implemented in Android browsers. The overall compatibility is shown in Table 3. It remains limited and thus may be impractical. A fully supported solution may require reliance on alternative browsers such as *Kiwi*⁵. Consequently, alternative ways of achieving similar functionality are also investigated. Three potential approaches are identified, namely the *WebView Approach*, the *Screen Approach*, and the *Capture Approach*. These approaches are not equivalent substitutes, because each assumes a different level of access to a media pipeline. In this context, the term media pipeline is used to refer to the sequence of technical processing

⁵ [kiwibrowser/src.next](https://github.com/kiwibrowser/src.next): Source-code for Kiwi Next, a Kiwi Browser auto-rebased with latest Chromium

steps through which the content passes, from being loaded, rendered, and, where applicable, captured for further processing.

3.5.1 WebView

The first approach involves the use of the Android *WebView* API within the Android SDK [47]. This API enables developers to render web content directly within the application's user interface. By leveraging this, JavaScript code can be dynamically injected into the loaded pages. The approach therefore consists of three main components, namely the instantiation of a *WebView*, the loading of web content, and the injection of JavaScript code for manipulation.

```
1  webViewClient = object : WebViewClient() {
2      override fun onPageFinished(view: WebView?, url: String?) {
3          injectJavaScript(this@apply)
4      }
5  }
6  private fun injectJavaScript(webView: WebView) {
7      val js = webView.context.assets.open("payload.js")
8          .bufferedReader().use { it.readText() }
9      webView.evaluateJavascript(js, null)
10 }
```

Code 1: Injecting JavaScript into Webpage (Kotlin)

The method of injecting JavaScript is shown in Code 1 above. There, the view is first instantiated, and JavaScript is then injected at a specific callback. Within this execution environment, media elements such as videos and images can be detected. The JavaScript *MutationObserver* interface can be used to identify dynamically added content. Whenever a `<video>` element is identified, the script may attach a manipulation pipeline to the respective object; this is shown in Code 2.

```
1  const domObserver = new MutationObserver(muts => {
2      muts.forEach(m => {
3          m.addedNodes.forEach(node => {
4              if (!(node instanceof Element)) return;
5              if (node.tagName==="VIDEO") modifyVideo(node);
6          });
7      });
8  });
9  domObserver.observe(
10     document.body,{ childList:true, subtree:true }
11 );
```

Code 2: Observing Video Element Changes in DOM (Js)

The frames of video elements can be read by using different methods as shown in Code 3:

1. The first method uses the Canvas API. Mozilla Developer Network (MDN) states that the *drawImage* method can draw the current frame of a `<video>`-element onto a canvas using a 2D rendering context. This is simple and widely supported [48]. However, when *drawImage* is called inside a *requestAnimationFrame*-Callback rendering loop, the sampling is driven by the browser's rendering/display refresh cycle rather than by video-frame presentation, so it is not guaranteed to align precisely with every frame [49, 50]. Alternatively, one may use the *requestVideoFrame*-Callback. It is triggered exactly when a new video frame is available, enabling more accurate timing [51, 52].
2. The second method extracts a video track from a stream object and processes individual frames through a WebCodes API reader, providing fine-grained control at the cost of higher complexity [53].

```
1 // First Method
2 const ctx = canvas.getContext("2d");
3 ctx.drawImage(srcVideo, 0, 0);
4 srcVideo.addEventListener("play", () => {
5   requestAnimationFrame(//call draw here)
6 });
7 srcVideo.requestVideoFrameCallback((now, metadata) => { // or here });
8 const pixels = ctx.getImageData(0, 0, canvas.width, canvas.height);
9 // Second Method
10 const track = srcVideo.captureStream().getVideoTracks()[0];
11 const processor = new MediaStreamTrackProcessor({ track });
12 const reader = processor.readable.getReader();
13 while (true) {
14   const { value: frame } = await reader.read();
15 }
```

Code 3: Reading Frames from Video Element (Js)

For this project's use case, option 1 bound to the display repaint loop refresh rate may be sufficient because the adaptation only needs to take place for frames that are actually displayed to the user. Option 2 provides better timing information and synchronization metadata, but other than that it may not add much benefit as it moves the processing into a more complex stream pipeline.

The computation may return different kinds of data types, and the adapted image can further be cached within an offscreen canvas. This type of canvas decouples it from the DOM and is transferable between contexts allowing the use of worker threads. After the

processing is completed, the processed result can be rendered back onto a visible frame, for example, by drawing it onto a visible target canvas this is shown in Code 4.

```
1 const targetCtx = targetCanvas.getContext("2d");  
2 targetCtx.drawImage(processingCanvas, 0, 0);
```

Code 4: Example Video Frame Drawing

The canvas may be positioned directly on top of the video element. This creates the visual impression that the edit is applied seamlessly.

The final pipeline may look like this:

1. read the current video frame into an intermediate/working canvas,
2. process the pixel data or apply a filter,
3. draw the processed result onto a visible target canvas,
4. place that visible canvas over the video element.

Since the modified canvas is the only output that should be visible, the original video element can be hidden to avoid rendering both. The video element should nevertheless remain active, as it is still used as the frame source for this processing step.

3.5.2 Screen Capturing

The next approach involves capturing the entire screen using the Android *MediaProjection* service [54]. With this method, frames can be streamed directly into a virtual display for processing. This approach requires explicit screen capture permission from the user and additional permissions to enable overlay rendering. Once captured, the frames can be processed, and the output may then be rendered onto a full-screen overlay window. This overlay may remain transparent to user interactions and allow underlying input events to pass through. However, this approach introduces a significant limitation. Since the screen is captured, transformations are applied globally unless additional processing is performed. To selectively manipulate only relevant content, it is necessary to first detect their spatial boundaries and restrict processing accordingly.

3.5.3 Capture and Render

The final approach involves capturing videos by directly accessing the underlying media source. This can be achieved, for example, by identifying and retrieving the corresponding resource URLs, which media are then fetched by the application. This download was successfully validated for Instagram based Reel URLs. Once (partially-) re-

trieved, the app may load the video into a player or decoder, such as *ExoPlayer* in combination with MediaCodec [55]. The stream may be decoded into frames, which are then routed to a target.

However, this approach also raises additional design questions that are not addressed by the mechanism alone. In particular, the application would need to define how the retrieved media is presented to the user, and how user interactions are handled. Since the mechanism does not directly augment an existing interface, the app would still need to provide an environment. This could be implemented through a custom view, but it may also be possible to combine this approach with the WebView-based approach described above.

4 Requirements

This section defines the requirements of the application. The functional requirements are illustrated in Figure 3. The PoC includes three functional requirements, of which only one, UC02, is mandatory. This requirement represents the primary objective as defined in the project goals.

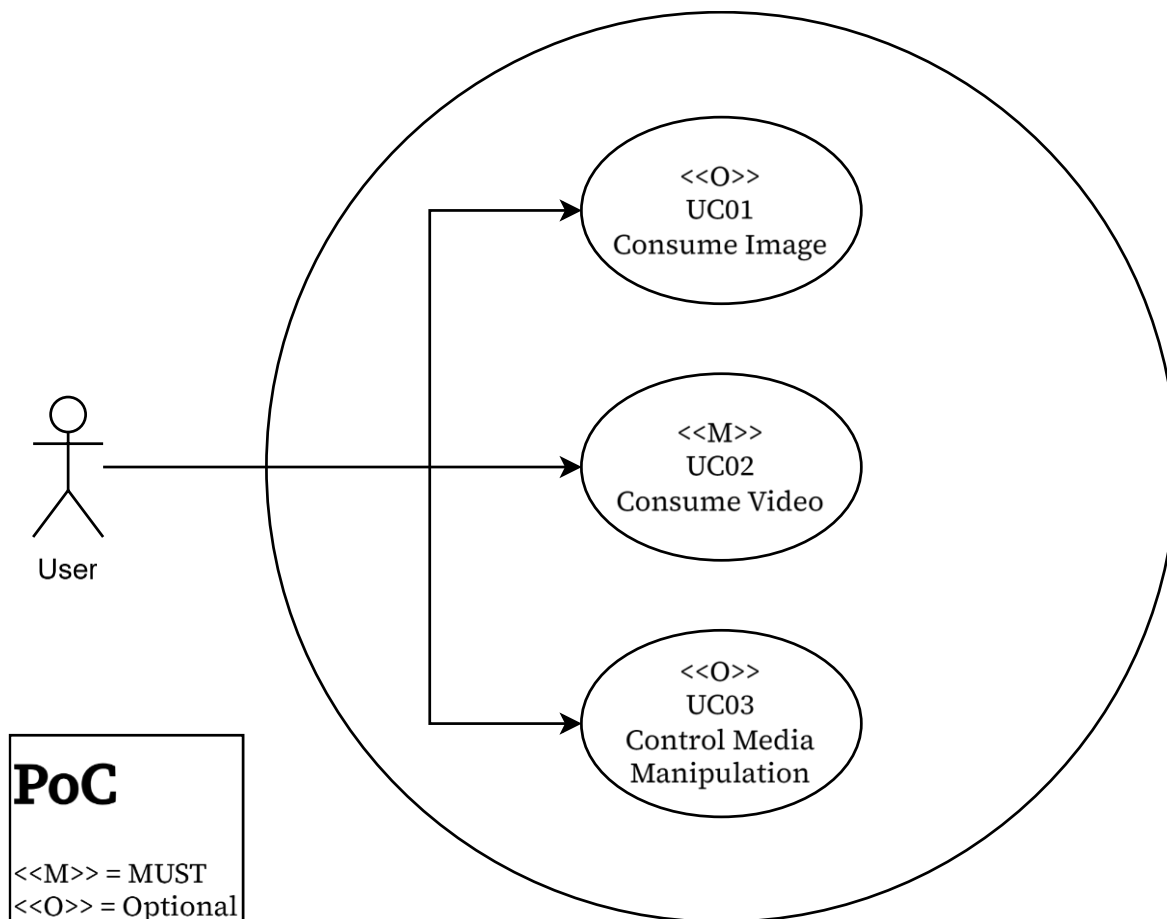


Figure 3: Use-Case Diagram

4.1 Functional Requirements

ID	UC01
Name	Consume Image
Use Case	As a user I want to consume social media images with more neutral emotional attributes without coercive restrictions.
Type	Optional
Actor	User
Main Success Scenario	1) The user browses specific social media website. 2) The user can observe the image on the feed. It is non-coercively switched to a target image which has a more neutral emotionality score as given by the regressor.
Alternative Scenario	In 2 the user can also open the specific view of the image. Here the same logic applies, and it's switched to an optimized image.
Additional Specification	Social Media Platform: Instagram

Table 4: UC01

ID	UC02
Name	Consume Video
Use Case	As a user I want to consume social media videos with more neutral emotional attributes without coercive restrictions.
Type	MUST
Actor	User
Main Success Scenario	1) The User browses specific social media website. 2) The user can observe the video on the feed. Its frames, or the full video, are non-coercively switched to the target frames, which have a more neutral emotionality score as given by the regressor.
Alternative Scenario	In 2 we define the following optional goal. If a user switches to an individual or even full screen view the same logic applies.
Additional Specification	Social Media Platform: Instagram

Table 5: UC02

ID	UC03
Name	Control Media Manipulation
Use Case	As a user I want to either enable or disable the manipulation of media
Type	OPTIONAL
Actor	User
Main Success Scenario	1) The User browses specific social media website. 2) The user can click on a button to enable or disable the manipulation of frames.
Alternative Scenario	-
Additional Specification	Social Media Platform: Instagram

Table 6: UC03

4.2 Non-Functional Requirements

ID	Description
NFR-1	It should run on an Android device Test Device: Samsung Galaxy S23 (Android 15)
NFR-2	The transformation should run on at least 1 social media platform preferably Instagram
NFR-3	The image/video transformation should happen (near-)real-time to be meaningfully usable. - Minimum: 33ms (30FPS) - Optimal: 16ms (60FPS) - Best: 8ms Test Device: Samsung Galaxy S23
NFR-4	A control element to enable/disable the filter should not cover any meaningful area of the screen
NFR-5	The application shall be an installable Android App (APK)

Table 7: NFR

4.3 Domain Model

As illustrated in Figure 4, the problem domain contains only a few key objects. A social media platform includes multiple image and video elements, whereby each video element is further composed of several frames. The objective of the model is to consume this visual input and generate a manipulated version that alters the conveyed emotionality. There may be multiple suitable models that achieve the same objective.

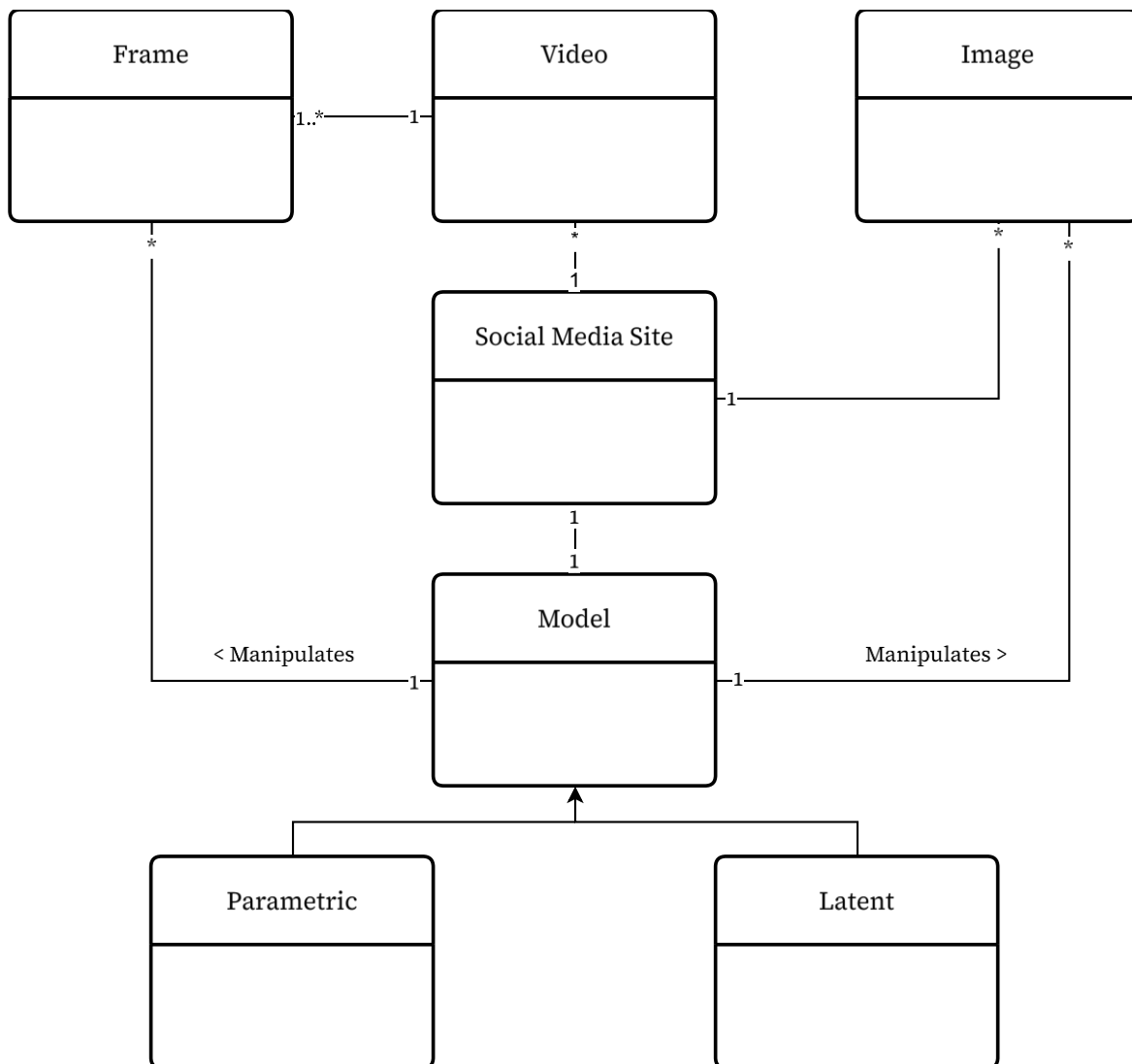


Figure 4: Domain Model

5 Design

In this chapter, the relevant design decisions are described. First, architecturally relevant design considerations are discussed, including the selected frame manipulation approach, model runtime, memory locality, and the use of global image transformations (Sec. 5.1). Based on these considerations, the resulting architecture is presented (Sec. 5.2).

5.1 Design Considerations

The design of the proof of concept is influenced by several constraints that result from the Android environment, the selected manipulation target, and the need to process content at runtime. Since the goal of the project is not only to demonstrate technical feasibility, but also to identify a realistic implementation path, the following considerations focus on trade-offs.

The first decision concerns the point at which frames are accessed and manipulated (Sec. 5.1.1). As discussed previously in chapter 3.5, this can be approached at different stages of the media pipeline.

A second consideration concerns the runtime used for executing machine learning models (Sec. 5.1.2). Different runtimes offer different levels of support, compatibility, and integration complexity. Since the proof of concept is intended for Android while still aiming for a certain degree of transferability, the runtime decision affects both the implementation effort and the long-term flexibility of the system.

Closely related to the runtime decision is the question of memory locality (Sec. 5.1.3). In this context, memory locality refers to where image data is stored and processed during the pipeline, for example in JavaScript, native Android code or GPU memory. This decision is relevant because unnecessary transfers may reduce real-time performance.

Finally, the design also considers the use of global image transformations (Sec. 5.1.4).

5.1.1 Frame Extraction Approach

The three approaches introduced in Chapter 3.5 are evaluated on the basis of the following criteria:

- I. technical feasibility,
- II. platform coverage,
- III. performance,
- IV. permissions and security,
- V. user experience (UX),
- VI. and maintainability.

Regarding *technical feasibility*, the WebView Approach relies on Android- and browser-native mechanisms, which may reduce complexity compared to the Screen Approach. The Capture Approach introduces uncertainty, as reliable media extraction may not always be achievable and could require further investigation.

For *platform coverage*, two dimensions must be distinguished, the runtime platform of the PoC and the target platform of the media that is being manipulated, for example Instagram or TikTok. On the runtime side, the WebView solution benefits from native components and could potentially be adapted to other environments such as iOS using comparable technologies [56]. In contrast, the Screen Approach depends strongly on platform-specific Android APIs and may therefore be harder to transfer. The Capture Approach may be less tightly coupled to a specific runtime environment depending on the implementation.

Concerning *target platforms*, the WebView Approach operates on the DOM level and may therefore be applied across various social media platforms, as long as media elements remain accessible within the page. Screen capturing is largely independent of the underlying platform and may work as long as media remains visible on the screen. The Capture Approach, however, depends on how the underlying media source can be identified, retrieved, and decoded. Since these mechanisms may vary significantly between target platforms, their applicability may be limited.

In terms of *performance*, the WebView Approach may be advantageous because it operates within the same environment in which the content is loaded and rendered. This can reduce the need for additional processing steps, like full-screen processing. In addition, the web context provides several potentially useful interfaces for media processing, including DOM access, Canvas APIs, WebGL, and related browser APIs. Screen capturing may introduce additional overhead due to full-screen processing, while Capture may require additional decoding and rendering steps.

From a *permission and security* perspective, the WebView Approach avoids intrusive permissions such as screen recording or overlay access but may introduce *Cross-Origin Resource Sharing (CORS)* issues. Screen capturing requires such permissions, while Capture introduces uncertainties related to external media access.

With respect to *UX*, the WebView solution allows modifications directly within DOM content, probably resulting in a more cohesive interaction. Screen capturing may interfere with the workflow due to required permissions and overlays whereas in the Capture Approach this is dependent on the implementation and thus unclear. The system would need to provide a UI for presenting the retrieved media and handling user interaction or be combined with a WebView-based interface in a hybrid solution.

Finally, regarding *maintainability*, the WebView Approach may be affected by changes in webpage structures but may remain relatively stable due to available standardized technologies. Screen capturing adds complexity through possible system-level dependencies, while Capture is particularly sensitive to external platform changes such as modifications in media delivery mechanisms.

Overall, the WebView-based Approach offers the most balanced trade-off in these regards and is therefore selected for proof of concept.

5.1.2 Model Runtime

Different machine learning runtimes can be used to execute models. For web applications, common options include *TensorFlow.js*⁶, Open Neural Network Exchange (ONNX) Web⁷, and *MediaPipe*⁸. On Android, typical choices include *TensorFlow Lite / LiteRT*⁹, *ONNX Runtime Android*¹⁰, *MediaPipe* and *ExecuTorch*¹¹. These runtimes differ in supported platforms, formats, hardware acceleration and integration effort.

For this proof of concept, we restrict the choice to ONNX-based runtimes. The main reason is ONNX's strong cross-platform compatibility, which is particularly relevant because the selected WebView-based approach may involve both native Android components and web-based execution contexts. ONNX offers flexibility in this regard, as it can be used either through ONNX Runtime Android in a native implementation or through ONNX Runtime Web in a web-based implementation. Regarding the processors, the web

⁶ [TensorFlow.js | Machine Learning for JavaScript Developers](#)

⁷ [Web | onnxruntime](#)

⁸ <https://ai.google.dev/edge/mediapipe/solutions/tasks>

⁹ [LiteRT: High-Performance On-Device Machine Learning Framework | Google AI Edge | Google AI for Developers](#)

¹⁰ [Build for Android | onnxruntime](#)

¹¹ <https://docs.pytorch.org/executorch/stable/index.html>

runtime documentation states that all operators are supported by WASM, whereas WebGL, WebGPU and WebNN currently support only a subset of operators [57]. This aspect is important in the context of the choice of global transformations as further discussed in 5.1.4. Unlike Web, ONNX Android uses native execution providers. The default CPU execution provider supports the full operator set, while Android-specific accelerators such as NNAPI and XNNPACK support only a subset [58].

Another important factor is that the reference work uses PyTorch [3]. ONNX is well suited for this setting, since PyTorch models can often be exported to ONNX without requiring a complete rewrite when targeting different environments. This may reduce implementation risk and may keep the PoC closer to reference implementations.

5.1.3 Memory Locality

However, only the choice of runtime is not sufficient to determine the architecture in this context. The inference location must be analyzed, especially in the context of the complete processing pipeline. In real-time manipulation, frames are continuously processed. The model is therefore not an isolated component, but one stage in a larger data path. A frame may originate from a camera buffer, video decoder, browser canvas or GPU texture, and after inference the result must be passed on to other components.

This matters because memory access and data movement are general performance bottlenecks in computer architecture. The *Memory Wall* describes the gap between processor speed and memory access speed, while work on efficient DNN processing shows that data movement can dominate throughput and energy consumption in neural-network workloads [59].

Therefore, a suitable inference location may not necessarily be the one with the lowest inference latency. A GPU may execute a model faster, but benefits are reduced if every frame must be copied and the output then must be copied back again. Similarly, CPU inference may be easier to integrate, but it may be inefficient if the surrounding pipeline already uses GPU-backed data or if the model is computationally demanding. Backend or hybrid inference [60] can provide more computing capacity, but it requires data to be transmitted over the network, which may introduce latency and privacy concerns. For this reason, a preferred architecture may be one that keeps data close to where it is produced and consumed, minimizing copies between domains.

In a web-based setup, the decision is further complicated by the browser abstraction. A web application can choose the APIs through which a video frame is accessed, but it usually cannot directly control the browser's internal memory representation or the exact transfers performed between APIs.

This is especially relevant when frames are passed from DOM media elements to processing APIs [61, 62]. For example, if the frame is needed for CPU-accessible inference, APIs such as *VideoFrame.copyTo()* method, from the WebCodecs API, copies contents into an *ArrayBuffer*, which makes the transfer explicit but may also introduce a memory copy [63]. The WebCodecs specification makes clear that user agents may allocate media resources in CPU memory, GPU memory or dedicated codec hardware, meaning that the exact memory location is not fully visible to the web application [64]. By contrast, more GPU-oriented APIs such as WebGPU's *importExternalTexture*¹² can expose a frame as a texture for rendering operations, which may avoid a full CPU readback [65, 66].

Using canvases with the Canvas API, we may expect the data to reside mostly on the GPU for WebCodec API video frame objects coming from video elements. However, the developer only has limited control over this behavior. One relevant option is the *willReadFrequently* setting. For this option, the HTML standard states:

“The *CanvasSettings* object also has a *will read frequently* boolean. When a *CanvasSettings* objects *will read frequently* is true, the user agent may optimize the canvas for readback operations” [67].

This again illustrates the difficulty of making precise assumptions about memory handling in web-based media processing.

A split architecture can add further movement costs. If processing is divided between the main thread, workers, iframes or separate components, data must be communicated between contexts. Transferable objects can avoid copying for resources such as *ArrayBuffer* or *ImageBitmap*, but ownership is moved, and the original object becomes unusable [68]. Shared memory mechanisms can reduce copying but introduce synchronization and security constraints [69]. If processing is moved into native or operating-system-level components, platform-specific mechanisms may be required. On Android, for example, *AHardwareBuffer* and *HardwareBuffer* can provide zero-copy sharing across processes by giving another process a shared view of the same memory region, but this requires native integration and buffer management [70, 71].

In the context of this work, these observations support a simple choice for proof of concept. The aim is not to prematurely optimize. Based on the considerations described above, a CPU/WASM-based ONNX Runtime Web setup is used as the initial baseline. Furthermore, the 2D Canvas API with the *willReadFrequently* flag shall be used. The baseline depends on the assumption that the model is sufficiently efficient, or can be optimized enough, to run on the CPU within the required limits. If this is not the case,

¹² not a baseline feature, because it does not work in some of the most used browsers

further steps, as discussed in Section 3.4, may become necessary. Nevertheless, independent of these decisions, the general data-movement problem remains relevant.

5.1.4 Global Transformations

For proof of concept, global image transformations (Apx. B) can in principle be applied at different points in the processing pipeline. The found options include:

- CSS filters,
- JavaScript or WebAssembly-based CPU processing,
- shader-based GPU processing,
- or integration directly into the model pipeline.

The most practical option would seem to be the application directly inside the model. If they can be exported or reimplemented as ONNX-compatible operations, this could reduce architectural complexity (Section 5.1.2) and may also make validation easier. Besides, the dependence on support, it also requires the computational graph to behave equivalently and with sufficient performance. For example, Gebhardt et al. uses a contrast formulation that uses the image average, that may impact inference time heavily [3].

A second simple consideration is the transformation using CSS. This is likely the simplest approach for those that are already supplied by the browser, such as brightness, contrast, saturation and blur. However, CSS filters provide only a limited set of predefined operations. While some are supported, others are not. No suitable implementation for tone and color curves was identified. Even for the effects that can be found, the operations may not match the original definitions. Therefore, CSS transformations seem practical for quick approximation but are not suitable for faithful reproduction.

A third option is shader-based processing, for example through WebGL2 or WebGPU. This approach is more complex, but it provides explicit control. This would require a separate implementation of the logic and synchronization with the inference. Further it introduces locality considerations that may be made partly independent of inference by splitting the source data into different views for rendering and inference. If the frame is being handled by a GPU-backed representation, we may expect shader-based processing to be efficient.

The fourth option is JavaScript-based CPU processing. This approach may be the easiest to understand and debug, since the transformations can be implemented directly on arrays. However, it is likely to be the least attractive option for real-time processing. Each frame would have to be made available in readable memory, processed pixel by

pixel, and then passed back. This may introduce overhead, especially at higher resolutions or frame rates. WebAssembly could improve performance compared, for example through explicit use of Single Instruction Multiple Data (SIMD). However, this could also increase implementation complexity, as it would likely have to be implemented in another language.

Based on these considerations, CSS filters appear unsuitable as the primary approach, because they lack the support for required operations. For proof of concept, the most relevant options are therefore direct integration into the model, shader-based processing, followed by JavaScript- or WebAssembly-based processing. Direct model integration is preferred if supported by ONNX operators. If unfeasible, shader-based processing provides a more flexible yet complex alternative, while JavaScript- or WebAssembly-based processing remains as a possible slow fallback.

5.2 Architecture

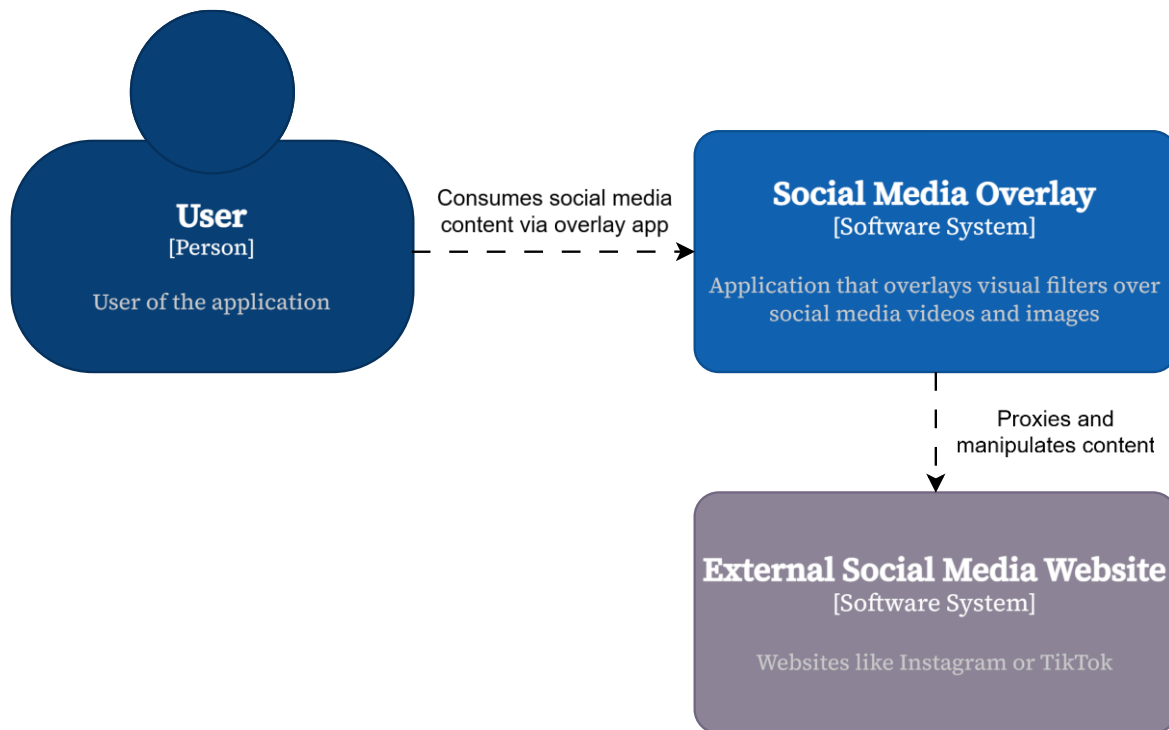


Figure 5: C4 System-Context Diagram

Using the *C4* framework [72], the container diagram is illustrated in Figure 5. The structure is kept simple, and interfaces with external systems are minimized. As a result, the software system comprises only a single logical container. The system further interacts with an external dependency via HTTP. This choice constrains the model to operate closer to the fetched data, thereby possibly reducing complexity.

The C4 components are shown in Figure 6. The container primarily consists of a *Web View Component*, which serves as the central entry point. This component proxies the website's data and injects the *Overlay System Component* implemented using common web technologies such as JavaScript. Within this web context, the *Machine Learning Runtime Component* is executed. The overlay system is responsible for reading the data, forwarding it to the machine learning runtime, and subsequently overlaying the processed output. For additional assets, a separate *Asset Loader Component* (see Section 7.1) is used. This component loads files from the filesystem and provides them to the Overlay System through the WebView within the web context.

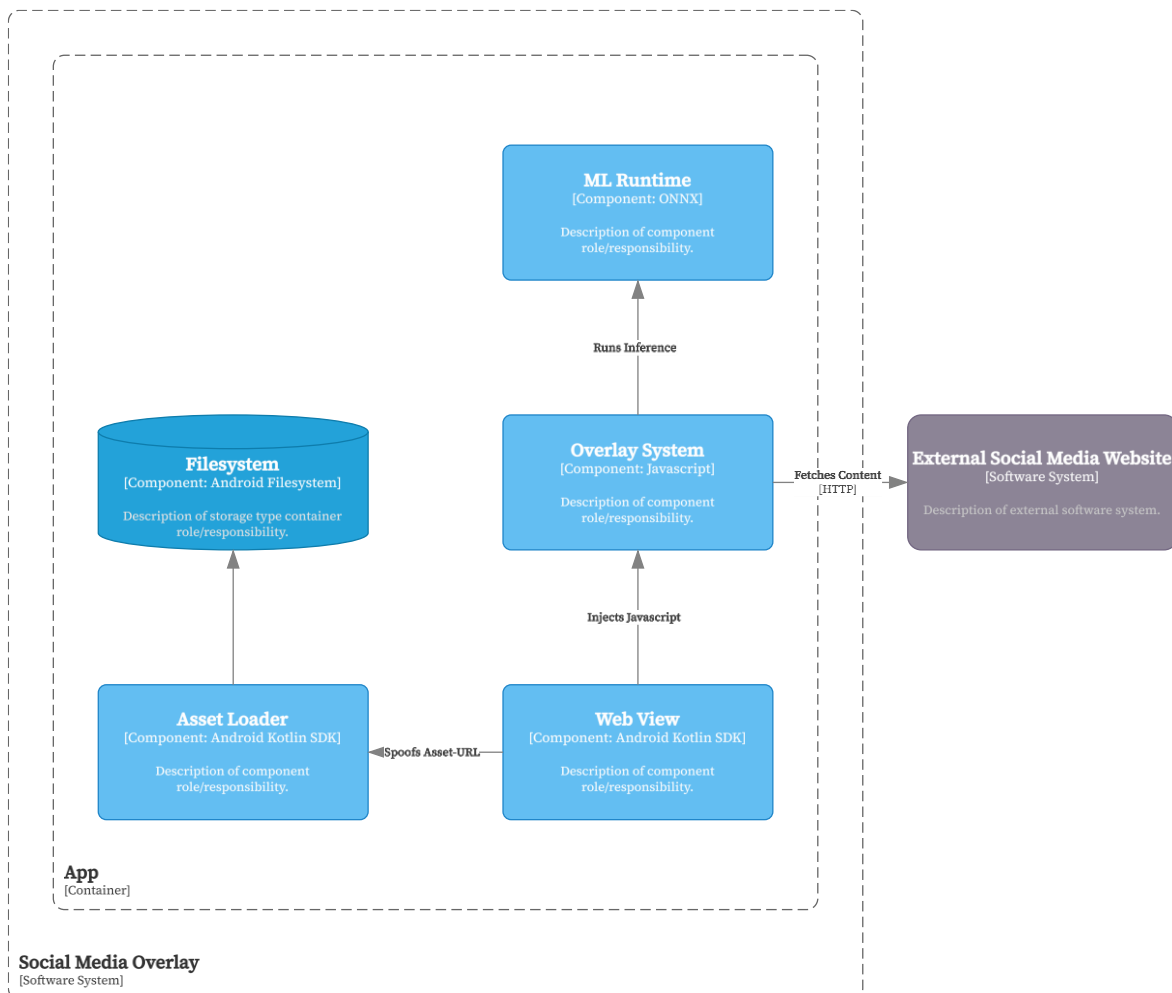


Figure 6: C4 Component Diagram

5.2.1 Architecturally Relevant Decisions (ARD)

In the context of the frame extraction and manipulation method (Section 5.1.1)
facing different possible alternatives
we decided to use the WebView Approach
and neglected the Screen or Media Capture Approach
accepting a simpler architecture with possibly fewer requirements but possible browser overhead.

In the context of model inference runtime and inference locality (Sec. 5.1.2, 5.1.3)
facing possible runtime alternatives and different inference locations like CPU, GPU or other accelerators
we decided to use ONNX Runtime Web (CPU WASM)
and neglected TensorFlow, MediaPipe, ExecuTorch or additional android native components
accepting simpler architecture but possibly slower inference latency.

In the context of C4 container split (Section 5.2)
facing concerns about increased complexity and higher latency
we decided on using a single container architecture (as in runtime boundary¹³)
and neglected hybrid or backend computation or additional Android execution boundaries
accepting constraints on model inference by the web browser context as well as the web ML runtime.

In the context of ML runtime dependencies (Section 5.1.2)
facing CORS issues within a WebView (Section 7.1)
we decided to implement an additional Asset Loader Component

¹³ [Container | C4 model](#)

<i>and neglected</i>	hybrid or backend computation and static dependency encodings like base64
<i>accepting</i>	an additional architectural component.
<i>In the context of facing</i>	Python dependency and runtime management (Section 7.2) the need for a reproducible setup that works without root privileges on the training cluster
<i>we decided</i>	to use UV
<i>and neglected</i>	Pip, Conda and Apptainer
<i>accepting</i>	that some widely used or more isolated environment approaches are not considered further, in exchange for a simpler, rootless workflow with lower development complexity.
<i>In the context of facing</i>	global image transformation (Sections 5.1.4 and 7.2) the need to reproduce transformations for real-time video
<i>we decided</i>	GLSL fragment shaders with WebGL2
<i>and neglected</i>	CSS filters, JavaScript-based CPU processing, CPU WebAssembly processing, direct in-model transformations, and WebGPU shaders
<i>accepting</i>	a separate shader implementation.

6 Method

This chapter describes the methodological basis for the modelling decisions. It introduces the modelling approach and its justifications (Sec. 6.1). The chapter then discusses adjustments that were made during development based on observed qualitative behavior (Sec. 6.1.1 to 6.1.3). In addition, the training and tuning procedure is outlined (Sec. 6.1.4). Finally, the choice of the control signal is described, as it determines how the intended manipulation is represented within the affect-space (Sec. 6.2).

An absolute formulation, for example through amortized prediction (Sec. 3.3) or distillation (Sec. 3.4.4), may require the generation of suitable training data and would introduce additional uncertainty regarding the amount, quality, and diversity of the required data. These considerations are detailed in Apx. C.

For this reason, our approach is based on the relative formulation proposed by Goetschalckx et al. in GANalyze (Sec. 3.2.2). The formulation is attractive because it removes the need for image synthesis, allows the reuse of pretrained components, and shifts the learning problem towards finding a direction in a latent space. Since this space may be lower-dimensional than the image space, learning such a direction may be considered more tractable.

Starting from the original loss formulation in Equation 5, the assessor is replaced by the emotion regressor from Gebhardt et al. (Equation 1). Furthermore, the formulation is adapted to use the pretrained MUNIT generator as described in the latent-style approach (Sec. 3.1.4).

$$\mathbb{E}_{s,c,\alpha} \left[\left(R(G(T_\theta(s, \alpha), c)) - (R(I) + \alpha) \right)^2 \right]$$

Equation 7:
Latent Style Loss

The resulting objective is shown in Equation 7. Here, s denotes the style vector, c the content vector, α the control magnitude, also referred to as the conditioning value, T_θ the learnable transformation function from Equation 6, G the MUNIT generator, and R the emotion regressor. The image I represents the original image before transformation.

However, this adapted formulation could not be made to work reliably. Several configurations and hyperparameter settings were tested (Apx. C). The experiments showed unstable qualitative results and fluctuating training losses without clear convergence. In addition, the approach introduced practical disadvantages for the intended application. Training was computationally expensive, and inference would have required the generator to be executed for every processed frame. This would have added a large component to the runtime pipeline and made the approach less suitable for real-time manipulation.

6.1 Parametric Generator

Building on the framework introduced by Mejjati et al. [33], a parametric generator (Sec. 3.2.1) is integrated into the GANalyze loss [23] from Equation 7.

In contrast to this prior formulation, the design eliminates the need for GAN components and replaces the generator with a fixed global transformation function T . The prior transformer function is replaced by a parametric generator f_θ that produces the parameters of T .

$$\mathbb{E}_{I, \alpha} \left[\left(R \left(T \left(f_\theta(I, \alpha) \right) \right) - (R(I) + \alpha) \right)^2 \right]$$

Equation 8:
Initial Parametric
Generator Loss

The original architecture proposed by Mejjati et al. [33] consists of five convolutional layers. A global average pooling operation is then applied to obtain a feature vector, which is split into two heads, each composed of two 128-dimensional fully connected layers. Adapting their concept, the architecture is split into three main components. First, the conditioning signal is encoded into a latent representation. In parallel, the input image is processed by a backbone network to produce features, which are subsequently aggregated into a vector. These two representations are then concatenated and passed through layers, whose heads predict the raw transformation parameters.

The capacity of the convolutional architecture presented by Mejjati et al. [33] is estimated to be approximately 6.5 million parameters. It is replaced with a *MobileNet*-based backbone. In particular, *MobileNetV4* [73] is employed, which is explicitly designed to optimize the accuracy-latency trade-off on edge devices. Compared to other models, it provides a favorable balance between accuracy and computational cost. According to the *MobileNetV4* paper, the corresponding *MNv4-Conv-S* variant achieves 73.8 percent top-1 accuracy on ImageNet, with 3.8 million parameters. On a Samsung Galaxy S23, it requires approximately 1.8ms on CPU and 2.0ms on GPU. For the implementation, a pretrained checkpoint provided by the timm library [74] is used.

The use of this backbone is motivated by several factors. First, it reduces the feature extractor from 6.5 million of the reference to 3.8 million parameters, before adding the task-specific layers. This lowers the expected computational cost. Second, the pre-trained backbone has presumably already learned general features from large-scale classification data. This may improve generalization and reduce the amount of data needed. Third, because *MobileNetV4* was explicitly optimized for efficient edge device inference, it may provide better practical latency.

6.1.1 Improving Transformation Identities

In the absence of additional inductive bias, the initial model is observed to produce strong edits even when the conditioning signal is zero. This is visible in Figure 7, where the edit is quite strong even when α is zero.



Figure 7: Example of Edited Image with Zero-Conditional

To mitigate this behavior, two modifications are introduced:

1. The conditioning signal is explicitly sampled at zero by setting the conditioning within a batch to zero with a given probability as shown in Equation 9.
2. A transformation is applied to the raw outputs of the prediction heads.

$$\alpha \sim U(-0.5, 0.5)$$

$$p(\alpha = 0) = 0.1$$

Equation 9:
Alpha Sampling

For the transformation, the aim is to enforce semantically meaningful parameter ranges and a well-defined identity point (see Apx. B). Due to the relative formulation of the loss, this identity must correspond to a conditioning value of $\alpha = 0$.

The two introduced changes modify the architecture. The final outputs are explicitly constrained to lie within a range by applying a function. During tuning, two formulations are tested, one based on a piecewise tanh function and one based on a sigmoid function. Both formulations map an unconstrained input $u \in \mathbb{R}$ into a bounded interval $[m, M]$ while ensuring that $f(0) = i$, where i is the identity value. The sigmoid-based formulation achieves this by shifting the input through a log term, centering around the initial value i . In contrast the tanh formulation explicitly treads the regions above and below i separately, allowing asymmetric scaling (See Apx. D). This may allow more fine-grained control near the lower and upper bounds. The choice of the transformation function is shown in the evaluation section 8.1.1.

Additionally, we experiment with multiplicative residual connections by scaling the raw outputs with the conditioning signal α . This alone in principle may ensure that the model's response is zero, but we found that the conditional signal tends to dominate the predicted parameter values.

6.1.2 Improving Image Preservation

Another qualitative observation during training is that negative values of α tend to darken the image rapidly, as shown in Figure 8.



Figure 8: Example Negative Conditioned Image

To account for this, a CLIP term is added to the objective, following the approach by Gebhardt et al. [3]. The CLIP loss, as shown in Equation 10, is implemented as the cosine distance between normalized image embeddings. As the intensity of the edit is intended to be controlled through α , no additional weight is introduced for the existing affect term. This results in our final objective function.

$$\mathcal{L}(\theta) = \left(R \left(T(f_{\theta}(I, \alpha)) \right) - (R(I) + \alpha) \right)^2 + \lambda \cdot \mathcal{L}_{CLIP}(I, T(f_{\theta}(I, \alpha)))$$

$$\mathcal{L}_{CLIP}(x, \hat{x}) = 1 - \frac{CLIP(x) \cdot CLIP(\hat{x})}{||CLIP(x)|| ||CLIP(\hat{x})||}$$

Equation 10:
Parametric
Generator
Objective

6.1.3 Improving Edit Diversity

The last qualitative observation during training is that the model may often collapse to similar edits, e.g. giving all images a green tint with negative α . Two additional changes are applied to improve diversity.

The first simple change is to introduce separate dense layers for associated outputs like the tone and color curves. This increases the capacity for these logically associated

values. A second more comprehensive change is to introduce *Feature-wise Linear Modulation* (FiLM) [75, 76]. As the name suggests, the fundamental idea is to apply feature-wise affine transformations instead of concatenating the conditioning to the features. Scaling and shifting are applied elementwise, or in convolutional networks, feature-map-wise, as shown in Equation 11.

$$FiLM(x) = \gamma(z) \odot x + \beta(z)$$

Equation
11: FiLM
Operation

An important advantage of this is that it enforces a clearer separation between content and control. FiLM is especially attractive because it may naturally extend from a univariate to a vector valued signal. This is an advantage over more explicit modifications such as multiplicative modulation, because those may require architectural changes to accommodate multidimensional conditionals.

We test this by fixing α and conducting experiments on the COCO 2017 validation set. The reported *parameter gains* are based on two diversity measures:

- The *per-feature standard deviation* is computed by taking one parameter, measuring standard deviation across the batch, and then averaging this. Some outputs are vectors. In that case, the standard deviation across images is first computed for each step of the curve, after which the resulting values are averaged.
- The *average pairwise L1 distance* measures how different the entire parameter vectors are between images. Every pair of images in the batch is compared. If the average pairwise L1 is high, it may suggest the model produces different parameters for different images, indicating higher diversity. Conversely, a low value suggests the parameters may be more similar.

The *edit metrics* are computed on the residuals, defined as the pixel-wise difference between the output and input images (*edits* = *in* – *out*):

- The *mean absolute magnitude* is computed by taking the absolute value of all edit pixels and averaging over all pixels. Smaller numbers suggest the model is barely changing the images. Higher numbers suggest generally larger edits
- Analogous to parameters, the *mean absolute pairwise difference of the edits* is also computed. Each edit map is flattened into a vector, and every pair of images is compared. A higher mean pairwise difference suggests that edits themselves vary a lot between images, whereas a lower suggests the model tends to apply similar changes.

- In addition, *Euclidean (L2)* distances between edit maps are computed to assess the spread of the generated edits. L2 emphasizes larger differences. The minimum, mean, and maximum pairwise distances are reported.

Final Parameter	Per-Feature Std Gain	Average Pairwise L1 Gain
Tone	+15.7%	+22.0%
Color	+23.0%	+28.5%
Exposure	+161.6%	+163.3%
Contrast	-44.3%	-43.1%
Saturation	+29.0%	+28.1%
Sharpness	-29.0%	-32.9%
Blur	+10.5%	+5.9%

Table 8: FiLM no FiLM Parameter Diversity Gain Comparison

Edit Metric	FiLM	No FiLM	Rel. Gain
Mean Absolute Magnitude	0.0575	0.0527	+9.8%
Mean Absolute Pairwise Difference	0.0773	0.0686	+12.6%
Pairwise Distance Minimum	44.60	29.41	+51.6%
Pairwise Distance Mean	82.67	72.82	+13.5%
Pairwise Distance Maximum	143.59	148.23	-3.1%

Table 9: FiLM no FiLM Edit Comparison

The results shown in Table 8 and Table 9 suggest that FiLM tends to increase diversity for most predicted parameters, particularly for tone and color, while the no-FiLM model shows higher variability in contrast and sharpness. The FiLM model also shows more diverse edits overall. In particular, the minimum pairwise L2 distance increases substantially (+51.6%), indicating that FiLM may reduce collapse. The mean edit differences also tend to be higher, while the maximum distance slightly decreases, suggesting fewer extreme outliers. Despite this increased diversity in parameter and edit space, the final output images are slightly less diverse under FiLM. The pixel-wise variation across outputs decreases from 0.2699 to 0.2572 (-4.7%). We assume this is due to a reduction in very extreme edits as, for example, contrast tends to change the whole image a lot. With these last adjustments, the final architecture is shown in Figure 9.

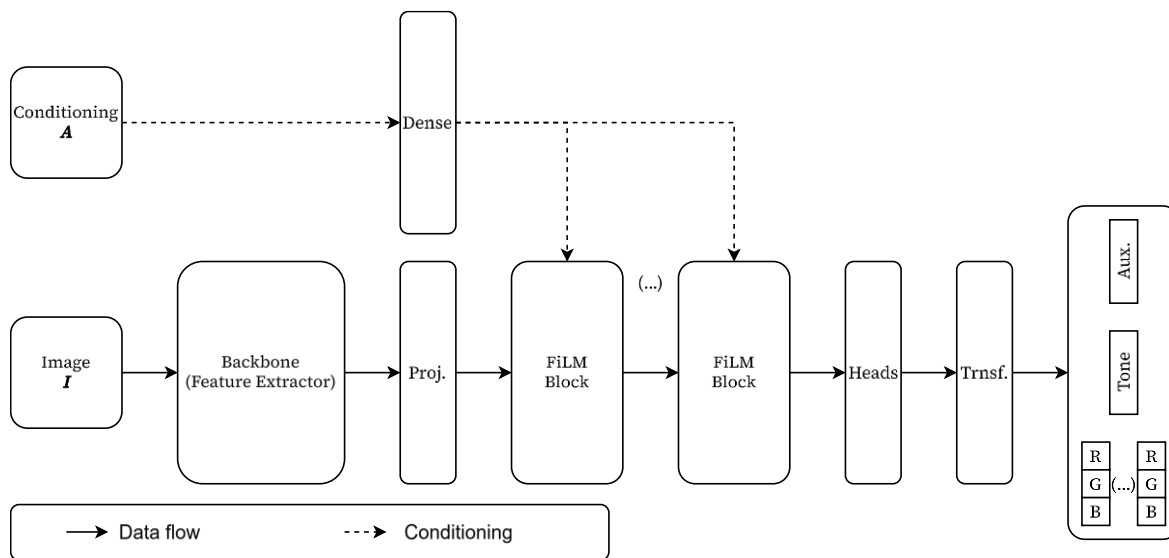


Figure 9: Model Architecture

The image I is processed by a configurable backbone and projected into a latent feature vector. The backbone can be pretrained or trained from scratch, and it can be frozen or fine-tuned. The conditioning value A is encoded by a configurable dense conditioning network and used to modulate the latent image representation through FiLM blocks. These FiLM blocks may alternatively be replaced by standard dense layers. This trunk has configurable depth and width. The shared latent representation is then passed to parameter-specific heads, including scalar, tone, and color heads. The tone and color heads have configurable depth and width and output curve parameters for the final image transformation. After the dense heads, an optional parameter transformation function can be applied, using either no mapping, a tanh-based mapping, or a sigmoid-based mapping, before the predicted parameters are used to transform the image. The model uses the leaky ReLU activation function.

6.1.4 Training and Tuning

The models are trained with the Adam optimizer, using a similar learning rate ($lr = 0.0001$) as given by GANalyze [23]. A balanced batch size of 32 is used, distributed across eight NVIDIA *Tesla P100 SXM2* GPUs resulting in four samples per GPU. All models employ a fixed CLIP term with a weighting factor of 0.05. Training is performed on the COCO training set for up to 25 epochs. This corresponds to about 2.5 million images seen during training. After each epoch, the model is executed on the evaluation set, and the resulting metrics are used to guide hyperparameter tuning. Although the models do not fully converge after 25 epochs, longer training runs make further experimentation increasingly impractical.

The training is guided by several loss components, including preservation losses based on L1 distance, CLIP similarity, KID and FID, as well as most importantly the affect score loss. During training, the modification parameter α is sampled uniformly from the range $[-0.5, 0.5]$. If the resulting modified target value exceeded the valid interval $[0, 1]$, it is clamped accordingly. The training is visualized in Figure 10 and the results are shown in Section 8.1.

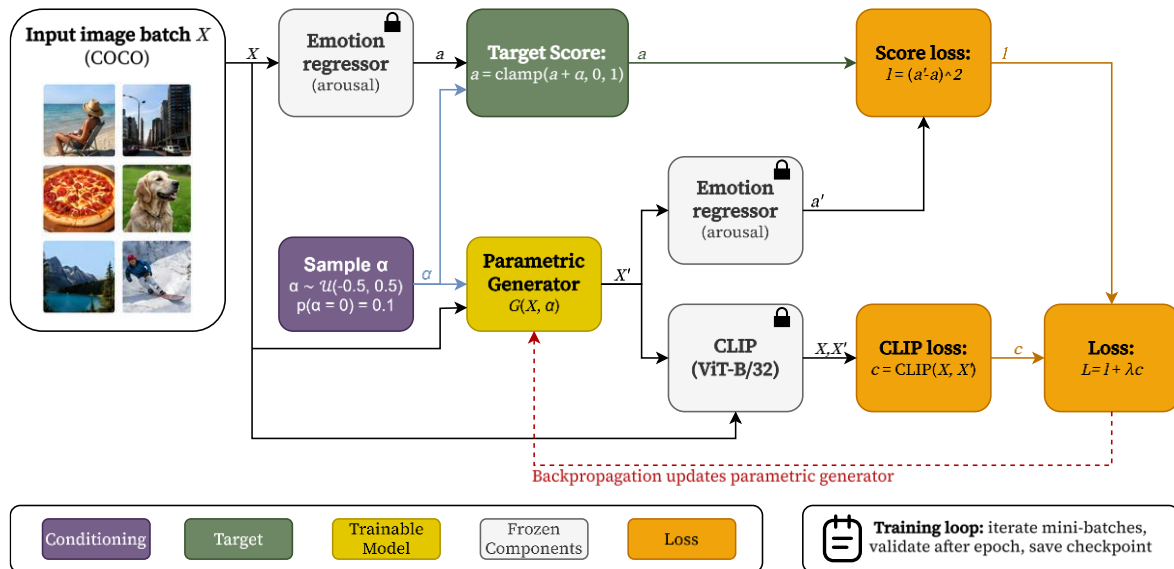


Figure 10: Training Process of the Conditional Parametric Model

During training, each input image is assigned a sampled conditioning value α , which defines the desired change in the target emotion score. A frozen pretrained emotion regressor first estimates the original image score, and the target score is obtained by adding α and clipping it to the valid range. The conditional parametric generator then predicts transformation parameters and produces an edited image. This edited image is passed again through the frozen emotion regressor, and the generator is optimized such that the predicted emotion score matches the target score. An additional CLIP-based semantic preservation loss is used to encourage the edited image to remain visually close to the original, while only the generator is updated during backpropagation.

6.2 Choice of Control Signal

In this setting, arousal was selected as the primary control signal instead of valence. Arousal is suggested to be more directly associated with activation in a single direction, whereas valence tends to increase engagement at the polarizing ends (Table 1). Thus, reducing arousal provides a clearer and more targeted objective, while indirect changes in valence are assumed to be less critical by us.

Such effects could be reduced by adding loss terms that penalize undesired changes in valence, or by extending the approach to a multivariate setting, the latter also discussed in Section 6.1.3. This extension is straightforward when FiLM conditioning is used, however, the semantic meaning of the loss changes. In the current setting, the objective can be interpreted directly through a scalar MSE between the intended and predicted score plus the additional tradeoff term. In the two-dimensional case, the target and predicted values could be represented as vectors and the loss computed as the MSE over both dimensions, that is, the squared errors of the individual dimensions averaged. In our experiments, this converged similarly compared to the univariate case.

While this may allow more explicit control over the affect space, additional steps are required to ensure independent manipulation along each direction. This could involve additional loss terms, training methods or architectural changes. For this reason, we consider the univariate variant sufficient for current proof of concept but provide an initial reference implementation.

7 Implementation

This chapter describes how the specification is implemented¹⁴. It first discusses selected issues that occurred during the implementation and explains where and why the final implementation differs from the initial design (Sec. 7.1 to 7.3). Afterwards, the chapter examines the complete end-to-end process (Sec. 7.4), the resulting UI (Sec. 7.5) and the deployment process (Sec. 7.6).

7.1 WebView CORS

When loading external resources from within the injected code, requests fail due to CORS restrictions. Since the engine is injected into the Instagram page, it runs within the web context of the `instagram.com` domain. As a result, the browser applies Instagram's origin and security policies to resource requests made by the injector. This results in denied cross-origin requests.

To mitigate this limitation, we use a separate Asset Loader component, as described in Section 5.1.1. The WebView uses this component to intercept requests to a predefined sub path of the current URL, for example `instagram.com/_onnx/test.txt`. Instead of forwarding these requests to Instagram, the component resolves them locally and returns the content from a specified location. This makes the files available to the web context as if they were served from the same origin.

7.2 Global Transformations: Independence from Image Sizes

The initial design intended that the image transformations are integrated directly into the forward pass and subsequently exported using ONNX (Section 5.1.4). However, issues arose when exporting models with dynamic input resolutions. This limitation was primarily caused by the LTS version of PyTorch. Of which, some do not support newer dynamo-based exporters. To resolve this, the environment was upgraded. This proved challenging due to missing root permissions on the training system. A solution was provided by Adrian Kretz from I3, who suggested using `uv`. This tool enables full rootless management, rendering the original setup using `Apptainer` obsolete.

However, parts of the implementation by Gebhardt et al. [3] required modification. In particular, functions from Kornia, such as `enhance.adjust`, which performs normaliza-

¹⁴ The overall structure of the codebase, including the build pipeline and the main dependencies used throughout the project are supplied in Apx. E.

tion and intensity transformations, exhibit incompatible branching behavior. These issues are resolved via monkey patching. Additionally, parts of the curve adjustment logic require modification due to export errors related to operator overloading. Certain expressions are implicitly converted leading to type mismatches in functions such as clamp. This issue is addressed by rewriting the operations to enforce use of Python scalar bounds.

We conduct a preliminary performance evaluation of the approach on a development system. Due to inefficiencies, the method proves to be insufficiently optimized, with a single frame exceeding 50 ms (Table 10). Due to this, we consider the next approach (see Section 5.1.4) and implement the transformations using fragment shaders in the *OpenGL Shading Language* (GLSL) for WebGL2.

7.2.1 Shader

The implementation of exposure, saturation, tone curve, color curve, and sharpening operations remains conceptually unchanged. These transformations are inherently local and can be expressed as per-fragment operations, meaning each pixel can be processed independently without requiring information from neighboring pixels.

The implementation of contrast differs structurally due to its dependence on a global mean. The reference by Gebhardt et al. [3] scales pixel intensities relative to the average luminance of the entire image, which requires a global reduction. Since fragment shaders do not efficiently support such operations, the mean is approximated instead. Blur also exhibits a structural difference between the implementations. The reference applies a Gaussian blur using a large 25×25 kernel. Larger kernels provide a closer approximation but incur computational cost. In contrast, the implementation uses a 7×7 sampling window. This reduces the number of required texture samples per pixel from 625 to 49.

Approach	Environment	Mean (ms)
ONNX	Rt Version 1.24.3, WebGPU	55.1
WebGL2 ¹⁵	GL2, Edge Version 147.0.3912.98	0.5

Table 10: Comparison of Transformation Performance

Table 10 shows the average latency measured on a Mozilla test video¹⁶. The experiments were conducted on a system equipped with an AMD Ryzen 7 5700X3D 8-Core Processor and an NVIDIA GeForce RTX 4070 Ti SUPER GPU. It is visible that compared to

¹⁵ WebGL2 timing operations required for benchmarking are not available within the Android WebView

¹⁶ “flower”, available at <https://interactive-examples.mdn.mozilla.net/media/cc0-videos/flower.mp4>

the shader the original transformations¹⁷ are not appropriate for real-time use. This is mainly attributable to the mean computation of the contrast and other spatial data dependencies.

7.2.2 Rendering Process

The rendering process for a video element is shown in Figure 11. The pipeline starts when a render loop is attached to the element and target canvas. In this setup, a callback is registered such that each new video frame or animation frame triggers rendering. If the browser supports it, the loop uses *requestVideoFrameCallback* to run when a new video frame is ready; otherwise, it falls back to *requestAnimationFrame*. On loop start, previous resources are released and the renderer is initialized for the target canvas. The renderer then creates a WebGL2 context for that canvas, which is the browser's interface to the GPU. When initializing, the renderer sets up all required GPU resources. This includes the compilation of shaders and its attachment to the GPU.

A key step is creating a *full-canvas quad*. This is a rectangle that covers the entire canvas. WebGL draws triangles only, so the rectangle is made from two triangles. The vertices for this quad are defined in *clip space* on the CPU and uploaded to the GPU as a *vertex buffer*. Clip space is the coordinate system used by the GPU after the vertex shader, where each axis runs from -1 to +1 across the visible area [77]. For example, (-1,-1) is the bottom-left of the canvas, and (+1,+1) is the top-right. Since the quad vertices span the full range [-1,+1], the two triangles cover the whole canvas. This means the quad will fill the viewport when drawn.

On each frame, the renderer does per-frame updates before drawing. It writes all current parameters, such as contrast, brightness and tone curves, into the shader uniforms. Since uniforms are constants for one draw call, this step ensures the shaders use the latest settings. It checks that the video is ready and then copies the current frame into the GPU texture created earlier.

Uniforms and textures are the two main ways to get data into the shaders. *Texture coordinates* (UV) are normalized (0 to 1) values that a fragment shader uses to look up the correct texel (texture pixel) from the frame. As MDN notes, texture coordinates are normalized to the [0.0, 1.0] range regardless of the actual image size [78]. For example, UV = (0.5, 0.5) will always sample the center of the texture, whether the image is 512 × 512 or 1920 × 1080.

¹⁷ A PyTorch wrapper module is used to perform the global transformations in the forward pass and returns the adapted image. It is exported using ONNX.

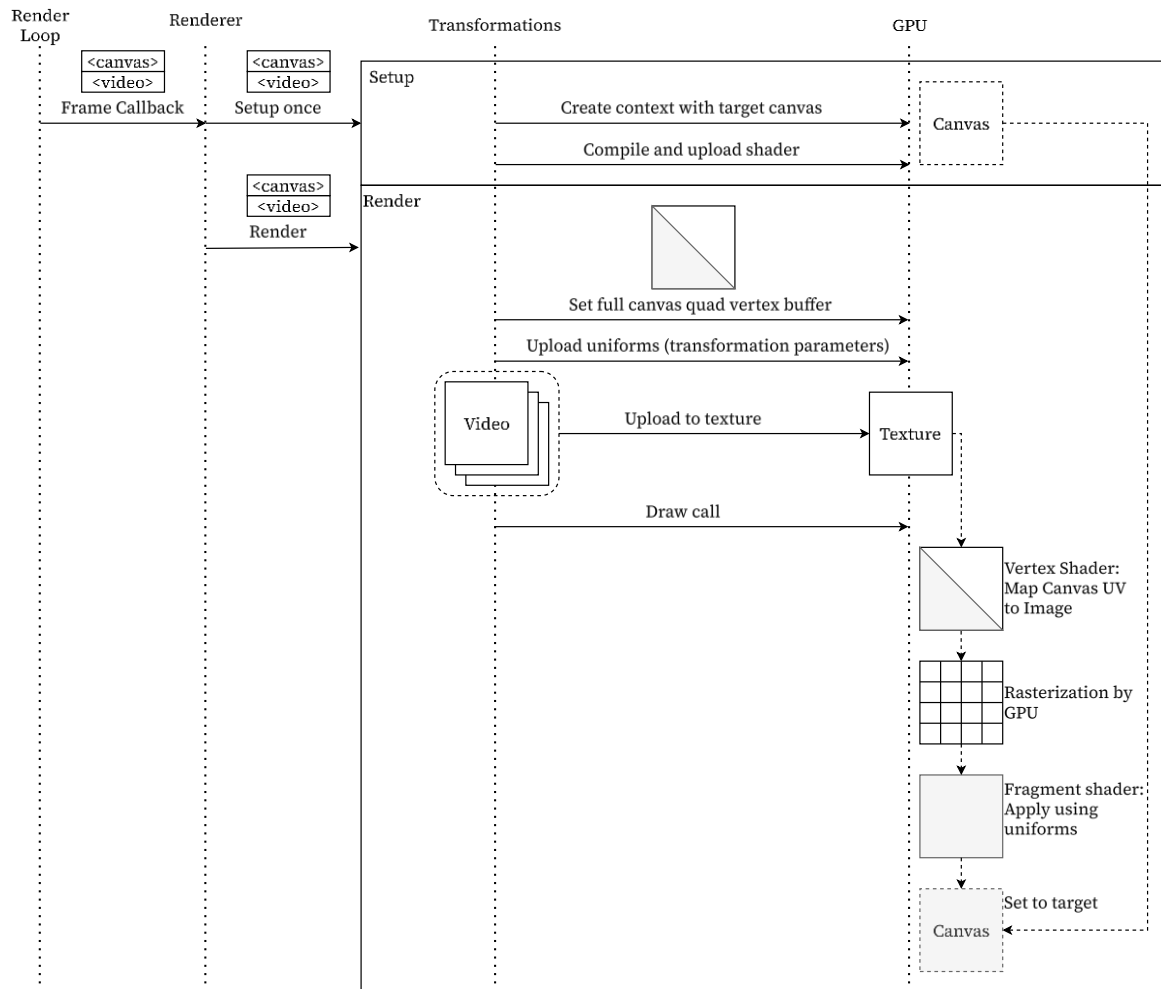


Figure 11: Rendering Process

Finally, the renderer issues the draw call for the six quad vertices. This tells the GPU to run the shader program for the full-canvas quad. Internally, the GPU pipeline proceeds as follows:

- *Vertex Shader:* The GPU executes the vertex shader once per vertex of the quad (six times). In our case, the vertex shader does two things. It assigns the clip-space position and it computes a canvas UV coordinate from that position. Simultaneously, the shader transforms that clip-space coordinate into a $[0,1]$ UV. This is needed because the fragment shader will run once per output pixel (fragment) and needs to know where that pixel lies on the canvas. By passing a normalized UV from the vertex shader, each fragment gets its own UV in $[0, 1]$.
- *Rasterization:* The GPU rasterizes the two triangles into fragments (potential pixels). Rasterization determines which screen pixels each triangle covers. In

effect, the fragment shader is run once per fragment, with the UV telling it the pixel's normalized position on the canvas. This stage happens automatically as part of the WebGL rendering pipeline.

- *Fragment Shader*: For each fragment, this shader is executed. It takes canvas-UV coordinate and maps it to an image-texture UV. After finding the correct texture UV, the shader samples the video texture at that point (fetches color). It then applies any pixel-wise transformations, such as multiplying by exposure factors, applying color curves or sharpening by sampling nearby texels. These operations use the uniform values uploaded earlier. Finally, the fragment shader writes out a color for that fragment. Since we render to the default framebuffer (the canvas), each fragment color becomes a pixel on the canvas. After all fragments are processed, the result is the fully-rendered frame on the canvas.

7.3 Scheduling and Smoothing

A key challenge is the associated cost with running inference on every frame. Frequent updates can put considerable pressure on the system, particularly when the inference time exceeds the duration of a single frame. In such cases, maintaining frame-synchronized updates may require buffering or other similar methods, which can negatively affect the UX. At the same time, less frequent updates may reduce the parameter quality, creating a trade-off between performance and accuracy.

To address this, a simple scheduling strategy is adopted in which inference is performed only once every n frames. ($n = 60$)¹⁸. The parameters are applied to subsequent frames until the next iteration.

However, this fixed interval represents a simplification and does not fully exploit the potential for more adaptive strategies. One of such strategy could be to dynamically trigger the inference based on thresholds, such as changes in content or confidence. In this context, a lightweight regressor, or a distilled and optimized variant thereof, could be used. For proof of concept, the simplifying assumption is made that short-form videos, such as Reels, remain stable over short time intervals.

To further reduce visible artefacts, temporal smoothing is applied before passing the estimations to the renderer, reducing changes between frames. Since the updates happen only at discrete intervals, their direct application leads to visible discontinuities.

¹⁸ This must be adapted to the specific target device

Temporal filtering is commonly used in video processing to reduce such undesired variations while preserving slower changes in the signal [79]. Each parameter is treated as a one-dimensional signal. The trajectories are smoothed using a low-pass filtering approach, since low-pass filters attenuate high-frequency variations. [80].

Each parameter is represented by a current value and a velocity term. At each rendered frame, the velocity is accelerated toward the newest target, then damped, and finally integrated. This is analogous to the standard spring-damper model, where a system is pulled toward an equilibrium point while damping reduces oscillations [81]. The use of a velocity term is also conceptually related to momentum-based updates in numerical optimization such as Adam. In discrete-time form, the update is given by Equation 12.

$$v_t = d \left(v_{t-1} + k(x_{target} - x_{t-1}) \right)$$

$$x_t = x_{t-1} + v_t$$

Equation 12:
Parameter
Smoothing
Update

Here, x_t is the smoothed parameter value at frame t , x_{target} is the most recent estimate value, v_t is the parameter velocity, k is the stiffness and d is the damping coefficient. The corresponding implementation is shown in Code 5 below.

```
1 velocity = velocity + (target - current) * stiffness
2 velocity = velocity * damping
3 current = current + velocity
```

Code 5: Parameter Smoothing Implementation

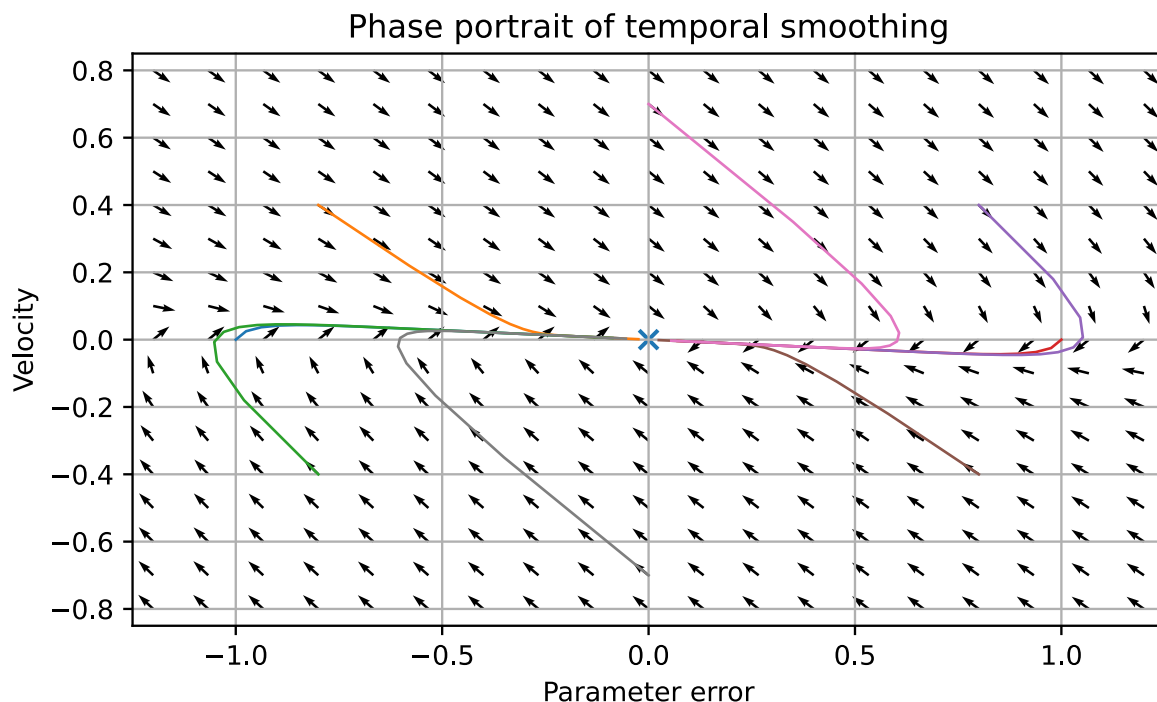


Figure 12: Phase Portrait of Chosen Smoothing Parameters

The phase plot in Figure 12 visualizes the behavior for chosen parameters (damping 0.5, stiffness 0.05). We tune parameter values empirically to select a balance between stability and responsiveness. The horizontal axis shows the parameter error, defined as the difference between the current smoothed value and the target value. The vertical axis shows the velocity term used by the smoother. The origin represents the stable equilibrium, where the current value equals the target value and the velocity is zero. The arrows indicate the one-step state update induced, and the plotted trajectories show evolutions from initial conditions. Since the trajectories converge toward the origin, the diagram illustrates that the rule reduces error over time while damping the velocity.

7.4 End-To-End Process

The final end-to-end process separates computationally intensive inference tasks from the main UI thread through the use of a dedicated WebWorker. This separation is implemented to prevent blocking operations during video processing and to preserve UI responsiveness. The main thread is responsible for user interaction, DOM observation, video frame capture, canvas rendering, and overlay management. Inference execution is delegated using the WebWorker API. The worker receives captured video frames through asynchronous message passing. This transfer mechanism allows frame processing to occur independently from other UI operations. The full process is visualized in Figure 13.

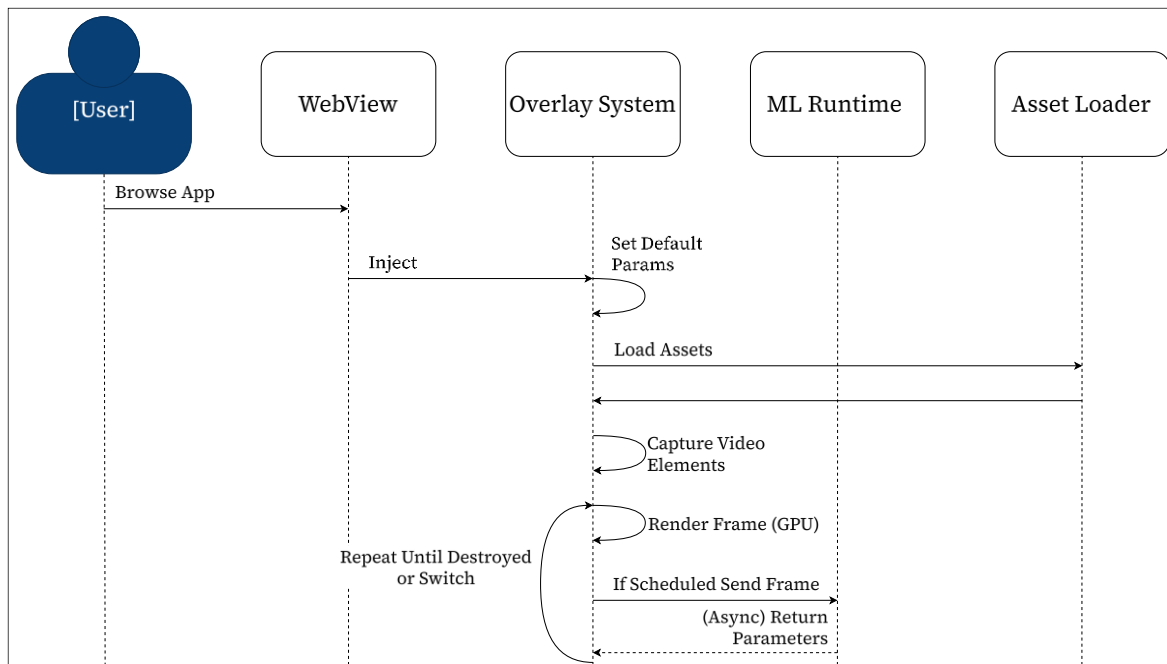


Figure 13: Simplified End-To-End Process

1. First, the user opens the application. The content is loaded through the WebView component, which acts as the proxy for the original page.
2. The WebView component injects the Overlay System into the browser context.
3. The Overlay System initializes. Required external resources, such as the ONNX runtime, are provided through the Asset Loader component to avoid access violations.
4. The Overlay System observes the DOM for video elements. When an element is detected, the rendering pipeline is attached to it. The parameters are set with no-op values, which are values that leave the image visually unchanged. The pipeline runs while the video is active and after each new frame is ready.
5. The Overlay System initializes the renderer and starts the first render pass using the identity parameters. After that, it directly issues an asynchronous task to the ML component. The frame is resized to the dimensions required by the backbone and is passed as a bitmap message. After the first directly issued task, an inference update is requested only every n -th frame.
6. Inside the ML runtime component, the worker receives the data, copies it to CPU and preprocesses it into the required model input format, converting it from RGBA to RGB and normalizing the values. It runs inference and posts the resulting parameters back to the main thread.
7. On the main thread, the Overlay System updates the parameter values when the result is received applying smoothing rules. These values are used for the following n render passes.

This processing pipeline is stopped for a specific element as soon as another element becomes the active target. This is determined by the selection of the one that is sufficiently visible in the viewport. A video becomes active once it intersects with more than a defined percentage of its area. If this active video differs from the previously processed one, the state is cleaned up. This stops the current pipeline, removes the associated resources, and attaches a new pipeline to the active element. Additional important considerations concerning the lifecycle and resource management in the context of memory pressure are outlined in Apx. G.

7.5 User Interface

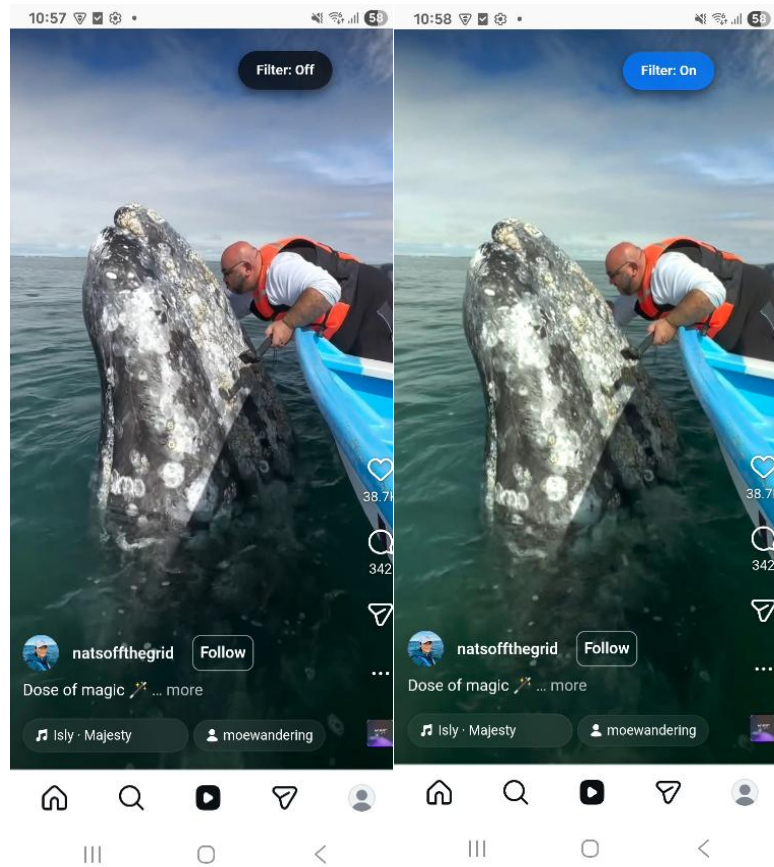


Figure 14: UI Design

The user interface as shown in Figure 14 is designed to remain non-intrusive to preserve the native experience. A small toggle button is displayed at the top right of the screen, allowing the user to enable or disable the video transformation. The button indicates the processing state through the labels “*Filter: Off*” and “*Filter: On*”. When the filter is disabled, the application behaves like a standard Instagram instance. Video content is displayed without additional processing.

Regardless of the button state, video elements are observed, the active video is detected, the canvas and renderer are initialized with identity parameters. When the button is switched on, the already attached renderer begins actively drawing the video into the overlay canvas, shows that canvas instead of the original video, and starts requesting ML updates.

7.6 Deployment

The prototype is not intended for publication in the Android store or for wider distribution. Since it manipulates content from an existing third-party platform, publication could raise legal issues.

Google Play's Intellectual Property policy states that apps must not infringe third-party rights, including copyright and trademark rights. It further notes that modified copyrighted content may still constitute a violation and that developers may be required to provide evidence of their rights to use such content. [82] In addition, Google Play restricts apps whose primary purpose is to provide a WebView of a website without permission from the website owner or administrator [83]. Since the prototype relies on displaying, interacting with, and modifying third-party web content, these policies would have to be assessed before any publication.

For usage, the application can be deployed to a device by generating an APK through Android Studio or the command line. The actual build process is automatic once the repository is set up locally.

8 Evaluation

This section presents the evaluation of the proposed model (Sec. 8.1) as well as the developed proof of concept (Sec. 8.3).

First, the model evaluation methodology is introduced, followed by the results of the fine-tuning process. Using the final model configuration, several aspects are analyzed using metrics and qualitative analysis. The chapter then turns to inference benchmarking where the model is compared with the optimization-based method. Within the same section, performance on the target hardware is reported to assess whether the approach is suitable for the intended scenario. Finally, the results of the prototype are discussed in relation to the validation criteria and requirements.

8.1 Model

The evaluation follows an approach similar to Gebhardt et al. [3]. Specifically, it assesses whether the models produce the desired manipulations while simultaneously preserving the quality as measured by FID, KID and L1. In addition, the generated edits are evaluated using the same image attribute metrics¹⁹. These are compared with previous findings to determine whether they remain consistent. The adapted²⁰ metric implementations are shown in Table 11 and the summarization of their correlation to the affect-space is shown in Table 12.

To examine whether the conditioning parameter α induces intended changes, a linear test is performed, following a similar rationale to the approaches proposed by Goetschalckx et al. [23] and Gebhardt et al. [3].

$$\mathbb{R}\left(T\left(f_{\theta}(I, \alpha)\right)\right)=\beta_1 \alpha+\beta_0+\varepsilon$$

Equation 13:
OLS Control
Model

The metrics are evaluated using 17 fixed α values within the range $[-0.5, 0.5]$ and utilize 5'000 randomly sampled images from the COCO 2017 test set, resulting in a total of 85'000 transformed images.

¹⁹ [Github regressor-guided-image-editing Low Level Image Metrics](#)

²⁰ Rewritten for use with torch

Metric	Description
Colorfulness	Computes color spread in LAB space as the standard deviation plus weighted mean of distances from the mean in the a and b channels.
Saturation	Converts image to HSV and computes the average saturation channel.
Mean Brightness Lighting Diversity	Converts image to grayscale and computes the average pixel intensity and computes the standard deviation of the L (lightness) channel in LAB, measuring variation in perceived brightness.
Blur	Estimates image sharpness via the variance of the Laplacian of the grayscale image. In this context higher means less blur. In the original formulation, this measure relies on an implementation in scikit-image ²¹ , for which no direct PyTorch equivalent was available. We employ a proxy based on the variance of the Laplacian because it is simple to implement, as it requires only a single convolution and a variance operation. This choice is supported by theory, according to which blur acts as a low-pass filter that attenuates high-frequency components such as edges [84, 85].
RMS Contrast	Computes global contrast as the standard deviation of grayscale pixel intensities.

Table 11: Image Attribute Metrics [3]

Attribute	Valence	Arousal	Source
Brightness	(+)		Brightness is typically linked to positive valence [86, 87]
Saturation	(+)	(+)	Higher saturation has been associated with positive valence and arousal [88, 89]
Contrast	(+)(-)		Contrast enhances valence when increased and reduces it when decreased [90]
Colorfulness	(+)(-)	(+)	Colored images amplify both positive and negative valence and are more arousing than grayscale images [91, 92]
Blur	(-)	(-)	Blur is known to reduce both valence and arousal [93]
Luminance Variability		(+)	Low-frequency changes in luminance are linked to higher arousal [94]

Table 12: Image Attribute Metrics Interpretation [3]

²¹ [Estimate strength of blur - skimage 0.26.0 documentation](#)

8.1.1 Training

In Figure 15 the results of training are shown. There, the general architecture consists of a dense trunk with a fixed width and a fixed depth (64, 3), followed by separate dense heads for color and tone, each again with fixed width and depth (64, 2). For the evaluation, five configurations were selected based on a preceding empirical search. *FiLM Sigmoid* denotes a model that includes FiLM blocks in the trunk and applies a sigmoid transformation to the prediction heads. *FiLM Sigmoid 128* follows the same design but increases the trunk width to 128 while keeping the depth fixed at 3. *FiLM TanH* also includes FiLM blocks in the trunk but uses a TanH transformation for the prediction heads instead of the sigmoid transformation. The *No FiLM* variants remove the FiLM blocks from the trunk and differ only in the prediction-head transformation, with one using the sigmoid-based and the other the tanh-based variant.

The performance is assessed using four metrics: overall loss, score MSE, score MAE, and CLIP preservation loss. The overall loss represents the combined optimization objective. Score MSE measures the mean squared error of the scores and penalizes larger deviations, while score MAE measures the mean absolute error and therefore reflects the average prediction error in a less outlier-sensitive way. The CLIP preservation loss captures how well the generated output preserves the semantic content.

Based on these metrics, the two sigmoid FiLM configurations achieve nearly identical overall losses and perform best among the evaluated variants. This suggests that the additional capacity of the wider variant is not necessary in the present setting, since it does not lead to a meaningful improvement but increases computational cost. The FiLM tanh configuration and the variants without FiLM show higher overall losses.

The same trend is observed for score MSE and score MAE, where both sigmoid FiLM configurations again achieve the best results and remain close to each other. A different pattern appears for the CLIP preservation loss. Here, the two sigmoid FiLM variants obtain almost identical values, indicating that the increased width also has little effect on the preservation objective. The FiLM tanh configuration achieves a lower CLIP loss, while the no-FiLM variants obtain the lowest CLIP loss overall. Thus, the sigmoid FiLM models perform best in terms of overall loss, score MSE, and score MAE, but at the cost of higher preservation loss compared with the no-FiLM baselines.

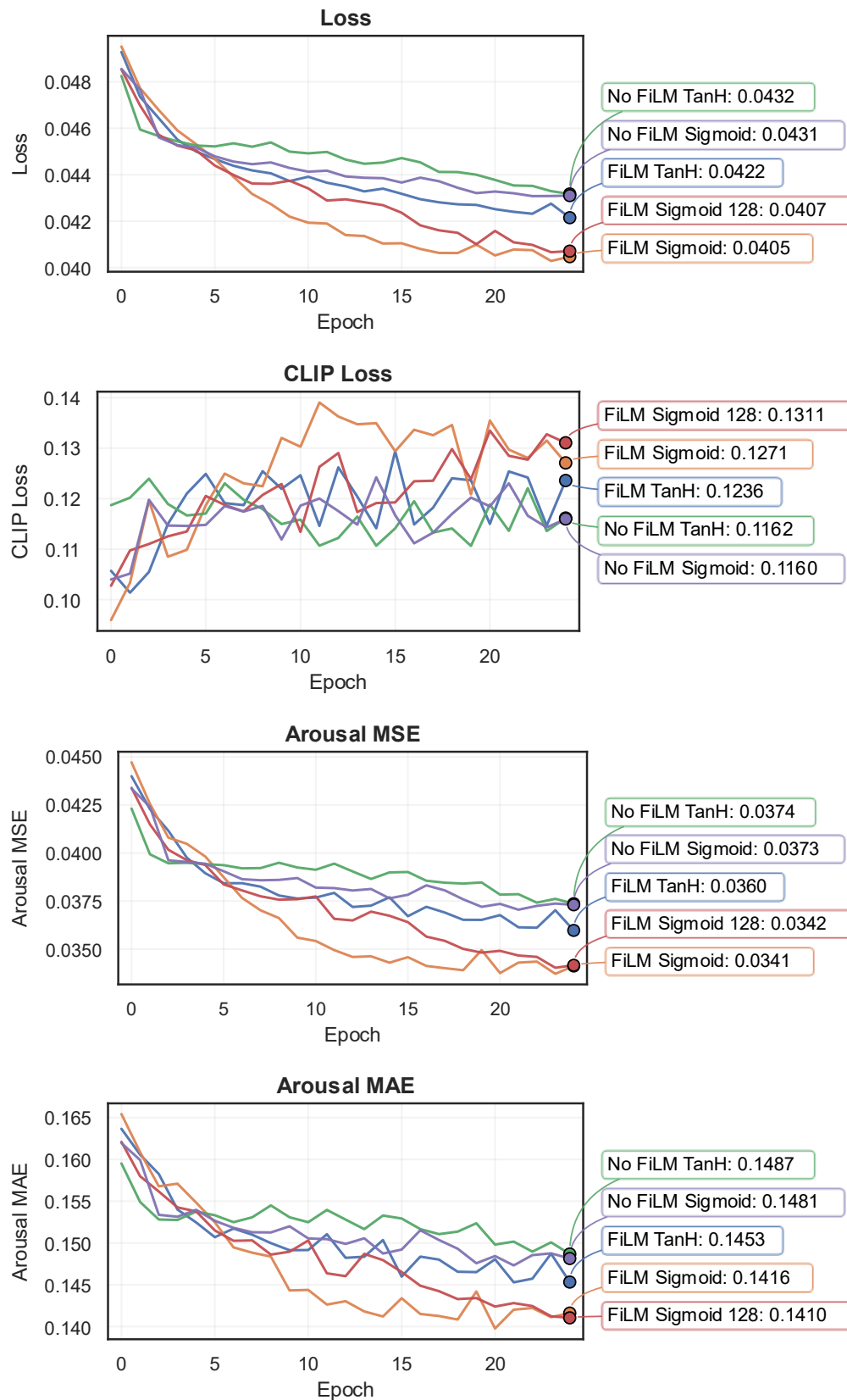


Figure 15: Finetuning Results (Validation Metrics)

Based on these results, the upcoming tests are conducted with the Sigmoid FiLM configuration as shown with the corresponding dimensions in Figure 16. The final capacity results in 2'610'169 total parameters of which 2'493'024 were frozen. The 117'125 trainable parameters use a 960-dimensional feature vector of the image supplied by the backbone. Compared to the original parametric generator formulation this corresponds to an overall capacity reduction of approximately 59.8%.

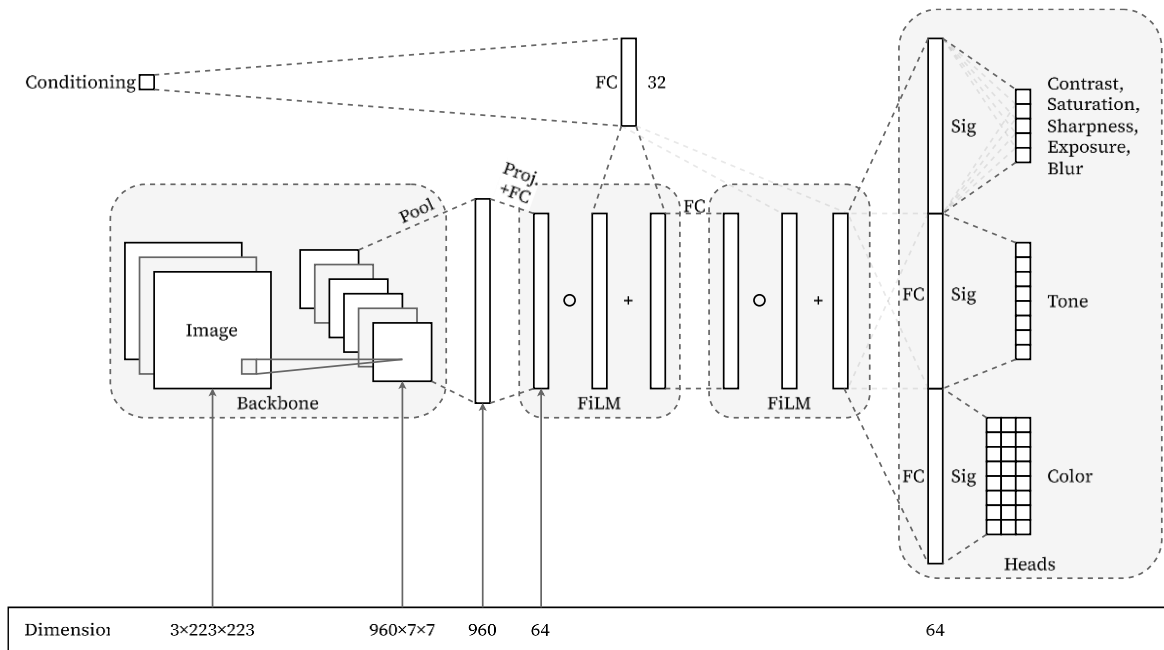


Figure 16: Final Model Configuration

The final configuration uses a MobileNet backbone to extract a $960 \times 7 \times 7$ spatial feature map from the input image. After global average pooling, this becomes a 960-dimensional feature vector, which is projected to the 64-dimensional shared trunk representation. The scalar conditioning value is encoded separately by a two-layer dense network, $1 \rightarrow 32 \rightarrow 32$. In each FiLM block, this 32-dimensional embedding is transformed by a modulation network, $32 \rightarrow 64 \rightarrow 128$, and split into γ and β , each of dimension 64. These FiLM parameters modulate the 64-dimensional latent representation by feature-wise scaling and shifting. The trunk contains three FiLM residual blocks with width 64. The final latent vector is passed to separate dense prediction heads. The scalar heads predict contrast, saturation, sharpness, exposure, and blur, while the tone and color heads have depth 2 and hidden width 64, producing an 8-step tone curve and three 8-step color curves. The raw outputs are then converted by the sigmoid-based transformation.

8.1.2 Image Preservation Metrics

Across all tested images, the evaluation yields an average FID of 67.31, KID of 0.0248, and L1 reconstruction error of 0.182. While these values indicate a noticeable deviation from the original images, they must be interpreted in light of the role of α as a control parameter. Moving away from the identity setting, both in the positive and negative direction, leads to an increase in L1, KID and FID. This reflects the stronger modifications required to alter the arousal level of the images.

However, the progression is not linear. Instead, it initially follows a seemingly quadratic pattern, with the error forming an approximately parabolic curve around the identity setting. This behavior is noteworthy, as the control is intended to produce approximately linear edits. After this initial increase, the error exhibits sharp jumps, indicating a transition to much stronger modifications. Beyond this point, changes flatten again, suggesting that further changes no longer lead to stronger deviations. In addition, not only the mean error increases, but also the variance of the error, suggesting that the effect is not uniform across images.

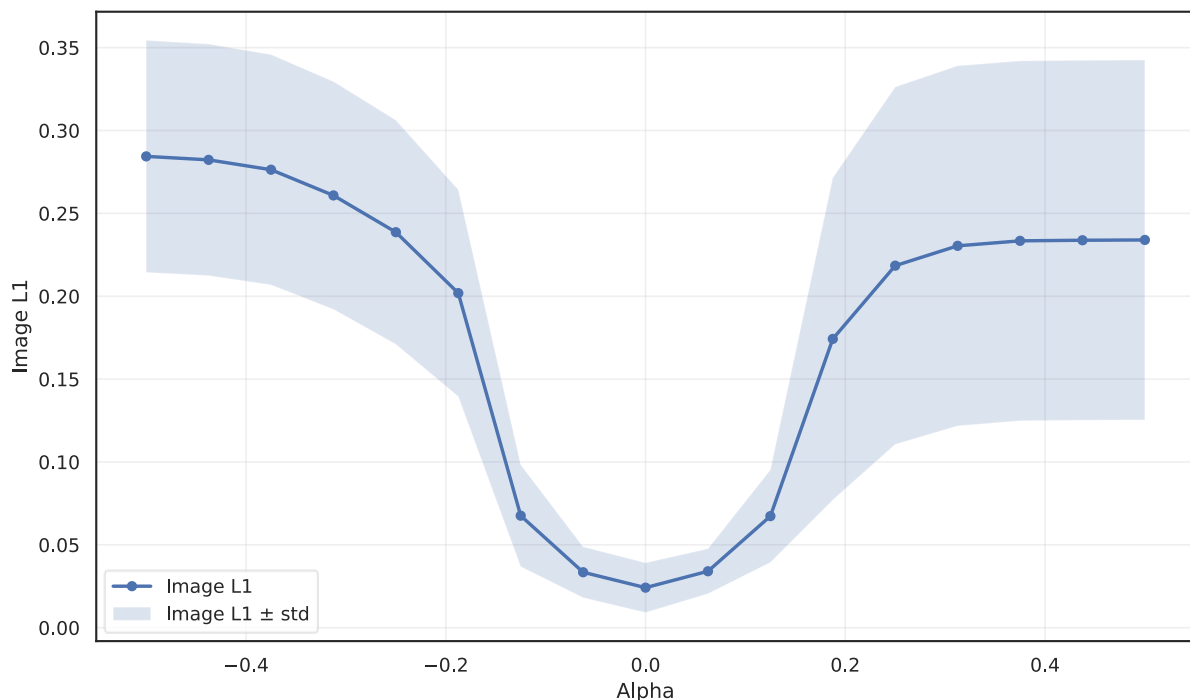


Figure 17: L1 over α

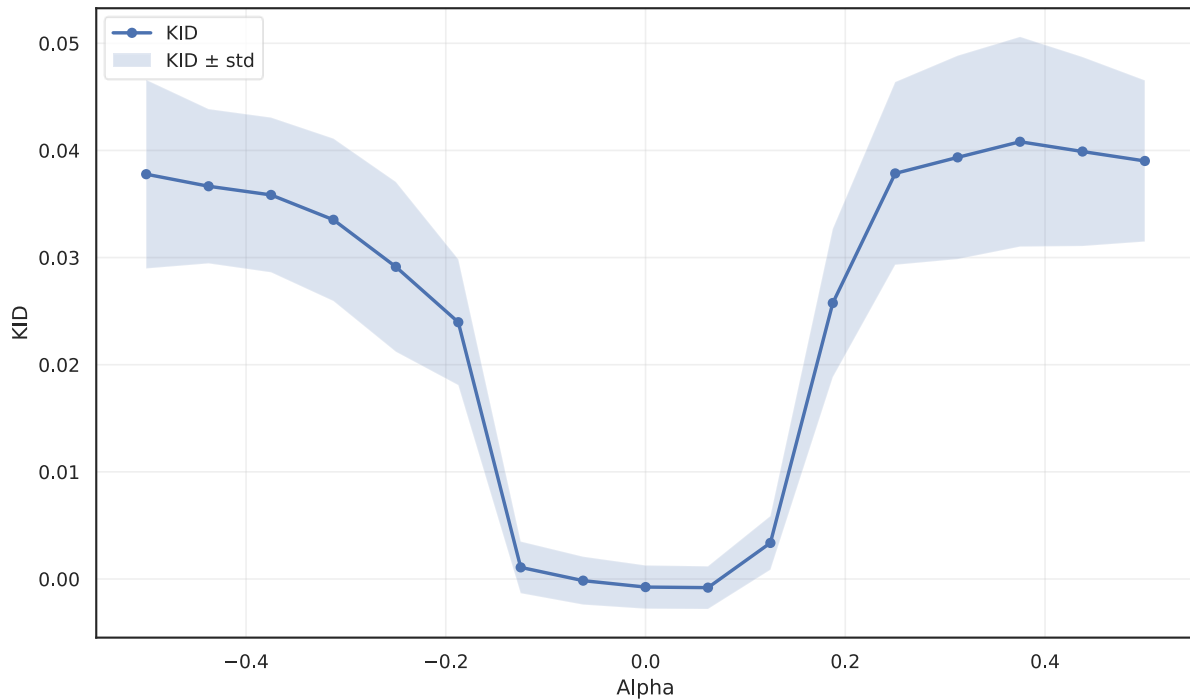


Figure 18: KID over α

8.1.3 Image Attribute Metrics

Consistent with previous findings linking higher *saturation* to increased arousal, a marked increase is observed with increasing α and a decrease for negative α values. However, this result differs from the findings reported by Gebhardt et al. [3], who showed that both style-based and parametric approaches increased saturation when transitioning toward a neutral affective point.

A similar pattern emerges for *colorfulness*. In our results, colorfulness rises sharply for positive α and decreases for negative α . This supports findings from prior literature, and aligns with the results reported by Gebhardt et al.

For *sharpness*, used here as a proxy for blur, prior literature reports that blur is associated with reduced arousal. Our findings are consistent with this relationship, as decreasing α leads to reduced sharpness, whereas increasing α results in sharper images. This pattern also aligns with the findings reported by Gebhardt et al.

Contrast, on the other hand, follows a different pattern, which may reflect the indirect effects described in the literature, as contrast is not directly linked to arousal. Prior work suggests that contrast is instead more closely associated with valence, as also shown by Gebhardt et al. In our results, contrast decreases for both positive and negative values of α .

A similar non-monotonic pattern is observed for lighting diversity, which decreases at both extremes of α . Although prior literature links certain forms of luminance variation, particularly *low-frequency luminance changes*, to arousal, our results show reduced lighting variability regardless of the direction of α . This also differs from the findings reported by Gebhardt et al., who observed greater changes irrespective of the approach used. However, in a broader sense, the results are still comparable, as both studies found patterns that did not fully correspond to the expected description.

Finally, *brightness* is primarily associated with valence in literature. In our results, brightness increases for negative α values and decreases for positive α values. This pattern is consistent with the interpretation of brightness as a valence-related feature. Gebhardt et al., in contrast reported a decrease in brightness when optimizing images toward their emotional reference condition, for both the style-based and parametric optimization approaches.

Overall, the image attributes that are directly linked to arousal, namely saturation, colorfulness, and sharpness, match the expected behavior supporting the intended role of α as control parameter. In contrast, more indirectly related attributes, such as contrast and brightness, show less straightforward patterns, which can be interpreted as consistent with their stronger association with valence. The evolution of the metrics is visualized in Figure 19 below.

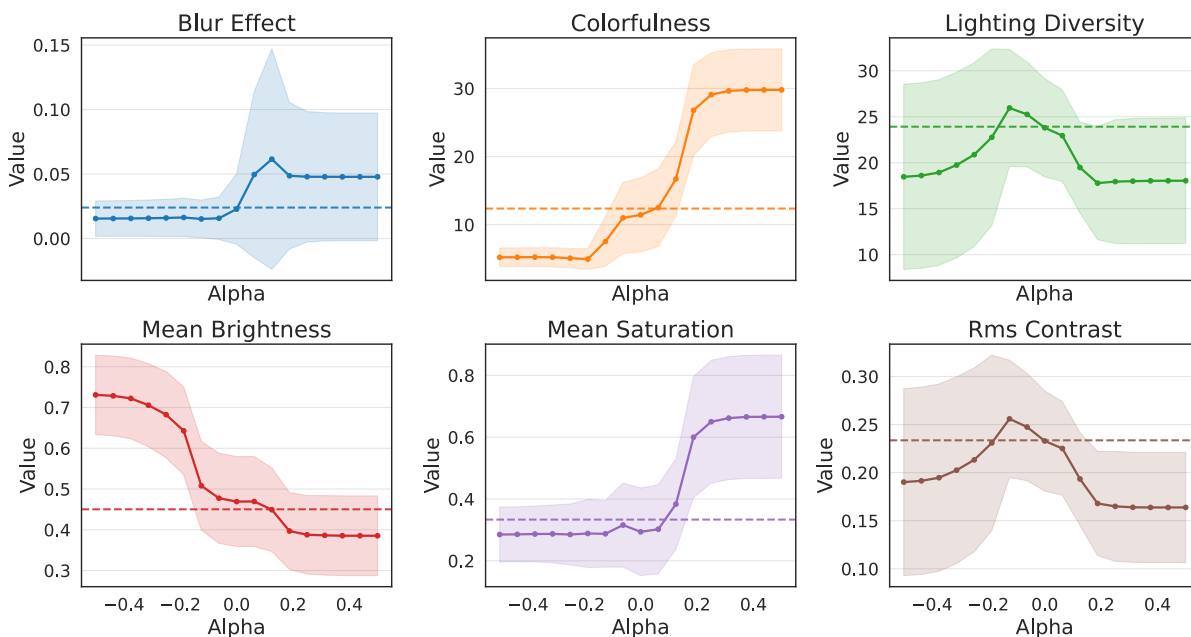


Figure 19: Evolution of Metrics over α

8.1.4 OLS

The OLS model results are shown in Figure 20. The model yielded a slope of $b_1 = 0.317$ ($p = 0$), indicating a statistically significant and approximately linear relationship between the control and the resulting change. However, the slope being below 1 suggests that the manipulation underestimates the intended effect size. The intercept was $b_0 = 0.006$ ($p = 4.35 \cdot 10^{-6}$), indicating a small bias. While numerically minor, this suggests that the transformation introduces a consistent offset in perceived arousal even at the identity. The model explains a moderate proportion of variance $R^2 = 0.668$. The remaining unexplained variance may reflect the non-linear behavior observed previously, particularly the sharp transition after the initial change. At the same time, image specific variation and noise may also contribute to the reduced fit. The mean absolute error between the observed change in arousal and α resulted in $MAE = 0.117$ and $MSE = 0.0341$.

As observed for the preservation metrics and the image attribute metrics, the model shows the strongest degree of image adaptation for conditioning values in the approximate range of ± 0.15 to ± 0.2 . This pattern appears largely symmetric for positive and negative conditioning values. Beyond this range, further increases in the conditioning value do not appear to produce additional visible changes, suggesting that the model converges toward a saturated adaptation state at higher absolute values.

This may be related to the training formulation, in which α values were uniformly sampled within the limited range of -0.5 to 0.5 in conjunction with the fixed CLIP loss term. At the same time, the observed saturation may also indicate that these low-level visual adaptations can only affect the predicted arousal to a limited extent. As a result, increasing α beyond a certain point may not be sufficient to further reduce predicted arousal without introducing more substantial changes to the image.

In addition to arousal, we find very small indirect effects on valence ($b_1 = 0.089$, $p = 0$), indicating that increasing arousal also leads to a slight linear increase in perceived valence. However, the explained variance is very low $R^2 = 0.091$, suggesting that this effect is weak and inconsistent across images. This may be consistent with the observations from the image attribute metrics, where indirect changes were also found in attributes associated with valence.

Overall, the OLS results are consistent with the image attribute metrics and support the interpretation of α as an arousal-related conditioning variable. The significant positive relationship between α and the observed change in the arousal score indicates that the conditioning direction is reflected in the model output.

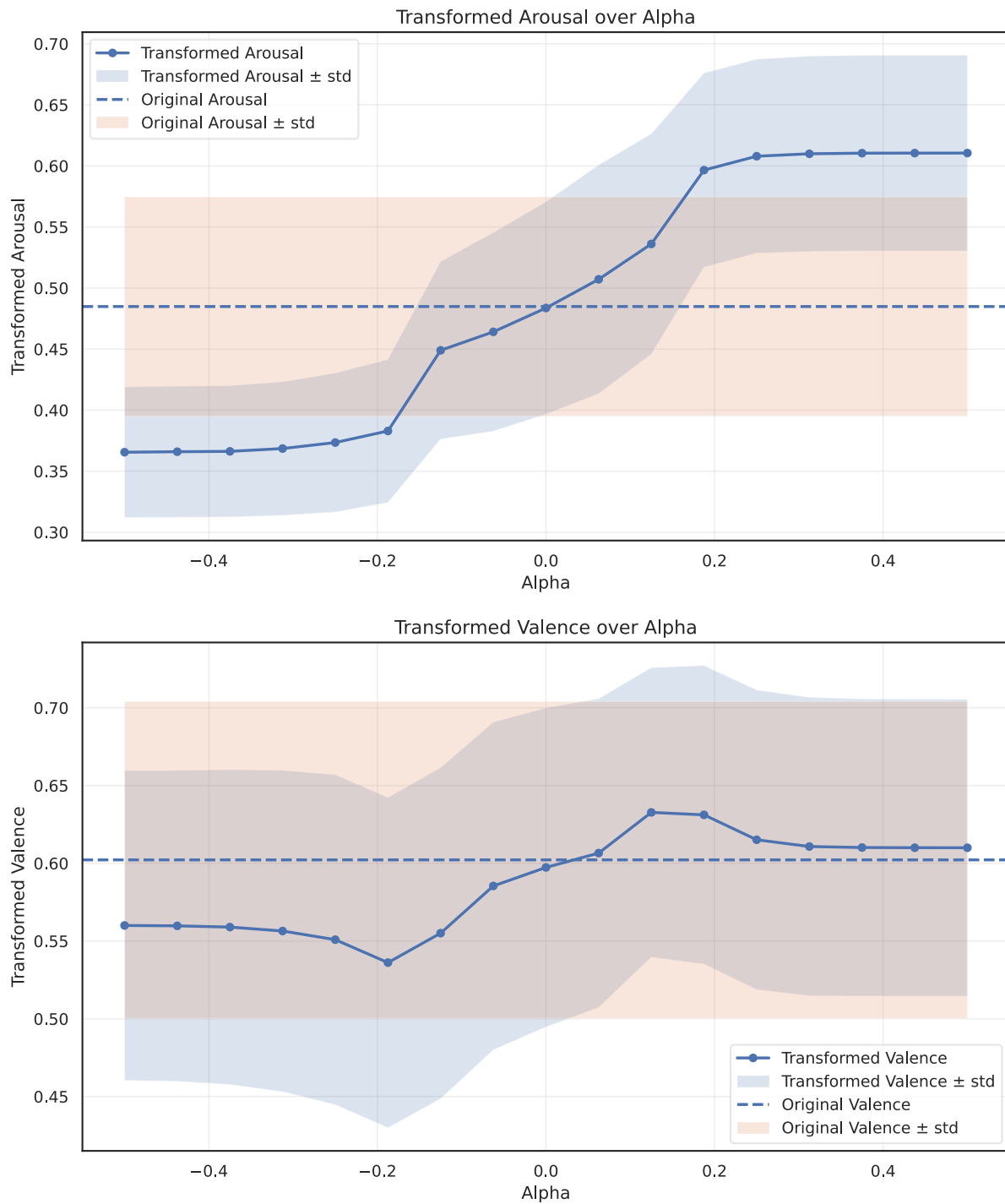


Figure 20: OLS Results

8.1.5 Qualitative Results

Qualitative results are illustrated in Figure 21 and Figure 22. In the first excerpt five images are shown with decreasing evolution of α going left to right. On the next page the same images are shown with increasing α .

As observed in the previously discussed sections, the modifications do not follow a strictly linear pattern. This behavior is also visible in the samples, where the strongest visual changes occur in the aforementioned range instead of increasing gradually across the full spectrum. In the negative direction, the edits often produce shifts that resemble a “vintage” appearance, while in some cases they tend to move the images closer to a black-and-white representation.

In contrast, edits in the positive direction increasingly converge towards a more red-dish color space, indicating a consistent shift in the visual representation. The color observation is in line with Elliot [95], who summarized research on the psychological and physiological arousal generated by color and found that red hues are associated with higher arousal, increased chroma or saturation correlates with heightened arousal, and blue or green hues are linked to lower psychological arousal, while relationships between arousal and factors such as lightness, yellow, or orange hues remain unsupported.

Interestingly, a shift towards a greener color space was especially observed in our earlier experiments before introducing image preservation and FiLM blocks. There, negative α values caused the images to shift strongly towards green tones (Section 6.1.3). This further supports the interpretation that the direction of the color shift is related to the intended modulation, although the final model appears to produce this effect in a more regulated manner.

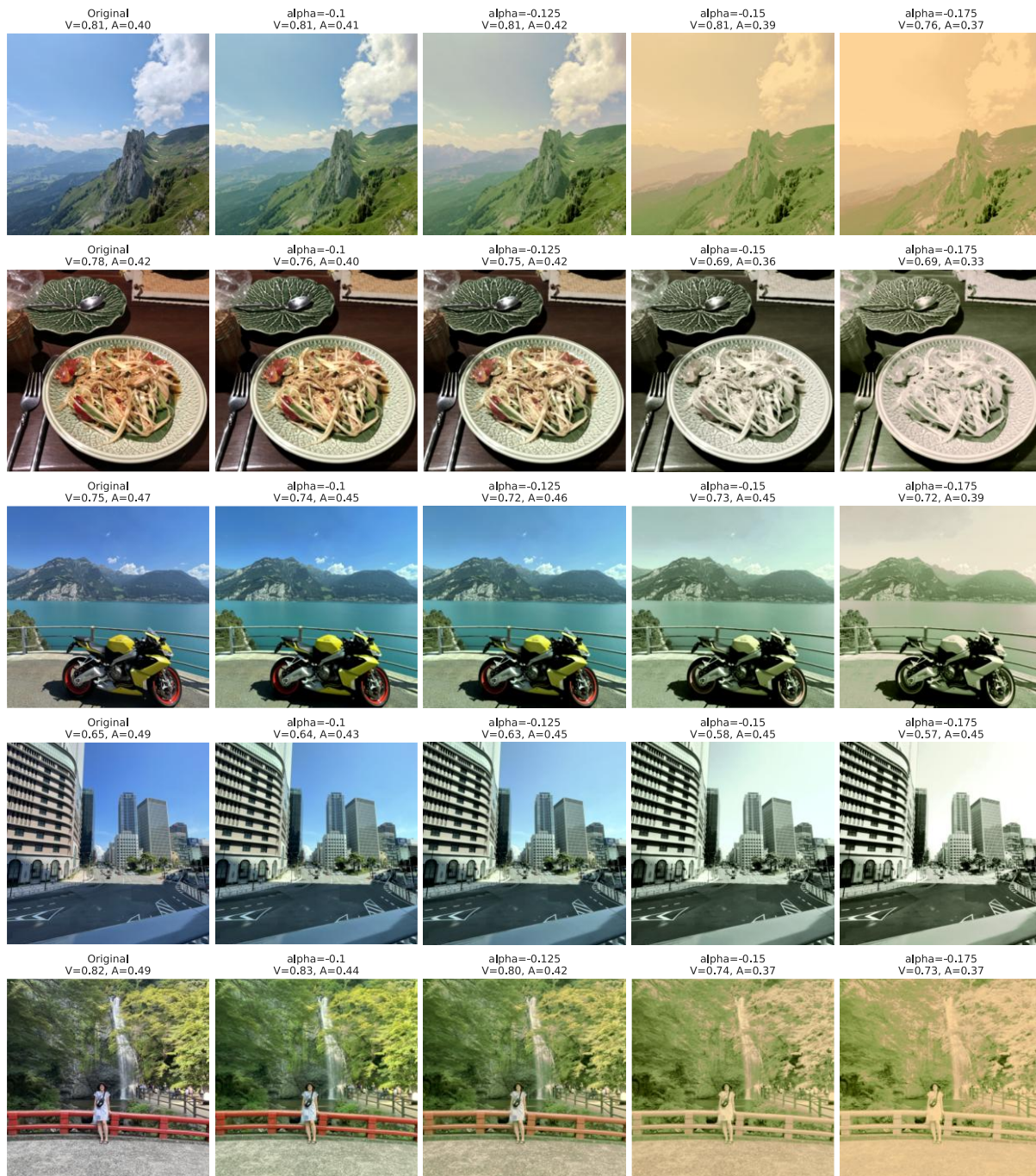


Figure 21: Negative Conditioned Samples

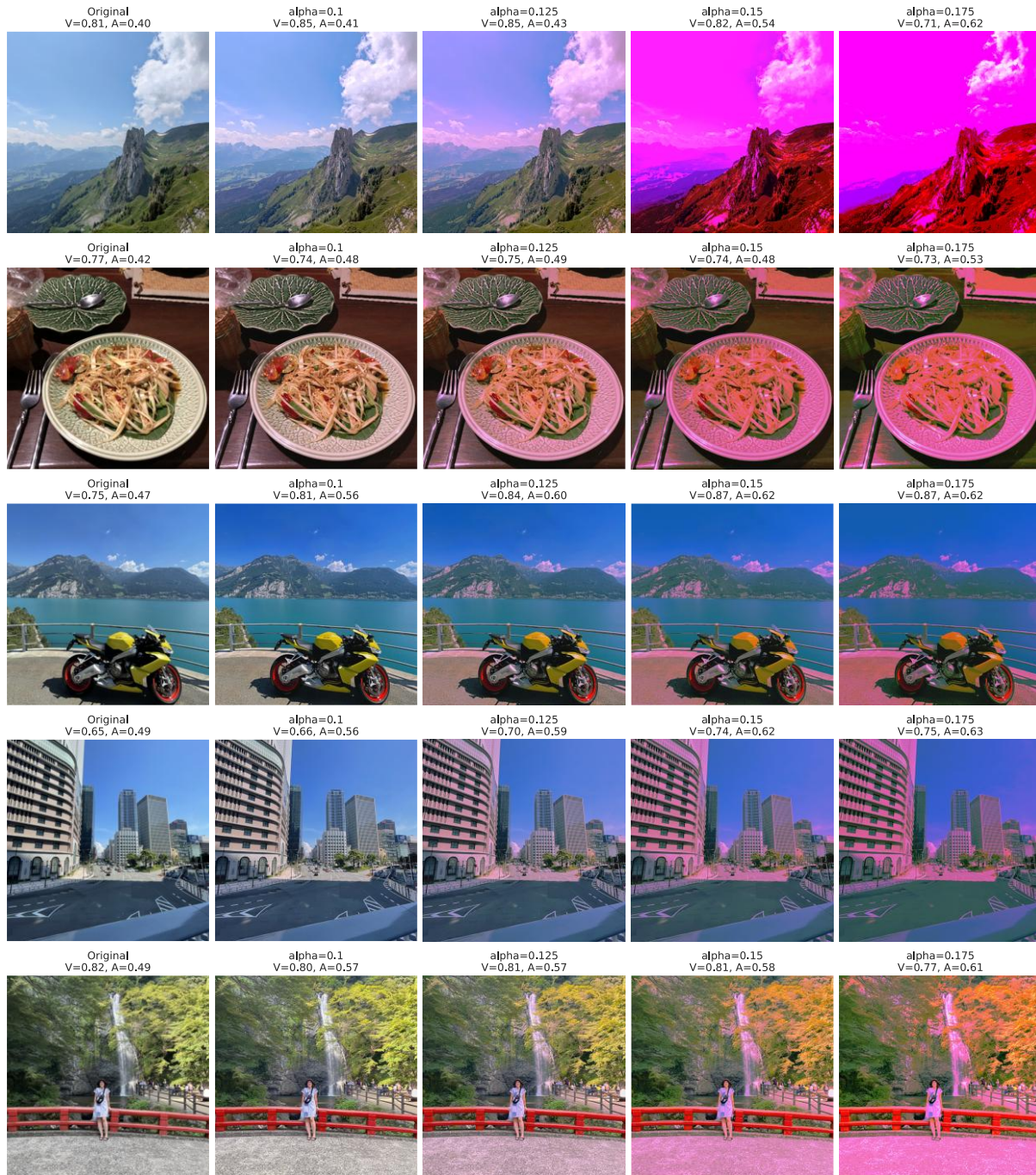


Figure 22: Positive Conditioned Samples

8.2 Performance Benchmarking

Performance validation is conducted in both the Python training environment on a NVIDIA Tesla P100-SXM2-16GB and the Samsung Galaxy S23 (Android 15).

In the Python environment, inference latency is evaluated using 5'000 images. The benchmark is focused on the model's forward-pass execution time and uses explicit CUDA synchronization to ensure timing. All measurements are performed with a batch size of 8 and fixed α values. The iterative optimization approach is benchmarked separately. This evaluation is conducted with a batch size of 1 over a total of 1'500 optimization steps.

The performance statistics on the training hardware are shown in Table 13. The learned model achieves very good per-frame performance, with all reported values remaining well below the 16ms threshold. A direct comparison with the original optimization-based approach should be interpreted with caution, since both methods follow different objective functions and the number of optimization steps required per image is not explicitly known. Nevertheless, using the reported iterative setting, the learned model shows a substantial performance gain of a factor of approximately 27 compared to the optimization-based method.

Method	Mean	Min	Median	Q95	Max
Learned (15'000 samples)	3.72	3.05	3.71	4.34	4.99
Optimization (1500 steps)	99.01	84.69	85.52	86.59	4062.40

Table 13: Per-Frame Performance on Training Hardware

On the target device, inference performance is measured within the worker thread using the Performance API. Timing data is recorded for 1'500 consecutive executions during active playback. The measurements are divided into four stages. The first stage measures the time required to put the captured image from the transferable bitmap onto the offscreen canvas. The second stage measures the time required to extract the image pixel data. The third stage covers preprocessing and inference. Finally, the total processing time is measured as the combined runtime of the stages.

The inference times on the target hardware are shown in Table 14. Compared to the mean latency of 3.718ms on the training hardware, the implementation results in a non-negligible higher mean of 40.48ms.

Step	Mean	Min	Median	Q95	Max
Copy to Canvas	0.05	0.00	0.00	0.10	3.30
Extract Pixel Data	8.98	2.70	7.90	17.00	42.20
Inference	40.48	21.80	37.90	61.60	114.50
Total	49.52	24.60	46.00	77.80	138.50

Table 14: Per Frame Performance on the Target Device (Samsung Galaxy S23)

Due to the current architectural design, which already limits the impact of increased latency and due to time constraints of the project, further optimization techniques, as discussed in section 3.4, have not yet been applied. In addition, the exact cause of the discrepancy has not been determined.

One possible explanation is the preprocessing step. The captured data must first be converted from RGBA to the RGB format expected by the model. This conversion is currently performed on the CPU. Since inference is also executed on the CPU, this should not introduce additional memory transfers, but it may still add processing overhead.

Executing the model on the GPU could potentially reduce latency. However, this step is not justified based on the expected performance of the underlying backbone. The MobileNetV4 paper reports very low inference times for the base architecture, including CPU-based execution (1.8 ms), suggesting that the backbone should be suitable without necessarily requiring GPU acceleration (2.0 ms) [73]. This comparison must, however, be interpreted carefully. Additional task-specific layers were added on top of the backbone, which increases the computational cost and may therefore affect the observed time. In addition, GPU execution was not pursued further because support within the WebView environment appeared to be limited at this stage. During the project, WebGPU could not be successfully executed in the emulator setup.

Another possible factor is overhead introduced by the browser runtime itself. Further analysis is therefore required to isolate the source of the increased latency. A useful next step would be to benchmark inference with static, preprocessed input data in the expected model format. This would help distinguish the actual inference time from overhead caused by image extraction, preprocessing, and browser-specific execution costs.

8.3 Application

The application validation covers both the Python-based components and the Android application. For the Python-based code, unit tests are added for central components as

well as for external code that was modified during the project. The main Android application is also validated using unit tests for selected parts of the logic. In addition, the functional requirements are verified through manual integration and system testing.

The protocol in Table 14 shows that the central functional requirement of the application was reached. The proof of concept successfully adapts social media videos in real time within the Android application. The social media platform is embedded through a WebView, while the adaptation logic is executed by injecting code into the web context. Video adaptation works both in the main feed and in the Reels section, and the filter can be activated and deactivated through a control element as expected.

The optional requirement for image adaptation was not reached. However, this feature could be integrated with relatively little additional effort, as the existing pipeline already provides the main mechanisms required for media detection, processing, and rendering.

The performance-related requirement was partially reached. The measured inference latency exceeded the intended per-frame target. However, the practical impact of this limitation was reduced through architectural mitigations, in particular the decoupling of rendering and inference. As a result, rendering does not wait for each inference result, and the system can still update the adaptation parameters regularly during playback. Temporal smoothing further reduces abrupt changes in effect intensity between these discrete updates. In addition, earlier loading issues related to external resources and CORS restrictions were addressed through asset-origin spoofing.

All other non-functional requirements were reached. The application could be built and deployed successfully on the tested emulator configurations and on the Samsung Galaxy S23. In addition, the WebView-based interaction with the Instagram platform worked as expected. During playback, no black frames, visible stuttering, blocked videos, or unintended interaction blocking by the toggle button were observed. The application remained stable during extended use, including repeated scrolling and repeated activation and deactivation of the filter.

UC	Process	Expectations	Remarks
NFR-1	Run build process using Android Studio and deploy to a Samsung phone and to an emulator. (x86_64 and arm64 for API 36.0, 36.1 and Prerelease 37.0; Samsung Galaxy S23)	The test and build process runs without any errors and the application is started on the remote device.	<i>Fully reached</i> ; No Remarks
NFR-2	Run the deployment to the test device. Browse the main feed as well as the reels section.	The functionalities work for the Instagram Web social media platform	<i>Fully reached</i> ; No Remarks
NFR-3	Run the deployment to the test device. Browse the main feed as well as the reels section. Consume at least 50 consecutive videos. Extract framerate numbers out of the android studio logs and aggregate.	Frame-rate logs matches testing devices paint cadence. Qualitatively: No black frames, no stuttering or blocked videos	<i>Fully reached</i> ; No black frames, visible stuttering, or blocked videos were observed during playback. Shader performance additionally assessed on separate hardware (See Section 7.2.1)
NFR-3	Run the deployment to the test device. Browse the main feed as well as the reels section. Run at least 1'500 model inference runs. Copy results into separate file and process aggregate results into average, minimum, median, q95 and maximum latency in milliseconds.	Minimum: 33ms (30FPS), Optimal: 16ms (60FPS), Best: 8ms	<i>Partially reached</i> ; See Section 8.2, Higher inference latency (approx. 50ms) mitigated using architectural changes namely using asynchronous processing.
NFR-4	Run the deployment to the test device. Browse the main feed as well as the reels section. Check if the toggle button obstructs any critical control element (back, home, action controls of the current video) in a way that decreases the UX.	The control button does not hide or obstruct any other control element that a user may want to use.	<i>Fully reached</i> ; No unintended interaction blocking was observed, as the button occupies a minimal screen region and does not interfere with scrolling or video interaction outside its bounding area.
NFR-4	Run the deployment to the test device. Browse the main	No errors in the log. The application remains responsi-	<i>Fully reached</i> ; The system maintained stable behavior

	feed as well as the reels section for an extended continuous duration (at least 30 minutes). During this apply continuous scrolling while repeatedly enabling and disabling the filter functionality.	ble and the reported latencies in the logs remain stable.	across repeated video transitions, and no crashes or rendering interruptions were observed. The filter toggle button remained synchronized with the global application state throughout repeated interaction cycles. No discrepancies were observed between the visual state of the UI button and the actual activation state of the rendering pipeline.
UC01	Run the deployment to the test device. Browse the main feed while having the toggle enabled.	Images are automatically adapted using the filter	<i>Not reached;</i> The Optional requirement was not implemented.
UC02	Run the deployment to the test device. Browse the main feed while having the toggle enabled. For part two open a video in full screen mode. For part three navigate back to the home screen. Open the reels section and consume videos.	Videos are adapted using the filter if the toggle is activated regardless of being in full-screen mode or within a feed.	<i>Fully reached;</i> No remarks
UC03	Run the deployment to the test device. Browse the main feed as well as the reels section. For different videos toggle the filter on and off.	The video adaptation activates and deactivates corresponding to the toggle. The behavior is reflected within the logs.	<i>Fully reached;</i> No remarks

Table 15: Validation Protocol

9 Conclusion

Building on prior iterative image-editing methods, a learned model was developed to adjust low-level visual properties based on regressor-estimated arousal. Instead of directly outputting edited images, the model predicts parameters for global image transformation functions, such as contrast or saturation, which are then applied to the input image.

The model relies on a relative loss formulation. During training, the loss optimizes the predicted change in arousal in relation to both the original image and a conditioning value. As a result, the model learns to select transformation parameters that shift the predicted arousal in the intended direction and magnitude while remaining anchored to the original prediction.

The selected configuration consists of a MobileNet feature extractor backbone, additional FiLM-based blocks, dense layers, and a final sigmoid transformation. The architecture remains adaptable, since its main components, the transformation function, and the image regressor used in the loss can be modified or replaced depending on the specific requirements. At this stage, the model remains univariate, as it is conditioned only with respect to arousal.

The model demonstrates sufficient conditional control over the predicted arousal change. This is supported by an OLS regression with $b_1 = 0.317$, $b_0 = 0.006$, $R^2 = 0.668$, $MAE = 0.117$, and $MSE = 0.034$. Indirect effects are also observed. The OLS results show a weak linear correlation with valence, while changes in low-level image attributes also indicate possible indirect effects on valence.

The observed transformations are consistent with prior findings on the relationship between low-level image attributes, such as saturation, and affective dimensions. This suggests that the model does not modify the affective space randomly.

On the training hardware, the model reaches an inference time of 3.72 ms, corresponding to a 27-fold speedup compared to the reference optimization baseline. On a Samsung Galaxy S23, the inference time increases to 40.48 ms. This increase is addressed in the proof-of-concept Android application through asynchronous prediction, fixed scheduling, and low-pass filtering, which reduce visible artifacts. The application uses a WebView as a proxy for Instagram, injects the system components into the web context, renders global image transformations with WebGL2 shaders, and overlays the filtered result when the system is enabled.

Finally, the system was validated on the target hardware. It can be deployed to compatible Android devices, depending on the supported version.

9.1 Discussion

At the beginning of this work stood a simple but difficult question:

How can people be encouraged to spend less time online without being forced to do so?

Many digital well-being tools answer this question with restrictions. They set timers, remove access, or rely on the user to resist content that was originally designed to be engaging in the first place. This project takes a less restrictive approach by exploring whether content can be adapted in a way that slightly reduces its stimulating effect.

Our results suggest that this idea is not only technically possible but may also be worth pursuing further. Sufficiently fast adaptation can be achieved by translating an iterative optimization approach into a learned framework that predicts parameters for global image transformation functions.

The proof of concept brings this idea into a more realistic setting. By integrating the model into a mobile application, the system is able to adapt videos directly during playback. This makes it possible to test the approach outside of isolated examples and within a less controlled environment of an actual feed.

The model changes video frames in a way that affects arousal as estimated by a regressor. This is not yet equal as proof that real users would feel calmer, watch less, or disengage more quickly. The work therefore demonstrates the technical possibility, not yet a psychological effect. Still, this possibility is relevant. If the intensity of online media can be modified continuously and with low enough latency, then a possible new design space may open up.

The model's behavior provides useful insight into this design space. The quantitative metrics and qualitative examples suggest that the transformations are systematic rather than random. However, they also show that the useful range of the control parameter α is limited. Very small changes may be too subtle to have a meaningful effect, while stronger changes risk becoming visually noticeable or disturbing. This tension is important. A non-coercive intervention should not feel like a punishment or a broken display. It has to work quietly in the background, reducing stimulation without destroying the content itself.

The observed non-linear behavior is especially interesting. It suggests that our model can preserve the original image up to a certain point, but that stronger edits in arousal may require increasingly visible changes. In practice, this means that an optimal version

of such a system would probably not maximize affective change only. Instead, it would search for a careful compromise between effect, visual quality, and user acceptance.

Another important fact is that the current model mainly controls arousal, while valence is only influenced indirectly. For this thesis, this focus is acceptable, because highly arousing content is closely related to engagement. However, it also limits the expressiveness of the system. A version that directly controls both dimensions could allow more precise and more nuanced adaptation. The modular architecture, especially the use of FiLM conditioning, makes such extensions realistic. The same principle could also be applied to other image-level scores, as long as a suitable regressor is available.

The approach is also not tied to one specific platform. Since the adaptation is applied to accessible DOM elements, the concept could in principle be transferred to other web-based social media platforms, such as TikTok, or to desktop browsers with plugin support. This extendibility is important, because digital well-being problems may not belong to a single application. They could emerge from patterns of interaction across platforms. A client-side, platform-adaptable filtering approach is more flexible than solutions built for only one closed ecosystem.

Overall, our thesis shows that learned affective filtering can bridge the gap between offline optimization and practical online media adaptation. In our view, this is the main contribution: It presents a possible path toward interventions that may not need to rely on force, guilt, or complete avoidance, but instead subtly reshape the conditions under which content is consumed.

9.2 Ethical Implications

The presented approach raises important ethical questions. Since the system changes visual content, it should not be used without the user's knowledge or consent. Otherwise, such filtering could become manipulative or be perceived as censorship. For this reason, any practical use of the system should be clearly initiated by the user with transparent information. It is also important to consider the position of content creators, whose visual choices may be altered by the filter. At the same time, the approach can be understood as shifting some agency back to users, who are often exposed to content that has already been optimized for attention and engagement. Future work should also examine and prevent misuse, for example by ensuring that adaptation is not used to push users toward bad emotional states.

9.3 Technical Limitations

Several technical limitations remain in the current PoC. *First*, inference performance on the target system was lower than on the training hardware. The implementation currently uses CPU-based WebAssembly execution, because GPU execution was not reliably available in the emulator. As a result, inference is executed on the CPU, while rendering happens on the GPU. *Second*, the current implementation uses a fixed scheduling strategy. Inference is executed every n -th frame, and the filter parameter α remains constant. While this makes the PoC simpler, it may not provide the best balance between responsiveness and visual consistency. *Third*, the prototype depends on the current Instagram front-end. The implementation assumes that video elements can be accessed and processed in a specific way. If Instagram changes how videos are structured, parts of the implementation could break. *Finally*, the fragment shader uses approximations for some transformations, in order to keep the application efficient enough.

9.4 Future Work

The most important direction for future work is the evaluation of the system with real users. The present work demonstrates that the technical approach is feasible, but it does not yet show how users perceive the intervention or whether it actually affects social media use. This would help identify a suitable intervention strength, since the current prototype uses a fixed conditioning value and does not yet determine which level of affective reduction is most effective or acceptable. Building on the technical limitations of the current proof of concept, future work could address several further improvements:

- I. *GPU acceleration*: Future work could investigate GPU acceleration, for example through WebGPU, so that image data can remain on the GPU throughout the processing pipeline.
- II. *Model compression*: Further improvements in inference speed could be achieved through model compression techniques such as quantization, pruning, or distillation.
- III. *Adaptive scheduling and control*: The current prototype uses a fixed inference schedule and a fixed conditioning value. Future versions could implement adaptive inference scheduling, where the model is updated only when necessary, and adaptive control of the filter parameter α , so that the intervention strength can change dynamically depending on the content or user preference.
- IV. *Temporal coherence*: Since the current system processes selected frames independently, future work could incorporate information from previous frames, for example through methods from recurrent networks.

- V. *Improved affective control:* A future version could control both valence and arousal simultaneously in a multivariate setup, instead of mainly targeting arousal.
- VI. *More efficient transformations:* Some transformations could be replaced with computationally more efficient alternatives. This applies particularly to spatially costly operations, such as the specific contrast adjustment used in this work.

10 Bibliography

- [1] M. Mohgler-Kuo, S. Dzemaili, S. Foster, L. Werlen und S. Walitza, «Stress and Mental Health among Children/Adolescents, Their Parents, and Young Adults during the First COVID-19 Lockdown in Switzerland,» *International Journal of Environmental Research and Public Health*, Bd. 18, p. 4668, 2021.
- [2] C. Stortz und C. Perissinotto, «"Problematische/Risikoreiche Bildschirmnutzung" In der Schweiz. Erkenntnisse und Empfehlungen der nationalen Expert:innengruppe "Onlinesucht",» *Synthesebericht 2021-2024*, 2024.
- [3] C. Gebhardt, R. Willardt, S. Sadat, C.-W. Ning, A. Brombach, J. Song, O. Hilliges und C. Holz, «Regressor-Guided Image Editing Regulates Emotional Response to Reduce Online Engagement,» arXiv, 2025.
- [4] J. A. Russel, «A circumplex model of affect,» *Journal of Personality and Social Psychology*, Bd. 39, pp. 1161-1178, 1980.
- [5] D. Garcia, A. Kappas, D. Küster und F. Schweitzer, «The dynamics of emotions in online interaction,» *Royal Society open science*, Bd. 3, Nr. 8, p. 160059, 2016.
- [6] J. Yu, «We look for social, not promotion: Brand post strategy, consumer emotions, and engagement,» *International Journal of Media & Communication*, Bd. 1, Nr. 2, pp. 28-37, 2014.
- [7] M. Schreiner, T. Fischer und R. Riedl, «Impact of content characteristics and emotion on behavioral engagement in social,» *Electronic Commerce Research*, Bd. 21, pp. 21-33, 2021.
- [8] S. Vosoughi, D. Roy und S. Aral, «The spread of true and false news online,» *Science*, Bd. 359, Nr. 6380, pp. 1146-1151, 2018.
- [9] J. Berger und K. Milkman, «What makes online content viral?,» *Journal of marketing research*, Bd. 49, Nr. 2, pp. 192-205, 2012.
- [10] K. Nelson-Field, E. Riebe und K. Newstead, «The Emotions that Drive Viral Video,» *Australasian Marketing Journal*, Bd. 21, Nr. 4, pp. 205-2011, 2013.

- [11] W. Brady, J. Wills, J. Jost, J. Tucker und J. Van Bavel, «Emotion shapes the diffusion of moralized content in social networks,» *Proceedings of the National Academy of Sciences*, Bd. 114, Nr. 28, pp. 7313-7318, 2017.
- [12] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger und I. Sutskever, «Learning Transferable Visual Models From Natural Language Supervision,» in *Proceedings of the 38th International Conference on Machine Learning*, 2021.
- [13] X. Huang, M.-Y. Liu, S. Belongie und J. Kautz, «Multimodal Unsupervised Image-to-Image Translation,» in *Proceedings of the European Conference on Computer Vision (ECCV)*, Munich, Germany, 2018.
- [14] MosaicML Inc, «ResNet Model Card,» [Online]. Available: https://docs.mosaicml.com/projects/composer/en/v0.14.0/model_cards/resnet.html. [Zugriff am 23 02 2026].
- [15] S. Albanie, «Github convnet-burden Repository,» [Online]. Available: <https://github.com/albanie/convnet-burden>. [Zugriff am 23 02 2026].
- [16] T. Zhang, F. Gao, J. Dong und Q. Du, «Remote Sensing Image Translation via Style-Based Recalibration Module and Improved Style Discriminator,» *IEEE Geoscience and Remote Sensing Letters*, Bd. 19, 2021.
- [17] O. Source, «Github open_clip Repository,» [Online]. Available: https://raw.githubusercontent.com/mlfoundations/open_clip/main/docs/model_profile.csv. [Zugriff am 23 02 2026].
- [18] Stability AI Ltd, «Announcing SDXL 1.0,» [Online]. Available: <https://stability.ai/news/stable-diffusion-sdxl-1-announcement>. [Zugriff am 23 02 2026].
- [19] Y. Zhang, E. Jin, Y. Dong, A. Khakzar, P. Torr, J. Stegmaier und K. Kawaguchi, «Effortless Efficiency: Low-Cost Pruning of Diffusion Models,» arXiv, Ithaca, NY, 2024.
- [20] TechPowerUp, «GPU Database NVIDIA GeForce RTX 4070 Ti SUPER,» [Online]. Available: <https://www.techpowerup.com/gpu-specs/geforce-rtx-4070-ti-super.c4187>. [Zugriff am 16 03 2026].

- [21] NanoReview, «Samsung Galaxy S23 Ultra Specs,» [Online]. Available: <https://nanoreview.net/en/phone/samsung-galaxy-s23-ultra>. [Zugriff am 16 03 2026].
- [22] Z. Yin, K. Chen, X. Bai, R. Jiang, J. Li, J. Liu, Y. Xian, H. Li, J. Yu und M. Zhang, «A Survey: Spatiotemporal Consistency in Video Generation,» *ACM Computing Surveys*, Bd. 58, Nr. 12, pp. 1-41, 2026.
- [23] L. Goetschalckx, A. Andonian, A. Oliva und P. Isola, «GANalyze: Toward Visual Definitions of Cognitive Image Properties,» in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) 2019*, 5744-5753, 2019.
- [24] X. Zhao, Y. Ma, Y. Guo und Y. Zhang, «Emotional Landscape Image Generation Using Generative Adversarial Networks,» in *Proceedings of the 26th ACM International Conference on Multimedia*, Seoul, Republic of Korea, 2018.
- [25] P. Lu, Y.-W. Chen, Z. Xu, W. Zhang, A. Liang und D. Xu, «EGAN: Emotional Generative Adversarial Network for Image Emotion Transfer,» *IEEE Transactions on Affective Computing*, Bd. 14, Nr. 1, pp. 164-176, 2023.
- [26] S. Weng, P. Zhang, Z. Chang, X. Wang, S. Li und B. Shi , «Affective Image Filter: Reflecting Emotions from Text to Images,» in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) , Paris, France , 2023*.
- [27] K.-C. Peng, T. Chen, A. Sadovnik und A. Gallagher, «A Mixed Bag of Emotions: Model, Predict, and Transfer Emotion Distributions,» in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015.
- [28] S. Zhao, X. Zhao, G. Ding und K. Keutzer, «EmotionGAN: Unsupervised Domain Adaptation for Learning Discrete Probability Distributions of Image Emotions,» in *Proceedings of the 26th ACM International Conference on Multimedia*, Seoul, Republic of Korea, 2018.
- [29] J. Yang, J. Feng und H. Huang, «EmoGen: Emotional Image Content Generation with Text-to-Image Diffusion Models,» in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, 2024.
- [30] J. Yang, J. Feng, W. Luo, D. Lischinski, D. Cohen-Or und H. Huang, «EmoEdit: Evoking Emotions through Image Manipulation,» in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

- [31] P. Zhang, S. Weng, C. Zhu, B. Tang, Z. Jia, S. Li und B. Shi, «Affective Image Editing: Shaping Emotional Factors via Text Descriptions,» *International Journal of Computer Vision*, Bd. 134, Nr. 1, 2026.
- [32] A. Rafaqat und A. Mohsen , «Automatic Image Transformation for Inducing Affect,» in *Proceedings of the British Machine Vision Conference (BMVC)* , Cardiff, United Kingdom , 2017.
- [33] Y. A. Mejjati, C. F. Gomez, K. I. Kim, E. Shechtman und Z. Bylinskii, «Look here! A parametric learning based approach to redirect visual attention,» in *Proceedings of the European Conference on Computer Vision (ECCV)* , Glasgow, United Kingdom , 2020 .
- [34] B. Amos, «Tutorial on amortized optimization,» *Foundations and Trends in Machine Learning*, Bd. 16, Nr. 5, pp. 592-732, 2023.
- [35] J. Benoit, K. Skirmantas, C. Bo, Z. Menglong, T. Matthew, H. Andrew, A. Hartwig und K. Dmitry, «Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,» in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [36] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam und D. Kalenichenko, «Quantization and Training of Neural Networks for Efficient,» in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [37] PyTorch, «PyTorch Quantization Reference API,» [Online]. Available: <https://docs.pytorch.org/docs/stable/quantization.html>. [Zugriff am 27 02 2026].
- [38] A. Polukhin, «Pruning: The History And Overview,» [Online]. Available: <https://polukhin.tech/2022/10/27/pruning-the-history-and-overview>. [Zugriff am 27 02 2026].
- [39] S. Han, H. Mao und W. Dally, «Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,» in *International Conference on Learning Representations (ICLR)*, 2016.
- [40] M. Paginini, «Pruning Tutorial PyTorch,» [Online]. Available: https://docs.pytorch.org/tutorials/intermediate/pruning_tutorial.html. [Zugriff am 27 02 2026].

- [41] G. Hinton, O. Vinyals und J. Dean, «Distilling the Knowledge in a Neural Network,» arXiv, 2015.
- [42] A. Vakunov und A. Svystun, «From massive models to mobile magic: The tech behind YouTube real-time generative AI effects,» Google Research, 21 08 2025. [Online]. Available: <https://research.google/blog/from-massive-models-to-mobile-magic-the-tech-behind-youtube-real-time-generative-ai-effects/#:~:text=1,faces%20in%20the%20video%20stream>. [Zugriff am 01 03 2026].
- [43] A. Kretz, «Github Repository insta-reel-mod,» [Online]. Available: <https://gitlab.ost.ch/adrian.kretz/insta-reel-mod>. [Zugriff am 21 03 2026].
- [44] «Google Chrome Help: Can I work on Google Chrome with extensions on a mobile phone?,» [Online]. Available: <https://support.google.com/chrome/thread/330237608/can-i-work-on-google-chrome-with-extensions-on-a-mobile-phone>. [Zugriff am 21 03 2026].
- [45] Mozilla, «Developing extensions for Firefox for Android,» [Online]. Available: <https://extensionworkshop.com/documentation/develop/developing-extensions-for-firefox-for-android/>. [Zugriff am 21 03 2026].
- [46] Geeks for Geeks, «How to Use Chrome Extensions on Android Mobile Browsers,» [Online]. Available: <https://www.geeksforgeeks.org/techtips/how-to-use-chrome-extensions-on-android-mobile-browsers/>. [Zugriff am 21 03 2026].
- [47] Android, «Android API Documentation: WebView,» [Online]. Available: <https://developer.android.com/reference/android/webkit/WebView>. [Zugriff am 21 03 2026].
- [48] mdn, «Manipulating video using canvas,» [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Manipulating_video_using_canvas. [Zugriff am 07 05 2026].
- [49] mdn, «API Documentation Window: requestAnimationFrame() method,» [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/Window/requestAnimationFrame>. [Zugriff am 07 05 2026].
- [50] mdn, «API Documentation DedicatedWorkerGlobalScope: requestAnimationFrame() method,» [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/DedicatedWorkerGlobalScope/requestAnimationFrame>.

US/docs/Web/API/DedicatedWorkerGlobalScope/requestAnimationFrame.
[Zugriff am 07 05 2026].

- [51] mdn, «API Documentation HTMLVideoElement: requestVideoFrameCallback() method,» [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLVideoElement/requestVideoFrameCallback>. [Zugriff am 07 05 2026].
- [52] T. Steiner, «Perform efficient per-video-frame operations,» web.dev, [Online]. Available: <https://web.dev/articles/requestvideoframecallback-rvfc>. [Zugriff am 07 05 2026].
- [53] mdn, «API Documentation HTMLMediaElement: captureStream() method,» [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement/captureStream>. [Zugriff am 07 05 2026].
- [54] Android, «Android Media Projection Documentation,» [Online]. Available: <https://developer.android.com/media/grow/media-projection?hl=de>. [Zugriff am 27 03 2026].
- [55] Android, «Android Media3-Exo-Player Documentation,» [Online]. Available: <https://developer.android.com/media/media3/exoplayer?hl=de>. [Zugriff am 27 03 2026].
- [56] Apple, «Developer WKWebView Documentation,» [Online]. Available: <https://developer.apple.com/documentation/webkit/wkwebview>. [Zugriff am 12 04 2026].
- [57] ONNX, «How to add machine learning to your web application with ONNX Runtime,» [Online]. Available: <https://onnxruntime.ai/docs/tutorials/web/>. [Zugriff am 20 03 2026].
- [58] ONNX, «How to develop a mobile application with ONNX Runtime,» [Online]. Available: <https://onnxruntime.ai/docs/tutorials/mobile/>. [Zugriff am 12 05 2026].
- [59] V. Sze, Y.-H. Chen, T.-J. Yang und J. Emer, «Efficient Processing of Deep Neural Networks: A Tutorial and Survey,» *Proceedings of the IEEE*, Bd. 105, Nr. 12, pp. 2295-2329, 2017.

- [60] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars und L. Tang, «Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge,» *ASPLOS*, pp. 615-629, 2017.
- [61] mdn, «GPUQueue: copyExternalImageToTexture() method,» [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/GPUQueue/copyExternalImageToTexture>. [Zugriff am 20 03 2026].
- [62] ONNX, «Using the WebGPU Execution Provider,» [Online]. Available: <https://onnxruntime.ai/docs/tutorials/web/ep-webgpu.html#keep-tensor-data-on-gpu-io-binding>. [Zugriff am 20 03 2026].
- [63] mdn, «API Documentation VideoFrame: copyTo() method,» [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/VideoFrame/copyTo>. [Zugriff am 06 05 2026].
- [64] w3, «WebCodecs Draft,» [Online]. Available: <https://www.w3.org/TR/webcodecs/>. [Zugriff am 06 05 2026].
- [65] mdn, «API Documentation GPUDevice: importExternalTexture() method,» [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/GPUDevice/importExternalTexture>. [Zugriff am 06 05 2026].
- [66] F. Beaufort, «What's New in WebGPU (Chrome 113),» Chrome For Developers, [Online]. Available: <https://developer.chrome.com/blog/new-in-webgpu-113>. [Zugriff am 06 05 2026].
- [67] WHATWG, «HTML Standard 4.12.5 The canvas element,» [Online]. Available: <https://html.spec.whatwg.org/multipage/canvas.html#concept-canvas-will-read-frequently>. [Zugriff am 12 05 2026].
- [68] Mozilla, «API Documentation Transferable objects,» [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Transferable_objects. [Zugriff am 06 05 2026].
- [69] mdn, «API Documentaion SharedArrayBuffer,» [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer. [Zugriff am 06 05 2026].

- [70] Android, «Native Hardware Buffer,» [Online]. Available: <https://developer.android.com/ndk/reference/group/a-hardware-buffer>. [Zugriff am 06 05 2026].
- [71] Android, «HardwareBuffer,» [Online]. Available: <https://developer.android.com/reference/kotlin/android/hardware/HardwareBuffer>. [Zugriff am 06 05 2026].
- [72] S. Brown, «The C4 model for visualising software architecture,» [Online]. Available: <https://c4model.com/>. [Zugriff am 18 04 2026].
- [73] D. Qin, C. Lechner, M. Delakis, M. Fornoni, S. Luo, F. Yang, W. Wang, C. Banbury, C. Ye, B. Akin, V. Aggarwal, T. Zhu, D. Moro und A. Howard, «MobileNetV4: Universal Models for the Mobile Ecosystem,» in *Computer Vision - ECCV 2024*, Milan, Italy, 2024.
- [74] timm, «Model card for mobilenetv4_conv_small.e2400_r224_in1k,» [Online]. Available: https://huggingface.co/timm/mobilenetv4_conv_small.e2400_r224_in1k. [Zugriff am 12 04 2026].
- [75] E. Perez, F. Strub, H. de Vries, V. Dumoulin und A. Courville, «FiLM: Visual Reasoning with a General Conditioning Layer,» *Proceedings of the AAAI Conference on Artificial Intelligence*, Bd. 32, pp. 3942-3951, 2018.
- [76] V. Dumoulin, E. Perez, N. Schucher, F. Strub, H. de Vries, A. Courville und Y. Bengio, «Feature-wise transformations,» *Distill*, Bd. 3, Nr. 7, 2018.
- [77] G. Tavares, «WebGL Fundamentals,» WebGLFundamentals, [Online]. Available: <https://webglfundamentals.org/webgl/lessons/webgl-fundamentals.html>. [Zugriff am 21 05 2026].
- [78] MDN, «Using textures in WebGL,» [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Using_textures_in_WebGL. [Zugriff am 21 05 2026].
- [79] M. Tekalp, *Digital Video Processing*, Bd. 2, Pearson, 2015.
- [80] R. Gonzalez und R. Woods, *Digital Image Processing*, Bd. 3, Pearson, 2008.
- [81] K. Ogata, *Modern Control*, Bd. 5, Prentice Hall, 2010.

- [82] Google, «Android Developer: Intellectual Property Policy,» [Online]. Available: <https://support.google.com/googleplay/android-developer/answer/9888072?hl=en>. [Zugriff am 30 05 2026].
- [83] Google, «Android Developer: Spam Policy,» [Online]. Available: <https://support.google.com/googleplay/android-developer/answer/9899034?hl=en-GB>. [Zugriff am 30 05 2026].
- [84] R. Gonzalez und R. Woods, Digital Image Processing, New York: Pearson, 2018.
- [85] Codegenes, «What's the Theory Behind Computing Image Variance for Blurriness Detection? How OpenCV Derives a Single Value from Laplacian Filter Output,» 16 01 2026. [Online]. Available: <https://www.codegenes.net/blog/what-s-the-theory-behind-computing-variance-of-an-image>. [Zugriff am 26 04 2026].
- [86] X.-P. Gao, J. Xin, T. Sato, A. Hansuebsai, M. Scalzo, K. Kajiwara, S.-S. Guan, J. Valldeperas, M. Lis und M. Billger, «Analysis of cross-cultural color emotion,» *Color Research & Application*, Bd. 32, Nr. 3, pp. 223-229, 2007.
- [87] E. Specker, H. Leder, R. Rosenberg, L. Hegelmaier, H. Brinkmann, J. Mikuni und H. Kawabata, «The universal and automatic association between brightness and positivity,» *Acta Psychologica*, Bd. 186, pp. 47-53, 2018.
- [88] C. Redies, M. Grebenkina, M. Mohseni, A. Kaduhm und C. Dobel, «Global image properties predict ratings of affective,» *Frontiers in psychology*, Bd. 11, p. 953, 2020.
- [89] J. Simola, K. Le Fevre, J. Torniainen und T. Baccino, «Affective processing in natural scene viewing: Valence and arousal,» *NeuroImage*, Bd. 106, pp. 21-33, 2015.
- [90] H. Yang, J. Han und K. Min, «Emotion variation from controlling contrast of visual contents through EEG-Based deep,» *Sensors*, Bd. 20, Nr. 16, p. 4543, 2020.
- [91] V. Bekhtereva und M. Müller, «Bringing color to emotion: The influence of color on attentional bias to briefly presented,» *Cognitive, Affective, & Behavioral Neuroscience*, Bd. 17, Nr. 5, pp. 1028-1047, 2017.
- [92] M. Cano, Q. A Class und J. Polich, «Affective valence, stimulus attributes, and P300: color vs. black/white and normal vs.,» *International Journal of Psychophysiology*, Bd. 71, Nr. 1, pp. 17-24, 2009.
- [93] A. De Cesarei und M. Codispoti, «Effects of picture size reduction and blurring on emotional engagement,» *PloS One*, Bd. 5, Nr. 10, p. e13399, 2014.

- [94] S. Delplanque, K. N'diaye, K. Scherer und D. Grandjean, «Spatial frequencies or emotional effects?: A systematic measure of,» *Journal of neuroscience methods*, Bd. 165, Nr. 1, pp. 144-150, 2007.
- [95] J. Alvarado, «Text-to-image models reveal specific color-emotion associations,» *Frontiers in psychology*, Bd. 16, Nr. 1593928, 2025.
- [96] M.-Y. Liu, X. Huang, S. Belongie und J. Kautz, «Multimodal Unsupervised Image-to-Image Translation,» in *Computer Vision - ECCV 2018*, Munich, 2018.
- [97] N. İbrahimoglu, M. Aytekin und F. Yıldız, «Knowledge Distillation from ResNet to MobileNet for Accurate On-Device Face Recognition,» *AIPA's International Journal on Artificial Intelligence: Bridging Technology, Society and Policy*, Bd. 1, Nr. 2, pp. 1-15, 2025.
- [98] S. Kim, «Knowledge Distillation for Enhanced Age and Gender Prediction,» *Mathematics*, Bd. 12, Nr. 17, p. 2674, 2024.
- [99] PyTorch, «PyTorch API Resize,» [Online]. Available: <https://docs.pytorch.org/vision/main/generated/torchvision.transforms.Resize.html>. [Zugriff am 31 03 2026].
- [100] mdn, «WebGPU API Browser compatibility,» [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGPU_API#browser_compatibility. [Zugriff am 20 03 2026].

Apx. A Registers

Glossary

Word	Meaning
CLIP	Contrastive Language–Image Pretraining is a multimodal model by OpenAI that learns to align images and text in a shared space, enabling zero-shot classification using natural language prompts.
COCO	Common Objects in Context is a large-scale dataset for object detection, segmentation, and captioning, containing over 200,000 images with multiple annotated objects (bounding boxes, masks, and captions).
DDP	PyTorch Distributed Data Parallel is a distributed training approach in PyTorch that parallelizes model training across multiple GPUs or nodes. Each process trains a separate replica of the model on a subset of the data, while gradients are synchronized across processes during backpropagation to keep the model parameters consistent.
Diffusion Scheduler	The component of a diffusion model that defines how noise is added or removed across timesteps.
FID	Fréchet Inception Distance is an evaluation metric for generative image models. It measures the distance between the feature distributions of real and generated images using representations from a pre-trained Inception network. The metric is commonly used to assess both image quality and diversity, where lower values indicate a closer match between the generated and real data distributions.
GAN	Generative Adversarial Network is a generative architecture consisting of two neural networks, a generator and a discriminator, that are trained against each other. The generator creates synthetic data, such as images, while the discriminator tries to distinguish generated samples from real ones.
GRU	Gated Recurrent Unit is a recurrent neural network architecture used to process sequential data. It is similar to an LSTM but has a simpler gated structure.
ImageNet	ImageNet is a large-scale image dataset with millions of labeled images across thousands of categories based on WordNet, widely used to train and benchmark computer vision models.
KID	Kernel Inception Distance is an evaluation metric for generative image models. It measures the difference between real and generated image distributions by comparing their feature representations from a pre-trained Inception network. Unlike some related metrics, KID is based on a polynomial-kernel maximum mean discrepancy and is unbiased for finite sample sizes.
Laplacian	Laplacian refers to a mathematical operator that measures how quickly the intensity of an image changes around each pixel. In image processing, it is often used to detect

edges and fine details, because these are areas where brightness changes strongly. The Laplacian is the divergence of the gradient, meaning that it combines the second-order changes in both spatial directions. It measures how fast that change itself changes.

LSTM	Long Short-Term Memory is a recurrent neural network architecture used to process sequential data. It uses memory cells and gates to retain relevant information over longer sequences and reduce the problem of vanishing gradients.
Monkey Patch	Monkey patching means changing or extending existing code at runtime, usually by replacing a method, function, or class behavior without modifying the original source code.
MUNIT	Multimodal Unsupervised Image-to-Image Translation is a framework for translating images between domains without paired data by separating them into content and style. [96] NVIDIA's implementation in Imaginaire provides a practical realization of this approach.
Neural Architecture Search (NAS)	Refers to a set of automated methods used to design neural network architectures. Instead of manually deciding how many layers a model should have, how these layers are connected, or which operations should be used, NAS searches through a predefined space of possible architectures and identifies designs that perform well on a given task.
Null-text optimization	An optimization method used during image editing with text-conditioned diffusion models. Instead of changing the model weights or the main text prompt, only the embedding of the empty or "null" text condition is optimized so that the model can better reconstruct the original image.
NNAPI	Android Neural Networks API, is an Android framework that allows machine learning models to run on hardware accelerators available on a device, such as GPUs, DSPs or dedicated neural processing units. In ONNX Runtime, NNAPI can be used as an execution provider to speed up inference on Android.
Pareto Optimal	A Pareto-optimal solution is a solution in which no objective can be improved without worsening at least one other objective. The concept is commonly used in multi-objective optimization to describe solutions that represent efficient trade-offs between competing goals.
ResNet	Residual Network is a neural network architecture that enables training very deep models using skip (residual) connections, with multiple variants differing in depth and block design.
RGB, CHW, HSV, grayscale	RGBA, HWC, LAB, Common image color spaces and tensor formats used in image processing, where RGB/RGBA describe color channels, CHW/HWC describe the order of tensor dimensions, HSV and LAB are alternative color spaces, and grayscale represents intensity-only images.
timm	A Python library that provides pre-trained image models, model architectures, training utilities, and related tools for computer vision tasks in PyTorch.

U-Net	A network architecture used for image-to-image tasks, such as segmentation, denoising, and diffusion models. It has an encoder-decoder structure with skip connections, where the encoder extracts image features at different resolutions and the decoder reconstructs the output while reusing information from earlier layers.
Wasserstein Loss	A loss function based on the Wasserstein distance, also known as the Earth Mover's Distance. In Wasserstein GANs, it measures how much "work" is needed to make the generated data distribution match the real data distribution. Instead of using a normal discriminator, WGANs use a critic whose output scores real and generated samples. This critic must be constrained to be 1-Lipschitz, meaning that its output is not allowed to change too sharply for small changes in the input. This is often enforced with a gradient penalty, which penalizes the critic when its gradient norm moves away from 1.
XNNPACK	Is an optimized library for neural network inference on mobile and edge devices. It is mainly designed to speed up CPU execution, especially for common operations used in deep learning models. In ONNX Runtime on Android, XNNPACK can be used as an execution provider to improve performance and energy efficiency

Figures

Figure 1: Goal Overview	8
Figure 2: Problem Overview	10
Figure 3: Use-Case Diagram	27
Figure 4: Domain Model	32
Figure 5: C4 System-Context Diagram	39
Figure 6: C4 Component Diagram	40
Figure 7: Example of Edited Image with Zero-Conditional	45
Figure 8: Example Negative Conditioned Image	46
Figure 9: Model Architecture	49
Figure 10: Training Process of the Conditional Parametric Model	51
Figure 11: Rendering Process	56
Figure 12: Phase Portrait of Chosen Smoothing Parameters	58
Figure 13: Simplified End-To-End Process	59
Figure 14: UI Design	61
Figure 15: Finetuning Results (Validation Metrics)	66
Figure 16: Final Model Configuration	67
Figure 17: L1 over α	68
Figure 18: KID over α	69
Figure 19: Evolution of Metrics over α	70
Figure 20: OLS Results	72
Figure 21: Negative Conditioned Samples	74
Figure 22: Positive Conditioned Samples	75

Tables

Table 1: Effects of Valence and Arousal on Online Engagement [3]	12
Table 2: Computational Inference Characteristics	15
Table 3: Browser Plugins on Android	22
Table 4: UC01	28
Table 5: UC02	29
Table 6: UC03	30
Table 7: NFR	31
Table 8: FiLM no FiLM Parameter Diversity Gain Comparison	48
Table 9: FiLM no FiLM Edit Comparison	48
Table 10: Comparison of Transformation Performance	54

Table 11: Image Attribute Metrics [3]	64
Table 12: Image Attribute Metrics Interpretation [3]	64
Table 13: Per-Frame Performance on Training Hardware	76
Table 14: Per Frame Performance on the Target Device (Samsung Galaxy S23).....	77
Table 15: Validation Protocol.....	80

Formulas

Equation 1: Emotion Regressor	12
Equation 2: Parametric Method	13
Equation 3: CLIP Similarity	13
Equation 4: Latent Method	13
Equation 5: GANalyze Loss.....	18
Equation 6: GANalyze Transformer Function	18
Equation 7: Latent Style Loss.....	43
Equation 8: Initial Parametric Generator Loss	44
Equation 9: Alpha Sampling	45
Equation 10: Parametric Generator Objective	46
Equation 11: FiLM Operation	47
Equation 12: Parameter Smoothing Update	58
Equation 13: OLS Control Model	63

Tools

Literature Research Management	Google Scholar, Swisscovery, ResearchGate, Arxiv
Data Analysis and Visualization	Excel, Jupyter, Python
Translation	Deepl, ChatGPT
Coding	Visual Studio Code, Microsoft CoPilot, ChatGPT, Codex
Text Creation, Text Optimization, Spelling and Grammar Checking	Word, ChatGPT
Collaboration and Project Management	Outlook, Teams, Gitlab, Jira
DevOps	Gitlab

Apx. B Image Transformation Functions

Transformation	PyTorch Shape	Allowed Range	Semantically Meaningful Range	Description	Additional Properties	Default (Identity) Value
Sharp	[B,1]	[0, +inf)	[0, 3]	Enhances edges via high-frequency amplification.		1.0
Exposure	[B,1]	(-inf, +inf)	[-0.1,1]	Scales brightness exponentially	exponential	0.0
Contrast	[B,1]	[0, +inf)	[0.5,3]	Scales difference from mean intensity		1.0
Saturation	[B,1]	[0, +inf)	[0.5, 3]	Scales color intensity		1.0
Tone	[B,1,K,1]	(-inf, +inf)	[0,3]	Piecewise tone curve	$K \geq 1$	1.0
Color	[B,3,K,1]	(-inf, +inf)	[0,3]	Rotates hue in HSV space	$K \geq 1$	1.0
Blur	[B,1]	(0, +inf)	(0,3]	Gaussian blur (sigma), 0.0 giving black images		Near 0

Applied in specific order:

1. Exposure
2. Saturation
3. Tone
4. color
5. contrast
6. sharp
7. blur

Apx. C Additional Modeling Considerations

Absolute Loss

A simple loss for a parametric approach leads to the following formulation, namely a simple *amortized regression loss*:

$$\arg \min_{\theta} \|y^*(x) - f_{\theta}(x)\|_2^2$$

Here, x denotes the input image, and $f_{\theta}(x)$ denotes the output of the learned parametric model. The term $y^*(x)$ refers to the optimal parameter vector obtained at the end of the original optimization process for the given input image. Thus, the loss encourages the model to directly predict the optimized parameters.

Alternatively, since the gradient of the objective function can be computed, the original objective can be reused, yielding an objective-based loss as shown by Amos [34]:

$$\arg \min_{\theta} y(f_{\theta}(x); x, e')$$

In the example above $y(f_{\theta}(x); x, e')$ represents the original objective of parametric optimization. There the input image x and the target emotion vector e' are kept fixed.

Finally, both approaches can be combined to incorporate information about the optimal solution as well as the objective-based update step. This can be achieved through a trade-off controlled by a hyperparameter λ , which determines the contribution of the latter:

$$\arg \min_{\theta} \|y^*(x) - f_{\theta}(x)\|_2^2 + \lambda \cdot y(f_{\theta}(x); x, e')$$

In this formulation, optimized parameters are required as targets. These would need to be generated to enable the form, which may become expensive for sufficiently large datasets. It is also unclear how much data and capacity would be required to learn this mapping effectively. Furthermore, this formulation does not provide any explicit control within the emotional space.

For a latent-style approach, distillation could be considered for a similar absolute loss. In principle, this would allow the construction of a student model with an architecture like the one shown by the Google researchers [42]. By generating samples using the optimization procedure and subsequently using these to train a student network. However, similar to the previous approach, a key factor makes this impractical. In the context of face identification, Ibrahimoglu et al. distilled a teacher ResNet model into a MobileNet

architecture using approximately 500k images [97]. In comparison, Kim used a smaller dataset of around 13k images for age and gender prediction [98]. Even with the lower data requirements reported in the latter work, generating a sufficiently large dataset could still require a substantial amount of time.

Based on the measurements in Section 8.2, this would require approximately 100 ms per optimization step on one of our tested GPUs. Assuming 300 steps per image, generating 13k images would require around 108 hours of computation. This estimate does not yet account for differences in convergence behavior or additional validation effort. Therefore, the required effort makes this approach impractical within the scope of proof of concept.

Latent-Style

For the data problem described in the absolute concept, the relative approach proposed in GANalyze [23] appears to be suitable to solve these issues.

While the paper does not report error metrics and therefore does not demonstrate training convergence, they do explicitly show that a direction in latent space can be found that modifies a valence score. Based on the pretrained MUNIT components, the learning objective can be reformulated as follows:

$$\mathbb{E}_{s,c,\alpha} \left[\left(R(G(T_\theta(s, \alpha), c)) - (R(G(s, c)) + \alpha) \right)^2 \right]$$

Here, the assessor is replaced by the emotion regressor R . In the original formulation, the generator operates on a single latent vector z . In contrast, the pretrained MUNIT generator variant G requires two latent codes, a content vector c and a style vector s . The content vector c is therefore kept fixed, while the learnable transformation is applied only to the style vector s with magnitude α . To further simplify the objective, the generation step is omitted in the second term.

$$\mathbb{E}_{s,c,\alpha} \left[\left(R(G(T_\theta(s, \alpha), c)) - (R(I) + \alpha) \right)^2 \right]$$

In this formulation, I represents the original image. Since the task may require finding direction in its 8-dimensional style space and the capacity of the transformer function is minimal, the data requirements could be relatively small for observing learning behavior. Based on this assumption, we train on the COCO 2017 validation set, which contains 5'000 images. In contrast, the original paper reports training on 400'000 sampled latent vectors [23].

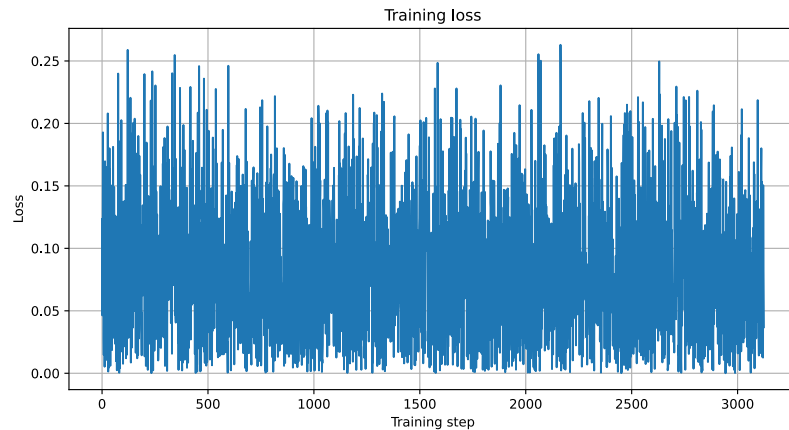
Another training detail concerns the resolution. Gebhardt et al. allow image dimensions of up to 1024×1024 [3]. To support this setting, our pipeline resizes and crops using PyTorch, which interpolates pixel values during resizing [99]. However, this resolution leads to substantial memory usage during training. Due to the resolution, the demands of the GAN components, and the averaging behavior of the regressor, we were limited to a batch size of two images per GPU, as larger batches exceeded the 16GB of VRAM available. To mitigate this, we first tried lower floating-point precision but later switched to *distributed data parallel* (DDP) training to make use of 8x NVIDIA *Tesla P100 SXM2* GPUs.

We sample α uniformly in the range $[-0.5, 0.5]$ following Goetschalckx et al. [23]. Results are shown in the figure below. It can be observed that the error does not converge but instead fluctuates. To investigate whether convergence could be achieved, we experimented with hyperparameter settings including:

- Resizing images to 480 pixels and cropping to 448 pixels, consistent with the regressor input requirements (additionally with image size 256×256)
- Training with and without averaging behavior of the regressor
- Using batch sizes: 1, 2, 4, 8, 16
- Using different sampling strategies for α were explored to examine whether the learned directions depend on its sign or range, including negative values only, positive values only, both positive and negative values, and positive and negative values constrained to the final regressor interval $[0,1]$. The latter is applied as follows; we use the regressor to get the valence rating using this rating we sample α such that the interval is respected
- Using different learning rates with different magnitudes of GANalyze [23] (0.002, 0.0002, 0.00002)
- Using additional fully connected layers up to depth 3 with width 8

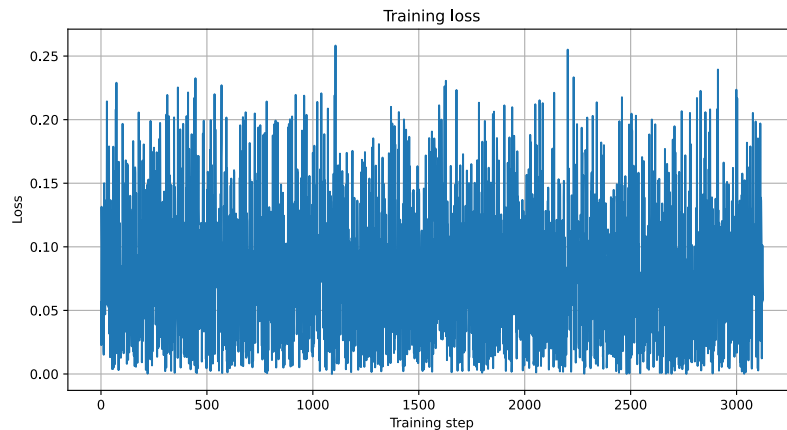
GANalyze Normal (checkpoints-ganalyze-normal-b2-ddp-final.pt)

Batchsize	8
Size	1024
LR	2e-4
Precision	32
Epochs	5
Average	True
Range	Full



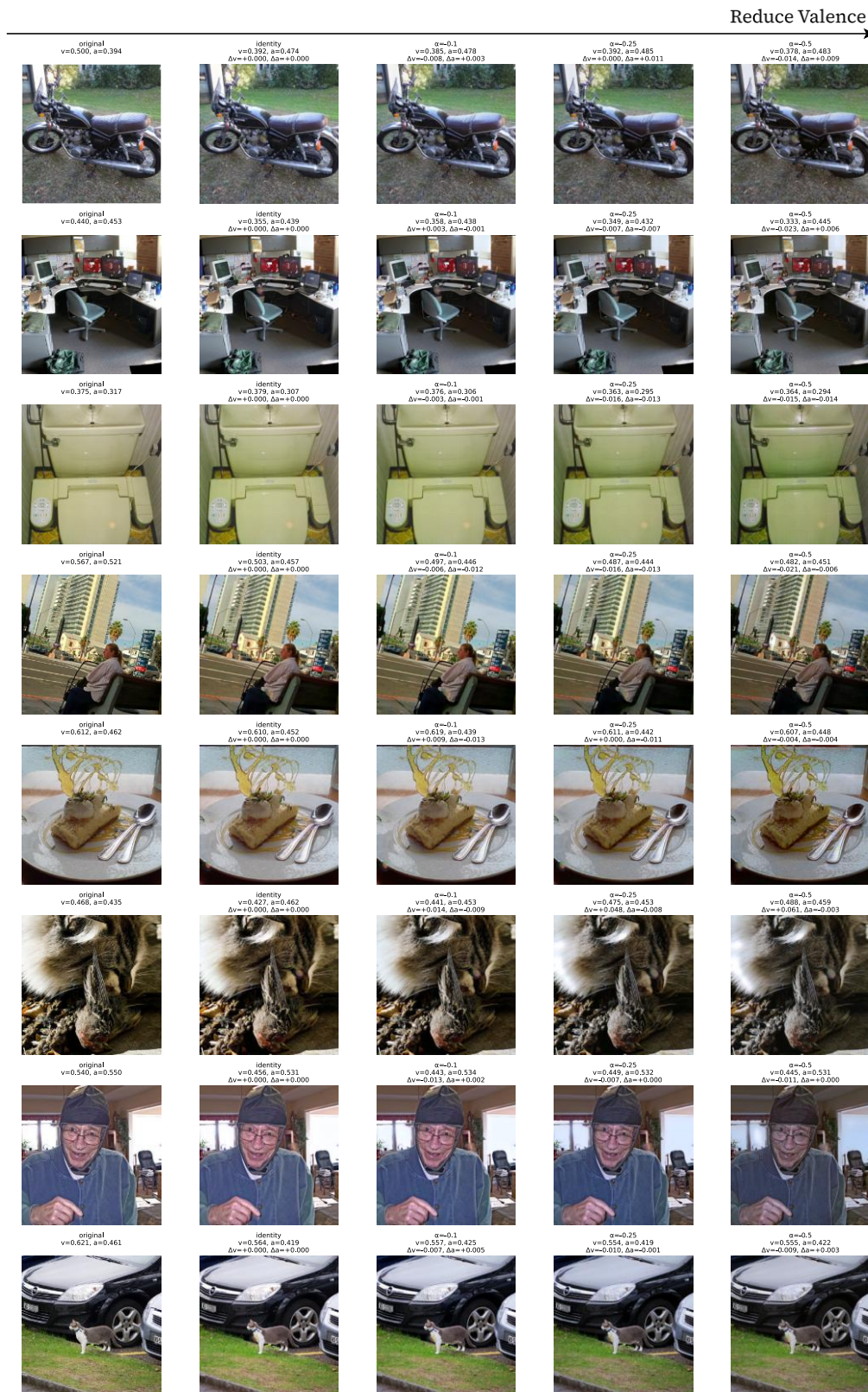
GANalyze Normal Dense 3 (checkpoints-ganalyze-dense-b2-ddp-final.pt)

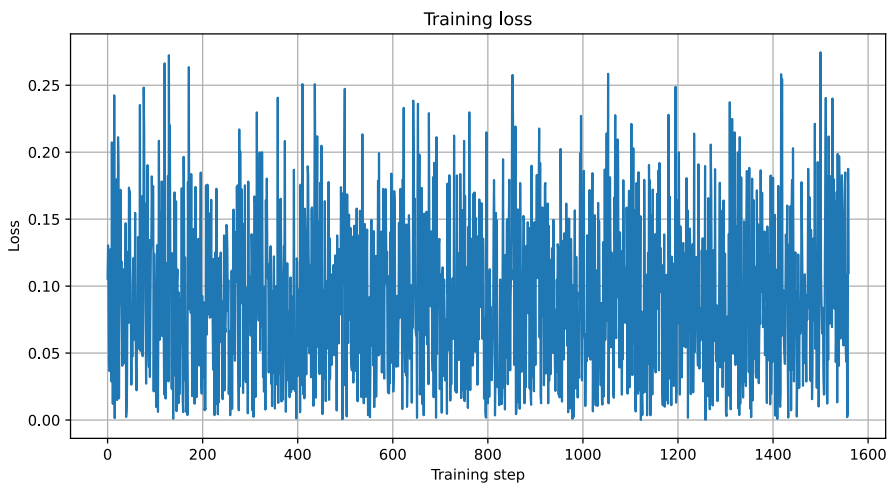
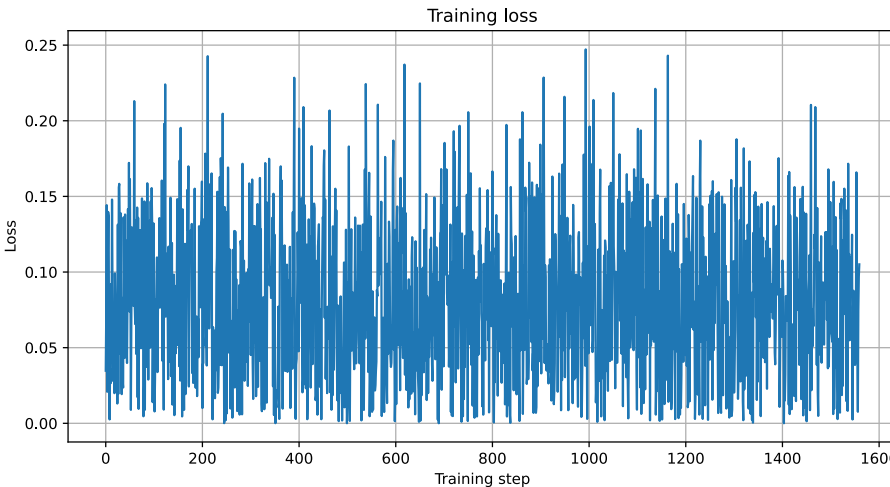
Batchsize	8
Size	1024
LR	2e-4
Precision	32
Epochs	5
Average	True
Range	Full
Depth	3

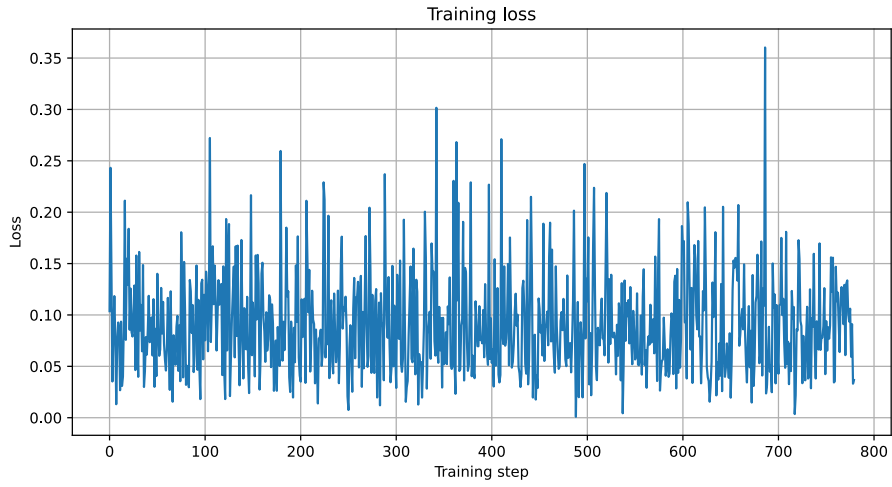
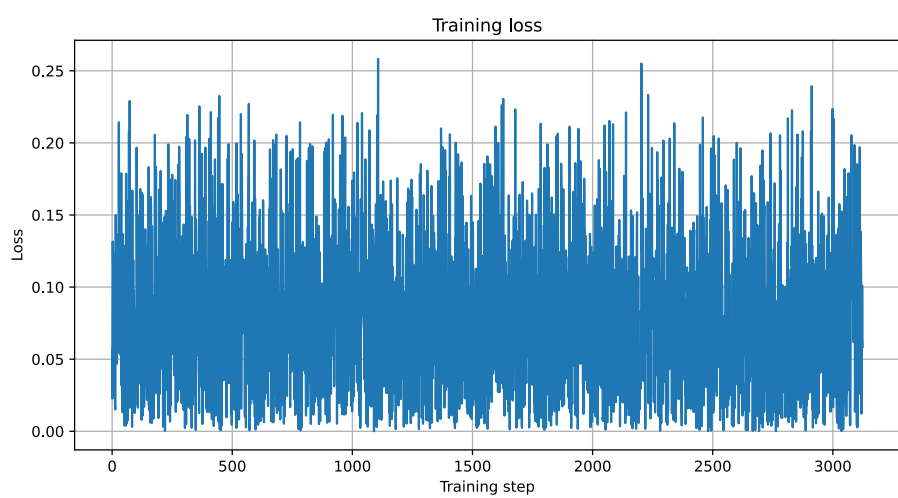


Generated results are shown below, the leftmost image shows the original input, followed by identity reconstruction, that is, the unaltered generator reconstructed image. The subsequent images illustrate incremental decreases of α . Although the loss seemingly does not converge, qualitative inspection suggests that forms of directional behavior are learned. However, the often weak transformations affect images inconsistently. A possible explanation is that there is no simple direction that corresponds to the target score. As a result, the capacity may be insufficient. But subtly increasing the dense capacity, as described above, did not lead to improvements.

Overall, we were unable to locate a configuration that worked reliably. A more rigorous evaluation would require quantitative analysis. However, this approach was not pursued further due to the lack of conclusive results and its primarily pragmatic motivation.



GANalyze Only Positive		
Batchsize	16	
Size	1024	
LR	2e-4	
Precision	32	
Epochs	5	
Average	True	
Range	Pos	
checkpoints-ganalyze-pos-b2-ddp-final.pt		
GANalyze Only Negative		
Batchsize	16	
Size	1024	
LR	2e-4	
Precision	32	
Epochs	5	
Average	True	
Range	Neg	
checkpoints-ganalyze-neg-b2-ddp-final.pt		
GANalyze Normal 256		
Batchsize	32	
Size	256	
LR	2e-4	
Precision	32	

Epochs	5	
Average	True	
Range	Full	
checkpoints-ganalyze-256-b4-ddp-final.pt		
GANalyze Normal Dense 3		
Batchsize	8	
Size	1024	
LR	2e-4	
Precision	32	
Epochs	5	
Average	True	
Range	Full	
Depth	3	
checkpoints-ganalyze-dense-b2-ddp-final.pt		

Apx. D Parameter Transformation Functions

Sigmoid

$$f(u, m, M, i) = m + (M - m)\sigma\left(u + \log \frac{i - m}{M - i}\right)$$

TANH

$$f(u, m, M, i, s) = \begin{cases} i + (M - i) \tanh\left(\frac{u}{s}\right), u \geq 0 \\ i + (i - m) \tanh\left(\frac{u}{s}\right), u < 0 \end{cases}$$

Apx. E Codebase

The project is organized as a monorepository with three major parts, each located in a separate directory. The first is the Android app, the second is the Overlay System, and the third contains the model training and evaluation code written in Python.

/	The root directory contains Gradle dependency management files of the android project
/app	Contains the source files of the android implementation
/ai-overlay-engine	Contains the web module that gets bundled into JavaScript files and injected into the Android WebView.
/python/	Contains all python code used to train and evaluate the models
/data	Contains the dataset, the pretrained component of Gebhardt et al. and the trained models.
/notebooks	Contain all high-level code for training and evaluation.
/src	Contains the core code and implementations from external sources (./ext/).
/src/test	Contains all unit tests.

Pipeline

The project is versioned using Gitlab and the CI/CD pipeline is structured into three sequential stages, *build*, *test*, and *deploy*. The Node-based component is built and tested using *npm*, but it is not handled as an independent pipeline stage. Instead, its lifecycle is integrated into the Android build process. During the Android pre-build phase, the Node component's dependencies are installed, its tests are executed, and its build is generated. Conceptually, the components are distinct parts of the system. However, from a pipeline perspective, the Node build is embedded to ensure that the application is only compiled if the web component is in a valid state.

Apx. F Dependencies

Android App For the Android environment, Gradle is used for dependency and build management, since it is the standard mechanism for configuring Android projects. The development is carried out in Android Studio, which supports different emulator configurations and allows physical devices to be connected for real testing.

Kotlin	Version 2.0
Junit	Used for unit testing
webkit	Provides the WebView Functionality

Overlay System For the TypeScript package npm is used for dependency management and running scripts. Development is carried out in Visual Studio Code, which provides full editor support for TypeScript, package configuration, and debugging.

Onnxruntime-web	Provides the ONNX runtime for the ai model
Prettier	Used for Typescript code formatting for development
TypeScript	Used for typed code instead of plain JS
Vite	Used for bundling the module into JS files
Vitest	For unit testing
Jsdom	Used to mock the DOM for tests

Python For the Python component *uv* is used for dependency management. All dependencies are located within the *pyproject.toml*. The justification for the usage is further detailed in Section 7.2.

Ipykernel	For the execution of the Jupyter notebooks
Kornia	Differentiable image transformations as well as color space transformations

Matplotlib	Used for figure plotting
ONNX	For exporting models into the Open Neural Network Exchange (ONNX) standard
Pandas	Used within notebooks for the technical evaluation
Seaborn	For figure plotting
Statsmodels	OLS regression within notebooks for the technical evaluation
Timm	Loading pretrained models like MobileNet
Torch	Main machine learning package

Apx. G Resource Ownership and Media Pressure Handling

In the WebView environment, rapid transitions between media elements create high resource pressure when decoding, rendering and inference are active at the same time. This is especially relevant for short videos, where users may switch between videos quickly.

Two main data paths are relevant that reference frame data within this application: the *rendering path* and the *AI path*.

Rendering is the more critical path. There, the current frame of the video element is uploaded into a texture. The shader processes this texture and draws the result to the overlay canvas. By sampling the frame through WebGL, it can indirectly keep references to native buffers that originate from Android's decoding. If the renderer continues to reference an old video, the backing buffer may remain alive longer than expected. This can happen for several reasons. The render loop is driven by callback, if it is scheduled before the switch, it may still be executed. In these cases, the system can keep the previous frame alive, even though the host application has already moved on. The scale of this problem is significant. A single uncompressed 1920×1080 RGBA frame contains approximately 8.3 MB per frame. At 60 frames per second, almost 498 MB of raw frame data can pass through the pipeline. This may create serious memory pressure.

The AI path has a similar, but shorter-lived, connection to data. A snapshot is created from the video element which is transferred to the worker. Inside the worker, the bitmap is drawn into an offscreen Canvas, copied into CPU memory, and then released. After this copy step, the model runs on CPU/WASM tensor data rather than on the original backed buffer. Therefore, this path can briefly touch frame-backed memory while the bitmap exists, but it should not retain it after the bitmap has been closed.

Failure occurs when these references are not cleaned up in a timely and synchronized way. During fast swiping, Instagram may start decoding the next video while the system still holds references. Because of this, its backing buffers cannot return to the pool. WebView then needs additional resources for the next video. If this sequence repeats, resources can accumulate eventually causing renderer termination.

To mitigate this, the overlay system separates resources for lifecycle management into three groups. *The first group* consists of per video overlay resources. These resources are always hard-cleaned on every active video switch. The implementation stops frame

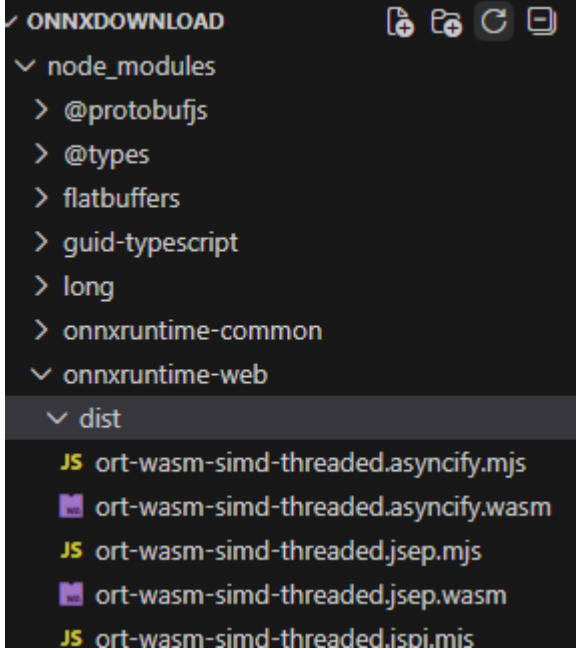
callbacks, clears the WebGL source texture, destroys the renderer, removes the overlay canvas, and discards stale inference results. *The second group* is the AI resource. The worker is kept persistent between videos to avoid setup overhead. It only uses very short lived references to the backing data. Finally, *the third group* consists of native resources; these are not managed by the developers.

The resulting lifecycle follows several important rules:

- At any moment, there must be only one active pipeline session attached to a video element. This means there is only one shader renderer and sender to the worker.
- On the worker side, only one inference must run at a time, and outdated in-flight requests must be discarded.
- The implementation must prioritize rendering over inference. Shader rendering is started first such that the visual output remains stable before any model processing is triggered. In practice, the original decoded video from the Instagram page remains visible during the setup phase. The renderer is then created, and the first filtered frame is drawn to the overlay canvas. Once this is ready, the application switches from the original view to the filtered canvas by hiding the original video. Only after this, the first request is sent to the worker.

Finally, a short cleanup gap is used before attaching the new active processing session. Native cleanup is not always immediate. This cleanup gap gives underlying system time to unwind old references before the next session starts.

Apx. H Download ONNX Web Dependencies

Require-ments: NodeJS, NPM	
Create a new direc- tory	
Install the ONNX Web runtime	npm install onnxruntime-web
Got to the depend- ency direc- tory	
Depending on your method downlod the mini- fied js file	- Ort.all.min.js - Ort.webgpu.min.js - Ort.wasm.min.js - Ort.webgl.min.js

Download the corre- sponding web as- sembly files or download all	*.wasm
--	--------