GENERATION OF TRAINING DATA FOR HANDWRITTEN TEXT RECOGNITION USING LATENT DIFFUSION MODELS

Dissertation
zur Erlangung des Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

KAI INGO WILHELM BRANDENBUSCH

Dortmund

2026

Tag der mündlichen Prüfung:	16.04.2026
Dekan:	Prof. Dr. Jens Teubner
Gutachter:	Prof. Dr.-Ing. Gernot A. Fink Prof. Dr. Stefan Harmeling

ACKNOWLEDGMENTS

I would like to take this opportunity to thank everyone who has accompanied and supported me throughout the journey that has led to this thesis. Looking back, these years have been both challenging and rewarding, and I am grateful to everyone who supported me along the way.

First and foremost, I would like to thank my supervisor, Prof. Dr.-Ing. Gernot A. Fink. Thank you, Gernot, for giving me the opportunity to pursue my doctoral research in your research group after you supervised my master's thesis. I greatly appreciate the freedom I had in shaping my research. Thank you very much for the many inspiring discussions we have had, your valuable feedback, and your continuous support over the years.

I would also like to express my gratitude to Prof. Dr. Stefan Harmeling for providing the second review of this thesis and for serving on the doctoral committee. I further thank Prof. Dr. Mario Botsch and Prof. Dr. Peter Buchholz for their participation in the committee, as well as Prof. Dr.-Ing. Peter Ulbrich for his guidance as my mentor in the structured doctoral program.

I would like to express my special gratitude to the people with whom I had the privilege of working in the research group over the years: Dr. Eugen Rusakov, Dr. Fernando Moya, Dr. Fabian Wolf, Dr. Philipp Oberdiek, Dr. Oliver Tüselmann, Dominik Koßmann, Arthur Matei, Tim Hallyburton, Tim Raven, and Nilah Ravi Nair. Thank you for the amazing years we spent together! Over the years, you have become more than just colleagues to me. In addition to the many inspiring discussions on research, teaching, and other topics, you made everyday life in the office truly enjoyable. I will always look back on countless great memories from our trips to conferences and summer schools. Furthermore, I will especially remember our wonderful shared experiences outside of work, from our monthly social events to other special occasions we shared as a group. I would particularly like to thank Eugen, who supported me especially in the early years of the research group and helped me find my way into research. I would also like to thank Fabian, who supported me during the late stages of my doctoral research, especially during the writing of this thesis, and who provided valuable feedback on the manuscript.

Finally, I would like to thank my family for their constant encouragement and support. It means a great deal to know that you are always there for me. A very special thank you goes to my partner, Jenny. You've always believed in me and supported me, no matter what. I cannot express how grateful I am to you!

CONTENTS

PUBLICATIONS

[Bra24] K. Brandenbusch. "Semi-Supervised Adaptation of Diffusion Models for Handwritten Text Generation." In: *arXiv: abs/2412.15853* (2024). DOI: 10.48550/arXiv.2412.15853.

This publication presents an initial version of the method proposed in this thesis along with preliminary results. The author of this work developed the proposed system for handwritten text recognition and the semi-supervised training procedure. All experiments were conducted and evaluated by the author of this thesis.

[Tüs+22] O. Tüselmann, K. Brandenbusch, M. Chen, and G. A. Fink. "A Weighted Combination of Semantic and Syntactic Word Image Representations." In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition*. Hyderabad, India, 2022, pp. 285–299. DOI: 10.1007/978-3-031-21648-0_20.

For this publication, the author of this thesis assisted in conducting the experimental evaluation and contributed to the discussion of the experimental results.

[BRF21] K. Brandenbusch, E. Rusakov, and G. A. Fink. "Context Aware Generation of Cuneiform Signs." In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. Lausanne, Switzerland, 2021, pp. 65–79. DOI: 10.1007/978-3-030-86549-8_5.

The author of this work was responsible for designing the generative model, developing the training procedure and conducting all experiments in this publication.

[WBF20] F. Wolf, K. Brandenbusch, and G. A. Fink. "Improving Handwritten Word Synthesis for Annotation-free Word Spotting." In: *Proc. of the Int. Conf. on Frontiers in Handwriting Recognition*. Dortmund, Germany (virtual), 2020, pp. 61–66. DOI: 10.1109/ICFHR2020.2020.00022.

The author of this work contributed to the evaluation and the discussion of the experiments of this publication.

[Rus+19] E. Rusakov, K. Brandenbusch, D. Fisseler, T. Somel, G. A. Fink, F. Weichert, and G. G. W. Mueller. "Generating Cuneiform Signs with Cycle-Consistent Adversarial Networks." In: *Proc. of the Int. Workshop on Historical Document Imaging and Processing*. Sydney, NSW, Australia, 2019, pp. 19–24. DOI: 10.1145/3352631.3352632.

For this publication, the author of this thesis contributed to the development of the proposed generative model and was responsible for conducting a part of the experiments.

NOTATION

When using mathematical expressions in this thesis all variables, functions and arguments are defined where they appear first. The following notation is used for specific mathematical elements:

$a, b, c, \ldots$	scalar
$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$	vector
$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \ldots$	matrix or tensor
$\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$	integer value for, e. g. counts or dimensionalities
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$	set
$\lvert\mathbf{A}\rvert$	the cardinality of set $\mathbf{A}$
$\boldsymbol{a}^{(i)}$	the i-th element in set $\mathbf{S} = \left\{\boldsymbol{a}^{(i)}\right\}_{i=1}^{n}$
$\boldsymbol{a}_i$	the i-th element of vector $\boldsymbol{a}$
$\boldsymbol{A}^T$	the transpose of $\boldsymbol{A}$
$\boldsymbol{A}_i$	the i-th row of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:j}$	the j-th column of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{ij}$	the element of matrix $\boldsymbol{A}$ at i-th row and j-th column
$[\boldsymbol{a}, \boldsymbol{b}]$	the concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$
$\mathcal{X}, \mathcal{V}, \mathcal{H}$	vector space
$\mathbb{R}, \mathbb{R}^{\mathsf{d}}$	space of real numbers or vectors with d dimensions
$\mathcal{I} \subseteq \mathbb{R}^{(\mathsf{h} \times \mathsf{w})}$	space of grayscale images
$\boldsymbol{I} \in \mathcal{I}$	an image
$p, P(x), P(y\lvert x)$	probabilities
$\boldsymbol{x} \in_R \mathbf{X}$	randomly uniformly sampling $\boldsymbol{x}$ from set $\mathbf{X}$
$\mathcal{N}(x; \mu, \sigma^2), \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	univariate and multivariate Gaussian distribution
$\mathcal{U}(a, b)$	uniform distribution of values in $[a, b]$
$f(x), f(\boldsymbol{x})$	scalar function with scalar or vector argument
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$	models or complex processing steps
θ	a set of model parameters
$\mathcal{A}_\theta$	models or complex processing steps parametrized by θ
$\mathcal{L}(\cdot)$	a loss function
t	timestep in the diffusion process
${}^t\boldsymbol{x}$, ${}^t\boldsymbol{X}$	latent vector or matrix at timestep t

If a function $f(\cdot)$ only defined with a support of $\mathbb{R}$ is applied to a vector, matrix, tensor or set, the function is applied elementwise unless stated differently. There may also be multiples of a matrix, tensor or vector that denote similar things. Then, this will be indicated in the exponent, e. g. $\boldsymbol{x}^{(i)}$ or $\boldsymbol{W}^{(l)}$, or with a bold subscript, e. g. $\boldsymbol{W}_{\boldsymbol{X}}$. Sets, scalars, models, functions and sets of parameters are enumerated or marked with a subscript, e. g. y_j, $\mathcal{L}_{MSE}$ or θ_k. Furthermore, we simplify the notation of sequences for better readability such that sequences of scalars are summarized to vectors, e. g. $\boldsymbol{x} = (x_1, x_2, \ldots, x_{\mathsf{N}})^T$. Sequences of vectors are summarized to a matrix $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(\mathsf{N})})^T$ were $\boldsymbol{X}_i = \boldsymbol{x}^{(i)}$ are the rows of the matrix.

We denote a multivariate standard normal distribution as $\mathcal{N}(\mathbf{0}, \mathbf{1})$ where $\mathbf{0}$ is the zero mean vector. To avoid confusion with the notation of an image $\boldsymbol{I}$, we denote the identity covariance matrix as $\mathbf{1}$.

1 INTRODUCTION

Handwriting has served as the primary medium for the preservation and transmission of information. It is used to create documents including personal records, official communication, legal documents, and literature. The creation of these documents involves the utilization of various scripts. Their range extends from scripts like cuneiform, which was utilized as early as the fourth millennium BCE, to contemporary handwriting that is in current use. While there are fundamentally different modern scripts (e. g. Latin, Chinese, or Bangla), writing styles also vary from writer to writer within the same script.

There is a significant interest in the automatic extraction of information from both modern documents and large collections of preserved historical documents, with the objective of automating their searchability. The process of information extraction involves the digitization of documents by scanning or taking an image with a camera, followed by the recognition of its contents. The latter step often requires a prior segmentation of the document into lines or words. In addition to numerous works aiming at implementing these steps with as little manual effort as possible, a research field has emerged that deals with the generation of handwritten words and documents.

Driven by the success of models like *generative adversarial networks* (GANs) and *denoising diffusion probabilistic models* (DDPMs) for the generation of natural images, the task of *handwritten text generation* (HTG) has attracted increasing attention in recent years. The objective of this task is to produce a realistic-looking image that depicts a specified word in handwriting, adhering to a desired calligraphic style. HTG can be utilized to convert digital text into a handwritten document for aesthetic enhancement or to support people who suffer from writing impairments. According to the foregoing definition of HTG, the content and, if applicable, the style of the generated word image are known. This allows for a more promising application, which is the generation of annotated training data for models addressing document analysis tasks like *handwritten text recognition* (HTR). The generated samples can be used to augment the existing training data with more diverse writing styles and an extended vocabulary. Furthermore, HTG facilitates the adaptation of HTR models to new sets of documents characterized by different handwriting styles through the synthesis of training samples with the respective writing styles.

Nevertheless, the task of HTG presents significant challenges. The word in the generated image must be written correctly and readable. This requirement is of particular importance when the generated images are utilized as training data. Furthermore, it is necessary to adhere to the objective of controlling the calligraphic style, thereby ensuring that the generated handwriting mimics the distinctive characteristics of a specific writer. In the process of generating data for a new dataset, the styles to be mimicked are not typically encountered during the training of the HTG model, thereby exacerbating the task.

In this thesis, a system for the generation of training data for an HTR model is presented. In particular, an HTG system based on a conditional *latent diffusion*

model (LDM) is proposed. Conditioning the model on text and writer embeddings from the novel style encoder allows generating writing styles not seen during training. Using the proposed system, the generation of training data for datasets with seen as well as for datasets with new writing styles is investigated. Furthermore, two semi-supervised training schemes that are proposed in this thesis allow the generative model to better adapt to unseen styles and help to considerably improve the generation results. These contributions are explained in more detail in the following section. Subsequently, an outline of this work is provided in Section 1.2.

1.1 CONTRIBUTIONS

This work makes several contributions to the task of HTG. The primary concern of this thesis is the generation of training data, a topic that is often considered a secondary aspect in works addressing HTG. A particular focus lies on the generation of training data for new datasets without annotations. This involves taking advantage of the unlabeled samples by integrating them into the training of the proposed model. Parts of the contributions are published in a technical report [Bra24] from the author of this thesis. The author was responsible for developing the HTG approach and for conducting the experimental evaluation. In the following, contributions from the technical report as well as novelties in this work are discussed in more detail.

HTG system supporting the generation of seen and unseen writing styles

In order to generate training data for another dataset than the one used to train the generative model, the model must be capable of mimicking styles based on example images. Therefore, an LDM that was proposed for HTG [Nik+23] is extended by a novel style encoder to remove the significant limitation in generation to styles from the training set. The style encoder is based on a masked autoencoder that allows training in a self-supervised fashion. Consequently, the necessity for annotations is eliminated, thereby enabling the incorporation of samples from a broader variety of datasets for training the style encoder. The content encoder in [Nik+23] solely processes the text conditioning while the style conditioning is passed separately to the LDM. The author of this thesis extends the content encoder with different methods for entangling the style and text conditioning before passing the combined conditioning to the LDM. The utilization of an LDM in the approach offers the advantage of enhanced computational efficiency, since the diffusion process is executed within the latent space of a pretrained autoencoder.

The generation quality of the proposed HTG model is further improved by *classifier-free guidance* (CFG), which has been shown to be beneficial for conditional generation of natural images [HS21]. However, only two other works take advantage of CFG for their HTG model [Dai+24; May+24]. In [Bra24], it has been shown that CFG is also beneficial for the presented HTG system. The author of this thesis proposed and investigated the improvements to the content encoder and the ability to generate images with new writing styles enabled by the style encoder in a previous work [Bra24].

Utilization of the HTG system for generating training data for HTR models

The generated images from an HTG system can be used in different ways. The conditional generation allows to generate labeled training data for downstream models like HTR models. Besides augmentation of the training set for the dataset the HTG model was trained on, training samples can be generated for new datasets. The latter application is particularly promising as it eliminates the manual creation of an annotated training set, a process that is time-consuming and prone to human error. Especially for DDPM-based HTG methods, this use case has been scarcely explored.

In this thesis, the generation of training data for a downstream HTR model is extensively explored. A comprehensive investigation is conducted into the various hyperparameters of the approach, encompassing the proposed methodologies for entangling style and text as well as CFG. The experiments are conducted with a particular focus on generating training data for a new dataset. The word distribution of a generated dataset is important for training an HTR model [WF24] but might be unknown for the new dataset. Therefore, different choices for words and styles for generating the training set are compared in this thesis. In a previous study [Bra24], the author published results from a part of these experiments that demonstrate the successful application of the proposed model for training data generation. Besides the exploration of more hyperparameters, experiments focussing on the application for different new datasets with distinct characteristics are conducted in this work. The experiments are thoroughly designed to ensure controlled conditions that facilitate the attainment of specific findings. Additionally, scenarios that closely resemble practical applications are investigated. Although the domain gap between the generated samples and the real data is smaller than with synthetic images rendered from computer fonts, it is still present. In order to narrow the performance gap to an HTR model trained on real samples, self-training [WF24] is utilized.

Semi-supervised training scheme for adaptation to new unlabeled datasets

Since the style embedding for conditioning the HTG model is extracted from example images, the presented method is capable of generating images from writers not seen during training. However, differences in the writing styles from the training set of the HTG model and the styles from the new dataset can have a negative impact on the generation quality. This problem can be mitigated, by including unlabeled images from the respective new dataset into the training of the HTG model. As the adversarial objective of GANs allows for an optimization without labels, the inclusion of unlabeled samples into the training was explored in a number of works [Fog+20; ZN21; ZN23; CPK23]. However, for the standard training of conditional DDPMs, labeled samples are required and the inclusion of unlabeled samples in the training has not been explored for HTG methods that use a DDPM.

In this thesis, two methods for incorporating the unlabeled samples from the new dataset into the training of the proposed HTG model in a semi-supervised fashion are presented. The inclusion of samples without annotated transcriptions but with writer IDs was explored by the author in a previous work [Bra24]. In this work, the author extends this approach to allow the inclusion of samples without any labels in the semi-supervised training. Thereby, the model can exploit information about new

styles from these examples and generation quality for the respective writing styles is improved. The specific hyperparameters of both training schemes are extensively evaluated using different benchmark datasets.

1.2 OUTLINE

This thesis is structured in six chapters. The current chapter introduces the problem of HTG, motivates the proposed approach, and highlights the contributions made in this work. The remainder of this thesis is organized as follows:

Chapter 2 - Fundamentals

The proposed HTG method in this thesis is based on artificial neural networks. Therefore, the fundamentals of these models are introduced in this chapter. Following a brief introduction to machine learning and computer vision, the perceptron is described as the basic building block of neural networks. This description is followed by an explanation of multi layer perceptrons and their training. Subsequently, more specific neural architectures like convolutional neural networks, recurrent neural networks and transformers are discussed.

Chapter 3 - Handwritten Text Generation

The subject of HTG has been addressed in a variety of works using different generative models. This chapter first presents important classes of generative models in general. Subsequently, a comprehensive review of the existing approaches for HTG and evaluation measures is presented. Finally, the contributions of existing methods to the generation of training data are discussed.

Chapter 4 - Training Data Generation

This chapter presents the proposed system for training data generation using HTG. Firstly, the problem of HTG is formally introduced. In the following sections, the latent diffusion model that is employed for HTG as well as the proposed style encoder are discussed in detail. Then, methods for training the model in a semi-supervised fashion are presented. Lastly, the HTR model used in this work and a self-training procedure for further refinement of that model are explained.

Chapter 5 - Experimental Evaluation

In this chapter, the proposed method is empirically evaluated through a series of experiments. After explaining the used datasets and evaluation measures, the results are presented in two parts. The first part provides insights in design choices of the proposed system and the semi-supervised training. In the second part, the proposed system is evaluated in a setting close to the practical application and compared to other works from the literature.

Chapter 6 - Conclusions

This chapter summarizes the results and main findings of this thesis. Furthermore, an outlook is provided on future research in the field of HTG for the generation of training data.

Appendix

After the main part of this thesis, additional details about the experimental setup and more detailed results are presented, followed by the bibliography.

2 FUNDAMENTALS

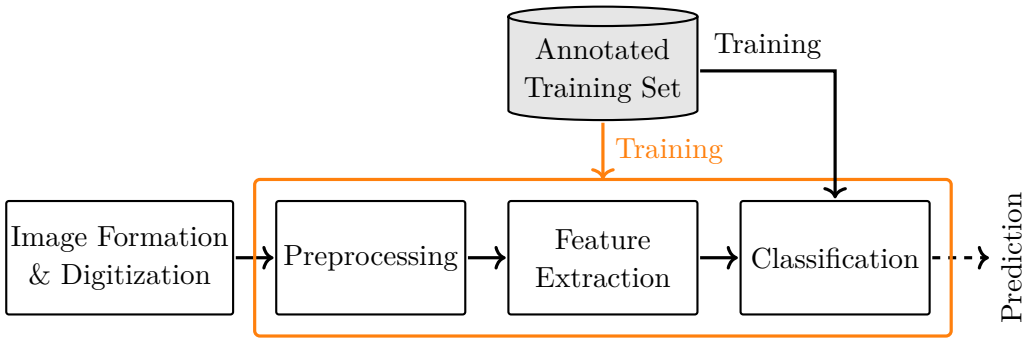


Figure 1: Processing steps in a typical computer vision system. The orange box indicates the steps that are combined and optimized end to end in deep learning models.

In the context of daily life, people are constantly perceiving their environment, subsequently guiding their actions and decisions. Examples include listening to a conversation, determining the taste of fruit in a juice, or identifying friends in photographs. Such processes involve the reception of raw data, so-called *patterns*, and the subsequent assignment of a category [DHS00, p. 1], which largely takes place unconsciously for humans. Therefore, many tasks like the aforementioned examples are very easy for humans. However, developing approaches to reproduce the results of human perception for machines is extremely complex. This process is termed *pattern recognition*. While this term considers all possible input modalities (e. g. visual, acoustic, etc.), the perception of only visual inputs (i. e. images) is referred to as *computer vision* (CV). Typical systems for CV are comprised of a common sequence of processing steps as depicted in Figure 1. In the literature, these steps are referred to by a variety of names and are subdivided at different levels of detail [WC11, pp. 3f.].

Before the first step of the actual CV system can be executed, the digital image to be processed must be captured. In the process of *image formation*, light which is reflected by surfaces in a three-dimensional scene is projected through camara optics onto a two-dimensional sensor [Sze22, p. 63]. The imaging chip then converts the incoming photons with different wavelengths into discrete digital samples to form the digital image.

Given a digitized image, the first step of most CV systems is the *preprocessing* of the image. This step aims at converting and improving the image to be more suitable for the subsequent steps [Sze22, p. 87]. It is important to emphasize that the term "suitable" is vague at this point. Preprocessing is intended to improve the final classification result and can therefore not be optimized independently of the remaining system in its application. Common preprocessing operations are the reduction of noise, the normalization of intensities or the alignment of image contents. Although it is often not described as a separate step, a *segmentation* or *grouping* of patterns in the image may be necessary [DHS00, pp. 10f.]. If multiple instances in an image should be categorized individually, the individual patterns have to be segmented first.

Paradoxically, the categorization of the instances would ease the segmentation. The complementary problem of grouping arises when objects are composed of multiple parts that are not connected (e. g. the letter "i").

After preprocessing, the images usually still contain information which is irrelevant for the categorization of the patterns [DHS00, p. 11]. The goal of *feature extraction* is to extract a numeric representation which helps to distinguish the categories. Features of patterns in the same category therefore should be similar while the features of patterns from different categories should be considerably different. Two fundamental procedures for constructing a feature extractor can be identified [Nie03, p. 165]. Heuristic methods rely on intuition and experience in the field of application. In contrast, analytic approaches compute optimal features with respect to a specified criterion.

The final step is the *classification* of the pattern. In this step, a category or class is assigned to the pattern based on the extracted features. A large variety of techniques is available for this purpose. Prominent examples are support vector machines, decision trees, and artificial neural networks [WC11, pp. 3–6]. Therefore, a *model selection* is required, encompassing the selection of the type of the classifier, its specific structure, and its capacity. At the same time, each classifier comes with a set of internal parameters, which have to be adapted to the respective classification problem. This is done by a learning algorithm, which optimizes these parameters based on a so-called *training set*. The training set comprises patterns that have been *annotated* with their known class.

The process of designing and training a classifier is accompanied by a number of challenges [WC11, p. 6]. Choosing a simple classifier (i. e. low number of trainable parameters), the classifier might not be capable to capture the underlying distributions of the training data. This problem is termed as *underfitting*. At the same time, choosing a complex classifier (i. e. high number of trainable parameters) might lead to the model *overfitting* the training set. In such a case, the accuracy is close to perfect on the training samples but highly erroneous for novel examples. However, the ultimate objective is to predict categories for novel patterns. This issue is referred to as the *generalization* performance of a classifier. In addition to the selection of an appropriate model, the collection of a *representative* training set is crucial for achieving high generalization performance [WC11, p. 7].

In the traditional computer vision, all the aforementioned steps are designed by hand [Sze22, p. 189]. Later, the classifier and, in the case of analytical approaches, the feature extraction are optimized based on the training samples. However, the methods and models for these steps have to be selected by hand. *Deep learning* facilitates the optimization of all steps, including preprocessing to a certain extent, based on training samples in an end-to-end fashion. This is achieved through the integration of the entire pipeline into one trainable model.

In addition to image classification, other prevalent tasks in CV include, for example, object detection, semantic segmentation, and image captioning [BB24, pp. 288f.]. Given the varying modality of the desired outputs (i. e. classes, coordinates, images, text), it is necessary to adjust the representation of the output accordingly. In a more general sense, the training examples are referred to as pairs of inputs and targets. The learning algorithm considered in the preceding description requires such pairs of inputs (e. g. an image) and targets (e. g. a class label). This approach to training a model is referred to as *supervised learning* [Sze22, p. 205]. Not in all cases,

a dataset with annotated targets is available. However, there are methods that learn to extract information like characteristics or distributions in such cases. The training of models from unlabeled data is termed *unsupervised learning*. A special case of unsupervised learning is *self-supervised learning*. Models are trained with labels that are not provided with the dataset. Instead, these labels are automatically derived from the unlabeled images [Sze22, p. 249]. While supervised and unsupervised learning represent the extreme cases, there are many scenarios in which both annotated and unlabeled data are available. In *semi-supervised learning*, unlabeled images are used to extend an annotated training set or vice versa to include additional information [ZG09, p. 9].

The method presented in this work mainly relies on artificial neural networks. Therefore, the fundamentals of these models will be explained in more detail in the remainder of this chapter. Beginning with the description of the perceptron, Section 2.1 provides details about the structure of a multi layer perceptron and its training. Different types of layers that are the main components of convolutional neural networks are described in Section 2.2. While the aforementioned architectures are specifically designed for structured data, recurrent neural networks that were developed for sequential data are discussed in Section 2.3. As a key technique in state-of-the-art models, different forms of the attention mechanism are explained in Section 2.4.

2.1 ARTIFICIAL NEURAL NETWORKS

The majority of the current CV systems are based on deep learning. However, the development of *artificial neural networks* (ANNs) already began in the mid-twentieth century. Early approaches were inspired by the information processing in biological neural networks like the human brain [BB24, p. 16]. Information is processed by transmitting electrical impulses through a complex network of basic processing units, so-called *neurons*. Simplified, incoming stimulating or inhibiting signals from connected neurons determine whether a neuron gets *activated* and outputs stimulating or inhibiting signals to subsequent neurons. While initial works developed mathematical models closely inspired by these processes, later works started to develop a more general foundation of ANNs beyond the neurobiological inspiration [BB24, p. 19]. This section begins with an explanation of early models, of which the perceptron is considered to be the most important. These basic building blocks can be combined to construct a multi layer perceptron which can then be trained for a variety of tasks.

2.1.1 *Perceptron*

The mathematical model which is considered one of the first artificial neurons was proposed by McCulloch and Pitts [MP43]. The neuron receives n binary inputs $x_1, x_2 \ldots, x_n$. The binary output is computed by comparing the sum of the inputs against a defined threshold $\tau \in \mathbb{N}$. Formally, the computation can be described by

$$f(x_1, x_2 \ldots, x_n) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} x_i \geq \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

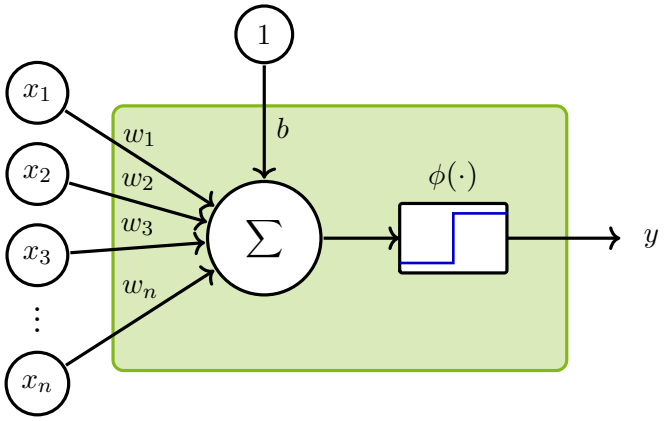


Figure 2: Visualization of the generalized version of the perceptron. The activation y is computed applying the activation function $\phi(\cdot)$ to the weighted sum of the inputs x_i with weights w_i and the bias b.

Rosenblatt extended this definition to include a weight for each input and a bias which is added to the sum [Ros58]. Furthermore, he presented a learning algorithm to adjust these weights. However, the processing of the perceptron by Rosenblatt was still limited to binary inputs. In 1969, the model was extended to work with real-valued inputs by Minsky and Papert [MP69]. For an input vector $\boldsymbol{x} \in \mathbb{R}^n$, weights $\boldsymbol{w} \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$, the generalized version of the perceptron computes

$$a = \sum_{i=1}^{n} w_i \cdot x_i + b, \tag{2}$$

$$y = \phi(a). \tag{3}$$

This common notation differentiates between the weighted sum a, which is also referred to as *pre-activation*, and the so-called *activation* y [BB24, p. 17]. $\phi(\cdot)$ is called the *activation function*. Figure 2 visualizes the structure of the perceptron. In the case of the perceptron presented by Rosenblatt [Ros58], the activation function is the step function

$$\phi(a) = \begin{cases} 1, & \text{if } a > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The importance of the activation function and the most common choices will be discussed later. Even though the activation function is non-linear, the perceptron is a linear model [GBC16, p. 14]. Therefore, it is not capable of solving classification problems which are not linearly separable like the *XOR* problem. This significant restriction was shown by Minsky and Papert [MP69] as part of their studies on the perceptron.

2.1.2 *Multi-Layer Perceptron*

The restriction to the solution of linearly separable problems can be circumvented by transforming the inputs using non-linear basis functions [BB24, pp. 171f.]. However, designing and choosing a suitable set of basis functions comes with significant limitations. It is therefore desirable to learn these functions from training data together with the perceptron. The most successful approach has been to use basis functions

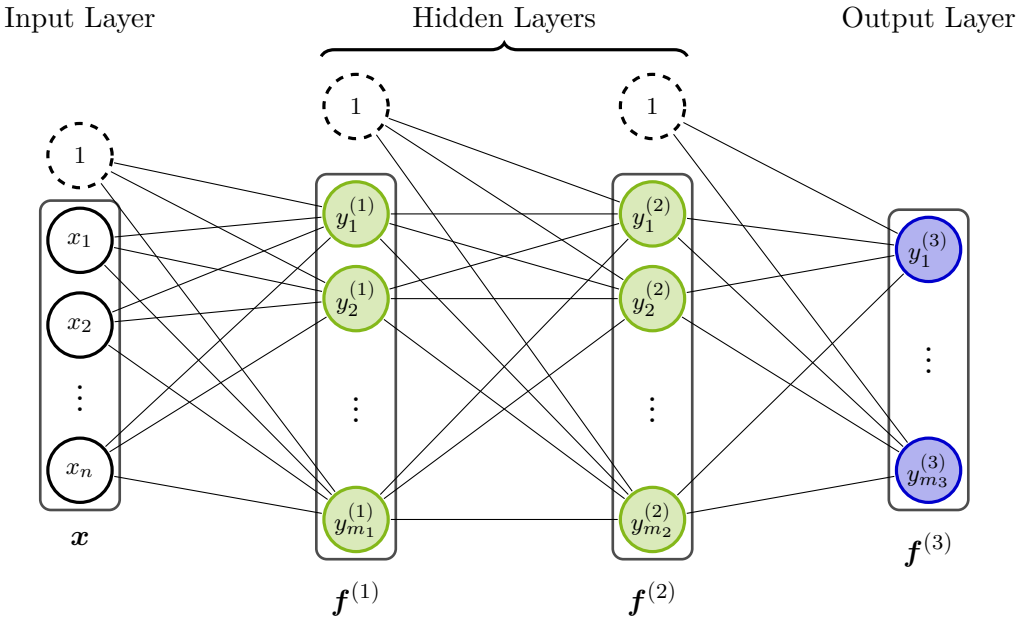


Figure 3: Example of an MLP with two hidden layers which are marked green. The neurons of the output layer are marked blue.

with the same form as the perceptron (cf. Equations 2 and 3) [BB24, p. 180]. This approach involves stacking multiple *layers* of perceptrons, which results in the name *multi layer perceptron* (MLP). An MLP comprises an *input layer*, at least one *hidden layer* and an *output layer*. Figure 3 shows an example for an MLP with two hidden layers. The layers are indexed by superscript (l) with $l = 1, \dots, L$ and L being the index of the output layer. L is also referred to as the *depth* of the model. Each layer can contain a different number of perceptrons which is specified by m_l and determines the *width* of the model. One important property of an MLP is the way its neurons are connected. Each perceptron in layer l receives the outputs of all perceptrons from layer $l-1$ as inputs. Consequently, this layer is often termed *fully connected*. It is important to note that there are no feedback connections in an MLP. Networks passing the outputs from one layer only to subsequent layers are called *feed forward neural networks*. Models that contain feedback connections will be discussed in Section 2.3.

Formally, the computation of an MLP can be written as a chain of the functions which are defined by the individual layers. Equations 2 and 3 can be rewritten using a dot product:

$$y = \phi(\boldsymbol{w}^T \boldsymbol{x} + b). \tag{5}$$

Layer l is then defined by a function $\boldsymbol{f}^{(l)}(\boldsymbol{y}^{(l-1)})$ comprising multiple parallel perceptrons:

$$\boldsymbol{y}^{(l)} = \boldsymbol{f}^{(l)}\left(\boldsymbol{y}^{(l-1)}\right) = \phi\left(\boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}\right). \tag{6}$$

The weights and biases of all m_l perceptrons are summarized in $\boldsymbol{W}^{(l)} \in \mathbb{R}^{(m_l \times m_{l-1})}$ and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{m_l}$, respectively[1]. The weight of the connection from neuron j in layer $l-1$ to neuron i in layer l is then given by $\boldsymbol{W}_{ij}^{(l)}$. Given an input vector $\boldsymbol{x}$ and

1 Without an activation function ϕ, the linear transformation $\boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)}$ is often referred to as *linear layer*.

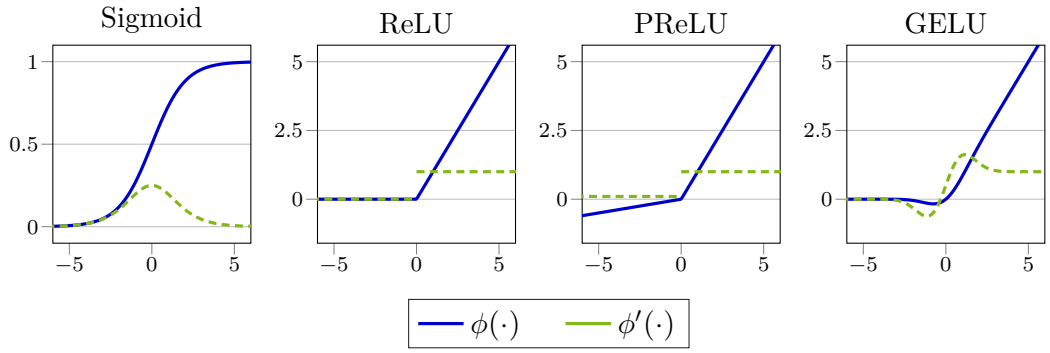


Figure 4: Examples of different activation functions $\phi(\cdot)$ and their derivatives $\phi'(\cdot)$. The slope of the PReLU for negative values of the argument is set to to 0.1 in the plot.

the three layers $\boldsymbol{f}^{(1)}$, $\boldsymbol{f}^{(2)}$ and $\boldsymbol{f}^{(3)}$ from the example in Figure 3, the output $\boldsymbol{y}$ is computed by

$$\boldsymbol{y} = \boldsymbol{f}^{(3)}\Big(\boldsymbol{f}^{(2)}\Big(\boldsymbol{f}^{(1)}\left(\boldsymbol{x}\right)\Big)\Big). \tag{7}$$

The main motivation for using an MLP instead of a perceptron is to extend modeling capabilities to non-linear problems. Achieving this objective necessitates the implementation of non-linear activation functions. Without non-linear activation functions, the network would be a composition of successive linear transformations which itself can be summarized in a single linear transformation [BB24, p. 183]. Additionally, an activation function is required to be differentiable, since the learning algorithm is based on gradients. In practice, however, activation functions that are not differentiable at certain input points can also be used [GBC16, p. 188].

In early works that used ANNs, the logistic sigmoid was a common choice as the activation function. It is applied to every pre-activation a independently and is defined as follows:

$$\phi(a) = \sigma(a) = \frac{1}{1 + \exp(-a)}. \tag{8}$$

A plot of the function and its derivative is shown in Figure 4 (left). The usage of the sigmoid activation function, however, comes with a significant drawback. For both, large positive and negative values, the gradients exponentially become very small. This results in *vanishing gradients* in the learning algorithm [BB24, p. 184]. This problem has been tackled by the introduction of the *rectified linear unit* (ReLU) which has become the most common activation function in state-of-the-art ANNs. The activation function is defined as

$$\phi(a) = \max(0, a). \tag{9}$$

A constant gradient of 1 for all positive inputs does not only avoid vanishing gradients for positive inputs but also results in more useful gradient directions in training [GBC16, p. 189]. However, the ReLU comes with some drawbacks. The derivative at $a = 0$ is not defined, which, however, can be ignored in practice [BB24, p. 185]. Another shortcoming is that the gradients for $a \leq 0$ are zero which inhibits gradient-based adjustments of the weights in these cases [GBC16, p. 189]. This problem is tackled by other activation functions that are inspired by the ReLU. The *leaky ReLU* [MHN13] avoids this problem by a small slope of 0.01 for negative values of a:

$$\phi(a) = \max(0, a) + 0.01 \cdot \min(0, a). \tag{10}$$

The approach was later generalized by replacing the fixed slope for negative values by a learnable parameter [He+15]. The resulting activation function is called *parametric ReLU* (PReLU).

Motivated by stochastic transformations of the input, the *Gaussian error linear unit* (GELU) was proposed [HG16]. The activation function is defined by

$$\phi(a) = a \cdot \Phi(a), \tag{11}$$

where $\Phi(a)$ is the cumulative distribution function of a normal distribution. Alternatively, the cumulative distribution function of a logistic distribution (i. e. the sigmoid) can be used for $\Phi(a)$. In the latter case, the activation function is also referred to as *sigmoid linear unit* (SiLU).

All previously described activation functions are applied individually to each pre-activation. For multiclass classification, we want the model to predict an output in a one-hot scheme. This means that for k mutually exclusive classes, the model should predict a vector $\boldsymbol{y} \in \mathbb{R}^k$ with $\boldsymbol{y}_i = 1$ for the target class with index i and zero for all other components of $\boldsymbol{y}$. In these cases, it is common to use the *softmax* function for the output layer [BB24, p. 197]:

$$\boldsymbol{y}_i = \boldsymbol{\phi}_i(\boldsymbol{a}) = \frac{\exp(\boldsymbol{a}_i)}{\sum_{j=1}^{k} \exp(\boldsymbol{a}_j)}. \tag{12}$$

It is important to note that the activation of neuron i depends on the pre-activations of all neurons in the respective layer. The softmax function enforces an output where $0 < y_i < 1, \forall i = 1, \ldots, k$ and $\sum_{j=1}^{k} y_j = 1$. Due to these properties, the output of the softmax is also considered to be a pseudo-probability distribution of the predicted classes.

The process of designing an MLP involves a number of choices like the number of layers, the number of neurons in each layer or the activation function. However, it was shown that a network with at least one hidden layer can approximate any real-valued multivariate function with arbitrary precision. This result is known as the *universal approximation theorem* [HSW89]. The fundamental prerequisite for this process is the presence of a sufficient number of hidden neurons within the model [GBC16, pp. 194f.]. Unfortunately, the theorem does not state how many hidden units are required. Furthermore, the learning algorithm does not guarantee that the model learns this approximation even if it is able to represent the function. Therefore, the practical implications of the theorem are limited.

2.1.3 *Training*

As discussed in the previous section, MLPs are universal function approximators. The aim of training an MLP can be formally defined by learning a function $\boldsymbol{f}(\boldsymbol{x})$ which approximates a target function $\boldsymbol{f}^*(\boldsymbol{x})$. For example, in multiclass classification, the goal is to learn a mapping $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ where $\boldsymbol{y}$ encodes the prediction of the class of the pattern $\boldsymbol{x}$. In the following, the learnable parameters (i. e. weights and biases) are summarized by θ and the function computed by the model is denoted as $\boldsymbol{f}_\theta(\cdot)$. The goal of training the model is to find a set of parameters θ^* to approximate the target function.

The learning algorithm for the perceptron proposed by Rosenblatt [Ros58], however, is only applicable to single layer networks [BB24, p. 18]. Training of MLPs became possible with the application of gradient-based optimization methods. The prediction of the model is compared to the desired output using an error function which is also referred to as a *loss function*. Subsequently, each parameter of the network is adjusted according to the derivative of the loss with respect to the parameter. The *back-propagation* algorithm was proposed by Rumelhart et al. [RHW86] and allows an efficient evaluation of the derivatives through all layers.

In supervised learning, this optimization is based on a representative, annotated training set $\mathbf{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{\mathsf{N}}$ with $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^k$. Given a loss function $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \in \mathbb{R}$, the error can be evaluated between the target $\boldsymbol{y}$ and the prediction $\hat{\boldsymbol{y}} = \boldsymbol{f}_\theta(\boldsymbol{x})$ by the MLP. The goal of the training is to adjust the parameters θ to minimize the accumulated error for the training set. However, the composition $\mathcal{L}(\boldsymbol{f}_\theta(\boldsymbol{x}), \boldsymbol{y})$ is usually an extremely complex function. Therefore, finding an analytical solution is considered infeasible and *gradient descent* is used for an iterative numerical optimization instead [BB24, p. 213]. The parameters θ are updated by taking steps in the direction of the negative gradient of the loss function:

$$\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_{\theta^{(t-1)}} \mathcal{L}(\boldsymbol{f}_{\theta^{(t-1)}}(\boldsymbol{x}), \boldsymbol{y}). \tag{13}$$

The parameter $\eta > 0$ is called *learning rate* and determines the step size of the parameter update [BB24, p. 214]. The superscript (t) indicates the number of the update step in this iterative optimization. Since gradient descent is an iterative optimization approach, it requires an initialization, i. e. an initial set of parameters $\theta^{(0)}$. The most commonly used approach is a random initialization of the weights which requires the choice of a suitable distribution. For layers using ReLU activation, the initialization proposed by He et al. [He+15] has become very popular. The initialization is based on a normal distribution with zero mean and a standard deviation of $\sqrt{2/m_l}$ where m_l denotes the number of hidden units in the respective layer.

Since the MLP is a composition of functions, the computation of the gradients for individual parameters $\boldsymbol{W}_{ij}^{(l)}$ using the back-propagation algorithm requires the application of the chain rule. For a composed function $h(g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x}))$, the chain rule is defined by:

$$\frac{\partial h}{\partial \boldsymbol{x}_i} = \sum_{j=1}^{k} \frac{\partial h}{\partial g_j} \cdot \frac{\partial g_j}{\partial \boldsymbol{x}_i}. \tag{14}$$

By applying the chain rule, the partial derivative of the loss function with respect to parameter $\boldsymbol{W}_{ij}^{(l)}$ can be decomposed as follows:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{f}_i^{(l)}} \cdot \frac{\partial \boldsymbol{f}_i^{(l)}}{\partial \boldsymbol{a}_i^{(l)}} \cdot \frac{\partial \boldsymbol{a}_i^{(l)}}{\partial \boldsymbol{W}_{ij}^{(l)}}, \tag{15}$$

where $\boldsymbol{a}_i^{(l)}$ is the pre-activation with $\boldsymbol{a}_i^{(l)} = \sum_{j=1}^{m_{l-1}} \boldsymbol{W}_{ij}^{(l)} \cdot \boldsymbol{f}_j^{(l-1)} + \boldsymbol{b}_i^{(l)}$. The second and third terms are computed in the same way for all layers l. The second term is the partial derivative of the activation with respect to the pre-activation and boils down to the derivative of the activation function:

$$\frac{\partial \boldsymbol{f}_i^{(l)}}{\partial \boldsymbol{a}_i^{(l)}} = \frac{\partial \boldsymbol{\phi}(\boldsymbol{a}_i^{(l)})}{\partial \boldsymbol{a}_i^{(l)}} = \phi'(\boldsymbol{a}_i^{(l)}). \tag{16}$$

At this point, the necessity of the activation function being differentiable becomes evident. According to Equation 2, the third term can be rewritten as the activation of the preceding layer $l-1$:

$$\frac{\partial \boldsymbol{a}_i^{(l)}}{\partial \boldsymbol{W}_{ij}^{(l)}} = \frac{\partial}{\partial \boldsymbol{W}_{ij}^{(l)}} \left[\sum_{k=1}^{m_{l-1}} \boldsymbol{W}_{ik}^{(l)} \cdot \boldsymbol{f}_k^{(l-1)} + \boldsymbol{b}_i^{(l)} \right] = \boldsymbol{f}_j^{(l-1)}. \tag{17}$$

In case of $l-1=0$, $\boldsymbol{f}^{(l-1)}$ is the input $\boldsymbol{x}$ of the MLP.

The computation of the first term differs for the output layer from that of the hidden layers. For the output layer (i. e. layer L), this term is the partial derivative of the loss function with respect to the activations $\boldsymbol{f}_i^{(L)}$ of the output layer. It is therefore dependent on the chosen error function which is required to be differentiable. For the hidden layers $l < L$, this term is defined recursively based on the gradients from the respective subsequent layer $l+1$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{f}_i^{(l)}} = \sum_{j=1}^{m_{l+1}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{f}_j^{(l+1)}} \cdot \frac{\partial \boldsymbol{f}_j^{(l+1)}}{\partial \boldsymbol{a}_j^{(l+1)}} \cdot \frac{\partial \boldsymbol{a}_j^{(l+1)}}{\partial \boldsymbol{f}_i^{(l)}}. \tag{18}$$

Given Equation 16, the second term evaluates to the derivative of the activation function. The third term is the partial derivative of pre-activation $\boldsymbol{a}_j^{(l+1)}$ (i. e. the weighted sum) with respect to its input $\boldsymbol{f}_i^{(l)}$ which boils down to the respective weight $\boldsymbol{W}_{ij}^{(l)}$. The first term can be computed by recursively applying Equation 18. This recursion from the last to the first layer serves as the underlying inspiration for the name of the algorithm: back-propagation.

There exist different approaches on how to use the training samples for the weight update in Equation 13 [BB24, pp. 214–216]. One approach is to compute the loss and its gradient for the entire training set each update step. This procedure is referred to as *batch gradient descent*. However, processing the whole training set becomes inefficient or even infeasible for large datasets. *Stochastic gradient descent* (SGD) [Bot10] tackles this problem by computing the error for only a single training sample per update step. This approach, however, only provides a very noisy estimate of the error for the whole training set. The estimation can be improved by computing the gradient based on small random sets of training samples instead, which are referred to as *mini batches*. The latter two approaches both require iterating over the training set. A complete pass over the entire training dataset is called *epoch*.

Besides a representative training set and the design of a suitable architecture, a fundamental requirement for the successful training by gradient descent is a careful choice of the learning rate η. Values that are too high can result in oscillation and divergence of the optimization [BB24, p. 218]. At the same time, values that are too small lead to a very inefficient training. There are several approaches enhancing the convergence behavior of the optimization. *Momentum* can be considered in the updates during the gradient descent to help reduce oscillations [BB24, p. 220]. Another improvement can be made by the introduction of a *learning rate schedule* [BB24, p. 222]. At the beginning of the training, a high learning rate is used that is reduced as the optimization progresses. The idea of adapting the learning rate during training motivated the research of more elaborate training algorithms like *AdaGrad* [DHS11] and *RMSProp* [TH12]. RMSProp and the idea of using momentum were combined in the *Adam* optimization algorithm [KB14]. By decoupling

the regularization mechanisms, Loshchilov and Hutter [LH19] proposed *AdamW* as an improved version of Adam.

2.2 CONVOLUTIONAL NEURAL NETWORKS

In CV, application tasks involve the analysis of image data. In an image $\boldsymbol{I}$, information is organized in a grid of pixels. Depending on whether the image is grayscale, color, or comprises measurements of other electromagnetic radiation e. g. X-Ray, each pixel contains a single value, a triplet (e. g. RGB), or any other number of channels. In theory, MLPs could be applied to work on images. However, they are not able to take advantage of the highly structured nature of image data [BB24, p. 289]. Furthermore, images usually have a high resolution (e. g. several million pixels) resulting in a high dimensional representation. Due to the layers of an MLP being fully connected, models would contain a huge number of parameters, thus making training infeasible.

With the development of *convolutional neural networks* (CNNs), inductive biases based on the structured nature of image data were introduced to ANNs. Thereby, the number of required model parameters could be significantly reduced while enhancing the generalization performance of the models [BB24, p. 289]. A first application of CNNs was shown by LeCun et al. [LeC+89] for recognizing handwritten digits on 28×28 pixel images. Most CNNs share a common structure comprising a stack of *convolutional* and *pooling* layers followed by an MLP. These layers are described in detail in Sections 2.2.1 and 2.2.2. In Section 2.2.3, improvements of the CNN architecture by residual connections and normalization layers are discussed.

2.2.1 *Convolutional Layers*

An operation which allows to take advantage of the structured nature of images and to reduce the number of parameters per layer is *convolution*. In general, the convolution operation $i * k$ is defined for two continuous real-valued functions $i : \mathbb{R} \mapsto \mathbb{R}$ and $k : \mathbb{R} \mapsto \mathbb{R}$ as follows [GBC16, p. 327]:

$$f(t) = (i * k)(t) = \int i(x)k(t - x)\mathrm{d}x, \tag{19}$$

with $x \in \mathbb{R}$ and $t \in \mathbb{R}$. The function $i(x)$ is often referred to as the *input* and $k(x)$ is termed *kernel*.

In the case of image data, however, the input is two-dimensional, discrete and finite. For discrete functions $i : \mathbb{Z} \mapsto \mathbb{R}$ and $k : \mathbb{Z} \mapsto \mathbb{R}$, the convolution operation is defined by [GBC16, p. 328]:

$$f(t) = (i * k)(t) = \sum_{-\inf}^{\inf} i(x)k(t - x), \tag{20}$$

with $x \in \mathbb{Z}$ and $t \in \mathbb{Z}$. A given image $\boldsymbol{I} \in \mathbb{R}^{(\mathsf{h} \times \mathsf{w})}$ can then be modeled as a function $I(i, j)$ which returns the pixel values at position (i, j). Outside the spatial dimensions of the image (i. e. h and w), the function is assumed to return zero. A similar

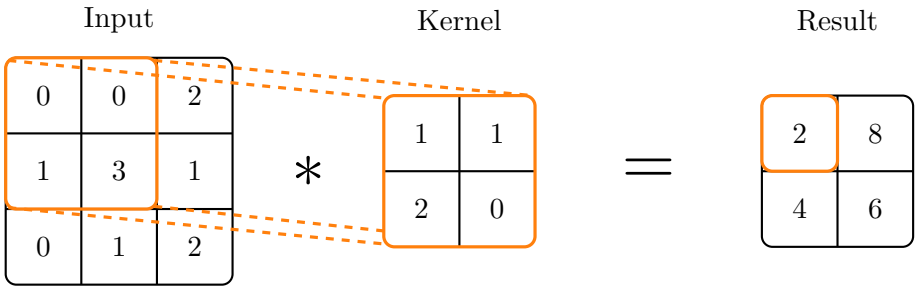


Figure 5: Example of a two-dimensional convolution operation for a discrete and finite 3×3 input image and a 2×2 kernel. For simplicity, the kernel is not flipped in this example. Therefore, the operation is a *cross-correlation* instead of a convolution as it is often used in ANN frameworks [GBC16, p. 329].

definition applies to kernel $K(i, j)$. The application of the discrete convolution along both dimensions is then given by [GBC16, p. 328]:

$$F(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n). \tag{21}$$

The result $F(i, j)$ of the operation is commonly referred to as a *feature map*. An example of this operation for a small 3×3 input image and a 2×2 kernel is presented in Figure 5. In the context of neural networks, each kernel can be considered a neuron with the respective parameters. Since the kernels are usually chosen to be smaller than the input [BB24, p. 294], there are only a few connections between the input and the neurons. Furthermore, these parameters are shared in the computations over all spatial positions in the input. Compared to a fully connected layer, the *sparse interactions* and *parameter sharing* result in significantly fewer parameters per neuron [GBC16, pp. 329f.]. At the same time, the parameter sharing results in an *equivariance to translation* [GBC16, p. 334].

For an image $\boldsymbol{I}$ with a finite size $\mathsf{h} \times \mathsf{w}$ and a kernel with a size $\mathsf{k} > 1$ the resulting feature map is smaller than the input image. The size $\mathsf{h}_{out} \times \mathsf{w}_{out}$ of the feature map can be computed by $\mathsf{h}_{out} = \mathsf{h} - \mathsf{k} + 1$ and $\mathsf{w}_{out} = \mathsf{w} - \mathsf{k} + 1$ for a kernel with size $\mathsf{k} \times \mathsf{k}$ [BB24, p. 294]. *Padding* can be used to extend the image size by appending p pixels to all borders of the image resulting in a feature map with the size $(\mathsf{h} - \mathsf{k} + 2\mathsf{p} + 1) \times (\mathsf{w} - \mathsf{k} + 2\mathsf{p} + 1)$. In order to maintain the size of the image after convolution, p needs to be set to $(\mathsf{k} - 1)/2$. The value of these pixels is typically set to zero.

In some applications, it can be advantageous to move the filter more than one pixel at a time. Therefore, the stepsize is adjusted by a parameter s called *stride*. The size of the resulting feature map is then calculated as follows [BB24, p. 294]:

$$\mathsf{h}_{out} = \left\lfloor \frac{\mathsf{h} - \mathsf{k} + 2\mathsf{p}}{\mathsf{s}} - 1 \right\rfloor \quad \text{and} \quad \mathsf{w}_{out} = \left\lfloor \frac{\mathsf{w} - \mathsf{k} + 2\mathsf{p}}{\mathsf{s}} - 1 \right\rfloor, \tag{22}$$

where $\lfloor \cdot \rfloor$ denotes flooring to the next lower integer.

The previous explanation assumed an input with only one channel which results in a single feature map. In order to be applicable to an input with $\mathsf{c} > 1$ channels (e. g. an RGB image), the dimensions of the kernel must be extended to $\mathsf{k} \times \mathsf{k} \times \mathsf{c}$. $\mathsf{k} \times \mathsf{k}$ individual parameters are learned for each input channel. The convolution with such a filter results in a single feature map and is analogous to a single neuron in a

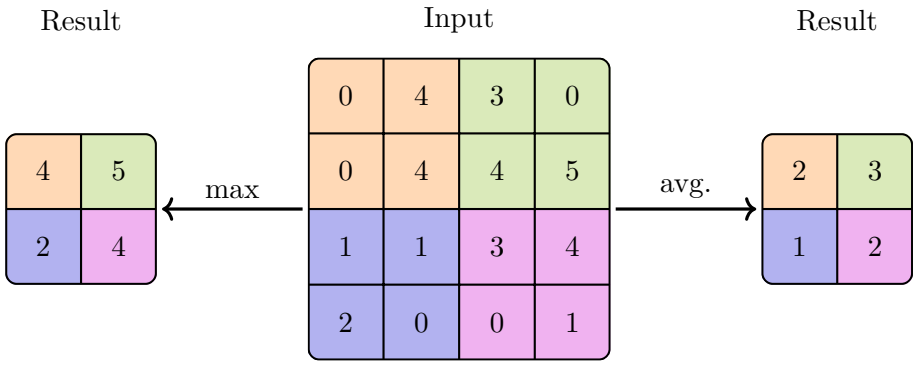


Figure 6: Visualization of *max-pooling* and *average pooling* with a window size of 2×2 and a stride of 2.

fully connected layer [BB24, p. 296]. To compute $\mathsf{c}_{out} > 1$ feature maps, the kernel needs to be further extended to a size of $\mathsf{k} \times \mathsf{k} \times \mathsf{c} \times \mathsf{c}_{out}$.

2.2.2 *Pooling Layers*

Besides convolutional layers, another essential operation for the hierarchical feature extraction in CNNs is *pooling*. Two main reasons for including pooling layers in the architecture are *local translation invariance* and *dimensionality reduction* [BB24, p. 297]. The operation works in a similar sliding window fashion like the discrete convolution. Similar to the kernel size, a parameter defines the size of the sliding window. The stride parameter determines the step size for moving the window. However, instead of aggregating the products of input values and learnable parameters, pooling applies a fixed function for aggregation. A common choice is the application of the *max* function on the inputs within the window. This type of pooling is referred to as *max-pooling* [ZC88]. One alternative is to compute the average of the inputs which is termed *average pooling* [BB24, p. 297]. Examples for both operations are provided in Figure 6. Given that input values in a feature map represent the strength of detecting a feature, these operations discard the positional information within the window but preserve information about the strength [BB24, p. 297]. Thereby, pooling achieves a local translation invariance, i. e. the representation becomes invariant to small translations of the input. This property is especially desirable for classification tasks.

Typically, each channel of a feature map is processed independently by pooling [BB24, p. 297]. While thereby maintaining the number of channels, a reduction in spatial dimensionality of the feature maps can be achieved by using a stride > 1. As fewer inputs are passed to the subsequent layer, this reduction improves the computational efficiency of the model [GBC16, p. 337].

As an alternative to the choice of a fixed window size and a fixed stride, these parameters can be determined according to the size of the input [BB24, p. 298]. Thereby, outputs of a fixed size can be computed for variable-sized inputs. This is particularly necessary if subsequent layers require a fixed size input (e. g. an MLP).

2.2.3 *Architecture Improvements*

The aforementioned layers comprise the main building blocks of CNNs. Nonetheless, techniques and modifications of the CNN architecture exist which aim to stabilize and accelerate the training process. One of these techniques is *normalization* that is essential for an effective training of ANNs in practice [BB24, p. 224]. There exist different forms of normalization. The input samples in the training set are often normalized to a fixed range or to have zero mean and unit variance. However, the update of the weights during the training process leads to changes in the distribution of the inputs in the subsequent layers which is referred to as *internal covariate shift* [IS15]. Ioffe and Szegedy, therefore, proposed to perform *batch normalization* (BatchNorm). In this operation, the intermediate activations are normalized to zero mean and unit variance per mini-batch during the training. In order to not limit the representational capability, two learnable parameters allow re-shifting and re-scaling the normalized activations. Initially, BatchNorm was proposed to speed up the training by reducing internal covariate shift. In later works, it has been shown that BatchNorm helps smooth the optimization landscape, which is a more important factor for accelerating the training process [San+18b].

A shortcoming of the BatchNorm is that the computation of the mean and variance for a mini-batch becomes noisy for small batch sizes [BB24, p. 229]. Instead of computing the activation statistics for a single neuron across a mini-batch, the mean and variance are computed across all neurons in the layer. This approach is termed *layer normalization* (LayerNorm) [BKH16]. The normalization across the neurons in one layer also enables the normalization of the activation per timestep in recurrent neural networks. A similar approach is used by *group normalization* (GroupNorm) [WH18]. In contrast to LayerNorm, in GroupNorm, mean and variance are computed across groups of channels instead of all channels. Thereby, GroupNorm has an improved representational power compared to LayerNorm [WH18].

In addition to normalization, *residual learning* is a technique utilized in the majority of state-of-the-art ANNs. The representational power of ANNs is mainly attributed to their hierarchical architecture comprising a stack of multiple layers of processing [BB24, p. 274]. Evidence for the importance of the depth of a network was provided by studies like [SZ15]. However, by adding more layers, training the models becomes increasingly difficult [BB24, p. 274]. This problem can be tackled by introducing so-called *residual connections* [He+16]. Instead of learning a mapping $\boldsymbol{h}(\boldsymbol{x})$, layers are trained to predict the residual $\boldsymbol{f}(\boldsymbol{x}) \coloneqq \boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{x}$. The idea of residual learning is implemented by adding shortcut connections in parallel to the respective layers:

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{x}. \tag{23}$$

In this formulation, $\boldsymbol{f}(\boldsymbol{x})$ can be implemented by a single layer or by a stack of multiple layers. A stack of layers including the residual connection is commonly referred to as a *residual block*. With the help of residual connections, networks with more than 100 layers were successfully trained [He+16]. Later, the success of this technique was explained by findings that residual connections prevent gradients in deep networks to resemble white noise [Bal+17].

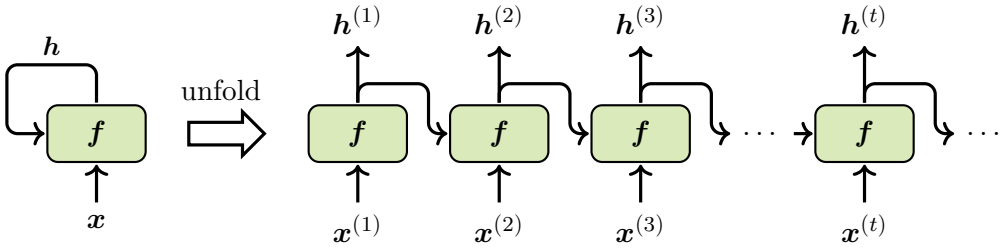


Figure 7: Unfolding of an RNN cell over timesteps t.

2.3 RECURRENT NEURAL NETWORKS

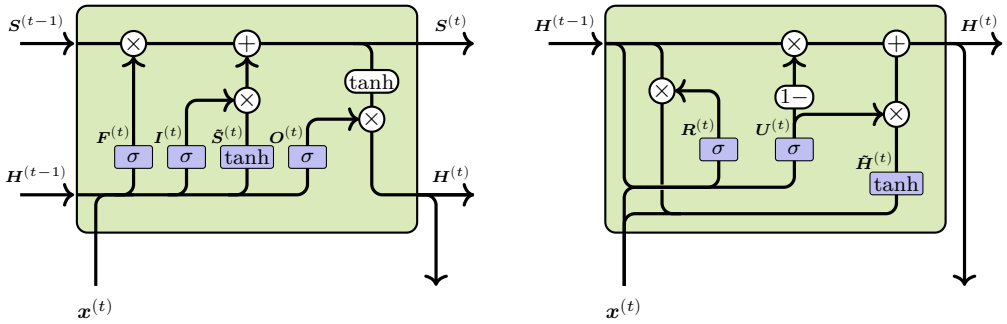
Similar to the inductive bias of CNNs for the processing of image data, ANNs were developed that are specialized for processing data with a sequential structure. A family of ANNs for this purpose are *recurrent neural networks* (RNNs) [RHW86]. Again, one key concept of these models is parameter sharing. Instead of sharing parameters for various spatial positions, RNNs share parameters over timesteps in a sequence. Thereby, most of these models are capable of processing sequences of variable length [GBC16, p. 367].

The input to the model is given as a sequence $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(T)})$ of vectors $\boldsymbol{x}^{(t)}$ where the superscript t denotes the timestep (i. e. position in the sequence). The computation of a typical RNN can then be described by

$$\boldsymbol{h}^{(t)} = \boldsymbol{f}\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}\right), \tag{24}$$

where $\boldsymbol{h}^{(t)}$ represents the *internal state* of the RNN [GBC16, p. 370]. This state can be used by the model to store information while processing the sequence. One problem with this internal state is its limited capacity in contrast to the input sequence that can have an arbitrary length. A possible implementation of the function $\boldsymbol{f}(\boldsymbol{h}^{t-1}, \boldsymbol{x}^{(t)})$ would be through a fully connected layer with a suitable activation function like the *tangens hyperbolicus* (tanh) [GBC16, p. 374]. Similar to the layer-wise organization of the neurons in an MLP, these so-called RNN *cells* are also arranged in layers.

The information processing of an RNN can be visualized in two different ways which are presented in Figure 7. In the compact representation (Figure 7 left), the input of $\boldsymbol{h}^{(t-1)}$ is indicated by a feedback connection with one timestep delay. *Unfolding* this connection over the timesteps (Figure 7 right) helps to illustrate the computation along the sequence with shared parameters. At the same time, the unfolded operations of the forward pass provide an indication how the gradients are back-propagated during training. For the *back-propagation through time* algorithm for the training of RNNs, we refer to [GBC16, Section 10.2.2]. Propagating the gradients over many timesteps, however, often results in vanishing or exploding gradients [GBC16, p. 396]. Thereby, the training of models for long-term dependencies becomes a challenging task. Vanishing or exploding gradients can be avoided by the introduction of *gates* in the computation of the internal state. The most popular variants of gated RNNs are long short-term memorys [HS97] and gated recurrent units [Cho+14] that are discussed in the following.



(a) LSTM (b) GRU

Figure 8: Illustation of the processing at timestep t within an LSTM call and a gated recurrent unit cell. Blue boxes mark functions with learnable parameters. White shapes indicate pointwise operations.

2.3.1 *Long Short-Term Memory*

The simplest form of an RNN only has a cell state which is updated based on the current inputs and the previous cell state (cf. Equation 24). Hochreiter and Schmidhuber [HS97] introduced the *long short-term memory* (LSTM) which has an additional internal cell memory that allows storing information across an arbitrary number of timesteps [GSC00]. Thereby, they enable back-propagation of errors for long sequences. The outputs of the cell and updates to the cell memory are computed via gates which protect the respective vectors from irrelevant perturbations [HS97]. Later, another gate was added to allow for forgetting parts of the cell memory [GSC00]. An illustration of the information processing in an LSTM cell is provided in Figure 8a.

While Equation 24 only considers the computations of a single cell, the following explanations will consider layers containing multiple parallel LSTM cells. One layer comprises l cells, each with a state vector $\boldsymbol{h}^{(t)} \in \mathbb{R}^{\mathsf{d}}$ and a memory vector $\boldsymbol{s}^{(t)} \in \mathbb{R}^{\mathsf{d}}$, where d denotes the dimensionality of the cell states. The matrix $\boldsymbol{H}^{(t-1)} \in \mathbb{R}^{(l \times \mathsf{d})}$ contains in each row the state vector of one of the l cells from the previous timestep $t-1$. At each timestep t, the current input vector $\boldsymbol{x}^{(t)} \in \mathbb{R}^{\mathsf{d}_x}$ is concatenated to each row of $\boldsymbol{H}^{(t-1)}$ to form $\tilde{\boldsymbol{X}}^{(t)} \in \mathbb{R}^{(l \times (\mathsf{d}+\mathsf{d}_x))}$. To avoid confusion with the timestep, the assignment of parameters to a function is indicated by a bold subscript (e. g. $\boldsymbol{W}_{\boldsymbol{X}}$), to differentiate the notation from indexing. Furthermore, the explicit notation of the bias is omitted[2].

At each timestep, the internal memory $\boldsymbol{S}_i^{(t)}$ of each LSTM cell i is updated. First, the *forget gate* determines which information in the memory should be deleted and which should be maintained. For each feature in the memory of each cell, a value between 0 and 1 is computed by the sigmoid activation function $\sigma(\cdot)$. The inputs for the sigmoid are given by applying a learnable linear transformation $\boldsymbol{W}_{\boldsymbol{F}} \in \mathbb{R}^{((\mathsf{d}+\mathsf{d}_x) \times \mathsf{d})}$ to the previous outputs and the current inputs:

$$\boldsymbol{F}^{(t)} = \sigma\Big(\tilde{\boldsymbol{X}}^{(t)} \boldsymbol{W}_{\boldsymbol{F}}\Big). \tag{25}$$

2 The computation of the pre-activation $a = \boldsymbol{w}^T \boldsymbol{x} + b$ can be reformulated to $a = \tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}$ with $\tilde{\boldsymbol{x}} = [\boldsymbol{x}, 1]$ and $\tilde{\boldsymbol{w}} = [\boldsymbol{w}, b]$.

Then, candidates $\tilde{\boldsymbol{S}}^{(t)}$ for updating the cell memory are computed based on $\tilde{\boldsymbol{X}}^{(t)}$:

$$\tilde{\boldsymbol{S}}^{(t)} = \tanh\Big(\tilde{\boldsymbol{X}}^{(t)}\boldsymbol{W}_{\boldsymbol{S}}\Big). \tag{26}$$

The *input gate* determines which parts of memory will be updated by these update candidates. The computation is similar to that of the forget gate but has its own parameters $\boldsymbol{W}_{\boldsymbol{I}}$:

$$\boldsymbol{I}^{(t)} = \sigma\Big(\tilde{\boldsymbol{X}}^{(t)}\boldsymbol{W}_{\boldsymbol{I}}\Big). \tag{27}$$

The memory of the cells is then updated according to the following equation:

$$\boldsymbol{S}^{(t)} = \boldsymbol{S}^{(t-1)} \odot \boldsymbol{F}^{(t)} + \boldsymbol{I}^{(t)} \odot \tilde{\boldsymbol{S}}^{(t)}, \tag{28}$$

where $\odot$ denotes an element-wise multiplication.

The output of the cells is then computed using the tanh of the cell memory and gating the returned information by the *output gate*:

$$\boldsymbol{O}^{(t)} = \sigma\Big(\tilde{\boldsymbol{X}}^{(t)}\boldsymbol{W}_{\boldsymbol{O}}\Big) \tag{29}$$

$$\boldsymbol{H}^{(t)} = \tanh\Big(\boldsymbol{S}^{(t)}\Big) \odot \boldsymbol{O}^{(t)}. \tag{30}$$

Following the unrolled illustration, the cell memory $\boldsymbol{S}^{(t)}$ and the cell state (and output) $\boldsymbol{H}^{(t)}$ are provided as inputs to the LSTM cells for the next step $t+1$.

2.3.2 *Gated Recurrent Unit*

An alternative to the LSTM was proposed by Cho et al. [Cho+14]. The *gated recurrent unit* (GRU) simplifies the design of the LSTM in two aspects. Its structure is shown in Figure 8b. First, the GRU does not model the memory and the output by two different variables but summarizes both representations in a single state $\boldsymbol{H}^{(t)} \in \mathbb{R}^{(l \times \mathsf{d})}$. Second, instead of having three gates (i. e. input, forget and output), the GRU only relies on two gates. Given the concatenation $\tilde{\boldsymbol{X}}^{(t)} \in \mathbb{R}^{(l \times (\mathsf{d}+\mathsf{d}_x))}$ of the previous state $\boldsymbol{H}^{(t-1)}$ and the current input $\boldsymbol{x}^{(t)}$, the *reset gate* is defined in the same fashion as the gates of the LSTM:

$$\boldsymbol{R}^{(t)} = \sigma\Big(\tilde{\boldsymbol{X}}^{(t)}\boldsymbol{W}_{\boldsymbol{R}}\Big), \tag{31}$$

with $\boldsymbol{W}_{\boldsymbol{R}} \in \mathbb{R}^{((\mathsf{d}+\mathsf{d}_x) \times \mathsf{d})}$. The values of the reset gate determine, which features from the previous states are considered in the computation of the candidates $\tilde{\boldsymbol{H}}^{(t)}$ for updating the cell states:

$$\tilde{\boldsymbol{H}}^{(t)} = \phi\Big(\Big[\boldsymbol{R}^{(t)} \odot \boldsymbol{H}^{(t-1)}, \boldsymbol{x}^{(t)}\Big]\, \boldsymbol{W}_{\boldsymbol{H}}\Big), \tag{32}$$

where $[\cdot,\cdot]$ denotes the concatenation of $\boldsymbol{x}^{(t)}$ to each row of the matrix $\boldsymbol{R}^{(t)} \odot \boldsymbol{H}^{(t-1)}$. For the activation function ϕ, the tanh was chosen in [Cho+14]. Similar to the definition of the other gates, the *update gate* is given by:

$$\boldsymbol{U}^{(t)} = \sigma\Big(\tilde{\boldsymbol{X}}^{(t)}\boldsymbol{W}_{\boldsymbol{U}}\Big). \tag{33}$$

However, the values of the update gate are utilized to combine forgetting of features and writing to the states:

$$\boldsymbol{H}^{(t)} = \boldsymbol{U}^{(t)} \odot \boldsymbol{H}^{(t-1)} + (\boldsymbol{1} - \boldsymbol{U}^{(t)}) \odot \tilde{\boldsymbol{H}}^{(t)}. \tag{34}$$

Thereby, entries from the previous state that should be deleted get replaced by the respective update candidates.

2.3.3 *Bidirectional RNNs*

The aforementioned RNN variants account for the causality of the input vectors. A state $\boldsymbol{h}^{(t_c)}$ only depends on the processed inputs $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t_c)})$ up to step t_c. However, information from future steps $t > t_c$ might be useful for predictions at step t_c [SP97]. For cases where the whole sequences are available offline, this information can be taken into account by employing one recurrent network for each direction in time. This idea of *bidirectional* RNNs was proposed by Schuster and Paliwal [SP97]. For an input sequence with T timesteps, one RNN processes the inputs from $t = 1, \ldots, T$ and computes the forward states $\overrightarrow{\boldsymbol{h}}^{(t)}$. Another RNN with its own set of parameters works in the opposite direction and computes the backward states $\overleftarrow{\boldsymbol{h}}^{(t)}$ for $t = T, \ldots, 1$. For each step t, $\overrightarrow{\boldsymbol{h}}^{(t)}$ and $\overleftarrow{\boldsymbol{h}}^{(t)}$ are concatenated. The concatenated states can then be further processed by output neurons [SP97]. It is important to note that both models work independently and neither of them has access to the states of the other model.

2.4 TRANSFORMER

The *transformer* is considered one of the most important developments in deep learning [BB24, p. 357] and surpasses the performance of RNNs and CNNs in several domains [BB24, p. 358]. Initially, transformers were proposed in the domain of *natural language processing* for machine translation (e. g. English to French). Machine translation, along with other sequence-to-sequence tasks like speech recognition and question answering, poses challenges that cannot be adequately addressed by a single RNN [SVL14]. RNNs map one sequence to another aligned sequence of the same length. The aforementioned tasks, however, require a mapping from an input sequence with length T to an output sequence with a possibly different length T'. At the same time, the relationships between input and output may be complex and non-monotonic.

Cho et al. [Cho+14] proposed an architecture to tackle these problems. Instead of using one RNN to map the input sequence directly to the output sequence, they suggest using two jointly trained RNNs. The *encoder* maps the input sequence $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(T)})$ to a sequence of state vectors $(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \ldots, \boldsymbol{h}^{(T)})$. The last state $\boldsymbol{h}^{(T)}$ is considered to be the summary (or context) $\boldsymbol{c}$ of the whole input sequence. The *decoder* then predicts an output $\boldsymbol{y}^{(t)}$ conditioned on the summary vector $\boldsymbol{c}$ and the previous output $\boldsymbol{y}^{(t-1)}$ in an autoregressive fashion. A similar approach was proposed by Sutskever et al. [SVL14]. However, they used the last state of the encoder only to initialize the first state of the decoder.

A major shortcoming of both approaches is that the encoder needs to encode all information from the input sequence into one vector of fixed length. Bahdanau et al. [BCB15] avoid this limitation by introducing an *attention* mechanism to the decoder. Instead of using a single vector $\boldsymbol{c}$, one context vector $\boldsymbol{c}^{(i)}$ for each output position i is computed as a weighted sum of the encoder states $\boldsymbol{h}^{(t)}$:

$$\boldsymbol{c}^{(i)} = \sum_{t=1}^{T} \alpha_{it} \boldsymbol{h}^{(t)}. \tag{35}$$

α_{it} is a scalar weight that determines the importance of the encoder state $\boldsymbol{h}^{(t)}$ for predicting the decoder state $\boldsymbol{s}^{(i)}$. The weights α_{it} are obtained by computing the softmax over the outputs of an alignment model:

$$\alpha_{it} = \frac{\exp(e_{it})}{\sum_{j=1}^{T} \exp(e_{ij})} \quad \text{with} \quad e_{ij} = a\Big(\boldsymbol{s}^{(i-1)}, \boldsymbol{h}^{(j)}\Big), \tag{36}$$

where $a(\cdot, \cdot)$ is the alignment model. Thereby, a soft alignment is obtained which allows the gradient to be propagated through all sub-models [BCB15].

Vaswani et al. [Vas+17] proposed to entirely remove all recurrent parts from models for sequence-to-sequence prediction. Their proposed transformer solely relies on the attention mechanism and significantly improved upon state-of-the-art approaches at that time. Instead of computing weights based on the predictions of an alignment model, they compute the attention coefficients by a dot product followed by a softmax. The vectors between which the attention coefficients are calculated are referred to as *queries* and *keys* [Vas+17]. Given m keys and n queries, these can be summarized in the matrices $\boldsymbol{K} \in \mathbb{R}^{(m \times \mathsf{d})}$ and $\boldsymbol{Q} \in \mathbb{R}^{(n \times \mathsf{d})}$, respectively, where d is the internal dimensionality of the attention mechanism. The output of the attention is then defined to be the weighted sum of so-called *values* $\boldsymbol{V} \in \mathbb{R}^{(m \times \mathsf{d}_v)}$:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{\mathsf{d}}}\right)\boldsymbol{V}, \tag{37}$$

where each value corresponds to one key. The softmax is applied on each row independently. The factor $1/\sqrt{\mathsf{d}}$ scales the dot product according to its variance[3] to avoid small gradients for large d [Vas+17]. Due to the scaling, this form of attention is also referred to as *scaled dot product attention*. As the computation does not rely on recurrent connections that require sequential processing, attention allows for a significantly better parallelization of the required computations [Vas+17]. Based on the definition in Equation 37, different variants of scaled dot product attention can be defined which will be described in the following.

2.4.1 *Self-Attention*

One variant is self-attention which relates different feature vectors within a single sequence in order to encode information from the entire sequence [Vas+17]. A set of inputs $(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)})$ with $\boldsymbol{x} \in \mathbb{R}^{\mathsf{d}_{in}}$ is summarized in the matrix $\boldsymbol{X} \in \mathbb{R}^{(n \times \mathsf{d}_{in})}$. Queries, keys and values are obtained by learnable linear transformations of the input $\boldsymbol{X}$:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{Q}}, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{K}} \quad \text{and} \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{V}}, \tag{38}$$

with $\{\boldsymbol{W}_{\boldsymbol{Q}}, \boldsymbol{W}_{\boldsymbol{K}}\} \in \mathbb{R}^{(\mathsf{d}_{in} \times \mathsf{d})}$ and $\boldsymbol{W}_{\boldsymbol{V}} \in \mathbb{R}^{(\mathsf{d}_{in} \times \mathsf{d}_v)}$.

In contrast to RNNs, attention does not take positional information within the sequence into account. In order to introduce information about absolute or relative positions into the attention mechanism, the addition of *positional encodings* was proposed [Vas+17]. These encodings can either be learned or be implemented by a defined function. Vaswani et al. proposed a fixed encoding based on sine and cosine

3 Assuming that the components of queries and keys are independent random variables with zero mean and unit variance the dot product of two d-dimensional vectors has variance d [Vas+17].

frequencies to encode positional information. For each vector in the input sequence, a positional encoding vector with the same dimensionality d_{in} is computed that is added to the input vector. The positional encoding $\boldsymbol{P} \in \mathbb{R}^{(n \times \mathsf{d}_{in})}$ is then defined by [BB24, p. 372]:

$$\boldsymbol{P}_{ti} = \begin{cases} \sin(t/10000^{i/\mathsf{d}}), & \text{if } i \text{ is even,} \\ \cos(t/10000^{(i-1)/\mathsf{d}}), & \text{otherwise.} \end{cases} \tag{39}$$

where the position in the sequence is specified by t and the dimension is given by i.

2.4.2 *Multi-Head Attention*

The modeling capacity can be increased by applying self-attention multiple times in parallel to the same sequence. Each self-attention is then referred to as an *(attention) head*. By employing multiple heads, the model can attend to information from different representation subspaces at different positions [Vas+17]. *Multi-head self-attention* (MHSA) comprises h parallel attention heads. Each head i has its own learnable parameters $\left\{\boldsymbol{W_Q}^{(i)}, \boldsymbol{W_K}^{(i)}\right\} \in \mathbb{R}^{(\mathsf{d}_{in} \times \mathsf{d})}$ and $\boldsymbol{W_V}^{(i)} \in \mathbb{R}^{(\mathsf{d}_{in} \times \mathsf{d}_h)}$ and computes:

$$\boldsymbol{H}^{(i)} = \text{Attention}(\boldsymbol{X}\boldsymbol{W_Q}^{(i)}, \boldsymbol{X}\boldsymbol{W_K}^{(i)}, \boldsymbol{X}\boldsymbol{W_V}^{(i)}). \tag{40}$$

The outputs $\boldsymbol{H}^{(i)}$ from all h heads are then concatenated and projected by another learnable linear transformation $\boldsymbol{W_O} \in \mathbb{R}^{((h \cdot \mathsf{d}_h) \times \mathsf{d})}$. The dimensionality of the outputs from the heads is typically set to $\mathsf{d}_h = \mathsf{d}/h$ [BB24, pp. 367f.].

2.4.3 *Cross-Attention*

In many cases, there is more than one sequence or set of vectors which should be set into context for the subsequent computations. Examples are sequence-to-sequence models like [Vas+17]. Here, the encoder solely aggregates information from the sequence by MHSA. The decoder gets the representation from the encoder as well as all previously predicted embeddings[4] as input. The representations from the encoder are then only used to compute the keys and values. The queries are computed based on the previously predicted outputs of the decoder. This form of attention which computes keys and values from a different sequence than the queries is referred to as *cross-attention*. Cross-attention can be extended to incorporate multiple heads in the same way as self-attention. This operation is then referred to as *multi-head cross-attention* (MHCA). MHSA and MHCA are the fundamental building blocks of transformers, which have contributed greatly to the development of powerful models like large language models.

4 Masking the inputs to only those that already have been predicted is only required in autoregressive tasks.

3 HANDWRITTEN TEXT GENERATION

Handwritten text generation (HTG) deals with the problem of generating an image showing a predefined word in handwritten text with a specific calligraphic style. The generated images can be be utilized in various ways, the most promising of which is the usage as training data for other document analysis systems. The majority of recent methods for HTG employ generative models that are based on *artificial neural networks* (ANNs). After covering the fundamentals of ANNs in Chapter 2, this chapter provides an overview on how ANNs can be used for HTG. Section 3.1 presents important classes of generative models in general and explains how the generation process is modeled using ANNs. These models serve as a basis for a number of strategies for addressing the task of HTG. Section 3.2 offers a comprehensive review of the literature on methods for HTG as well as approaches for evaluating the generation results. Finally, the application of works on HTG for training data generation is discussed in Section 3.3, providing the motivation for this thesis.

3.1 GENERATIVE MODELS

This section introduces different classes of generative models independently from the type of data that is generated. While the types of data that can be generated include various modalities such as text, audio, images, and video, the most relevant advancements for the development of HTG models were researched for the generation of natural images. Due to their excellent modeling capabilities, ANNs provide a good foundation for building deep generative models. The goal in generative modeling is to find a distribution $p_\theta(\boldsymbol{x})$ that approximates the underlying distribution $p(\boldsymbol{x})$ of a dataset $\mathbf{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{\mathsf{N}}$ with $\boldsymbol{x} \in \mathbb{R}^{\mathsf{d}}$. This objective can be expressed in terms of maximizing the likelihood function $p(\boldsymbol{x}|\theta)$. However, $p(\boldsymbol{x})$ can be very complex, making a direct approximation of the distribution very hard. Instead of modeling $p_\theta(\boldsymbol{x})$ directly, common approaches are to use *autoregressive modeling* or *latent variable models*.

In autoregressive modeling, the distribution is decomposed into a product of conditional distributions. This approach is explained in more detail in Section 3.1.1. When using latent variable models, $p_\theta(\boldsymbol{x})$ is decomposed into the distribution $p_\theta(\boldsymbol{z})$ over a latent variable $\boldsymbol{z}$ and the conditional distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. The distribution $p_\theta(\boldsymbol{x})$ is obtained by marginalizing over $\boldsymbol{z}$:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z}) \mathrm{d}\boldsymbol{z}. \tag{41}$$

Usually, $p_\theta(\boldsymbol{z})$ is chosen to be some simple distribution like a multivariate standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is approximated by a model. The optimization of $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ by maximum likelihood is, however, intractable depending on the model [Bon+22]. In such cases, proxy functions are optimized instead. Often, trade-offs are made between efficient optimization, efficient sampling and sample quality [Bon+22]. Common approaches based on latent variable models are varia-

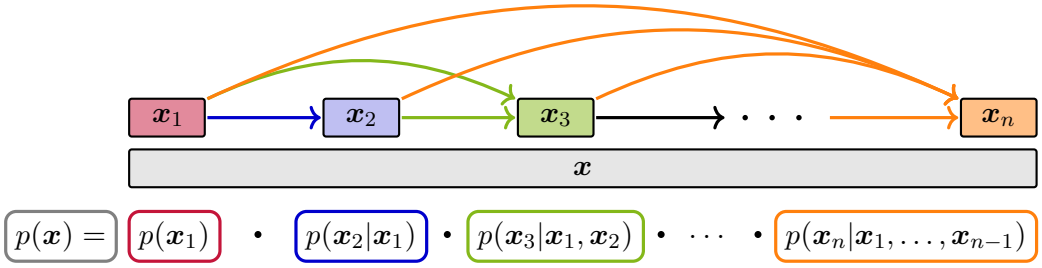


Figure 9: Example of an autoregressive model for $p(\boldsymbol{x})$, where $p(\boldsymbol{x})$ is modeled by the product of conditional probabilities of its components $\boldsymbol{x}_i$.

tional autoencoders, normalizing flows, generative adversarial networks, and diffusion models. A special focus in the following explanations is given on generative adversarial networks and diffusion models because they form the basis of most HTG methods.

3.1.1 *Autoregressive Models*

In *autoregressive models*, the joint distribution $p(\boldsymbol{x})$ is represented as the product of a sequence of conditional distributions. By recursively applying the product rule of probability, the joint distribution can be rewritten to

$$p(\boldsymbol{x}) = p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\mathsf{d}) = \prod_{i=1}^{\mathsf{d}} p(\boldsymbol{x}_i|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}), \tag{42}$$

where $\boldsymbol{x}_i$ are the ordered components of $\boldsymbol{x} \in \mathbb{R}^\mathsf{d}$. This modeling approach is illustrated in Figure 9. By this way of modeling, maximum likelihood estimation can be performed by minimizing the negative log likelihood [Bon+22]:

$$-\ln p(\boldsymbol{x}) = -\sum_{i=1}^{\mathsf{d}} \ln p(\boldsymbol{x}_i|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}). \tag{43}$$

A notable disadvantage of autoregressive models is a slow sampling since the process requires a sequential evaluation of the individual conditional distributions [Bon+22]. Assumptions about dependencies within the modeled sequences can be introduced to accelerate the sampling and simplify the model. For example, a *Markov* chain of order k can be assumed to simplify the conditional probabilities to [BB24, pp. 351f.]:

$$p(\boldsymbol{x}_i|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}) = p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}, \ldots, \boldsymbol{x}_{i-k}). \tag{44}$$

Autoregressive models can be implemented using different architectures like *multi layer perceptrons* (MLPs), *convolutional neural networks* (CNNs), *recurrent neural networks* (RNNs) or using self-attention. The models are particularly important for applications in *natural language processing*. Well-known examples are the successful *large language models* based on generative pretrained transformers [Rad+18; Rad+19; Bro+20; Ope+23]. These models were extended to more modalities, including the generation of images in an autoregressive fashion [Ope25]. While most state-of-the-art methods for the generation of natural images employ diffusion models, Sun

et al. [Sun+24] recently demonstrated, that autoregressive models achieve competitive generation performance. However, the first attempts to autoregressive modeling of image generation were made much earlier. Larochelle and Murray [LM11] used an MLP where they masked weights to model the conditioning on only preceding observations. They trained their *neural autoregressive distribution estimator* for generating handwritten digits from a binarized version of the MNIST dataset. The approach was later generalized to model real-valued distributions [UML13]. Besides MLP-based models, architectures based on RNNs [TB15; OKK16] and CNNs [Oor+16] were proposed.

3.1.2 *Normalizing Flows*

Another generative approach that allows a direct optimization of the likelihood during training is that of *normalizing flows*. The idea is to transform a probability density through a sequence of invertible mappings. This sequence is referred to as *flow*. The repeated application of the *rule for change of variables* ensures that a valid, i. e. *normalized*, distribution is obtained [RM15].

The *base distribution* $p_{\boldsymbol{z}}(\boldsymbol{z}), \boldsymbol{z} \in \mathbb{R}^{\mathsf{d}}$ can be simple, e. g. $\mathcal{N}(\mathbf{0}, \mathbf{1})$. The transformation is defined as a non-linear function $\boldsymbol{f}_\theta(\boldsymbol{z}) : \mathbb{R}^{\mathsf{d}} \mapsto \mathbb{R}^{\mathsf{d}}$. Mapping $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$ from the simple distribution to $\boldsymbol{x} = \boldsymbol{f}_\theta(\boldsymbol{z})$ results in the following distribution:

$$p_\theta(\boldsymbol{x}) = p_{\boldsymbol{z}}(\boldsymbol{f}_\theta^{-1}(\boldsymbol{x})) \cdot \left\| \det \frac{\partial \boldsymbol{f}_\theta^{-1}}{\partial \boldsymbol{x}} \right\|, \tag{45}$$

where $\boldsymbol{f}_\theta^{-1} / \partial \boldsymbol{x}$ is the Jacobian matrix of the inverse of $\boldsymbol{f}_\theta$ and det denotes the determinant of a matrix. The flow $\boldsymbol{f}_\theta$ can also be composed of K transformations $\boldsymbol{f}_{\theta_i}^{(i)}$:

$$\boldsymbol{z}^{(K)} = \boldsymbol{f}_{\theta_K}^{(K)} \circ \boldsymbol{f}_{\theta_{K-1}}^{(K-1)} \circ \cdots \circ \boldsymbol{f}_{\theta_1}^{(1)} \left(\boldsymbol{z}^{(0)} \right), \tag{46}$$

with $\boldsymbol{z}^{(0)} = \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$ and $\boldsymbol{z}^{(K)} = \boldsymbol{x}$. The repeated application of Equation 45 then results in the following log-likelihood [Bon+22]:

$$\ln p_{\theta_K}^{(K)} \left(\boldsymbol{z}^{(K)} \right) = \ln p_{\boldsymbol{z}}(\boldsymbol{z}^{(0)}) - \sum_{i=1}^{K} \ln \left\| \det \frac{\partial \boldsymbol{f}_{\theta_i}^{(i)}}{\partial \boldsymbol{z}^{(i-1)}} \right\|. \tag{47}$$

By using the flow in Equation 46, the base distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$ can be transformed into an arbitrarily complex density $p_{\theta_K}^{(K)}(\boldsymbol{z}^{(K)})$ [Bon+22]. Figure 10 shows an example of this stepwise transformation by a normalizing flow from composed functions.

Sampling from the resulting distribution can be efficiently performed by sampling $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$ and applying $\boldsymbol{f}_\theta$. However, Equation 45 imposes two requirements on $\boldsymbol{f}_\theta$: (i) The function $\boldsymbol{f}_\theta$ must be invertible and differentiable. Thereby, the implementation of $\boldsymbol{f}_\theta$ by means of ANNs is considerably constrained. A consequence of requiring $\boldsymbol{f}_\theta$ to be invertible is that $\boldsymbol{z}$ has to have the same dimensionality as $\boldsymbol{x}$, making ANNs inefficient for high-dimensional data [Bon+22]. (ii) For efficient training, an efficient method for computing the determinant of the Jacobian of each of the transforms is required.

One approach that meets these requirements is to use *coupling flows* [DKB14]. The input vectors $\boldsymbol{x}$ are split in parts of d' and $\mathsf{d} - \mathsf{d}'$ components, denoted by

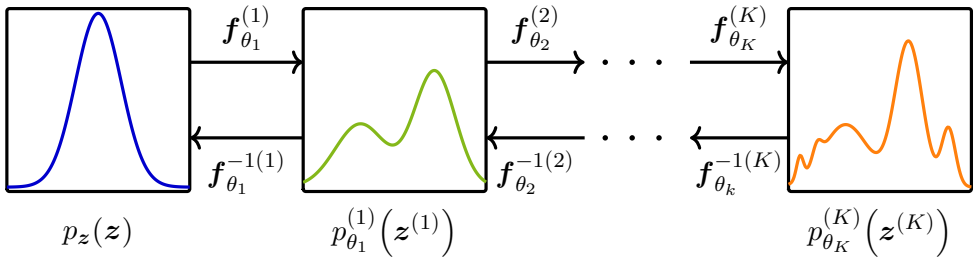


Figure 10: Example for a normalizing flow that is composed of invertible functions $\boldsymbol{f}_{\theta_i}^{(i)}$ and transforms the simple base distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$ into the complex distribution $p_{\theta_K}^{(K)}(\boldsymbol{z}^{(K)})$.

$\boldsymbol{x}_{1:\mathsf{d}'}$ and $\boldsymbol{x}_{(\mathsf{d}'+1):\mathsf{d}}$, respectively. While the first part is preserved, the second part is transformed by an invertible transformation $\boldsymbol{h}(\cdot)$ that is parameterized depending on the first part:

$$\boldsymbol{x}_{1:\mathsf{d}'} = \boldsymbol{z}_{1:\mathsf{d}'}, \tag{48}$$

$$\boldsymbol{x}_{(\mathsf{d}'+1):\mathsf{d}} = \boldsymbol{h}\left(\boldsymbol{z}_{(\mathsf{d}'+1):\mathsf{d}}; \boldsymbol{f}_\theta(\boldsymbol{z}_{1:\mathsf{d}'})\right). \tag{49}$$

For example, $\boldsymbol{h}$ can be implemented by an affine transformation [DSB17]. The approach results in a simple Jacobian, but the expressiveness is limited [Bon+22]. The generation of images requires more complex models which can be built by stacking numerous coupling flows. *Autoregressive flows* [PPM17] are a generalization of coupling flows that are universal approximators [Bon+22]. However, they either require sequential sampling [PPM17] or sequential density estimation during training [Kin+16]. Besides the aforementioned types, there exists a variety of alternatives like convolutional or residual flows [Bon+22]. Dinh et al. [DKB14; DSB17] demonstrated that normalizing flows are capable of generating images. Although normalizing flows received less attention than other generative models like generative adversarial networks or diffusion models, recent work [Zha+25] demonstrated that a transformer-based variant of a masked autoregressive flow can achieve competitive results for natural image generation compared to state-of-the-art diffusion models.

3.1.3 *Variational Autoencoders*

An auto-associative ANN or *autoencoder* (AE) is a model that can be used for learning internal representations. These models comprise an *encoder* and a *decoder* network. The encoder $\mathcal{E} : \mathbb{R}^{\mathsf{d}} \mapsto \mathbb{R}^{\mathsf{d}_z}$ maps an input $\boldsymbol{x}$ into a latent representation $\boldsymbol{z} = \mathcal{E}(\boldsymbol{x})$. The decoder $\mathcal{D} : \mathbb{R}^{\mathsf{d}_z} \mapsto \mathbb{R}^{\mathsf{d}}$ maps the representation back to $\hat{\boldsymbol{x}} = \mathcal{D}(\boldsymbol{z})$. The goal is to minimize the reconstruction error $\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}})$, where $\mathcal{L}(\cdot)$ can, for example, be implemented as the mean squared error. In order to obtain a meaningful latent representation $\boldsymbol{z}$, it is essential to prevent the AE from learning a trivial solution like the identity function. For this purpose, different constraints can be introduced to the architecture of the AE or to the training process. A common choice is to limit the number of dimensions $\mathsf{d}_z < \mathsf{d}$ of the latent representation. Alternatively, sparcity of the representation can be enforced by regularization. A different approach that requires the AE to learn about the structure of the data is to corrupt the input.

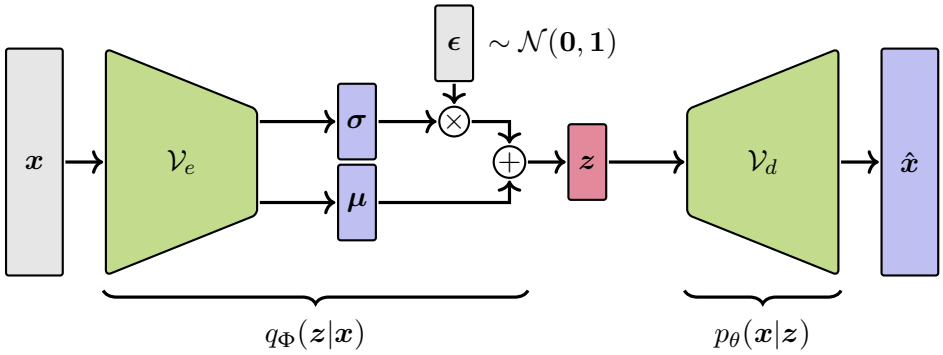


Figure 11: Variational autoencoder with the encoder $\mathcal{V}_e$ predicting the parameters of a Gaussian distribution with a diagonal covariance matrix. $\boldsymbol{z}$ is sampled using the reparametrization trick and passed to the decoder $\mathcal{V}_d$ to predict $\hat{\boldsymbol{x}}$. $\times$ and $+$ are elementwise operation.

Examples for this procedure are denoising AEs [Vin+08] and *masked autoencoders* (MAEs) [He+22]. Since MAEs are part of the proposed method of this thesis, they are explained in more detail in Section 4.3. AEs, however, cannot be directly used for generation [BB24, p. 564].

Instead of deterministic encoders and decoders, probabilistic models can be used. The resulting latent variable model can be used for generation and is referred to as *variational autoencoders* (VAEs) [KW13]. Given a latent distribution $p_\theta(\boldsymbol{z})$, the decoder $\mathcal{V}_d$ is trained to model the conditional probability $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. However, the marginal likelihood (cf. Equation 41) is intractable. *Variational inference* allows to optimize the *evidence lower bound* (ELBO)

$$\ln p_\theta(\boldsymbol{x}) \geq -D_{KL}(q_\Phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) + \mathbb{E}_{q_\Phi(\boldsymbol{z}|\boldsymbol{x})}\left[\ln p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] \tag{50}$$

instead of directly maximizing the log-likelihood. $D_{KL}(\cdot||\cdot)$ denotes the *Kullback-Leibler* (KL) divergence. For the variational inference, the true posterior $p(\boldsymbol{z}|\boldsymbol{x})$ is approximated by $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$. Instead of computing the approximation $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ per sample in the training set, the process is amortized by using an ANN to predict $q_\Phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ for each sample $\boldsymbol{x}^{(i)}$. This ANN is the encoder $\mathcal{V}_e$ of the VAE. Typically, $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ is modeled by a Gaussian distribution with diagonal covariance matrix, where the encoder predicts mean $\boldsymbol{\mu}^{(i)}$ and standard deviation $\boldsymbol{\sigma}^{(i)}$ for each sample $\boldsymbol{x}^{(i)}$ [BB24, pp. 572f.]. In order to optimize Equation 50 with back-propagation, the sampling $\boldsymbol{z} \sim q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ is rewritten using the *reparameterization trick* [KW13]:

$$\boldsymbol{z} = \boldsymbol{g}_\Phi(\boldsymbol{\epsilon}, \boldsymbol{x}), \quad \text{with} \quad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}), \tag{51}$$

where $\boldsymbol{g}_\Phi(\cdot)$ is a differentiable transformation. If $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ is a Gaussian that is parameterized in the previously described way, the reparameterization is given by $\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. The structure of a VAE applying this reparameterization is depicted in Figure 11. Furthermore, choosing $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ and $p_\theta(\boldsymbol{z})$ as Gaussian distributions has the advantage that the KL divergence can be computed and differentiated without estimation [KW13].

After training, new samples can be generated by sampling $\boldsymbol{z} \sim p_\theta(\boldsymbol{z})$ and forwarding $\boldsymbol{z}$ through the decoder. The encoder can be discarded. The VAE can also be extended for conditional generation [BB24, p. 576]. Therefore, the encoder and the decoder are conditioned on a variable $\boldsymbol{c}$ that, for example, encodes a class label.

Training the VAE by optimizing the ELBO can result in the following problems. When $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ converges to $p_\theta(\boldsymbol{z})$ as enforced by the KL divergence term in Equation 50, the dependency on $\boldsymbol{x}$ gets ignored and the generated outputs become blurry [BB24, pp. 576f.]. This problem is referred to as *posterior collapse*. On the other hand, if $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ is very different from $p_\theta(\boldsymbol{z})$, reconstructions can be highly accurate during training, but when generating samples from $p_\theta(\boldsymbol{z})$ the decoder is unable to generate realistic results [BB24, p. 577]. Higgins et al. [Hig+17] addressed these problems and introduced a hyperparameter β which weights the contribution of the KL divergence to the overall objective. Dosovitskiy and Brox [DB16] attribute the problem of blurry generation to the expectation term (second term in Equation 50), that serves as a reconstruction error. They replace this term with their proposed loss, which combines an adversarial loss, a loss in feature space as well as a loss in the image space. Thereby, ideas from generative adversarial networks are incorporated in their VAE. Another approach to avoid a posterior collapse is the vector quantized VAE, which incorporates ideas from vector quantization to learn a discrete latent representation [OV17]. This approach was further improved by including a perceptual reconstruction loss [Zha+18] and a patch-based discriminator [Iso+17] for an adversarial objective [ERO21]. Furthermore, it was shown that encoding the sample in two stages helps avoiding the second problem (i. e. $q_\Phi(\boldsymbol{z}|\boldsymbol{x})$ is very different from $p_\theta(\boldsymbol{z})$) [DW19]. Another very successful two-stage model that uses a VAE in combination with a diffusion model is the latent diffusion model [Rom+22] that is discussed in more detail in Section 3.1.5.

3.1.4 *Generative Adversarial Networks*

A family of generative models that largely influenced the development of methods for HTG are *generative adversarial networks* (GANs) [Goo+14]. Similar to normalizing flows, the distribution $p_\theta(\boldsymbol{x})$ is modeled by a non-linear mapping $\boldsymbol{x} = \mathcal{G}_{\theta_g}(\boldsymbol{z})$ of random vectors $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$ from a latent distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$, e. g. a standard normal distribution. The function $\mathcal{G}_{\theta_g}(\cdot)$ is called *generator*. However, since the choice of $\mathcal{G}_{\theta_g}(\cdot)$ is not restricted, in general, the likelihood cannot be evaluated in closed form and thus cannot be used for optimization [BB24, p. 534]. The idea of adversarial training is to use a second network $\mathcal{D}_{\theta_d}(\boldsymbol{x})$ that predicts whether the sample is *real* or *fake*. In this context, samples from the training set $\mathbf{D}$ are considered real images while samples generated by the generator are termed fake samples. Therefore, the model $\mathcal{D}_{\theta_d}(\cdot)$ is referred to as *discriminator*. This adversarial setup is visualized in Figure 12.

Both models are implemented by ANNs with parameters θ_g and θ_d, respectively, and trained jointly. In the initial work [Goo+14], the discriminator has a single sigmoidal output that represents the probability of input $\boldsymbol{x}$ being a real sample. With labels $y = 1$ for real samples and $y = 0$ for fake samples, the model is trained to maximize

$$\mathcal{L}_d(y, \mathcal{D}_{\theta_d}(\boldsymbol{x})) = y \ln(\mathcal{D}_{\theta_d}(\boldsymbol{x})) + (1 - y) \ln(1 - \mathcal{D}_{\theta_d}(\boldsymbol{x})). \tag{52}$$

At the same time, $\mathcal{G}_{\theta_g}(\cdot)$ is trained to minimize the loss

$$\mathcal{L}_g(\mathcal{G}_{\theta_g}(\boldsymbol{z})) = \ln(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\boldsymbol{z}))). \tag{53}$$

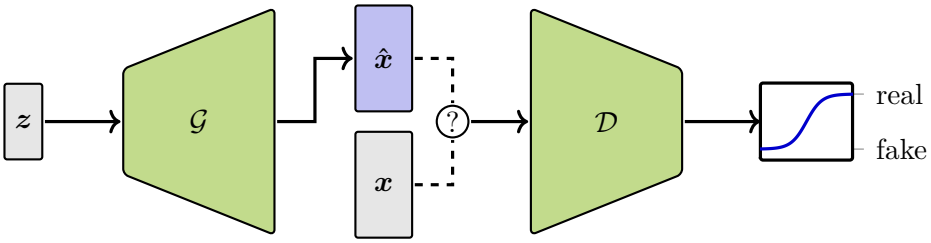


Figure 12: Generative adversarial network with a generator $\mathcal{G}$ that predicts $\hat{\boldsymbol{x}}$ based on the latent vector $\boldsymbol{z}$. The discriminator $\mathcal{D}$ is trained to predict a probability of a given sample being real (i. e. $\boldsymbol{x}$) rather generated by $\mathcal{G}$ (i. e. $\hat{\boldsymbol{x}}$).

The objective for the joint optimization of θ_g and θ_d can then be written as

$$\underset{\theta_g}{\operatorname{argmin}} \underset{\theta_d}{\operatorname{argmax}} \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})} \left[\ln(\mathcal{D}_{\theta_d}(\boldsymbol{x}))\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})} \left[\ln(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\boldsymbol{z})))\right], \tag{54}$$

where $\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}$ denotes the expected value over the data distribution $p(\boldsymbol{x})$ which boils down to sampling $\boldsymbol{x} \in \mathbf{D}$ for a given training set $\mathbf{D}$. $\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}$ is typically approximated by sampling $|\mathbf{D}|$ times from $p_{\boldsymbol{z}}(\boldsymbol{z})$ [BB24, p. 535]. The optimization is performed by gradient descent and gradient ascent using back-propagation and alternates between updating the generator and the discriminator. After training, the discriminator can be discarded. New samples can be generated by sampling $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$ and applying the generator to map $\boldsymbol{z}$ to a sample $\hat{\boldsymbol{x}} = \mathcal{G}_{\theta_g}(\boldsymbol{z})$.

The previously described model allows only for unconditional generation. In order to provide control over the content of the generated samples, *conditional generative adversarial networks* (CGANs) were proposed [MO14]. The generator as well as the discriminator are therefore conditioned on extra information $\boldsymbol{c}$ (e. g. an encoded class label) which is passed to the models as an additional input. Instead of conditioning the discriminator on the class label, Odena et al. [OOS17] trained the discriminator to predict the generated class in addition to predicting whether the given image is real.

Despite their successful application in generating realistic images, GANs are remarkably difficult to train [AB17]. The problem is that the optimization of the adversarial objective necessitates the finding of a Nash equilibrium, a task which gradient descent techniques are not designed for [Sal+16]. One typical problem is *mode collapse*, where the generator is able to generate samples from one or a few modes only and misses the other modes from the underlying distribution $p(\boldsymbol{x})$ [BB24, p. 536]. Another problem can occur if the discriminator improves. Then the loss function in Equation 53 saturates and the gradients vanish. This was tackled by Goodfellow et al. [Goo+14] by maximizing $\ln(\mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\boldsymbol{z})))$ instead of minimizing Equation 53. Another approach that mitigates this problem is to replace the cross-entropy loss by a least-squares loss function resulting in the following objectives [Mao+17]:

$$\underset{\theta_d}{\operatorname{argmin}} \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})} \left[(\mathcal{D}_{\theta_d}(\boldsymbol{x}) - 1)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})} \left[\mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\boldsymbol{z}))^2\right], \text{ and} \tag{55}$$

$$\underset{\theta_g}{\operatorname{argmin}} \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})} \left[(\mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\boldsymbol{z})) - 1)^2\right]. \tag{56}$$

Arjovsky et al. [ACB17] proposed to optimize the *Wasserstein-1* distance which eliminates the vanishing gradients problem and offers improved stability [Bon+22].

At the same time, it introduces other problems like the requirement to clip the weights [Bon+22]. Aside from these loss functions, other functions such as the hinge loss [LY17] can be used for the training process.

Early GANs were implemented by MLPs (e. g. [Goo+14]). As discussed in Section 2.2, MLPs have considerable shortcomings when applied to images, especially for high resolutions. Radford et al. [RMC15] successfully implemented GANs using CNNs that allowed generation of images with a resolution of 64×64 pixels. Karras et al. [Kar+18] suggested progressive growing of the models. They started the training of generator and discriminator for a spatial resolution of 4×4 pixels and incrementally added layers to increase the resolution up to 1024×1024 pixels. Brock et al. [BDS19] demonstrated that scaling the number of trainable parameters as well as increasing the batch size is sufficient to enable high resolution image generation without progressive growing. Another technique that helps to improve the generation quality is to use a *patch discriminator* [Iso+17]. Instead of using the whole image as input, the discriminator only classifies smaller patches in a sliding-window fashion.

The concept of generating samples from a latent random vector was later expanded to generation conditioned on an input image. This approach allows utilizing GANs for *image-to-image translation*. Pathak et al. [Pat+16] designed a model similar to an AE but trained it for inpainting of images. By combining a reconstruction loss with an adversarial loss, they trained models which generate sharp results and learn a representation that captures the semantics of visual structures. Instead of inpainting, Isola et al. [Iso+17] developed a GAN for image-to-image translation between different domains (e. g. sketches or segmentation maps to photorealistic images). Their model was also trained with a combination of an adversarial loss and a reconstruction loss. Following more closely the original GAN design, they experimented with providing a random noise vector $\boldsymbol{z}$ as input to the generator in addition to the conditioning to encourage variation in the generation results. They discovered that models learned to ignore the noise and utilized dropout to enforce variation.

The reconstruction loss, however, requires pairs of images to be provided as training data. Zhu et al. [Zhu+17] propose to exploit cycle consistency to avoid this problem and allow *unpaired* image-to-image translation. Given two domains, $\mathcal{I}_A$ and $\mathcal{I}_B$, they train one GAN with a generator $\mathcal{G}_B : \mathcal{I}_A \mapsto \mathcal{I}_B$ that maps images from domain $\mathcal{I}_A$ to $\mathcal{I}_B$. The corresponding discriminator $\mathcal{D}_B(\cdot)$ is trained to distinguish real images from domain $\mathcal{I}_B$ from images generated by $\mathcal{G}_B$. At the same time, they train a generator for the inverse mapping $\mathcal{G}_A : \mathcal{I}_B \mapsto \mathcal{I}_A$ and a corresponding discriminator $\mathcal{D}_A(\cdot)$. In addition to an adversarial loss for each of the GANs, they enforce cycle consistency using a reconstruction loss between an image $\boldsymbol{X} \in \mathcal{I}_A$ and $\mathcal{G}_A(\mathcal{G}_B(\boldsymbol{X}))$. The same loss is used for the opposite direction between an image $\boldsymbol{Y} \in \mathcal{I}_B$ and $\mathcal{G}_B(\mathcal{G}_A(\boldsymbol{Y}))$. By using the cycle consistency loss for mapping from $\mathcal{I}_A$ to $\mathcal{I}_B$, the approach does not require pairs of images from both domains.

3.1.5 *Diffusion Models*

Denoising diffusion probabilistic models (DDPMs) [Soh+15; HJA20] are a class of generative models that yield state-of-the-art results in image generation [DN21]. By gradually adding noise to the input data, a complex data distribution $p(\boldsymbol{x})$ is con-

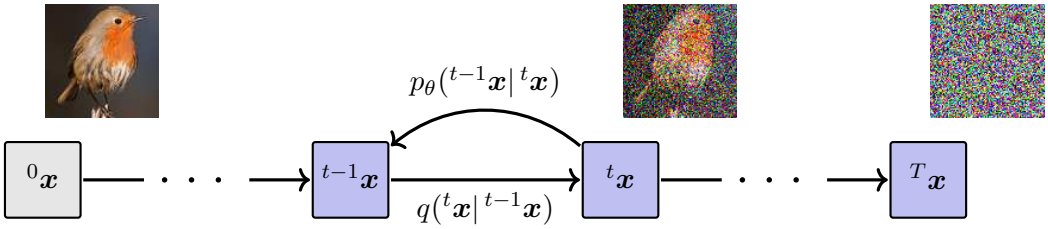


Figure 13: Stepwise addition of noise to a sample ${}^{0}\boldsymbol{x}$ by the diffusion process $q({}^{t}\boldsymbol{x}|\,{}^{t-1}\boldsymbol{x})$. A sample can be generated from random noise ${}^{T}\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ by reversing this process. The reverse process $p_\theta({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x})$ is learned by a model. The bird image is taken from the STL-10 dataset [CNL11].

verted into an analytically tractable distribution, e. g. a standard normal distribution. Learning transitions to reversing this process, allows for generating samples [HJA20]. These processes are depicted in Figure 13.

The addition of noise is done in a Markov chain that is referred to as *forward process* or *diffusion process*. At each timestep $t = 1, \ldots, T$, Gaussian noise is added to the data ${}^{0}\boldsymbol{x} = \boldsymbol{x} \sim p(\boldsymbol{x})$ with a defined variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$ [HJA20]. The resulting chain is defined by:

$$q({}^{1}\boldsymbol{x}, \ldots, {}^{T}\boldsymbol{x}|\,{}^{0}\boldsymbol{x}) \coloneqq \prod_{t=1}^{T} q\left({}^{t}\boldsymbol{x}|\,{}^{t-1}\boldsymbol{x}\right) \quad \text{and} \tag{57}$$

$$q({}^{t}\boldsymbol{x}|\,{}^{t-1}\boldsymbol{x}) \coloneqq \mathcal{N}\Big({}^{t}\boldsymbol{x}; \sqrt{1-\beta_t}\cdot {}^{t-1}\boldsymbol{x}, \beta_t \mathbf{1}\Big), \tag{58}$$

where ${}^{1}\boldsymbol{x}, \ldots, {}^{T}\boldsymbol{x}$ are latent variables for every timestep. This modeling of the forward process allows to directly sample ${}^{t}\boldsymbol{x}$ at an arbitrary timestep t from the following distribution:

$$q({}^{t}\boldsymbol{x}|\,{}^{0}\boldsymbol{x}) = \mathcal{N}\Big({}^{t}\boldsymbol{x}; \sqrt{\bar{\alpha}_t}\,{}^{0}\boldsymbol{x}, (1-\bar{\alpha}_t)\mathbf{1}\Big), \tag{59}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ [HJA20]. Reparametrization of Equation 59 results in the following computation of ${}^{t}\boldsymbol{x}$:

$${}^{t}\boldsymbol{x}({}^{0}\boldsymbol{x}, t) = \sqrt{\bar{\alpha}_t}\,{}^{0}\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}). \tag{60}$$

For a sufficiently large T and a suitable variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$, the latent ${}^{T}\boldsymbol{x}$ follows almost a standard Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$ [ND21].

For the generation of samples, the process has to be reversed into a stepwise removal of noise. The respective process would require knowing $q({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x})$, which depends on the entire data distribution [ND21]. Instead, the distribution is approximated by a model $p_\theta({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x})$. Starting at ${}^{T}\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$, the transitions of the reverse process are defined as:

$$p_\theta({}^{0}\boldsymbol{x}, \ldots, {}^{T}\boldsymbol{x}) \coloneqq p({}^{T}\boldsymbol{x}) \prod_{t=1}^{T} p_\theta\left({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x}\right) \quad \text{and} \tag{61}$$

$$p_\theta({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x}) \coloneqq \mathcal{N}\Big({}^{t-1}\boldsymbol{x}; \boldsymbol{\mu}_\theta({}^{t}\boldsymbol{x}, t), \boldsymbol{\Sigma}_\theta({}^{t}\boldsymbol{x}, t)\Big). \tag{62}$$

The forward process $q\left({}^{1}\boldsymbol{x}, \ldots, {}^{T}\boldsymbol{x}|\,{}^{0}\boldsymbol{x}\right)$ and the backward process $p_\theta\left({}^{0}\boldsymbol{x}, \ldots, {}^{T}\boldsymbol{x}\right)$ together form a VAE [ND21]. The only difference is that the forward process is fixed in contrast to the encoder of a VAE which is learned from data.

Training

Similar to a VAE, the model for the reverse process $p_\theta({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x})$ is trained by optimizing the ELBO [HJA20] that can be rewritten as

$$\begin{aligned}\mathcal{L}_{ELBO} \coloneqq \mathbb{E}_q\Bigg[& D_{KL}\Big(q({}^{T}\boldsymbol{x}|\,{}^{0}\boldsymbol{x})||p({}^{T}\boldsymbol{x})\Big) \\ & + \sum_{t>1} D_{KL}\Big(q({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x},{}^{0}\boldsymbol{x})||p_\theta({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x})\Big) - \log p_\theta({}^{0}\boldsymbol{x}|\,{}^{1}\boldsymbol{x})\Bigg].\end{aligned} \tag{63}$$

Conditioning the forward process on ${}^{0}\boldsymbol{x}$ results in tractable distributions

$$q({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x},{}^{0}\boldsymbol{x}) = \mathcal{N}\Big({}^{t-1}\boldsymbol{x}; \tilde{\boldsymbol{\mu}}_t({}^{t}\boldsymbol{x},{}^{0}\boldsymbol{x}), \tilde{\beta}_t\mathbf{1}\Big), \tag{64}$$

with

$$\tilde{\boldsymbol{\mu}}_t({}^{t}\boldsymbol{x},{}^{0}\boldsymbol{x}) \coloneqq \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\,{}^{0}\boldsymbol{x} + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\,{}^{t}\boldsymbol{x} \quad \text{and} \quad \tilde{\beta}_t \coloneqq \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{65}$$

Given the parametrization of the reverse process from Equation 62, the KL divergences in Equation 63 can be evaluated in closed form [HJA20]. Fixing the variance $\boldsymbol{\Sigma}_\theta({}^{t}\boldsymbol{x},t) = \sigma_t^2\mathbf{1}$ of the reverse process to time dependent constants (e. g. $\sigma_t^2 = \beta_t$) and reformulating the closed form solution of the KL divergence using Equations 60 and 65 results in the following target for the model $\boldsymbol{\mu}_\theta({}^{t}\boldsymbol{x},t)$:

$$\frac{1}{\sqrt{\alpha_t}}\left({}^{t}\boldsymbol{x} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\right). \tag{66}$$

Since ${}^{t}\boldsymbol{x}$ is provided as an input to the model, the model can alternatively be trained to predict $\boldsymbol{\epsilon}$ by $\boldsymbol{\epsilon}_\theta({}^{t}\boldsymbol{x},t)$. Omitting scaling terms that have been found to be not beneficial for sampling quality [HJA20], the model can be trained using the following simplified objective:

$$\mathcal{L}_{simple} \coloneqq \mathbb{E}_{t,{}^{0}\boldsymbol{x},\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta({}^{t}\boldsymbol{x},t)\|^2\right]. \tag{67}$$

A detailed derivation of this objective is provided in the work of Ho et al. [HJA20]. Optimization of the objective is implemented by stochastic gradient descent on uniformly sampled timesteps t. Given this parametrization of the reverse process in terms of $\boldsymbol{\epsilon}$-prediction, the objective resembles denoising score matching over multiple noise scales [SE19; HJA20]. In this case, $\boldsymbol{\epsilon}_\theta({}^{t}\boldsymbol{x},t)$ is considered the (diffusion) score and gives an estimate of the gradient of the log-likelihood with respect to the noisy data ${}^{t}\boldsymbol{x}$ [HS21].

Sampling

For the generation of samples, ${}^{t-1}\boldsymbol{x} \sim p_\theta({}^{t-1}\boldsymbol{x}|\,{}^{t}\boldsymbol{x})$ has to be sampled [HJA20]. Given the noise prediction $\boldsymbol{\epsilon}_\theta({}^{t}\boldsymbol{x},t)$ from the network, ${}^{t-1}\boldsymbol{x}$ can be computed from ${}^{t}\boldsymbol{x}$ by

$${}^{t-1}\boldsymbol{x} = \frac{1}{\sqrt{\alpha_t}}\left({}^{t}\boldsymbol{x} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta({}^{t}\boldsymbol{x},t)\right) + \sqrt{\beta_t}\boldsymbol{\epsilon}, \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{1}). \tag{68}$$

Starting at timestep $t = T$, this removal of noise is repeated in order to obtain a generated image ${}^{0}\boldsymbol{x}$. This process is computationally very expensive as it necessitates the simulation of a Markov chain for many (e. g. $T = 1000$) steps [SME21].

Algorithm 1 : Training of a conditional diffusion model applying classifier-free guidance [HS21].

Input : Labeled dataset $\mathbf{D} = \{(\boldsymbol{x}^{(0)}, y_0), \dots, (\boldsymbol{x}^{(N)}, y_N)\}$;
Models : Conditional DM $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t, y)$;
Parameters : Probability of unconditional training p_{uc}; Number of diffusion steps T; Variance schedule $\{\beta_t \in (0,1)\}_{t=1}^T$;

for $t \leftarrow 1$ **to** T **do**
  $\bar{\alpha}_t \leftarrow \prod_{i=1}^t 1 - \beta_i$; ▷ Pre-compute alphas
repeat
  $({}^0\boldsymbol{x}, y) \in_R \mathbf{D}$;
  $y \leftarrow \emptyset$ with probability p_{uc};
  $t \sim \mathcal{U}(1, T)$; ▷ Sample timestep
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$; ▷ Sample noise
  ${}^t\boldsymbol{x} \leftarrow \sqrt{\bar{\alpha}_t}\,{}^0\boldsymbol{x} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$; ▷ Apply forward process
  Update weights θ according to $\nabla_\theta \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t, y)\|$;
until *converged*;

Consequently, more efficient approaches for sampling have been investigated. Song et al. [SME21] propose a generalization of DDPMs by a class of non-Markovian diffusion processes. While preserving the original training objective, their modelling allows to consider forward processes with less than T steps. The number of steps in the reverse process are correspondingly reduced. Recent sampling strategies enable a further reduction of the number of sampling steps while maintaining high generation quality [Lu+22a; Lu+22b; ZC23; Zha+23].

Conditional Generation

The explanation so far has focused on unconditional DDPMs. In order to leverage the full potential of generative models, it is essential to control for the content of the generation by conditioning the model. Examples for such conditioning are class labels (e. g. [DN21; HS21; Son+21]), text prompts (e. g. [Nic+22]) or other modalities. Song et al. [Son+21] proposed a conditioning mechanism for score-based generative models, which also include DDPMs. Gradients of a time-dependent classifier are included in the stochastic differential equation for sampling to control the generated content. Following this idea, Dhariwal et al. [DN21] adapted the approach to the sampling process of DDPMs. First, a time-dependent classifier $c_\Phi(y|\,{}^t\boldsymbol{x}, t)$ is trained on noisy images ${}^t\boldsymbol{x}$ where y is the class label. $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t)$ is then extended by the addition of the gradient of the classifier

$$\tilde{\boldsymbol{\epsilon}}_\theta({}^t\boldsymbol{x}, t) = \boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t) - \sqrt{1-\bar{\alpha}_t} \cdot w_{gs} \cdot \nabla_{{}^t\boldsymbol{x}} \log c_\Phi(y|\,{}^t\boldsymbol{x}, t). \tag{69}$$

By this formulation, sampling is guided toward the classifier predicting the given conditioning y. w_{gs} is a weight that allows to scale the influence of the guidance by the classifier gradients. Conditioning the DDPM on the class label in addition to guiding the sampling process helps to improve generation quality [DN21].

A major shortcoming of classifier guidance is the necessity of a classifier that is trained to work on noisy data. To avoid this problem, Ho and Salimans [HS21]

Algorithm 2 : Guided sampling from a conditional diffusion model applying classifier-free guidance [HS21].

Input : Conditioning y (e. g. class label);
Output : Generated sample $\hat{\boldsymbol{x}}$;
Models : Trained conditional DM $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t, y)$;
Parameters : Guidance scale w_{gs}; Number of diffusion steps T; Variance schedule $\{\beta_t \in (0,1)\}_{t=1}^T$;

${}^T\hat{\boldsymbol{x}} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$;
for $t \leftarrow T$ **to** 1 **do**
  $\bar{\alpha}_t \leftarrow \prod_{i=1}^t 1 - \beta_i$; ▷ Compute alpha
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ if $t > 1$, else $\boldsymbol{\epsilon} = \mathbf{0}$; ▷ Noise for reparametrization
  $\tilde{\boldsymbol{\epsilon}} \leftarrow (1 + w_{gs})\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{x}}, t, y) - w_{gs}\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{x}}, t, y = \emptyset)$; ▷ Guidance
  ${}^{t-1}\hat{\boldsymbol{x}} \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left({}^t\hat{\boldsymbol{x}} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\tilde{\boldsymbol{\epsilon}}\right) + \sqrt{\beta_t}\boldsymbol{\epsilon}$; ▷ Denoising step
return ${}^0\hat{\boldsymbol{x}}$;

proposed *classifier-free guidance* (CFG) which does not require a classifier. Instead, they train an unconditional *diffusion model* (DM) $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t)$ and a conditional model $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t, y)$. The sampling process is then guided by using a linear combination of conditional and unconditional scores:

$$\tilde{\boldsymbol{\epsilon}}_\theta({}^t\boldsymbol{x}, t, y) = (1 + w_{gs})\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t, y) - w_{gs}\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t), \tag{70}$$

where w_{gs} determines the strength of the guidance. In this approach, the conditional and the unconditional models can be implemented by a single model where the conditioning is set to a null label $\emptyset$ with some probability p_{uc} during training:

$$\boldsymbol{\epsilon}_\theta\left({}^t\boldsymbol{x}, t\right) = \boldsymbol{\epsilon}_\theta\left({}^t\boldsymbol{x}, t, y = \emptyset\right). \tag{71}$$

Algorithm 1 and Algorithm 2 describe the procedures for training a DM with CFG and sampling with CFG, respectively.

Latent Diffusion Models

Training and sampling of DMs in the pixel space is extremely expensive regarding GPU compute power and time. To reduce these costs, Rombach et al. [Rom+22] proposed to separate these processes into a compressive and a generative stage. The compressive stage is implemented by an autoencoder (AE) based on the model from Esser et al. [ERO21] which is trained to provide a lower-dimensional latent space by removing high-frequency details. The encoder $\mathcal{V}_e$ computes the latent space representation $\boldsymbol{Z} \in \mathbb{R}^{(\mathsf{h}/f \times \mathsf{w}/f \times c)}$ for a given RGB image $\boldsymbol{I} \in \mathbb{R}^{(\mathsf{h} \times \mathsf{w} \times 3)}$. Here, f denotes the downsampling factor. For sampling, the generated latent representations are mapped back to the pixel space by the decoder $\mathcal{V}_d$ to form the image $\hat{\boldsymbol{I}} = \mathcal{V}_d(\boldsymbol{Z}) = \mathcal{V}_d(\mathcal{V}_e(\boldsymbol{I}))$. The AE is trained using a perceptual loss [Zha+18] and a patch-based adversarial loss [Iso+17]. Additional regularization is implemented to avoid arbitrarily scaled latent spaces. Besides a vector quantization layer [OV17] Rombach et al. tried a KL divergence to a standard normal distribution. In the latter case, their AE is similar to a VAE.

After training the AE, the DM is trained in the lower-dimensional latent space of the AE. Thereby, the DM can focus on important, semantic information [Rom+22]. At the same time, training of the DM is more efficient due to the dimensionality reduction. DMs following this two-stage approach are termed *latent diffusion models* (LDMs).

3.2 METHODS FOR HANDWRITTEN TEXT GENERATION

The primary domain that has driven the development of generative models (e. g. [Goo+14; HJA20; Rom+22]) is that of natural images. Consequently, these models provide an excellent foundation for application in other domains like handwritten documents. The task of generating images depicting handwriting is called handwritten text generation (HTG). Approaches tackle the task at different levels ranging from the generation of individual characters (e. g. [AM24; Cha+18]), words (e. g. [AMM19; Kan+20b; Nik+23; Ria+24]), and lines (e. g. [Dav+20; Kan+22]), to the generation of entire paragraphs (e. g. [May+24]). There also exists works on the generation of handwritten characters or words in other scripts like Bangla (e. g. [Fua+24]), Chinese (e. g. [Ren+24]) or ancient scripts like cuneiform (e. g. [BRF21]). Moreover, the generation of scene text presents similar challenges and is addressed by some works on HTG like [Kri+21; Zhu+23]. In addition to the generation of handwritten text, research is also being conducted on the generation of handwritten mathematical expressions (e. g. [CLD24]). Other document-related applications of generative models are the generation of fonts (e. g. [Tha+24]) and the generation of tables (e. g. [Ham+24]).

This thesis focuses on the generation of handwritten word images with Latin script. Given a word in the form of an ASCII string, the task of HTG is to generate an image depicting the respective word in handwriting. A common extension is to generate the word mimicking the style of the handwriting of a specific writer. To emphasize this second goal, some works refer to the task explicitly as *stylized* HTG. These objectives give rise to two fundamental challenges. First, textual correctness has to be ensured. This involves ensuring that the generated word is spelled correctly and that the generated handwriting is readable. The second challenge is the accurate imitation of the handwriting style of the desired writer. In particular, it is insufficient to merely mimic the stroke width and the texture of the background. Instead, each writer has a unique way of writing individual characters and ligatures (i. e. connections of two characters within a word).

The methods addressing this task can be grouped according to the underlying procedure. Early approaches generated individual glyphs (i. e. characters) which are then aligned and connected to construct a word image. These *glyph-based* methods are briefly reviewed in Section 3.2.1. In contrast, recent approaches directly generate entire words. These can be divided into the genration of *online* and *offline* handwriting. In online handwriting, the word is recorded as a sequence of pen-tip locations. Section 3.2.2 presents methods for online HTG which predict and render these pen trajectories to obtain word images. In the case of offline handwriting, only images of the words are given and HTG methods directly predict the pixel values. These methods are discussed in detail in Sections 3.2.3, 3.2.4, and 3.2.5.

In contrast to tasks like classification or *handwritten text recognition* (HTR) where *correctness* is well-defined, the evaluation of generation results is a challenging task. Various metrics to assess different aspects of generated image such as visual quality or correct content have been proposed. Section 3.2.6 discusses the advantages and shortcomings of these metrics for the evaluation of generated handwriting.

3.2.1 *Glyph-based HTG*

Most approaches for glyph-based HTG can be divided into two phases. First, samples of characters written by one writer are collected and analyzed. Second, a synthesis model renders word images based on that collection.

For collecting samples, Wang et al. [Wan+05] use handwritten paragraphs of about 200 words where each letter appears at least five times. The paragraphs are then automatically segmented to obtain individual characters. Konidaris et al. [Kon+07] collect image templates for each character manually. To avoid erroneous segmentation and to still make the process more efficient, Haines et al. [HMB16] used a human-in-the-loop process for a semi-automatic collection of character samples. Lin et al. [LW07] avoid the requirement of segmentation by asking the user to write isolated characters, special letter pairs and multi-letter words. In the application case in [TRG09], the collection step is not included. Since the method aims not at imitating a specific handwriting style, an existing database of handwritten characters is used. The collected samples are then used to create character templates by skeletonization [TRG09; HMB16] or to learn shape models [Wan+05; LW07]. Only Konidaris et al. [Kon+07] directly use the extracted template images which are only adjusted according to their baseline.

All methods follow a similar series of steps in the synthesis phase. First, the individual glyphs are sampled from the templates or generated by the shape models. Optionally, the connection to neighboring characters can be taken into account [LW07]. Then the letters are aligned according to their baseline and scaled to ensure correct relative sizes between the characters. At the same time, the spacing between the glyphs must be adjusted. As the goal is the generation of cursive handwriting, ligatures need to be generated. This step can be performed heuristically for all characters [Wan+05; TRG09] or based on the style of the respective writer [LW07; HMB16]. In [Kon+07], this step is skipped. For a realistic appearance, methods based on shape models and skeletons require the rendering of the strokes and the background. This can be done according to the collected samples [HMB16; TRG09] or based on pen pressure captured during the collection of writer samples [LW07]. Variations like geometric deformation are introduced on the level of individual glyphs as well as on the word-level.

While being able to synthesize realistic looking word images, the majority of these methods require a lot of manual effort for obtaining and preparing the individual character glyphs. At the same time, the synthesis pipeline requires a careful design. The requirement for expensive manual effort can be avoided when words are rendered from true type fonts [Kan+20c; KJ16]. Nonetheless, it is questionable whether such methodologies can be regarded as HTG. Krishnan et al. [KJ16] used 750 publicly available fonts to create a synthetic dataset containing nine million images. In a later work [KJ19], they used it for pretraining a model to learn representations for word

spotting. They showed that there exists a domain gap between their synthetic images and real word images and therefore proposed using transfer learning to close this gap. Kang et al. [Kan+20c] used 387 freely available computer fonts and a text corpus of over 430 000 words to generate a synthetic dataset of 5.6 million word images. They present an unsupervised writer adaptation approach to mitigate performance drops caused by the domain gap.

3.2.2 *Online HTG*

In online handwriting, a word is represented by the recorded trajectory of the pen-tip during the writing of a word. This representation is a sequence of real-valued coordinates defining the position of the pen-tip for the recorded timesteps. The coordinates can either be absolute coordinates or offsets from the previous point. If the pen position was recorded even when the tip did not touch the surface, an additional binary value can be used to indicate whether the pen was lifted. Given the need to model these sequences in the context of online HTG, autoregressive models emerge as a natural choice for this task.

Graves [Gra13] proposed a method for online HTG that is based on a *long short-term memory* (LSTM). In order to model the real-valued coordinates, the LSTM was trained to predict the parameters of a mixture of bivariate Gaussians, following the idea of mixture density networks [Bis94]. The lifting of the pen is predicted using a Bernoulli distribution. While allowing for the generation of realistic handwriting samples, HTG requires a mechanism to control the generated text. However, the discrepancy between the lengths of the input character sequence and the predicted coordinate sequence poses a significant challenge in learning the alignment between these sequences. Graves proposed to use convolution with a "soft window" along the text input. The result is then used as additional input for the LSTM to control the generated word. The readability of the handwriting could be improved by biasing the sampling process towards more probable outputs. This is implemented by decreasing the standard deviation of the mixture components by a chosen bias and adjusting the mixture weights accordingly. Thereby, the bias enables the trade-off between readability and variability in the generated samples. Graves also approaches the imitation of a specific writing style by "priming" the model. A real sample from the writer is used as input to the model before generating the desired word. Instead of generating only the requested word, the word to be generated is appended to the transcription of the real sample.

Inspired by the success of GANs in various applications, Ji and Chen [JC19] used adversarial learning to train the model from Graves [Gra13]. They use the model as a generator and add a discriminator which is specifically designed to work on features that encode geometrical and order information of stroke points. Ingle et al. [Ing+19] also base their synthesis method on [Gra13]. They add a style embedding module to the model. This style embedding module receives intermediate representations from the synthesis model as inputs. As these inputs are independent of the generated word, the authors assume that the respective features encode style information. The style vector is obtained by computing the sum of the outputs of the style embedding module and is provided as an additional input to the layer that predicts the mixture

density parameters. By computing the style vectors for each writer, they can generate samples with the respective style without the need for priming the model.

Another RNN-based HTG system was proposed by Aksan et al. [APH18]. To introduce an additional source of variability compared to LSTMs or gated recurrent units, they employ a *variational RNN* (VRNN) [Chu+15]. However, they modified the VRNN to have two latent variables instead of only one in order to model style and text as two separate latent variables. The content (i. e. the text) is modeled by a categorical latent variable while the style is captured by a continuous latent variable. Similar to [Gra13], the outputs are modeled by bivariate Gaussian and Bernoulli distributions. In addition to reconstruction terms and KL divergence terms for the latent distributions, Aksan et al. introduce additional loss terms. A classification loss should enforce that content information is encoded in the categorical latent variable. To provide control of the spacing between words in lines and the spacing between characters within a word, binary "end of character" and "beginning of word" signals are predicted. An additional log-likelihood term for these signals is added to the loss during training.

Kotani et al. [KTT20] use an encoder-decoder architecture [Cho+14] and extend the idea of separate content and style representations. Instead of using a style representation per writer, they distinguish between writer-level and character-level style. They train two separate networks. One network predicts writer- and character-specific style vectors. The other network predicts a character-specific linear transformation from a latent global writer style to the writer- and character-specific style vector. By this factorization of the style vectors, their model is able to generate handwriting in new writing styles as well as generating characters that are not in the training set by estimating their style vector.

Luhman and Luhman [LL20] are the first to employ a diffusion model (DM) to the generation of online handwriting. For the conditioning on a writing style, they, however, use a pretrained MobileNetV2 [San+18a] to extract style features from an offline handwriting sample. An attention mechanism between character-level embeddings and style features allows entangling both conditions. The resulting sequence is added to the sequence of character embeddings. The combined embeddings are then transformed and used to condition the DM.

3.2.3 *GAN-based offline HTG*

In the domain of offline handwriting, methods directly operate on images of handwriting. Typically, images are obtained through the scanning of handwritten documents. Alternatively, they can be captured by other devices like a smartphone camera [LCL21]. Consequently, temporal information of the strokes is not available. Only the resulting graylevel (or color) values for the pixels in the captured image are given.

Given their success in image generation, GANs [Goo+14] were proposed for offline HTG. A seminal approach addressing offline HTG with GANs was presented by Alonso et al. [AMM19]. A generator $\mathcal{G}$ is trained to map a random noise vector to a handwriting image. The discriminator $\mathcal{D}$ is trained to discriminate between real images and images generated by $\mathcal{G}$. In order to control which word is generated, $\mathcal{G}$ is additionally conditioned on a string encoding computed by an LSTM. The

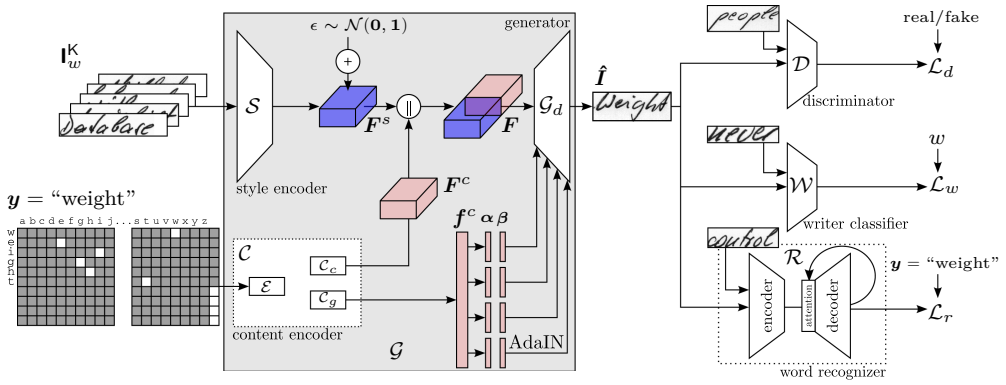


Figure 14: Architecture of GANwriting. A handwriting image $\hat{\boldsymbol{I}}$ is generated with the style of writer w and the text $\boldsymbol{y}$ by the HTG model $\mathcal{G}$. The writer embedding computed by $\mathcal{S}$, the character-wise content encoding computed by $\mathcal{C}_c$ and the global string encoding computed by $\mathcal{C}_g$ are fed to the generator $\mathcal{G}_d$. For training the model, a discriminator $\mathcal{D}$, a writer identifier $\mathcal{W}$ and an HTR model $\mathcal{R}$ are used to compute the overall loss. Figure taken from [Kan+20b] and adapted to match the mathematical notation in this work.

readability of the generated image is enforced by employing an auxiliary HTR model $\mathcal{R}$. In contrast to [OOS17], a separate model is used instead of training $\mathcal{D}$ for both tasks. To prevent $\mathcal{R}$ from learning to recognize erroneous generated handwriting images, the model is only trained on real samples. $\mathcal{G}$ is then trained by optimizing both the adversarial loss from $\mathcal{D}$ and the recognition loss from $\mathcal{R}$. However, the different architectures and loss functions from $\mathcal{D}$ and $\mathcal{R}$ require a careful balancing of the gradients when updating $\mathcal{G}$ [AMM19]. Furthermore, the introduction of the HTR model leads to the requirement of handwriting images with transcriptions in contrast to the basic GAN (only a generator and a discriminator) which can be trained in an unsupervised fashion. The previously described combination of the generator, discriminator, and an auxiliary model constitutes the foundation for many subsequent methods. Nevertheless, the initial approach from Alonso et al. [AMM19] is subject to certain limitations that were subsequently addressed by other works.

A significant constraint of [AMM19] is its inability to control the style of the generated handwriting. With *GANwriting*, Kang et al. [Kan+20b] addressed this shortcoming by extending the system by a *style encoder*. The overall architecture is depicted in Figure 14. The style encoder computes calligraphic features based on a set of $\mathsf{K} = 15$ random example images from a writer. These features can then be used to condition the generator on the desired writing style. For the textual conditioning, they compute a character-wise as well as a global embedding for a given string. Variations in the style of a writer can be imitated by randomly sampling different sets of examples and by adding random noise to the writer embedding. Similar to the HTR model in [AMM19] that enforces the generation of readable words, Kang et al. [Kan+20b] introduce a writer classifier to guide the generation of the desired calligraphic style. The writer classifier is trained only on real samples optimizing a cross-entropy loss. For generated images, the loss provides an additional training signal for the generator. Nevertheless, this approach entails the disadvantage of necessitating the annotation of writer IDs in addition to the transcriptions which are required by the HTR model.

The extraction of writing styles by a style encoder has been employed in numerous other works on HTG [Kan+20a; Dav+20; Gua+20; GW21; Bhu+21; Mat+21; Liu+21; Kri+21; Kan+22; Gan+22; Wan+22; ZN23; Luo+23; PCC23; Van+25; Wan+25]. However, there are some considerable differences. The style encoder from Davis et al. [Dav+20] receives the predicted transcription of the style sample allowing a character specific style extraction. Furthermore, the spacing between letters is modeled explicitly by an additional *spacing network*. Other works [Bhu+21; Wan+22; ZN23; PCC23; Van+25] employ transformer models to allow for the capturing of global style information by the style encoder. By cross-attention in the decoder, the generator can then incorporate the style information into the generation process depending on the given textual conditioning. Wang et al. [Wan+25] follow this idea but extend the encoder by a state space model [GGR22; GD24; Liu+24] to capture style features at various levels.

Most approaches use a separate model for their style encoder and writer classifier. Gan et al. [Gan+22] simplified this setup by using the first layers of the writer classifier for the extraction of style features. At the same time, this setup allows to pretrain the model to obtain more robust style features and circumvent the necessity of retraining for different generators [Gan+22]. This simplification was adopted by Zdenek and Nakayama [ZN23] for their style encoder that is based on a vision transformer. However, they train the model jointly with their GAN.

Besides modifications to the architecture of the style encoder, additional objective functions have been utilized to improve the imitation of styles. To ensure that the style was disentangled from the content of the style image and that the generator correctly used the encoded information, a cycle consistency of style features is utilized by several methods [GW21; Bhu+21; Kri+21; Gan+22; ZN23; PCC23; Van+25; Wan+25]. This consistency is enforced by a loss term defined by the distance (e. g. an L1 distance) between the style encodings extracted from the style example and the extracted encoding from the generated image. In addition, certain works [GW21; Liu+21; Gan+22; ZN23] use a KL divergence to regularize the latent space of the style encoder, thereby enforcing it to follow a standard normal distribution. Style encodings sampled from $\mathcal{N}(\mathbf{0}, \mathbf{1})$ allow the generation of handwriting images with random styles in addition to the imitation based on examples. However, the incorporation of additional loss terms further amplifies the necessity for weighting these terms or for balancing the gradients.

There exist other approaches [Dav+20; May+20; Gua+20; Liu+21; Kri+21] able to imitate a specific style, which do not employ an auxiliary writer classifier. Instead they rely on reconstruction losses (e. g. an L1 distance) and perceptual losses [JAF16] between real and generated samples with identical content and style. Consequently, these methodologies do not require the annotation of writer IDs, but solely the transcription of the training images.

Mayr et al. [May+20] propose a hybrid approach of online and offline HTG. They first approximate the pen trajectory (i. e. online handwriting) based on a handwriting image. For the generation and imitation of writing styles with respect to strokes and shapes they use the online HTG system from Graves [Gra13]. The result is then rendered to obtain an image. For imitating the remaining aspects of the style like ink, stroke width, or background texture, they employ image-to-image translation [Iso+17].

Besides the aforementioned aspects, the GAN-based approaches to offline HTG also differ in the number of samples that are required for style extraction. Various works [Kan+20b; Bhu+21; Mat+21; Wan+22; PCC23; Van+25; Wan+25] followed Kang et al. [Kan+20b] and used $\mathsf{K} = 15$ style samples for extracting and encoding style information for a writer. However, K is usually an adjustable hyperparameter. The previously mentioned methods that do not use writer labels, only use a single image for style extraction. Other works [Kan+20a; GW21; Gan+22; ZN23] that assume to have writer labels available also address style imitation based on a single example from the respective writer.

Another distinguishing characteristic of handwriting as opposed to natural images is its variability in width and aspect ratio. *ScrabbleGAN* [Fog+20] was the first work tackling this challenge for offline HTG and generating handwriting images of variable width. Instead of conditioning their generator on an entire word representation, Fogel et al. proposed to use a filter bank with a filter for each character in the alphabet. The initial feature map is then generated by concatenating the filters of the characters of the word to be generated. A fully convolutional generator is then used to generate an image of the respective word based on this feature map. Overlapping receptive fields between neighboring characters allow for the generation of ligatures. The multiplication of the initial feature map with a random noise vector allows for variance in writing style but not for the imitation of a specific style.

Gan et al. followed this method for generating words of arbitrary length for their models *HiGAN* [GW21] and *HiGAN+* [Gan+22]. Zdenek and Nakayama [ZN21; ZN23] argued, that using one filter per character becomes inefficient for large alphabets (e. g. Chinese). Therefore, they proposed to use a single base filter that transforms the sequence of character embeddings for the respective word. The resulting sequence is then concatenated horizontally to form a base feature map. Similarly, Liu et al. [Liu+21] concatenate one-hot encodings of the characters horizontically to create a base feature map. Instead of making the width of the base feature map directly proportional to the length of the word, Davis et al. [Dav+20] learn the width by predicting *spaced text* based on the word and the extracted style. Bhunia et al. [Bhu+21] and Wang et al. [Wan+25] use a model based on a transformer encoder and a transformer decoder. As the transformer decoder receives the encoded string as an input, an output with a corresponding length is generated.

Instead of using learned filters of one-hot encodings to create a base map, several works render the word to be generated using a computer font. The resulting image [Kri+21; CPK23; Luo+23] or the sequence of character images [PCC23; Van+25] is then used as input to the generator that provides a base feature map with a suitable aspect ratio. At the same time, this approach to encode the textual conditioning provides information about the basic shape of the respective characters. Therefore, Pippi et al. [PCC23] refer to the characters rendered from a computer font as *visual archetypes*. Figure 15 shows the architecture of *Vatr++* [Van+25] which is an improved version of the HTG model proposed by Pippi et al. [PCC23] and relies on visual archetypes.

The hybrid approaches [Gua+20; May+20] only perform the style transfer in the offline handwriting domain. Therefore, they rely on the provided [Gua+20] or generated [May+20] online handwriting to account for the variable length of words. Other approaches like the extension of GANwriting to the generation of text lines [Kan+22] do not allow for the generation of words or sentences of arbitrary length. However,

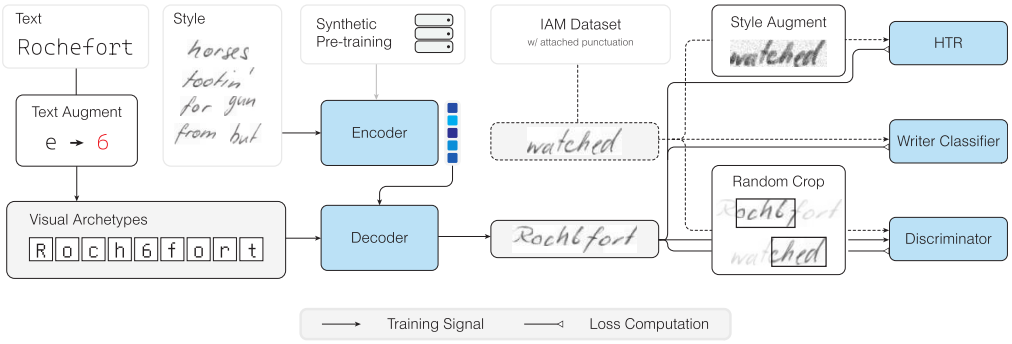


Figure 15: Architecture of Vatr++. Writer embeddings are computed by a style encoder that is pretrained on synthetic data. The content encoding is provided by a linear projection of the visual archetype representation of the string to be generated. A training of the generator is guided by a discriminator, a writer classifier and an HTR model. Figure taken from [Van+25].

they allow for a variable length up to a chosen maximum by padding shorter strings with placeholders (e. g. whitespaces).

In addition to approaches for dealing with the two central challenges, imitation of specific styles and generation of words of arbitrary length, further improvements to HTG systems were proposed. Mattick et al. [Mat+21] extended GANwriting by an additional local discriminator [Iso+17] in order to reduce pen-level artifacts in the generated images. The idea of using an additional local discriminator was also implemented in subsequent works [Gan+22; Luo+23]. Gan et al. [Gan+22] improve the style transfer by utilizing a contextual loss [MTZ18]. This loss relies on high-level style features and allows slight spatial deformations of the generated images compared to the real images. Wang et al. [Wan+22] addressed stroke level artifacts by an additional focal frequency loss term [Jia+21]. Zdenek and Nakayama [ZN21] suggested to additionally condition the generator on vertical layout information based on baseline, mean line, ascenders and descenders in the target word. Vanherle et al. [Van+25] proposed to improve style extraction by attaching punctuation marks to neighboring words instead of treating them as separate words.

In summary, there exist a variety of GAN-based approaches addressing the task of offline HTG. The primary focus of these studies is the extraction and generation of handwriting styles of specific writers. To this end, a range of models and enhanced optimization criteria were examined. Furthermore, methodologies for generating words of an arbitrary length were presented, as well as methods for reducing artifacts in the generated images.

3.2.4 *DDPM-based offline HTG*

While the application of GANs allowed to guide the training process towards learning content and style generation by using multiple loss terms, this results in the necessity of gradient balancing for the training to converge [Dav+20]. Besides this disadvantage of GAN-based HTG, it has been demonstrated that denoising diffusion probabilistic models (DDPMs) exhibit superior performance in generating natural images when compared to GANs [DN21]. Consequently, these models are considered a promising foundation for HTG methods. The first approach, which success-

fully used a DDPM for online HTG, was published as early as 2020 [LL20]. However, DDPM-based methods for offline HTG were not presented until 2023 [Zhu+23; Din+23; Nik+23].

Zhu et al. [Zhu+23] proposed a DDPM based on a UNet that can be conditioned on different combinations of encodings of the text, the style and the image. The image conditioning is obtained by attention pooling [Lee+19] of features from an HTR model. The conditioning should encode general visual features like texture and colors [Zhu+23]. The text condition is obtained by multiplying the classifier weights of the pretrained HTR model with the one-hot encoded characters of a string. The personal style of writers is encoded by a learned embedding for each writer based on its writer ID. Depending on which of these conditions are given, the model can be used for synthesis, augmentation, recovery and imitation of known styles.

Inspired by a DDPM that was presented for the generation of handwritten Chinese characters [Gui+23], Ding et al. [Din+23] conditioned their DDPM on glyph images instead of text. This is similar to GAN-based methods like [Luo+23]. The style embedding is provided by learned and normalized embeddings based on the writer IDs. Since DDPMs generate images at different noise levels during training, the approach of using an auxiliary classifier to guide the training complicates the training procedure [HS21]. Instead of using an HTR model and a writer classifier like [Kan+20b], Ding et al. utilize classifier-free guidance (CFG) for their DDPM. They adopt it to their model that is conditioned on two modalities. Thereby, the impact of the style and content conditioning in the sampling process can be adjusted by two independent guidance scales.

Both of the previously discussed works perform the diffusion process in the pixel space. In order to increase the computational efficiency of the model, Nikolaidou et al. [Nik+23] proposed *Wordstylist*, which is based on a latent diffusion model (LDM). They use a pretrained VAE to map word images into a latent space. The reverse diffusion process is learned by a UNet that operates in the latent space. Similar to [Din+23; Zhu+23], the style embedding is learned based on writer IDs. For the content conditioning, a sequence of learned character embeddings is processed by self-attention. The architecture during training is depicted in Figure 16.

As is the case with this thesis, a number of works use Wordstylist as a foundation. Gurav et al. [GCK24] modify the architecture to enable the generation of longer words and text lines. The processing of the content and style conditioning is kept identical with Wordstylist. In another work [GKC24], the text encoding of Wordstylist is concatenated with the *pyramidal histogram of shapes* (PHOS) [Bha+22] representation of the generated word. The PHOS representation of a word is a concatenation of histograms at different scales counting the occurrences of sub-shapes in characters. Thereby, more detailed structural information is provided to the generative model [GKC24]. Lin et al. [Lin+25] also improve upon Wordstylist by providing structural information about the generated word. In contrast to [GKC24], they provide this information by a glyph image of the generated word. The word is rendered using a computer font and the resulting image is encoded by the same VAE as used in their LDM. The resulting latent representation is then concatenated with the noisy latent variable of the diffusion process and used as the input for the DM. Riaz et al. [Ria+24] follow a similar idea for providing glyph information but train their DM directly in the pixel space instead using a more efficient latent space.

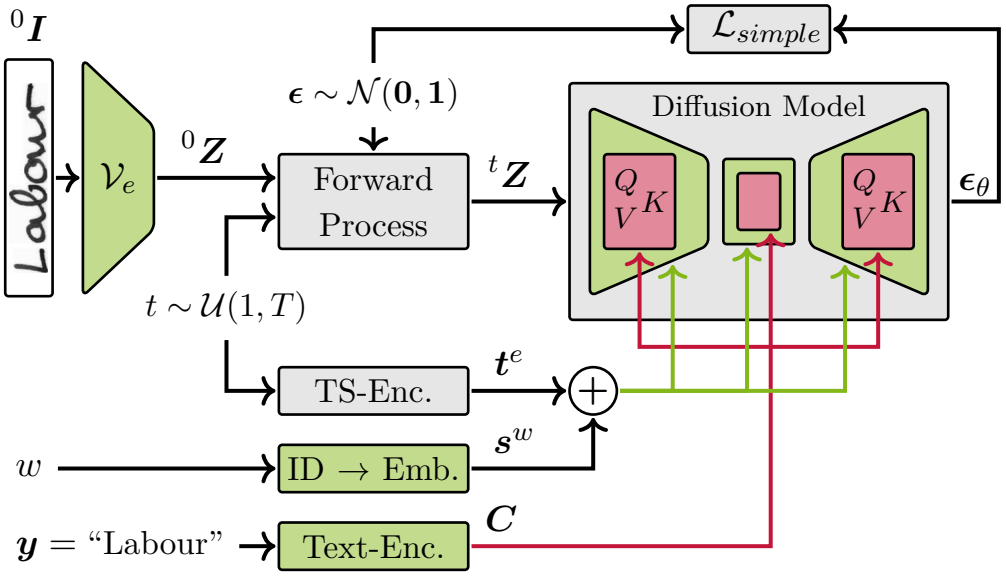


Figure 16: Architecture of Wordstylist [Nik+23] during training. The input image $\boldsymbol{I}$ is projected into the latent space by the encoder $\mathcal{V}_e$. The DM is conditioned on the encoded timestep $\boldsymbol{t}^e$ of the diffusion process, the encoding $\boldsymbol{C}$ of the string to be generated, and a style conditioning, that that is obtained by mapping writer ID w to a learnable embedding $\boldsymbol{s}^w$. The model is trained to predict the noise $\boldsymbol{\epsilon}$ that is added to the latent representation $\boldsymbol{Z}$ in timestep t.

However, all of these works lack the ability to imitate the style from a writer that is not in the training set. Nikolaidou et al. [Nik+24a] addressed this significant limitation with their recent model, dubbed *DiffusionPen*. It is an enhanced version of their prior Wordstylist model that was developed independently at the same time as the approach [Bra24], which this work is mainly based on. Similar to several GAN-based methods, the style embedding is extracted using a set of sample images from a writer instead of learning an embedding for each writer ID. Nikolaidou et al. [Nik+24a] employ a MobileNetV2 [San+18a] which they train with a classification and a triplet loss for learning a style representation. During the training of the LDM, the weights of the style encoder are kept fixed and style embeddings are computed by using $\mathsf{K} = 5$ samples per writer.

Another DDPM-based approach that allows to generate unseen styles was presented by Dai et al. [Dai+24]. Their style encoder comprises two sub-networks. One model extracts information directly based on the style image. The second network extracts information from a high-pass filtered version of the style image. The extraction of discriminative features is encouraged by a contrastive loss on the high-frequency style features. The encodings from both networks are combined using multi-head cross-attention with queries that are based on glyph images of the characters in the word to be generated. In contrast to DiffusionPen, the model can imitate writing styles based on a single style example. Furthermore, Dai et al. utilize CFG for training and sampling.

The first approach that attempts to generate offline handwriting on paragraph level using a DDPM was presented by Mayr et al. [May+24]. To allow the extraction of writing styles from new writers, they utilize a trained writer classifier, in which the final layer is discarded. Similar to other approaches, they employ an LDM and

use CFG. In contrast to the previously described methods, they do not use a VAE that was pretrained on natural images (e. g. from [Rom+22]) but train their VAE on handwritten documents.

In summary, several approaches based on DDPMs have been explored for offline HTG. While some models learn the reverse diffusion process in the image space, other methods take advantage of the increased computational efficiency of LDMs. However, methods like CFG are only utilized by a few works. Furthermore, only three of the reviewed works allow the generation of writing styles that were not present in the training set.

3.2.5 *Autoregressive offline HTG*

While the task of generating offline handwriting has largely been dominated by GAN-based and later also DDPM-based methods, Pippi et al. [Pip+25] recently presented an autoregressive approach for offline HTG. In analogy to LDMs, they train the autoregressive model in the latent space of a VAE. Using a large synthetic dataset of text lines, they pretrain the VAE optimizing a loss comprising a reconstruction term, a KL divergence term, an HTR term, and a writer classification term. The latent representation of a text line image computed by the VAE is considered as a sequence of representations corresponding to vertical slices in the image. The autoregressive model is an encoder-decoder transformer with the same architecture as [Raf+20]. The encoder computes an embedding sequence based on the string that should be generated. The decoder receives a sequence of embedding vectors computed by the VAE encoder for a style image. Due to the autoregressive task, the self-attention on this sequence is masked accordingly. At the same time, the decoder can attend to the full text embedding sequence by cross-attention. The VAE decoder is then used to reconstruct the image from the sequence generated by the transformer decoder. Training is performed by solely optimizing the *mean squared error* (MSE) between the embedding vectors extracted from the style image and those predicted by the decoder. Thereby, the training process is more controllable and less resource-demanding compared to GAN-based and DDPM-based methods. New text lines can be generated by providing a style image to the model and appending the desired string to the transcription of that image.

3.2.6 *Evaluation of HTG*

The evaluation of HTG systems is a challenging task [Pip+23]. There exists a plethora of evaluation measures, most of which have been adopted from the evaluation of natural image generation. The difficulty of the evaluation of generated styled handwriting images lies in the fact that different aspects have to be taken into account at the same time [Gan+22; ZN23]. Similar to natural images, the generated images should look realistic and exhibit high visual quality. Additionally, it is essential that the generated images show the correct content. The generated handwritten word must be readable and correctly spelled. Specific to the problem of HTG is that the generated handwriting oftentimes should mimic the style of a specific writer. Furthermore, the generated images should be diverse while still capturing the original data distribution accurately. These criteria can be used to group

metrics according to the aspects they take into account: Visual quality, readability, style imitation, and diversity and fidelity.

A natural way to assess the visual quality is to conduct a user study in which humans have to identify the generated images in a mix of real and generated samples. Based on the results, different metrics like the accuracy, precision, or recall can be computed [EB24]. However, an evaluation that involves human feedback is costly. Therefore, metrics that can be computed automatically based on the generated images are appreciable. A straight-forward approach is to measure the reconstruction error between real and generated samples in terms of *peak signal-to-noise ratio*. Alternatively, the *structural similarity index measure* [WSB03] can be used as a measure of structural similarity which is assumed to approximate perceived image quality [WSB03].

The computation of the aforementioned metrics, however, is only possible if all generated images can be compared to a corresponding real image which cannot be guaranteed in general for HTG applications. Therefore, metrics based on the sets of real and generated images were proposed. The *geometry score* [KO18] compares geometrical properties of estimates of the underlying data manifolds of the real dataset and the generated dataset. Another metric that was used by Gan et al. [Gan+22] for the evaluation of their HTG model is the *inception score* [Sal+16]. It is computed based on the predicted label distribution for all generated images using an InceptionV3 model [Sze+16] that was trained on ImageNet [Den+09]. A shortcoming of the inception score is that it does not take real samples into account [Heu+17].

The most commonly used metric for an assessment of the visual quality is the *Fréchet inception distance* (FID) [Heu+17]. An InceptionV3 model that was trained on ImageNet is used to extract features from a set of real and a set of generated samples. Instead of using the predicted label distribution, the outputs of the last average pooling layer are used as features. Based on the assumption that these features follow a Gaussian distribution, the FID is defined by the *Fréchet distance* between these distributions. The *kernel inception distance* (KID) [Bińkowski+18] follows a similar idea but without the assumption of Gaussian distributions. Instead of the Fréchet distance, the *maximum mean discrepancy* between the estimated distributions is used as the score.

When applied to evaluate the visual quality of generated handwriting images, however, the aforementioned metrics have considerable shortcomings [Kan+22]. Since the InceptionV3 model is trained on natural images, it is doubtful that the model learned to extract handwriting-related features. At the same time, the architecture was designed to process fixed-size inputs, which are typically square. In contrast, for a fixed height, images of handwriting have variable width which mainly depends on the length of the displayed word. Kang et al. [Kan+22] addressed these issues and proposed a new metric referred to as *variable-length Fréchet inception distance* (vFID). Their proposed metric accounts for the variable width by replacing the average pooling by a temporal pyramid pooling [Wan+17]. In order to obtain handwriting related features, they train the InceptionV3 model for writer identification on handwriting images from the IAM database [MB02].

Pippi et al. [Pip+23] proposed the *handwriting distance* (HWD), that follows a similar idea as the FID and vFID. However, they employ a VGG16 [SZ15] backbone that is pretrained for font classification on a synthetic dataset of text images. They extract a set of feature vectors per image to account for the variable width of hand-

writing images. The HWD per writer is obtained by the Euclidean distance of the mean feature vector from the real and from the generated images. The overall HWD is defined as the average of all per-writer HWDs.

However, all of the previously explained metrics only focus on the visual quality and do not consider correct content nor correct style. Therefore, Gan et al. [Gan+22] proposed to assess performance in style imitation by performance measures of proxy tasks. They use the same InceptionV3 model for writer identification as in [Kan+22] and measure its *writer identification error rate* (WIER) on generated images. The correctness and readability of the generated content can be assessed in a similar fashion. An HTR model is trained on real images and used to predict the transcriptions of the generated word images. The error rate in these predictions is then used as a performance measure. This evaluation approach was first used in [ZN21] and [Mat+21].

Two other metrics that were proposed to measure diversity and fidelity of images generated by GANs have been adapted to the evaluation of HTG systems (e. g. [ZN23; Nik+24b; Gua+20; Nik+23]). In GAN-test [SSA18], a classifier is trained on real samples. Its prediction accuracy on the generated images is then used as a measure of fidelity of the generated images, requiring correctness as well as a good generation quality. Applied to HTG, this metric is similar to the previously described measure of readability [ZN23]. As GAN-test does not account for the diversity of the generated samples, Shmelkov et al. [SSA18] proposed GAN-train for this purpose. This metric is defined by the accuracy on real test samples of a classifier that has been trained only on generated samples. Using these samples as training data requires the generative model to be able to generate a representative set with respect to the original dataset. For the evaluation of HTG systems, an HTR model is used instead of a classifier. While requiring correctly generated content, it reflects the model's ability to replicate the variability in the writing styles in the original training set [Nik+24b]. At the same time, the metric indicates the usefulness of the generated samples for training a downstream model, e. g., an HTR model.

Some works on HTG (e. g. [May+20; Kan+22; CPK23]) focus on a few specific metrics which are considered favorable by the respective authors. Other works (e. g. [Gan+22; ZN23; Nik+24a]) base their evaluation on several of the aforementioned measures to account for different aspects in the generation of handwriting images or propose a structured evaluation protocol (e. g. [Nik+24b]). However, in light of the aforementioned variety of metrics, which focus on different aspects and present various shortcomings, a common evaluation protocol has yet to be established for HTG.

3.3 DISCUSSION

In the previous section, an overview on different methods for HTG is presented. The majority of these methods focus on the visual quality of the generated samples through either human evaluation or evaluation measures such as FID, KID, or HWD. Furthermore, the readability of the generated samples is typically assessed. While visual quality and readability are important criteria, they offer no insights into the usefulness of generated images if they are to be utilized for training other models. This use of generated handwriting images is implemented and evaluated to varying

Table 1: Overview of works on HTG categorized based on the underlying generative model and the evaluated setting for training data generation. Autoregressive models are abbreviated with *AR*.

Setting	GAN-based	DDPM-based	AR
(a)	[AMM19; Fog+20; Kan+20a; Gua+20; GW21; ZN21; Liu+21; Kan+22; CPK23; Luo+23; ZN23; Wan+25]	[Zhu+23; Din+23; Nik+24a; GKC24; GCK24]	
(b)	[Gua+20; ZN21; ZN23]	[Zhu+23; Din+23; Nik+23; Nik+24a; Ria+24; GCK24]	
(c)	[Fog+20; ZN21; GW21; Bhu+21; Kan+22; Luo+23]	[Zhu+23; May+24]	[Pip+25]

degrees by the reviewed works. In particular, the generation of training data for a new dataset without available annotation is of interest. In this context, the annotated dataset, which the generative model is trained on, is termed *source dataset*. The new and potentially unlabelled dataset is referred to as the *target dataset*. The application of HTG for training data generation can be categorized as follows:

(a) Augmentation of the source dataset. The HTG model is trained on the source dataset and then used to generate samples. These samples are used in addition to the source dataset for training an HTR model. Several works explored this application of their HTG systems (cf. Table 1, first row). Although this augmentation technique has been demonstrated to enhance recognition performance, it is considerably more computationally expensive than conventional augmentation methods. However, significant improvements can be achieved with this approach, if the source dataset is very small (e. g. only 5 000 samples) [ZN21; ZN23]. A reranking based on the readability of the generated samples can further enhance the results [Din+23; May+24]. As the readability is evaluated by a pretrained HTR model, this technique is only applicable when augmenting a labeled dataset.

(b) Training using only generated samples. The HTG model is trained on the source dataset and generates a set of training samples. The HTR model is then only trained on the generated samples. This approach is not desirable in practical application, as available annotated data, i. e. the source dataset, is not used for training the HTR model. However, it provides a proxy measure for how well the generated samples are suited as training data. This setting is therefore explored in various works on HTG (cf. Table 1, second row).

(c) Generation of training samples for an unlabeled target dataset. The HTG model is trained on the annotated source dataset. Depending on the method, unlabeled samples from the target dataset can be included in a semi-supervised fashion [Fog+20; ZN21; ZN23; CPK23]. The HTG model is then used to generate samples which better correspond to the target dataset in terms of writing styles and the lexicon. The labeled source dataset and the generated samples are then used to train the HTR model which is evaluated on the test set of the target dataset,

afterwards. While this application scenario is explored for a number of GAN-based approaches, only two works employ a DDPM for this task (cf. Table 1, third row). Kang et al. [Kan+22] argue, that it is also realistic to have a few samples from the target dataset for which labels are available. They include these samples from the target dataset in the training of the HTR model. Pippi et al. [Pip+25] take a further step in this direction, using not a real dataset (e. g., IAM) as their source dataset, but rather a synthetic dataset they have created themselves with over 2 million examples.

The discussed categorization of works on HTG with respect to their application for training data generation (cf. Table 1) reveals that in most cases, the generated samples were used to augment the training set of the source dataset for an HTR model. Training of the HTR model exclusively on generated samples has primarily been explored in works that present a DDPM-based HTG system. The most promising application of training data generation, namely the generation of annotated samples for a new, unseen target dataset, is considerably less explored in the HTG literature. Notably, only two works employing a DDPM have conducted a limited number of experiments that address this task. Furthermore, integrating unlabeled samples from the target dataset in a semi-supervised training was only explored by four methods based on GANs. For HTG systems based on DDPMs, a comparable semi-supervised training of the model has not been investigated yet. The findings thus underscore the necessity for continued research in this area.

4 TRAINING DATA GENERATION

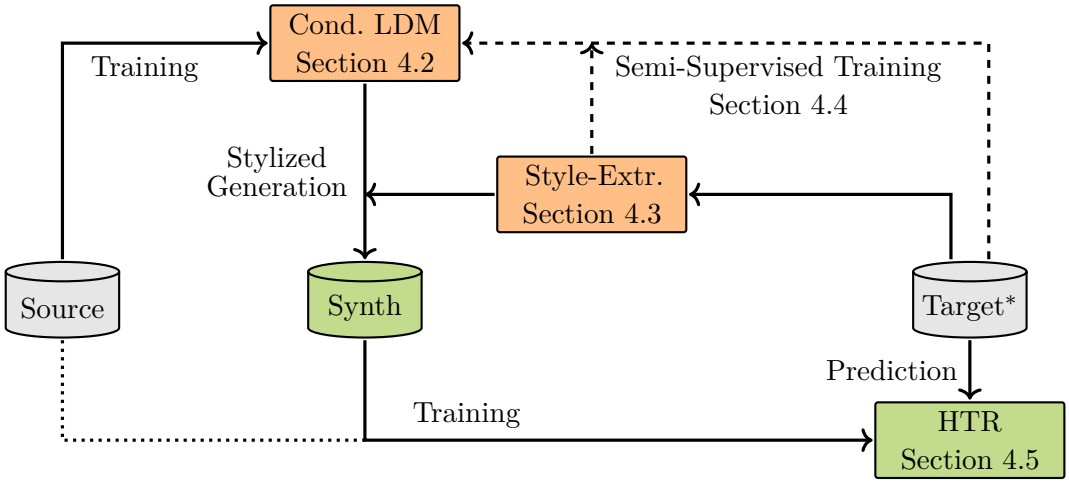


Figure 17: Overview of the proposed system for the generation of training data for an HTR model, which is applied on a target dataset. The source dataset provides full supervision with annotated transcriptions and writer IDs. For the target dataset, either only writer IDs or no annotations at all are available.

In the previous chapter, we gained a comprehensive overview of approaches and their specific applications to the problem of *handwritten text generation* (HTG). Most approaches focus on the generation of samples with a highly realistic appearance instead of samples which are suitable as training data. In contrast, the primary goal of this work is the development of an HTG system which is used to generate labeled training samples for an *handwritten text recognition* (HTR) model. Before presenting our method, the problem of HTG is first formally described in Section 4.1. While HTG models can be used to augment the dataset on which they were trained on, the far more interesting application is how these models can be adapted to generate training data for unseen datasets for which no annotations are available. In this application, however, the difficulty arises that the calligraphic styles to be generated can differ significantly from those seen during training of the generative model, often resulting in poor generation quality. In the following, the annotated dataset will be referred to as the *source dataset* and the unseen dataset without annotations will be referred to as the *target dataset*.

Our contribution to the problem of HTG is a system which can be adapted to new calligraphic styles from an unlabeled datasets by incorporating samples from the target dataset in a semi-supervised training scheme. The obtained system is capable of generating a synthetic dataset with writing styles similar to the target dataset. The generated dataset can then be used to train an HTR model that operates on the target dataset. An overview of our approach is presented in Figure 17. A conditional *latent diffusion model* (LDM) including our improved *content encoder* constitutes the backbone of the proposed system. Given a string as a textual conditioning and

an embedding encoding the calligraphic style, the content encoder combines the information into a conditioning sequence for the LDM which is then used to generate the given text in the desired writing style. Section 4.2 describes the LDM and the processing of the conditioning by the content encoder in detail. While the character encoding is learned by our content encoder, we train a separate *style encoder* (SE) for computing the style embeddings. For this purpose, we employ a *masked autoencoder* (MAE) which is trained in a self-supervised fashion on unlabeled images which can be from the source as well as the target dataset. The architecture and the training of the model as well as the computation of the style embeddings are described in Section 4.3. While the model described so far is technically capable of generating word images in the style of an arbitrary writer (given a few style examples), empirical evidence shows that the generation quality for styles not seen during training is often poor. In order to facilitate the transfer of the LDM for generating styles from the target dataset, we propose a semi-supervised training scheme. Thereby, we are able to exploit information about the new styles either from examples that are only annotated with writer IDs or from examples without any annotation. The training procedures for both settings is described in Section 4.4.

The ultimate goal of the method presented in this work is to improve the recognition performance of an HTR model on a target dataset. The obtained HTG system is used to generate a synthetic dataset containing samples with a calligraphic style similar to those in the target dataset. The HTR model is then trained on this stylized synthetic dataset. The model used in this thesis is described in Section 4.5. Additionally, the section provides details on how *self-training* can be applied to adapt the recognizer further to the target dataset.

4.1 PROBLEM FORMULATION

As introduced in Section 3.2, the goal of HTG is to generate a realistic looking image depicting the given string in a handwriting with a desired style. Let $\mathbf{A}$ be a set of all characters that we want to generate (e. g. letters, digits, punctuation). $\mathbf{A}$ is referred to as our *alphabet*. Next, the *lexicon* $\mathbf{L_A}$ is defined to be the set of all strings which only contain characters in $\mathbf{A}$. Given a set of writers $\mathbf{W}$, HTG can be defined as the problem of finding a mapping $\mathcal{G} : \mathbf{L_A} \times \mathbf{W} \mapsto \mathcal{I}$ such that a generated image $\hat{\boldsymbol{I}} = \mathcal{G}(\boldsymbol{y}, w)$ depicts the handwritten word with the desired transcription $\boldsymbol{y}$ while resembling the calligraphic style of writer w. While generative models in other domains like natural image generation usually work on RGB images $\mathcal{I}_{\text{RGB}} \subseteq \mathbb{R}^{(\mathsf{h} \times \mathsf{w} \times 3)}$, this work focuses only on grayscale images $\mathcal{I} \subseteq \mathbb{R}^{(\mathsf{h} \times \mathsf{w})}$, due to the target task of HTR. h and w denote the height and the width (in pixels) of the images.

For most HTG methods including GAN-based and DDPM-based approaches, $\mathcal{G}$ is learned based on an annotated multi-writer dataset $\mathbf{D} = \{(\boldsymbol{I}^{(i)}, \boldsymbol{y}^{(i)}, w_i)\}_{i=1}^{\mathsf{N}}$. The set of all writers in $\mathbf{D}$ is denoted as $\mathbf{W_D}$ and the set of all transcriptions in $\mathbf{D}$ is denoted as $\mathbf{L_D}$. The latter is also referred to as the lexicon or the *vocabulary* of the dataset. For an annotated dataset, every image $\boldsymbol{I}^{(i)} \in \mathcal{I}$ is labeled with its transcription $\boldsymbol{y}^{(i)} = (y_1, y_2, \ldots, y_{l_i})^T \in \mathbf{L_D}$ and its corresponding writer identifier $w_i \in \mathbf{W_D}$. Note that the timestep index t is omitted if $t = 0$, e. g. $\boldsymbol{I}^{(i)} = {}^0\boldsymbol{I}^{(i)}$ for better readability when not describing the diffusion process itself.

A key objective of learning $\mathcal{G}$ is that the model is capable of generating images for strings $\boldsymbol{y} \in \mathbf{L_D}$ as well as strings $\boldsymbol{y} \notin \mathbf{L_D}$. These cases are termed *in-vocabulary* (IV) and *out-of-vocabulary* (OOV), respectively. Many HTG models do not directly learn the generation of a specific style from the writer ID w but from a set of example images of the respective writer. The subset of all images written by writer w is referred to as $\mathbf{I}_w = \{\boldsymbol{I}^{(i)} | w_i = w\}$ with $\mathsf{N}_w = |\mathbf{I}_w|$ and $\mathbf{I}_w^{\mathsf{K}} \subseteq \mathbf{I}_w$ being a subset of $\mathsf{K} \leq \mathsf{N}_w$ random examples from writer w. The specification of a writing style based on a set of example images instead of a symbolic writer ID enables the generation of calligraphic styles which are not present in $\mathbf{D}$. In addition to the distinction of IV and OOV, we therefore also distinguish between seen and unseen calligraphic styles.

4.2 CONDITIONAL IMAGE GENERATION

The central component of our approach is the generative model including its conditioning mechanism which is explained in detail in this section. There are different choices of generation mechanisms which were reviewed in Section 3.1. In HTG, *generative adversarial networks* (GANs) and *denoising diffusion probabilistic models* (DDPMs) are the most frequently used generative methods with a clear trend towards DDPMs in state-of-the-art works. Therefore, we also choose to use a DDPM for our HTG system. In particular, we employ an LDM for performing the diffusion process in the latent space of a powerful pretrained *variational autoencoder* (VAE) which results in high generation performance while significantly reducing the computational costs [Rom+22]. Despite the choice of the underlying generative method, the task of HTG additionally requires the development of a conditioning mechanism to control for the textual content as well as the calligraphic style.

Our proposed generative model is based on a model named Wordstylist [Nik+23]. However, we introduce significant changes to this approach. In contrast to Wordstylist, we condition our generation process on writer embeddings from a dedicated model instead of a learnable embedding for every writer from the training set. This enables the generation of writing styles which are not present in the training set which is essential for the goal of generating training data with styles from a target dataset. Details of how the writer embeddings are obtained will be discussed later in Section 4.3. In addition to the changes in the conditioning passed to the model, we also investigate more sophisticated ways of processing and entangling the style and text conditioning. In this section, first, the overall system will be described. Training and sampling will be explained without going into detail about the internal mechanisms of the individual components. These architectural features will then be detailed in Section 4.2.1 and Section 4.2.2.

Training

The overall architecture of our system during training is depicted in Figure 18. Algorithm 3 formally describes the procedure for training the LDM and the content encoder using *classifier-free guidance* (CFG). An image $\boldsymbol{I}$ from writer w, its transcription $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T$ and a set $\mathbf{I}_w^{\mathsf{K}}$ of examples from writer w are provided. A style embedding vector $\boldsymbol{s}^w = \mathcal{S}(\mathbf{I}_w^{\mathsf{K}})$ for the writer is either computed by our SE or randomly drawn from the style cache $\mathbf{S}_w$ (see Section 4.3.3). The given

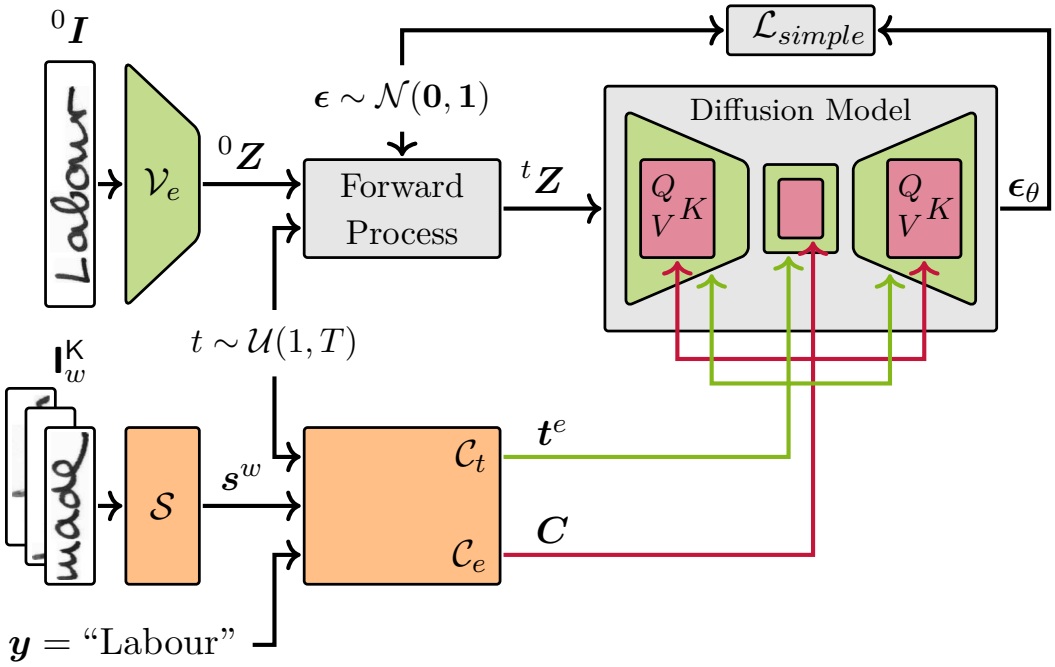


Figure 18: Overview of the proposed system during training given an input image $\boldsymbol{I}$, its transcription $\boldsymbol{y}$ and a set $\mathbf{I}_w^{\mathsf{K}}$ of examples from the writer of $\boldsymbol{I}$. The system comprises the encoder $\mathcal{V}_e$ of the VAE, style embedding function $\mathcal{S}$, content encoder $\mathcal{C}_e$ and the diffusion model (DM) implemented by a UNet architecture. The blocks containing “Q,K,V” indicate the spatial transformers for incorporating the content embedding $\boldsymbol{C}$. The DM $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})$ is trained to predict the noise $\boldsymbol{\epsilon}$ added in step t of the forward diffusion process. Instead of computing the style embedding by $\mathcal{S}$ from samples $\mathbf{I}_w^{\mathsf{K}}$, it could also be drawn from a precomputed cache $\mathbf{S}_w$. The proposed changes to Wordstylist are highlighted in light orange.

image ${}^0\boldsymbol{I}$ is passed to the VAE encoder $\mathcal{V}_e$ to compute its latent representation ${}^0\boldsymbol{Z} = \mathcal{V}_e({}^0\boldsymbol{I})$. Then, ${}^t\boldsymbol{Z}$ is obtained by adding noise in the forward diffusion process according to a randomly sampled timestep $t \sim \mathcal{U}(1, T)$. The string $\boldsymbol{y}$, the writer embedding $\boldsymbol{s}^w$, and the timestep t are passed to the content encoder which comprises two modules. The task of the first module is to generate a conditioning sequence $(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \ldots, \boldsymbol{c}^{(l')})^T = \mathcal{C}_e(\boldsymbol{y}, \boldsymbol{s}^w)$. The second module computes the timestep embedding $\boldsymbol{t}^e = \mathcal{C}_t(t, \boldsymbol{s}^w)$. Depending on the selected method for including the style conditioning, the style conditioning may be contained in $(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \ldots, \boldsymbol{c}^{(l')})$ or in $\boldsymbol{t}^e$ (see Section 4.2.2). Since some of these options append a vector to the conditioning sequence, this is indicated by denoting the length of the sequence by $l' \in \{l, l+1\}$. For better readability, the l' embedding vectors $\boldsymbol{c}^{(i)} \in \mathbb{R}^{\mathsf{d}_c}$ of the sequence are denoted as the matrix $\boldsymbol{C} = (\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \ldots, \boldsymbol{c}^{(l')})^T$ with $\boldsymbol{C} \in \mathbb{R}^{(l' \times \mathsf{d}_c)}$.

We change the commonly used notation $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{x}, t, y)$ for predicting the noise from the noisy latent ${}^t\boldsymbol{x}$, one conditioning y and the timestep t to $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})$. Thereby, we emphasize that the architecture of our *diffusion model* (DM) works on feature maps with spatial extent. Furthermore, the discrete timestep needs to be converted into a continuos vector representation such that it can be included in the calculations of the model. Especially, since we are combining the timestep embedding with the style embedding in one of our strategies, it is important to make the vectorial representation of the timestep explicit.

Algorithm 3 : Training of our LDM applying classifier-free guidance [HS21]. The weights θ do not only include the weights from the LDM but also from the content encoder modules. The weights of the VAE stay unchanged.

Input : Labeled dataset $\mathbf{D} = \{(\boldsymbol{I}^{(0)}, \boldsymbol{y}^{(0)}, w_0), \ldots, (\boldsymbol{I}^{(N)}, \boldsymbol{y}^{(N)}, w_N)\}$; Sets of writer embeddings $\mathbf{S}_w$ for all $w \in \mathbf{W_D}$;

Models : Conditional LDM $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})$; VAE encoder $\mathcal{V}_e$; Content encoder modules $\mathcal{C}_e$ and $\mathcal{C}_t$;

Parameters : Probability of unconditional training p_{uc}; Number of diffusion steps T; Variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$;

repeat
- $(\boldsymbol{I}, \boldsymbol{y}, w) \in_R \mathbf{D}$;
- $\boldsymbol{Z} \leftarrow \mathcal{V}_e(\boldsymbol{I})$;
- $\boldsymbol{s}^w \in_R \mathbf{S}_w$;
- $\boldsymbol{y} \leftarrow \boldsymbol{y}_\emptyset$ with probability p_{uc}; ▷ `Independently mask text`
- $\boldsymbol{s}^w \leftarrow \boldsymbol{s}^\emptyset$ with probability p_{uc}; ▷ `Independently mask style`
- $t \sim \mathcal{U}(1, T)$;
- $\boldsymbol{t}^e \leftarrow \mathcal{C}_t(t, \boldsymbol{s}^w)$; ▷ `Encode timestep and style`
- $\boldsymbol{C} \leftarrow \mathcal{C}_e(\boldsymbol{y}, \boldsymbol{s}^w)$; ▷ `Encode text and style`
- $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$;
- $\bar{\alpha}_t \leftarrow \prod_{i=1}^{t}(1 - \beta_i)$;
- ${}^t\boldsymbol{Z} \leftarrow \sqrt{\bar{\alpha}_t}\,{}^0\boldsymbol{Z} + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$;
- Update weights θ according to $\nabla_\theta \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})\|^2$;

until *converged*;

Based on the latent ${}^t\boldsymbol{Z}$, the timestep embedding $\boldsymbol{t}^e$ and the conditioning $\boldsymbol{C}$, the diffusion model predicts the noise $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})$ in the latent space. The loss is computed using the squared distance between the applied noise $\boldsymbol{\epsilon}$ and the predicted noise $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})$ as described by Equation 67. The weights of the LDM $\boldsymbol{\epsilon}_\theta(\cdot)$ and the content encoder modules $\mathcal{C}_e(\cdot)$ and $\mathcal{C}_t(\cdot)$ are updated according to the gradients of the loss. During training, the weights of the VAE encoder $\mathcal{V}_e$ stay unchanged. The VAE decoder $\mathcal{V}_d$ is not needed during training, which reduces computing costs.

In order to be able to later guide the sampling based on the conditioning, we use CFG [HS21] which requires to jointly train a conditional and an unconditional model. The joint training works by replacing the conditioning by a `[null]`-token $\emptyset$ with the probability of p_{uc} for every sample in each iteration. As we implement style and text as two separate conditions, we introduce a `[null]`-token $\boldsymbol{s}^\emptyset$ for the style embedding and a sequence of j repetitions of a `[null]`-character-token $\boldsymbol{y}^\emptyset = (y_{\emptyset,1}, y_{\emptyset,2}, \ldots, y_{\emptyset,j})$ for the text conditioning. The replacement of text and style conditioning are applied independently both with probability p_{uc}.

Sampling

After training, our LDM-based system can be used to synthesize handwritten word images. The sampling procedure is illustrated in Figure 19 and formally described by Algorithm 4. For generating a handwritten word image, the conditioning is provided by a string $\boldsymbol{y}$ and a set $\mathbf{I}_w^\mathsf{K}$ of writer examples. While the style embedding vector $\boldsymbol{s}^w$

Algorithm 4 : Guided conditional sampling of our LDM.

Input : String $\boldsymbol{y}$; Writer samples $\mathbf{I}_w^{\mathsf{K}}$;
Output : Generated image $\hat{\boldsymbol{I}}$;
Models : Conditional LDM $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})$; VAE decoder $\mathcal{V}_d$; Content encoder modules $\mathcal{C}_e$ and $\mathcal{C}_t$;
Parameters : Guidance scale w_{gs}; Number of diffusion steps T; Variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{T}$;

${}^T\hat{\boldsymbol{Z}} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$;
$\boldsymbol{s}^w \leftarrow \mathcal{S}(\mathbf{I}_w^{\mathsf{K}})$;
$\boldsymbol{C} \leftarrow \mathcal{C}_e(\boldsymbol{y}, \boldsymbol{s}^w)$; ▷ Conditional content embedding
$\boldsymbol{C}^\emptyset \leftarrow \mathcal{C}_e(\boldsymbol{y}^\emptyset, \boldsymbol{s}^\emptyset)$; ▷ Unconditional content embedding
for $t \leftarrow T$ **to** 1 **do**
  $\boldsymbol{t}^e \leftarrow \mathcal{C}_t(t, \boldsymbol{s}^w)$; ▷ Conditional timestep embedding
  $\boldsymbol{t}^\emptyset \leftarrow \mathcal{C}_t(t, \boldsymbol{s}^\emptyset)$; ▷ Unconditional timestep embedding
  ▷ Linear combination of unconditional and conditional noise
  $\tilde{\boldsymbol{\epsilon}} \leftarrow (1 + w_{gs})\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{Z}}, \boldsymbol{t}^e, \boldsymbol{C}) - w_{gs}\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{Z}}, \boldsymbol{t}^\emptyset, \boldsymbol{C}^\emptyset)$;
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ if $t > 1$, else $\boldsymbol{\epsilon} = \mathbf{0}$;
  $\bar{\alpha}_t \leftarrow \prod_{i=1}^{t}(1 - \beta_i)$;
  ${}^{t-1}\hat{\boldsymbol{Z}} \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left({}^t\hat{\boldsymbol{Z}} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\tilde{\boldsymbol{\epsilon}}\right) + \sqrt{\beta_t}\boldsymbol{\epsilon}$; ▷ Denoising step
$\hat{\boldsymbol{I}} \leftarrow \mathcal{V}_d({}^0\hat{\boldsymbol{Z}})$;
return $\hat{\boldsymbol{I}}$;

could be randomly drawn from a set of cached writer embeddings $\mathbf{S}_w$, we believe that it is more common in the application scenario to only have a small set $\mathbf{I}_w^{\mathsf{K}}$ for writer w during sampling. Thus, we denote the computation of the style embedding by $\boldsymbol{s}^w = \mathcal{S}(\mathbf{I}_w^{\mathsf{K}})$.

Since we employ an LDM as our generative model, the image generation is done by iterating the reverse process. Before the iterations, the conditioning $\boldsymbol{C} = \mathcal{C}_e(\boldsymbol{y}, \boldsymbol{s}^w)$ is computed. The reverse process starts at timestep $t = T$ with the latent ${}^T\hat{\boldsymbol{Z}} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ being noise from a multivariate standard normal distribution. In each iteration, the timestep embedding $\boldsymbol{t}^e = \mathcal{C}_t(t, \boldsymbol{s}^w)$ is computed and the noise $\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{Z}}, \boldsymbol{C}, \boldsymbol{t}^e)$ is predicted by the model. In addition to this conditional prediction, an unconditional prediction of the noise is required to enable guided sampling [HS21]. Therefore, we compute the unconditional content embedding $\boldsymbol{C}^\emptyset \leftarrow \mathcal{C}_e(\boldsymbol{y}^\emptyset, \boldsymbol{s}^\emptyset)$ and the unconditional timestep embedding $\boldsymbol{t}^\emptyset \leftarrow \mathcal{C}_t(t, \boldsymbol{s}^\emptyset)$. Based on the unconditional noise prediction $\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{Z}}, \boldsymbol{t}^\emptyset, \boldsymbol{C}^\emptyset)$ and the conditional noise prediction $\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{Z}}, \boldsymbol{t}^e, \boldsymbol{C})$, the noise removed from the latent ${}^t\boldsymbol{Z}$ is computed according to Equation 70. These steps are repeated for T timesteps in order to obtain ${}^0\hat{\boldsymbol{Z}}$. While we describe the noise removal as originally proposed by Ho et al. [HJA20], this step could be replaced by more recent sampling algorithms (e. g. [Lu+22a; Lu+22b; ZC23; Zha+23]) to reduce the number of required denoising steps. Finally, the VAE decoder generates the image $\hat{\boldsymbol{I}} = \mathcal{V}_d({}^0\hat{\boldsymbol{Z}})$.

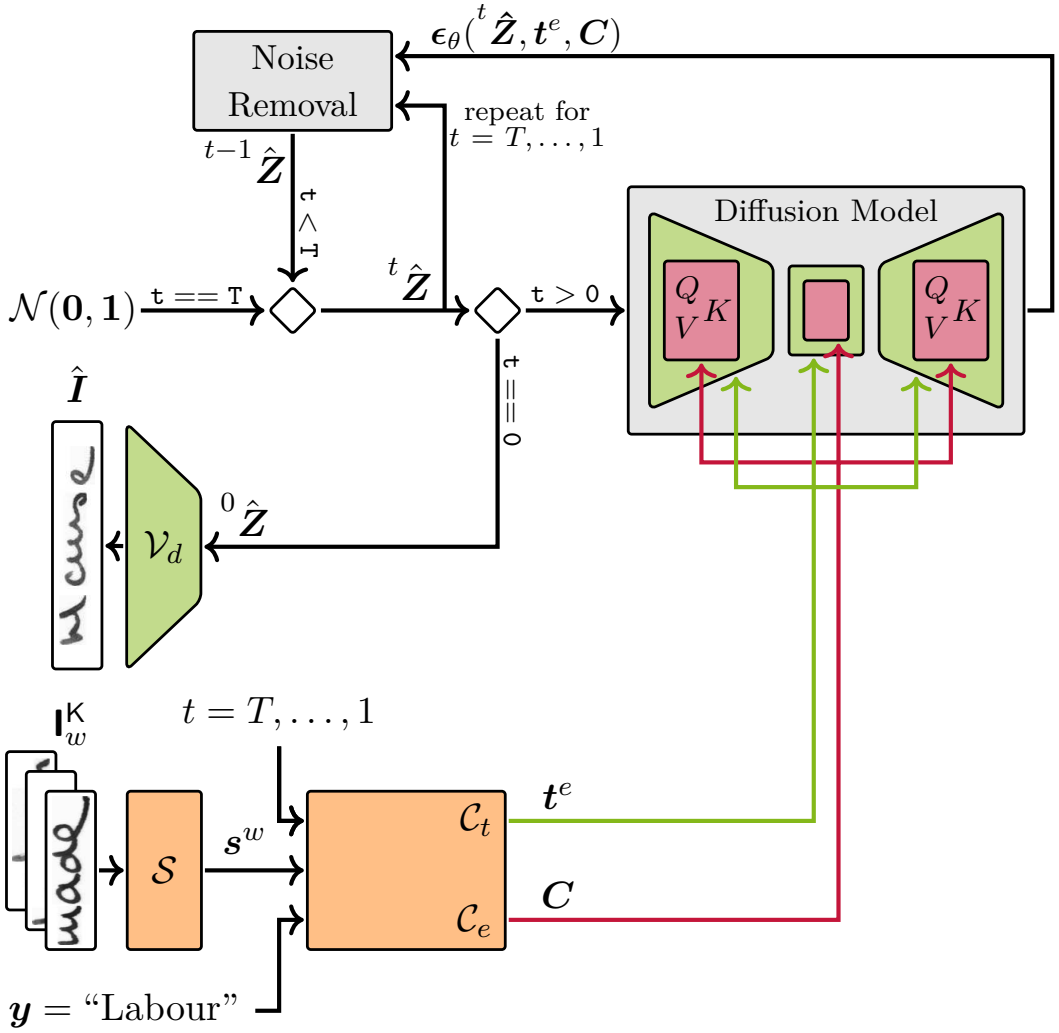


Figure 19: Overview of the proposed system when sampling an image given the target string $\boldsymbol{y}$ and a set $\mathbf{I}_w^{\mathsf{K}}$ of examples from the desired writer. Starting from random noise ${}^T\hat{\boldsymbol{Z}} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$, the model predicts noise $\boldsymbol{\epsilon}_\theta({}^t\hat{\boldsymbol{Z}}, \boldsymbol{t}^e, \boldsymbol{C})$ which is then removed from ${}^t\hat{\boldsymbol{Z}}$ to obtain ${}^{t-1}\hat{\boldsymbol{Z}}$. This procedure is repeated for T timesteps to compute ${}^0\hat{\boldsymbol{Z}}$ which is passed to the decoder part $\mathcal{V}_d$ of the VAE to obtain the final image $\hat{\boldsymbol{I}}$.

4.2.1 *Latent Diffusion Model*

Our generative model is based on the LDM proposed by Rombach et al. [Rom+22]. The LDM is composed of a large-scale pretrained VAE and the actual DM implemented by a UNet. By using a VAE, the diffusion process is shifted from the pixel space to the latent space (see Section 3.1.5). The latent representations computed by the VAE only have $1/8$ the size of the corresponding spatial dimensions in the image space. Since the conditioning is only introduced for the UNet and the VAE is left unconditional [Rom+22], we can focus on only training the UNet. We employ a pretrained VAE[1] and keep its weights frozen during training. In the following, we will only discuss the architecture of the UNet in detail. For details about the VAE, we refer to the work from Rombach et al. [Rom+22].

The UNet architecture was initially proposed for biomedical image segmentation [RFB15]. The key idea of this architecture is to extend the contracting network of fully convolutional models [LSD15] by a symmetric expanding path. While the contracting path mainly captures contextual information, high resolution features at different stages of the contracting path are concatenated with features from the expanding path to enable precise localization of features. This concept of up- and down-sampling with connections between feature maps with the same spatial size can be illustrated in a *U-shape* (cf. Figure 20). This architecture concept was used by Ho et al. [HJA20] for the implementation of their DM. However, they base their UNet on wide residual blocks [ZK16] with *group normalization* (GroupNorm) [WH18]. In order to specify the timestep of the diffusion process, the timestep is encoded by a sinusoidal position embedding [Vas+17] and is added in each residual block [HJA20]. Context is aggregated at the 16×16 resolution feature map by self-attention [Vas+17]. The architecture was further improved [Son+21; DN21] by making the model deeper, using *multi-head self-attention* (MHSA) at additional resolutions and using BigGAN [BDS19] blocks. For conditioning the model on a class label in addition to the diffusion timestep, *adaptive GroupNorm* (AdaGN) [ND21; DN21] was proposed. Given intermediate activations $\boldsymbol{h}$ and $\boldsymbol{y} = [\boldsymbol{y}^{(s)}, \boldsymbol{y}^{(b)}]$ obtained from a linear transformation of the combination of timestep and the class embedding, the layer is defined as:

$$\text{AdaGN}(\boldsymbol{h}, \boldsymbol{y}) = \boldsymbol{y}^{(s)} \, \text{GroupNorm}(\boldsymbol{h}) + \boldsymbol{y}^{(b)}. \tag{72}$$

The combination of timestep and class embedding is simply implemented by addition [ND21]. Besides the incorporation of the conditioning to the DM, gradients of a classifier are used during sampling [DN21]. In order to allow for more flexible conditioning beyond class labels or blurred variants of the input image, cross-attention [Vas+17] (see Section 2.4.3) can be used [Rom+22]. Intermediate activations of the DM can thereby attend to embeddings of the conditioning computed by a domain specific encoder. The cross-attention is introduced to the architecture as part of a transformer. Such a transformer is composed of blocks of self-attention, a position-wise *multi layer perceptron* (MLP) followed by cross-attention [Rom+22]. These transformers are then used to replace the self-attention blocks of the U-Net [DN21]. Furthermore, CFG [HS21] is used for conditional generation.

1 We load the pretrained weights from the Huggingface repository https://huggingface.co/runwayml/stable-diffusion-v1-5, accessed: 25.08.2023.

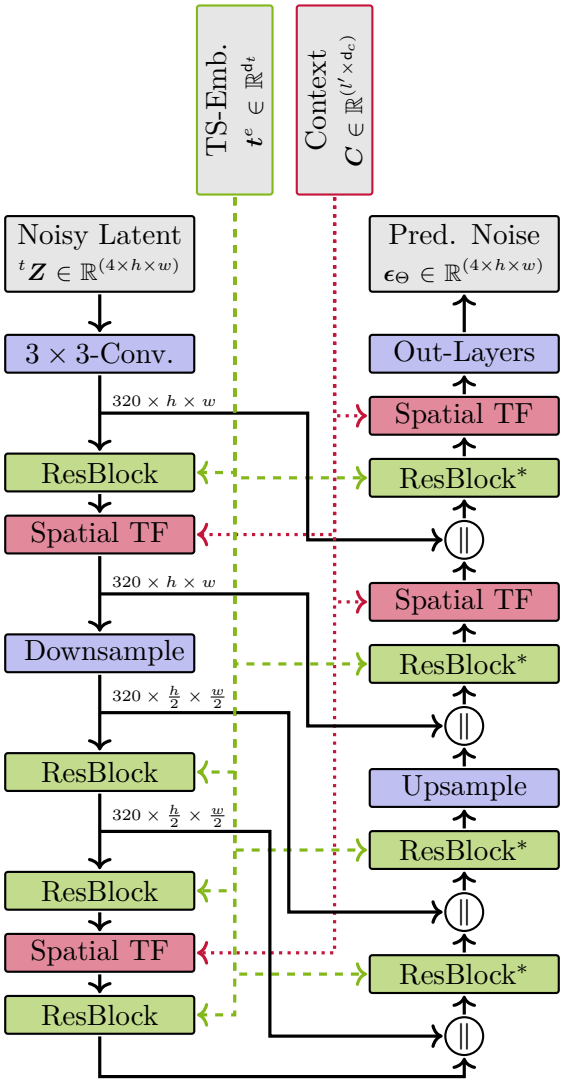


Figure 20: Architecture of the UNet used for noise prediction in our LDM and Wordstylist [Nik+23]. The timestep embedding $\boldsymbol{t}^e$ and the content embedding $\boldsymbol{C}$ used as the context are provided by our content encoder. The concatenation along the feature dimension is denoted by the operator $\|$. The residual blocks (ResBlocks) marked with * are used to project the features to have 320 dimensions.

The conditional LDM by Rombach et al. [Rom+22] was proposed for the generation of natural images. These images usually have larger spatial dimensions (at least 256×256 in [Rom+22]) compared to handwritten text images which are usually processed with a height of 32 or 64 pixels. Furthermore, for the domain of natural images, huge datasets (e. g. 400 million images [Sch+21]) are available for training. In contrast, datasets containing images of handwritten words (e. g. [MB02]), usually contain less than 100k samples. We, therefore, change the model following the approach in [Nik+23]. We reduce the number of residual blocks, the number of attention heads and the internal feature dimensionality. Furthermore, the latent representation of the VAE is downsampled only once within the UNet in order to ensure a minimum spatial resolution of 4 pixels in the smallest feature map for input images with a height of 64 pixels.

The overall architecture of our UNet used to learn the conditional noise prediction $\boldsymbol{\epsilon}_\theta({}^t\boldsymbol{Z}, \boldsymbol{t}^e, \boldsymbol{C})$ is depicted in Figure 20. $\boldsymbol{C}$ and $\boldsymbol{t}^e$ are computed by our content encoder, which we discuss in more detail in Section 4.2.2. It is important to note, that both models, the UNet and the content encoder, are always trained jointly. As mentioned above, the UNet is mainly composed of *residual blocks* (ResBlocks) incorporating the timestep embedding and *spatial transformers* (spatial TFs) with cross-attention on the context. The input to the model is the latent representation ${}^t\boldsymbol{Z} = \mathcal{V}_e({}^t\boldsymbol{I})$ where w and h are $1/8$ of width and height of $\boldsymbol{I}$. The 4 feature maps from the VAE encoder are projected to the 320 feature dimensions used throughout our model by a 3×3 convolutional layer. After a ResBlock and a spatial TF, the spatial dimensions of the feature maps are downsampled once by a 3×3 convolutional layer with stride 2. Together with another ResBlock, all these layers can be considered as the contracting part of the UNet where intermediate representations are passed to the expanding part. In between both parts, our model has a stack of a ResBlock, followed by a spatial TF and another ResBlock. The output of these blocks is then concatenated along the feature dimension with the corresponding features from the contracting part. The resulting 640 feature dimensions are projected back to 320 dimensional features by the following ResBlock. The concatenation of feature maps followed by a ResBlock is repeated before upsampling the spatial dimensions. The upsampling is computed by nearest neighbor interpolation followed by a 3×3 convolutional layer. Then, concatenation of higher resolution features followed by a ResBlock and a spatial TF is repeated twice. GroupNorm [WH18], a *sigmoid linear unit* (SiLU) [HG16] activation function and a convolutional layer compute the final output (summarized as "Out-Layers" in Figure 20) of the DM with the same size as the input.

The internal structure of both main building blocks, the ResBlock and the spatial TF, will be explained in the following. The structure of the ResBlock used for including the timestep embedding is depicted in Figure 21. As proposed in previous works [ND21; DN21], a class conditioning of the model can also be implemented by adding the respective embedding to the timestep embedding. Following Wordstylist, we explore this approach as one option for conditioning the DM on the style embedding. However, we do not use adaptive GroupNorm. Instead, we add the linear projection of the embedding vector to every position in the feature map of the UNet [Nik+23]. A 3×3 convolutional layer maps the intermediate activations from the UNet to the required dimensions. Similarly, a 1×1 convolutional layer is employed for the residual connection.

The other important component of our UNet ist the spatial TF which is shown in Figure 22. As proposed by Rombach et al. [Rom+22], this block can be utilized for conditioning the model on various modalities. Since the *multi-head cross-attention* (MHCA) within the spatial TF was developed to work on sequences (e. g. prompts), it is the natural choice for the integration of the textual conditioning. In contrast to Wordstylist, our content encoder can encode style information in this sequence as well (see Section 4.2.2). In the spatial TF, GroupNorm is first applied to the input feature map followed by a linear projection through a 1×1 convolution. In order to apply attention, the spatial dimensions are then flattened into a single sequence dimension. This sequence is then processed by a stack of MHSA, MHCA and a *feed forward neural network* (FFN) each preceded by *layer normalization* (LayerNorm) [BKH16]. By the MHSA, information can be distributed along the different

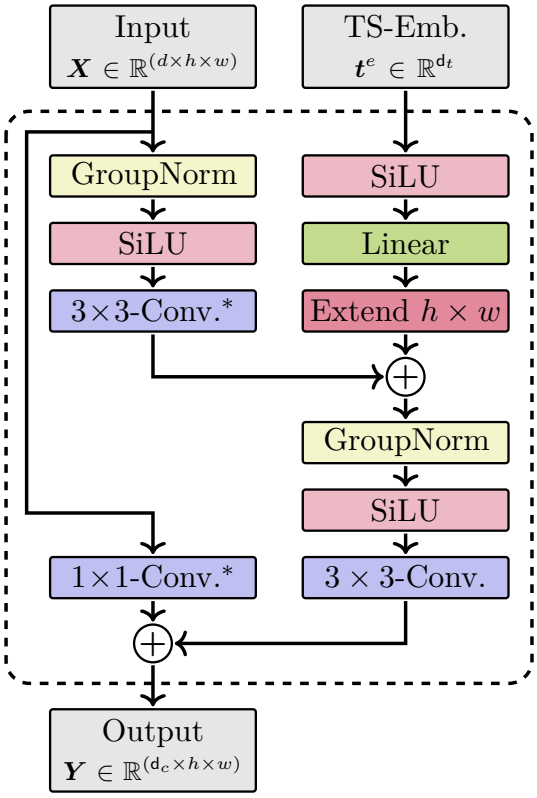


Figure 21: ResBlock used in our UNet to incorporate information from the timestep and possibly the writing style depending on the chosen style inclusion. "Extend $h \times w$" repeats a given vector to construct a feature map of size $\mathsf{d}_c \times h \times w$. The convolutional layers marked with * are used to project the features with dimension d to d_c-dimensionsional features.

positions in the feature map. The resulting features are then used to compute the queries in the MHCA. Keys and values are computed from the context $\boldsymbol{C}$. Thereby, spatial positions can attend to different information provided by the context. The FFN is then applied translationally equivariant to each vector in the sequence. However, instead of two linear layers with a *rectified linear unit* (ReLU) activation function inbetween, in later works (e. g. [Nik+23; Nik+24a; Rom+22]), the first linear layer and the ReLU was replaced by a *gated linear unit* (GLU) [Dau+16]. The GLU is defined as the component-wise product of a linear projection of the input with so-called gates. The gates were initially implemented by a linear projection followed by sigmoid activations resulting in the following definition of the GLU for input $\boldsymbol{x}$:

$$\mathrm{GLU}(\boldsymbol{x}) = \sigma(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \otimes (\boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)}), \tag{73}$$

where $\sigma(\cdot)$ is the sigmoid function and $\otimes$ denotes the component-wise product of two vectors [Dau+16]. The linear transformations are parametrized by $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$. Later, the use of other activation functions instead of the sigmoid was proposed [Sha20]. In particular, the *Gaussian error linear unit* (GELU) [HG16] is used as an activation function in the GLU:

$$\mathrm{GEGLU}(\boldsymbol{x}) = \mathrm{GELU}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \otimes (\boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)}). \tag{74}$$

While we only employ a single stack of MHSA, MHCA and a FFN to reduce the number of trainable parameters, this block can be repeated to build larger models.

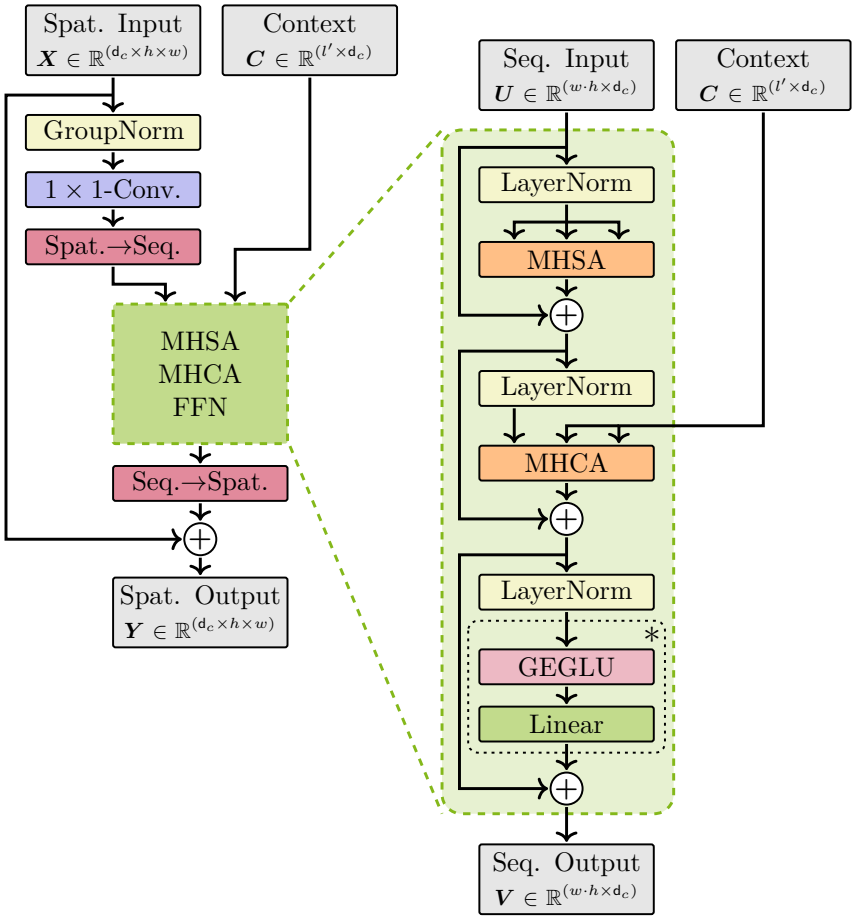


Figure 22: Spatial Transformer used in our UNet to incorporate information from the sequence of context vectors into the intermediate activations of the UNet. "Spat.→Seq." and "Seq.→Spat." transform the spatial dimensions into one sequence dimension and vice versa. While not required by multi-head cross-attention (MHCA), the dimensions of context and input are similar in our case. The feed forward neural network marked with * is applied identically to every vector in the sequence.

4.2.2 *Content Encoder*

One of the major changes compared to Wordstylist [Nik+23] is our proposed content encoder. While DMs for the generation of natural images are usually conditioned on a single modality (e. g. a class-label [ND21] or a prompt [Nic+22]), the HTG task requires simultaneous conditioning on two modalities, textual content and calligraphic style. In Wordstylist, the calligraphic style is represented by a class (i. e. writer ID) which is mapped to an embedding and then added to the embedding of the timestep [ND21; DN21]. For the textual conditioning, the cross-attention-based approach [Rom+22] is used. By this way of including the style conditioning, there is only interaction between the character embeddings and features that possibly contain style information in the MHCA of the spatial TFs of the UNet. Apart from the major limitation described in Section 4.3, the question arises whether two such intertwined modalities should be introduced into the model independently of each other. Both conditions, text and style, are closely related, i. e. every character has a

specific appearance for a writer. Therefore, we assume that incorporating the style information with the text conditioning before passing it to the UNet, might help improve generation performance. Furthermore, the timestep embedding is always added to all positions in the feature maps within the ResBlocks (see Figure 21). While the global distribution of information about the timestep of the denoising process is intuitive, style information might contain information only relevant for some positions in the image. MHCA allows a model to jointly attend to different features at different positions in a given vector sequence [Vas+17]. Therefore, providing the style information to the context for the MHCA in the spatial TF allows for new ways of propagating of the information along the intermediate activations of the UNet.

Figure 23 shows the overall architecture with the different possible paths of incorporating the style information. The content encoder receives three inputs: (i) A string of k characters $\boldsymbol{y} = (y_1, y_2, \ldots, y_k)^T$, $k \leq l$ with a maximum length of l characters which should be generated, (ii) an embedding vector $\boldsymbol{s}$ encoding information about the desired writing style, and (iii) the timestep $t = 1 \ldots, T$. Since we want to obtain a context and the timestep embedding, our content encoder comprises two modules $\mathcal{C}_e$ and $\mathcal{C}_t$. The computation of the context based on text conditioning is extended to also consider the style embedding and is denoted by $\mathcal{C}_e(\boldsymbol{y}, \boldsymbol{s})$. The output of this module is a sequence $(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \ldots, \boldsymbol{c}^{(l')})^T$ summarized in the matrix $\boldsymbol{C} \in \mathbb{R}^{(l' \times \mathtt{d}_c)}$ where l' can be l or $l+1$ depending on the chosen approach for including the style embedding. The timestep embedding $\boldsymbol{t}^e \in \mathbb{R}^{\mathtt{d}_t}$ is a single vector computed based on the timestep and the style embedding by the module $\mathcal{C}_t(t, \boldsymbol{s})$. However, we structure the following explanations according to the two conditions text and style instead of the modules.

Processing of the Text Conditioning

The first processing steps for the text condition are equal to those of Wordstylist. The given k characters $(y_1, y_2, \ldots, y_k)$ are first mapped to their corresponding learnable $\mathtt{d}_e$-dimensional character embeddings. Then, the sequence is padded to a fixed length l using a learnable `[pad]`-token to obtain the sequence $(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \ldots, \boldsymbol{h}^{(l)})$. In Wordstylist, the next step is the addition of sinusoidal positional encodings [Vas+17] (see Section 2.4.1) to introduce positional information to each vector. To allow the model to distribute information within the sequence, the final conditioning is computed by self-attention:

$$(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \ldots, \boldsymbol{c}^{(l)}) = \text{Self-Attention}(\boldsymbol{h}^{(1)} + \boldsymbol{p}^{(1)}, \ldots, \boldsymbol{h}^{(l)} + \boldsymbol{p}^{(l)}), \tag{75}$$

where $\boldsymbol{p}^{(i)}$ is the sinusoidal positional encoding for position i. We propose different additional steps before and after the self-attention to provide different options of incorporating the style information.

Incorporation of the Style Conditioning

As discussed earlier, the incorporation of calligraphic information into the context $\boldsymbol{C}$ could potentially benefit the distribution of information within the activations of the UNet. Therefore, we explore different approaches which we term *style inclusions*

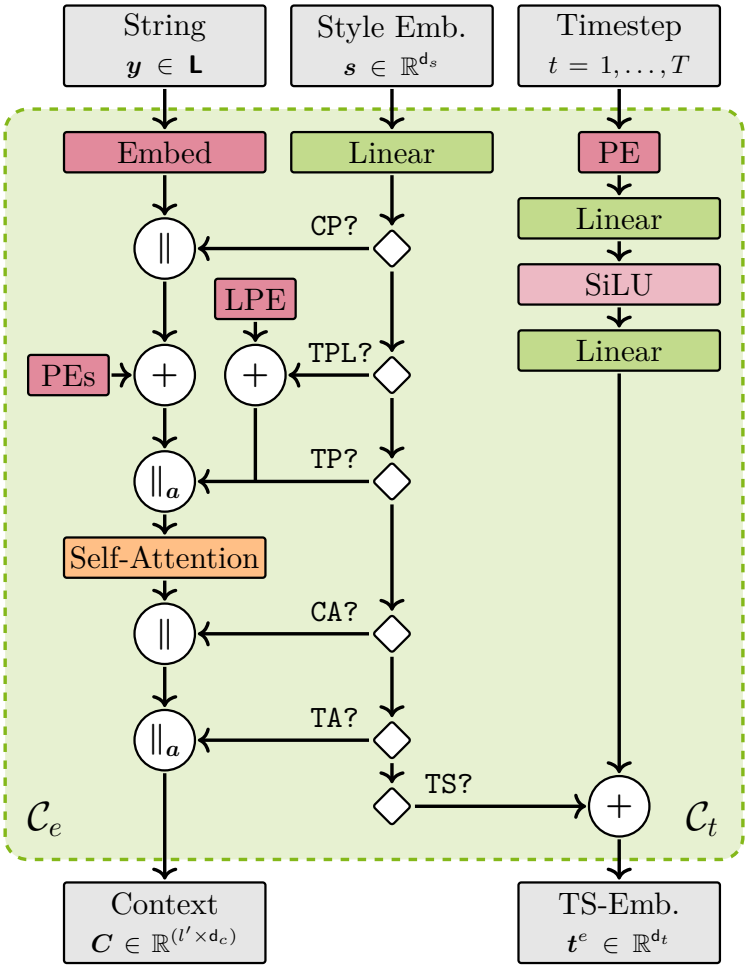


Figure 23: Computation of the content embedding $\boldsymbol{C} = \mathcal{C}_e(\boldsymbol{y}, \boldsymbol{s})$ and the timestep embedding $\boldsymbol{t}^e = \mathcal{C}_t(t, \boldsymbol{s})$ for the different proposed style inclusions. "PE(s)" and "LPE" denote positional encodings and learned positional embeddings, respectively. Operands are $+$ for component-wise addition, $\|$ for concatenation of two vectors along their feautre dimension and $\|_{\boldsymbol{a}}$ for appending a vector to a sequence.

(SIs). As a first step, we use a learnable linear projection $\boldsymbol{W} \in \mathbb{R}^{(\mathsf{d}_s \times \mathsf{d}_x)}$ to allow for adjustments of the provided style embedding including its dimensionality:

$$\boldsymbol{s}' = \boldsymbol{W}\boldsymbol{s}, \qquad \boldsymbol{s} \in \mathbb{R}^{\mathsf{d}_s}. \tag{76}$$

Depending on the chosen SI, d_x is set to the dimensionality of the timestep embedding d_t or the dimensionality of the character embedding d_e.

A straight-forward idea to enable the model to learn to distribute style information jointly with the textual information within the context is to allow the self-attention of the content encoder to also attend to $\boldsymbol{s}'$. This idea can be implemented in different ways. First, $\boldsymbol{s}'$ can be appended to the padded sequence of character embeddings before passing the sequence to the self-attention:

$$(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \ldots, \boldsymbol{c}^{(l+1)})_{\mathtt{TP}} = \text{Self-Attention}(\boldsymbol{h}^{(1)} + \boldsymbol{p}^{(1)}, \ldots, \boldsymbol{h}^{(l)} + \boldsymbol{p}^{(l)}, \boldsymbol{s}'). \tag{77}$$

Optionally, a learnable positional embedding can be added:

$$(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \ldots, \boldsymbol{c}^{(l+1)})_{\mathtt{TPL}} = \text{Self-Attention}(\boldsymbol{h}^{(1)} + \boldsymbol{p}^{(1)}, \ldots, \boldsymbol{h}^{(l)} + \boldsymbol{p}^{(l)}, \boldsymbol{s}' + \boldsymbol{p}^{\theta}). \tag{78}$$

We use a learnable positional embedding $\boldsymbol{p}^{\theta}$ for the style embedding token instead of a sinusoidal encoding, as it is not part of the original sequence of characters and therefore has no inherent position. Instead, we leave the virtual placement up to the model. We refer to these options as `TP` and `TPL`, respectively. Another option for incorporating $\boldsymbol{s}'$ prior to the self-attention is the concatenation of the style embedding vector to all character embedding vectors along their feature dimension:

$$(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \dots, \boldsymbol{c}^{(l)})_{\texttt{CP}} = \text{Self-Attention}([\boldsymbol{h}^{(1)}, \boldsymbol{s}'] + \bar{\boldsymbol{p}}^{(1)}, \dots, [\boldsymbol{h}^{(l)}, \boldsymbol{s}'] + \bar{\boldsymbol{p}}^{(l)}). \quad (79)$$

As this increases the dimensionality of the vectors in the sequence from $\mathtt{d}_e$ to $2 \cdot \mathtt{d}_e$, the dimension of the positional encoding must be adjusted accordingly (indicated by $\bar{\boldsymbol{p}}^{(i)}$). We denote this option for SI as `CP`.

In addition to SIs allowing the processing of style information by the self-attention, we also explore options to include the style embedding into the context without any modifications before passing it to the UNet. These options still allow the MHCA in the spatial TFs of the UNet to attend to the style features and are, therefore, significantly different from the addition within the ResBlocks. The presented approaches are analogue to those previously explained except that no positional encodings have to be taken into account. The style embedding vector can be appended to the sequence:

$$(\bar{\boldsymbol{c}}^{(1)}, \bar{\boldsymbol{c}}^{(2)}, \dots, \bar{\boldsymbol{c}}^{(l)}) = \text{Self-Attention}(\boldsymbol{h}^{(1)} + \boldsymbol{p}^{(1)}, \dots, \boldsymbol{h}^{(l)} + \boldsymbol{p}^{(l)}), \quad (80)$$

$$(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \dots, \boldsymbol{c}^{(l+1)})_{\texttt{TA}} = (\bar{\boldsymbol{c}}^{(1)}, \dots, \bar{\boldsymbol{c}}^{(l)}, \boldsymbol{s}'), \quad (81)$$

which we refer to as `TA`. This option only includes style information in one of the context vectors. Alternatively, the style embedding vector can be concatenated to all context vectors before passing it to the UNet:

$$(\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \dots, \boldsymbol{c}^{(l)})_{\texttt{CA}} = ([\bar{\boldsymbol{c}}^{(1)}, \boldsymbol{s}'], \dots, [\bar{\boldsymbol{c}}^{(l)}, \boldsymbol{s}']). \quad (82)$$

This SI, which we refer to as `CA`, provides style information in every vector of the input sequence for the MHCA but does not influence the self-attention on the character embedding sequence.

We also consider the addition of the style embedding to the timestep embedding like in Wordstylist. In this case, the context embedding is not influenced by the style embedding at all. Instead, this is the only case, where the style embedding is actually considered in the computation of $\mathcal{C}_t(t, \boldsymbol{s})$. The only difference to the implementation in Wordstylist is, that we apply a learnable linear projection of the given style embedding to obtain $\boldsymbol{s}'$. This is necessary for our method as the given vector might have a different dimensionality than the timestep embedding. We define the SI called `TS` as

$$\boldsymbol{t}^e = \boldsymbol{W}^{(2)}(\text{SiLU}(\boldsymbol{W}^{(1)}\boldsymbol{p}^{(t)} + \boldsymbol{b}^{(1)})) + \boldsymbol{b}^{(2)} + \boldsymbol{s}', \quad (83)$$

where $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$ are the parameters of two linear layers. $\boldsymbol{p}^{(t)}$ is the sinusoidal positional encoding corresponding to timestep t.

For all inclusion methods, the computed content embedding vectors $\boldsymbol{c}^{(i)}$ have the same dimensionality $\mathtt{d}_c = \mathtt{d}_e$ as the character embeddings. The only exceptions are the concatenation-based inclusions `CP` and `CA` where $\mathtt{d}_c = 2 \cdot \mathtt{d}_e$. If the style embedding is appended to the sequence (i. e. `TP`, `TA`), the returned sequence has a length of $l + 1$ instead of l.

4.3 STYLE ENCODER

As previously described, the conditioning on the writing style in Wordstylist is implemented by mapping the writer ID to a learnable embedding. For writers seen during training, realistic images even for OOV generation were generated [Nik+23]. Based on the embeddings from two writers from the training set, interpolation in the latent space of the learned writer embeddings has been shown to work for the generation of new writing styles. However, this mapping inhibits the generation of word images with specific styles from other datasets not seen during training which is a necessary capability for the specific problem addressed in this thesis. A common approach to overcome this substantial limitation is the computation of writer embeddings based on example images $\mathbf{I}_w^{\mathsf{K}}$ from a writer w. Some approaches learn this conditioning jointly with the generative model (e. g. [Kan+20b; ZN23]) others pretrain a dedicated model for computing style embeddings (e. g. [Gan+22; Nik+24a]) which are then used for conditioning a generative model. We adopt the latter approach and train a separate model for style extraction as this allows for the selection of training data different from that of the generative model. Many of these models are trained optimizing a classification or a contrastive objective (e. g. [Nik+24a; Gan+22]) and therefore require samples labeled with writer IDs. One possible solution is to use synthetic data (e. g. [Van+25]). However, the risk is that the domain gap is too large and the model does not adequately capture the stylistic features of real handwriting. Therefore, we choose a model for our style encoder (SE), which allows for self-supervised training on real handwriting eliminating the need for annotated samples. We employ a modified version of Text-DIAE [Sou+23] which is based on a MAE [He+22]. In the following, the architecture of the model will be explained. Then, its training as an MAE will be described. Finally, it will be discussed how the obtained model can be used to compute writer embeddings based on a given set of samples from a writer or based on a single image.

4.3.1 *Architecture*

The MAE [He+22] is composed of an encoder $\mathcal{M}_e$ and a decoder $\mathcal{M}_d$. An overview of the models and the processing of images is provided in Figure 24. The encoder $\mathcal{M}_e$ is the backbone of a *vision transformer* (ViT) [Dos+21] which has been a very influential model in the computer vision community, first introduced for the task of image classification. In order to be processed by a transformer [Vas+17], a given image $\boldsymbol{I} \in \mathbb{R}^{(\mathsf{h}\times\mathsf{w}\times\mathsf{c})}$ first has to be reshaped into a sequence of vectors. Therefore, the image is first divided into $\mathsf{N}_p = \mathsf{hw}/\mathsf{p}^2$ non-overlapping patches of size $\mathsf{p}\times\mathsf{p}$ and flattened into vectors $\boldsymbol{p}^{(i)} \in \mathbb{R}^{(\mathsf{p}^2\times\mathsf{c})}$. The obtained sequence forms the matrix $\boldsymbol{P} = (\boldsymbol{p}^{(1)}, \boldsymbol{p}^{(2)}, \ldots, \boldsymbol{p}^{(\mathsf{N}_p)})^T$ with its rows containing the flattened patches. To keep the notation short, we summarize the patch extraction and flattening using the function $\mathcal{P}$ with $\mathcal{P}(\boldsymbol{I}) = \boldsymbol{P}$ and the inverse operation $\mathcal{P}^{-1}(\boldsymbol{P}) = \boldsymbol{I}$. Before passing $\boldsymbol{P}$ to the transformer blocks, a linear transform $\boldsymbol{W}$ is applied and a learnable 1D positional embedding $\boldsymbol{W_{PE}}$ is added:

$$\boldsymbol{X}^{(0)} = \boldsymbol{P}\cdot\boldsymbol{W} + \boldsymbol{W_{PE}}. \tag{84}$$

As the ViT was originally proposed for image classification, before adding the positional embedding $\boldsymbol{W_{PE}}$, a learnable `[class]`-token was prepended to the sequence

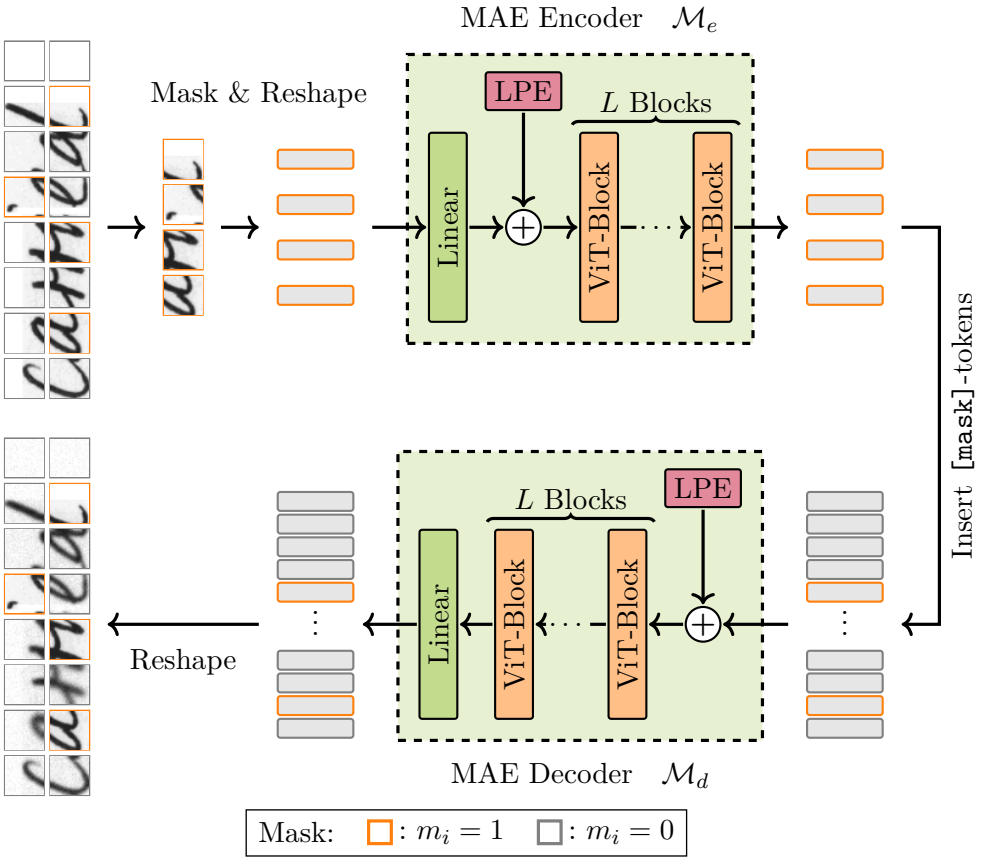


Figure 24: Architecture of the MAE with a ViT-based encoder $\mathcal{M}_e$ and the corresponding decoder $\mathcal{M}_d$. The computations during masked training are illustrated based on a handwriting image. "LPE" denotes the learned positional embeddings.

$\boldsymbol{P} \cdot \boldsymbol{W}$. Its corresponding representation computed by the transformer blocks was then for the classification by an MLP [Dos+21]. We omit the use of this token since we do not apply the model to a classification problem but need representations for all patches. The embedding sequence $\boldsymbol{X}^{(l)}$ is processed by L transformer blocks indexed by (l) comprising MLP and MHSA sub-layers surrounded by residual connections [He+16] to obtain the final representation $\boldsymbol{X}^{(L)}$:

$$\boldsymbol{X}'^{(l)} = \text{MHSA}(\text{LayerNorm}(\boldsymbol{X}^{(l-1)})) + \boldsymbol{X}^{(l-1)}, \tag{85}$$

$$\boldsymbol{X}^{(l)} = \text{MLP}(\text{LayerNorm}(\boldsymbol{X}'^{(l)})) + \boldsymbol{X}'^{(l)}. \tag{86}$$

The MLP consists of two layers with a GELU [HG16] after the first layer. Note that the MLP is applied separately to each vector in the sequence, therefore, being translationally equivariant [Dos+21]. Figure 25 depicts the structure of one of these blocks which we will refer to as *ViT-blocks*. For simplicity, we denote the latent representation $\boldsymbol{X}^{(L)} = \boldsymbol{Z}$. In contrast to the transformer block presented by Vaswani et al. [Vas+17], the LayerNorm [BKH16] is applied before the MLP and MHSA sub-layers in the ViT [Dos+21]. This change was inspired by previous studies [BA19; Wan+19] on the placement of the LayerNorm in the transformer block.

A similar stack of L ViT-blocks followed by a linear projection forms the decoder $\mathcal{M}_d$ predicting the flattened reconstructed patches. Also with the decoder, learnable positional embeddings $\boldsymbol{W}_{\boldsymbol{PD}}$ are added to the latent representation $\boldsymbol{Z}$ before they

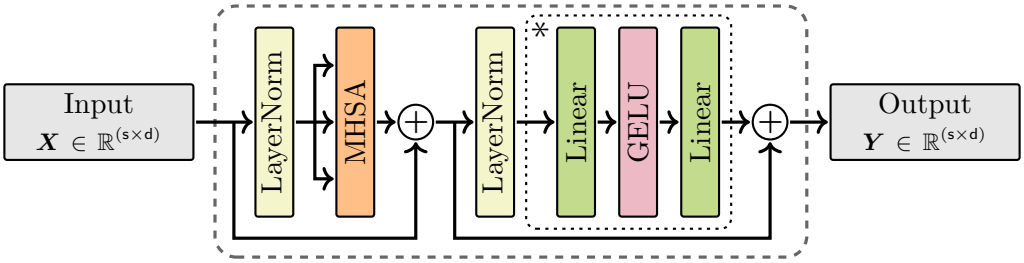


Figure 25: Structure of a ViT-block [Dos+21]. The MLP marked with * is applied to each vector in the sequence separately.

are passed to the ViT-blocks [He+22]. The joint encoding and reconstruction of an image $\boldsymbol{I}$ can then be described by:

$$\boldsymbol{Z} = \mathcal{M}_e(\mathcal{P}(\boldsymbol{I})), \tag{87}$$

$$\hat{\boldsymbol{I}} = \mathcal{P}^{-1}(\mathcal{M}_d(\boldsymbol{Z})). \tag{88}$$

4.3.2 *Training*

As described so far, the model only represents a transformer-based *autoencoder* (AE). The choice of the training procedure is essential for the model to learn expressive representations. Based on previous works on denoising AEs [Vin+08] and applications in natural language processing [Dev+19], He et al. [He+22] found that masked autoencoding is a simple but very effective approach for self-supervised training. The task is to reconstruct the original image from a masked version of itself.

In each training iteration, a random binary mask $\boldsymbol{m} = (m_1, m_2, \ldots, m_{\mathsf{N}_p})^T \in \{0,1\}^{\mathsf{N}_p}$ is generated per image $\boldsymbol{I}$ with $P(m_i = 0) = p_{mask}$ and $p_{mask} \in (0,1)$. The patches $\boldsymbol{P}$ are then divided into visible patches $\boldsymbol{P_V}$ and masked patches $\boldsymbol{P_M}$ [He+22]. $\boldsymbol{P_V}$ contains all rows $\boldsymbol{p}^{(i)}$ from $\boldsymbol{P}$ for which $m_i = 1$. The remaining rows form $\boldsymbol{P_M}$. For the visible patches, their latent representations $\boldsymbol{Z_V} = \mathcal{M}_e(\boldsymbol{P_V})$ are computed. The addition of the learnable position embeddings is computed according to the original positions of the visible patches in the image.

Before decoding, a learnable `[mask]`-token is inserted in every row i in $\boldsymbol{Z_V}$ where $m_i = 0$. We denote the resulting matrix as $\boldsymbol{Z'}$ which is then passed to the decoder to compute the reconstructed flattened patches $\hat{\boldsymbol{P}} = \mathcal{M}_d(\boldsymbol{Z'})$. At this point, the addition of the learnable positional embeddings $\boldsymbol{W_{PD}}$ of the decoder is essential as the same `[mask]`-token has been inserted at every masked position without any information about the corresponding position in the image [He+22]. While the decoder computes predictions for all patches, only those corresponding to masked patches are considered in loss computation. The loss is then defined by the *mean squared error* (MSE):

$$\mathcal{L}_{rec}(\boldsymbol{P}, \hat{\boldsymbol{P}}) = \frac{1}{\mathsf{N}_p} \sum_{i=1}^{\mathsf{N}_p} (1 - m_i) \cdot \mathrm{MSE}(\boldsymbol{P}_i - \hat{\boldsymbol{P}}_i). \tag{89}$$

While following the architecture of Text-DIAE [Sou+23], we do not adopt the additional proposed training objectives of that work. This choice is inspired by their ablation study showing the effectiveness of pretraining for HTR using only the masking objective. Their evaluation, however, only demonstrated that textual information

must be encoded by the model. In our application of the model, we need the embeddings to encode stylistic information instead. For a good reconstruction in pixel space, however, encoding only textual information is not sufficient. This consideration suggests that the model also has to learn an encoding of stylistic information in order to accurately reconstruct the strokes of the handwriting from only a few visible patches. Although this implies that only part of the embedding contains stylistic information, in our opinion the advantage of not requiring annotated data outweighs possibly more style focused representations of models trained with supervision.

4.3.3 *Computation of Writer Embeddings*

In our proposed method for the generation of training samples for a target dataset, the previously explained MAE is used as our style encoder. The aforementioned training allows the model to be trained independently of the remaining system even including unlabeled samples from the target dataset. Once a trained MAE is available, it can be used to compute writer embeddings based on a set of K example images $\mathbf{I}_w^{\mathsf{K}}$ from any given writer w. We only use the encoder $\mathcal{M}_e$ of the trained MAE. Based on our application to embed writing styles, we denote the learned latent space of $\mathcal{M}_e$ as the *style space*. First, we compute a set of latent representations for all patches from all given images:

$$\mathbf{Z}_w = \left\{ \mathcal{M}_e(\boldsymbol{I}) | \boldsymbol{I} \in \mathbf{I}_w^{\mathsf{K}} \right\}. \tag{90}$$

In order to summarize the information into one vector, we define a function $\mathcal{S}$ to compute the average of the embeddings of all patches from all images:

$$\mathcal{S}(\mathbf{I}_w^{\mathsf{K}}) = \frac{1}{\mathsf{N}_p \mathsf{K}} \sum_{i=1}^{\mathsf{K}} \sum_{j=1}^{\mathsf{N}_p} \boldsymbol{Z}_j^{(i)}. \tag{91}$$

$\boldsymbol{Z}_j^{(i)} \in \mathbf{Z}_w$ denotes the representation of patch j from the i^{th} example image in $\mathbf{I}_w^{\mathsf{K}}$. While a lot of information is likely to be lost through the averaging, much of this loss is indeed desired. In particular, the encoded information about the textual content is expected to partially average out when the images in $\mathbf{I}_w^{\mathsf{K}}$ show different words. By averaging over the patches of a single image, however, we cannot differentiate information by character. While it would be desirable to preserve this information, without a character-level segmentation, aggregation per character is not straightforward. However, we believe that global calligraphic information should be sufficient for a decent imitation of writing styles.

Although the MAE allows for an unsupervised training, this computation of embeddings requires either knowledge or inference of writer IDs, except for the trivial case with $\mathsf{K} = 1$. If $\mathsf{K} = 1$, however, we do not average out the textual content and other instance specific details by the approach described so far. In order to regularize the influence of the textual content and to allow for variations of style embeddings computed from one image, we introduce the parameter $p_r \in (0, 1]$ defining which share of the patches per image is considered:

$$\mathcal{S}_{pr}(\mathbf{I}_w^1) = \frac{1}{\mathsf{N}_p p_r} \sum_{i=1}^{\mathsf{N}_p} m_i \boldsymbol{Z}_i, \tag{92}$$

where m_i is a binary indicator with $P(m_i = 1) = p_r$, whether a patch embedding in considered in the computation of the style embedding. Though Equation 91 could also be extended to use a subset of the patches for $\mathsf{K} > 1$, we limit this modification to the case $\mathsf{K} = 1$. Furthermore, we only use Equation 92 during training. For sampling, we always compute writer embeddings based on Equation 91.

To avoid a bias towards a specific selection of $\mathbf{I}_w^{\mathsf{K}}$ or a subset of patches, it is a common practise to randomly draw a new set $\mathbf{I}_w^{\mathsf{K}}$ each time $\mathcal{S}$ is computed (e. g. [Nik+24a; Van+25; Gan+22]). In practice, this requires several forward passes through $\mathcal{M}_e$ every iteration thereby slowing down the training. Instead, we prepare a cache $\mathbf{S}_w$ of style embeddings by repeating the computation N_{se} times:

$$\mathbf{S}_w = \left\{ \mathcal{S}(\mathbf{I}_w^{\mathsf{K},i}) \right\}_{i=1}^{\mathsf{N}_{se}}, \tag{93}$$

where i indicates that every $\mathbf{I}_w^{\mathsf{K},i}$ is an independent random subset of examples from writer w. At the same time this augmentation introduces variance and fosters the exploration of the style space. The application based on $\mathcal{S}_{pr}$ works similarly, however only computing a set of embeddings for the respective image instead of a writer.

4.4 TRAINING WITH LESS SUPERVISION

In theory, the approach described so far is able to generate handwritten word images for unseen writers. However, differences between the styles seen during training and the style requested for generation (i. e. from the target dataset) might deteriorate the generation quality. Since, this is exactly the scenario in which our model should be used for training data generation, it is essential to improve the transfer of the generative model to the target dataset. In this scenario, however, we cannot assume to have samples from the target dataset labeled with their transcription. As discussed in Section 3.2.3 for GAN-based HTG systems, unlabeled samples can be integrated in a semi-supervised fashion [Fog+20; ZN21; CPK23]. In these systems, labels are usually only required for additional loss terms (e. g. a recognition loss). While these works do not explicitly explain how unlabeled samples are integrated in the training, it can be assumed that the computation of loss terms requiring labels is simply skipped for unlabeled examples.

DMs, on the other hand, usually rely on CFG which requires to provide the conditioning as an input to the model. Consequently, labels must be provided for all images used in conditional training. In addition to conditional training, the core idea of CFG is to train an unconditional model that shares its weights with the conditional model. The generation is then guided based on predictions from both models [HS21]. In this thesis, we present two approaches for training our LDM in a semi-supervised fashion taking advantage of this simultaneous training. In particular, by the training described in Section 4.2, our system implicitly learns an unconditional model, a style-conditional model, a text-conditional model and a model conditioned on both modalities. With both of our semi-supervised approaches, we aim at improving the unconditional[2] and the writer-conditional models by incorporating samples from the target dataset. By providing a training signal with style conditioning for the target dataset, we want to support the exploration of the style space in the regions

2 only if the style conditioning is dropped due to CFG training

of the target styles which might not be covered by the samples from the source dataset. At the same time, the fully labeled samples (writer ID and transcription) from the source dataset, are used to train the model conditioned on both modalities. Thereby, the model should learn the correspondences between the text and style conditioning regarding the respective handwritten word images (i. e. the appearance of the characters for a given style conditioning).

Our first training scheme relies on samples where only writer IDs are annotated. Even if no reference annotation is available, assumptions can usually be made about the dataset being written by a single or by multiple writers. Furthermore, for many document types like letters or notes it is a reasonable assumption that each page has been written by a single person. Based on these assumptions, pseudo-writer IDs could be assigned based on the page a word image was cropped from. Accordingly, we can compute writer embeddings with variations as in Equation 93 for the target dataset. For training, we augment the fully labeled source dataset with samples $(\boldsymbol{I}, \boldsymbol{y}^{\emptyset}, w)$ from the target dataset where we replace the transcription with a repetition of the learnable `[null]`-character-token. The replacements of the writer embedding with $\boldsymbol{s}^{\emptyset}$ and the transcriptions with $\boldsymbol{y}^{\emptyset}$ is handled in the same way as in the fully supervised training in Algorithm 3. To entirely avoid the use of any labels from the target dataset, we propose a second variant of the semi-supervised training that does not require any writer IDs. This is enabled by the use of the function $\mathcal{S}_{pr}$ instead of function $\mathcal{S}$ allowing the computation of varying style embeddings based on a single image.

4.5 HTR MODEL & SELF-TRAINING

Although not part of the generative process itself, an HTR model is at the heart of our application which is the generation of training data. In this section, first, the chosen HTR model is described in Section 4.5.1. Training and prediction based on *connectionist temporal classification* (CTC) are explained in Section 4.5.2. Afterwards, in Section 4.5.3, it is discussed how an initial model trained on synthetic data (e. g. rendered fonts or generated by DMs) can be improved by self-training on the target dataset.

4.5.1 *Architecture of the HTR Model*

We decide for the HTR model presented by Retsinas et al. [Ret+22] as a collection of previously researched ideas [RSN21; Ret+21a; Ret+21b; TRM21] for word and line image recognition. Besides a simple and comparably cheap training, it enables a direct comparison to results from related works [Nik+23; Nik+24a]. The model is a CNN-LSTM-based architecture where a *convolutional neural network* (CNN) backbone extracts features followed by a *bidirectional long short-term memory* (BiLSTM) for the prediction [Ret+22]. The architecture of the model is depicted in Figure 26.

After a 7×7 convolution with stride 2, the backbone comprises a stack of residual blocks [He+16] (note that these differ from the more advanced ResBlocks discussed in Section 4.2.1) with 3 convolutions, batch normalization [IS15] and ReLU activations. After stacks of 2, 4 and 4 residual blocks each, the spatial dimensions of the feature maps are halved by max-pooling. Starting with 32 dimensional features af-

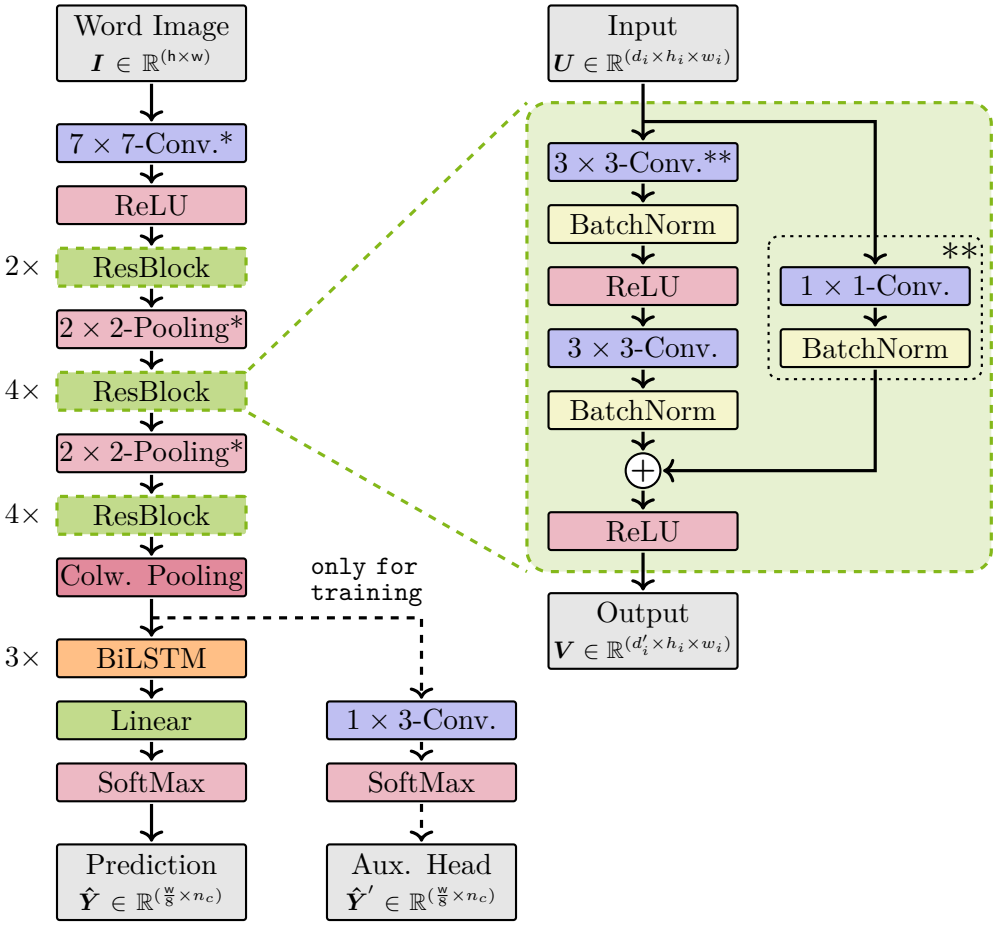


Figure 26: Architecture of the HTR model with the recurrent head and the auxiliary head which is only used during training. The model outputs a sequence of $\mathsf{w}/8$ pseudo-probability distributions over the n_c character classes. The column-wise max-pooling is abbreviated as "Colw. Pooling". Layers marked with * use a stride of 2 to reduce the spatial dimensions of the feature maps. The structure of the ResBlocks is depicted in the right part of the figure. The dimensionality of the feature vectors is adjusted by the layers marked with **. If the dimensionality does not change, an identity mapping is used for the residual connection.

ter the first convolutional layer, the number of feature dimensions is doubled at the beginning of each of the three stacks up to 256 features. Before passing these features to the BiLSTM, a column-wise max-pooling is applied to obtain translational invariance along the vertical direction. This operation for flattening is preferred over concatenation as on the one hand computational costs can be reduced and on the other hand only character (i. e. horizontal) positions are of interest. The final prediction is computed by two heads. The recurrent head consists of three BiLSTM layers which aggregate context information. A linear layer is used to project each feature vector from the predicted sequence of the last BiLSTM layer to activations per character class. The second head produces a similar sequence of vectors with scores per character class but uses a single 1D convolution with kernel size 3. For both heads, a softmax function maps the activations to a sequence of pseudo-probability distributions over the character classes. These sequences are decoded into the predicted

string by CTC as described in the following section. The second convolution-based head is only used as an auxiliary training path and is omitted during inference.

Since the architecture has an inductive bias with regard to the processed images, we follow the preprocessing described by the authors [Ret+22]. We resize all images to have a maximal height ≤ 64 pixels and a width ≤ 256 pixels. By padding the image to the desired size, the aspect ratio can be preserved. Besides the application of an affine augmentation, gaussian noise is added during training. With the transcriptions, we follow the proposed encapsulation of the string by spaces in order to account for the margins of most word images.

4.5.2 *Connectionist Temporal Classification*

Performing HTR by using a model to predict word classes is considered infeasible due to the huge amount of possible word classes. This problem is solved by predicting characters instead which form a finite set. However, this raises a new problem. Models like the one previously described predict character probabilities per position in the feature map. Usually this results in many more predictions than characters in the transcription requiring an alignment of predictions and characters. This problem is tackled by the CTC [Gra+06] offering a decoding scheme and a differentiable objective function for training such a model. For this approach, an additional so called "blank" label is introduced. This label is essential in encoding possible alignments of the prediction and the transcription. Note that this symbol despite its name does not equal the white space label. Given a sequence of pseudo-probability distributions over character classes predicted by a model, each path forms a possible prediction. Decoding aims to find the path with the highest overall probability, which is defined by the product of pseudo-probabilities along the path. A simple approximation is the best path decoding which chooses the character with the highest pseudo-probabilities at each step. The predicted string is computed by first removing all consecutive repeating characters followed by the removal of the blank symbol. Thereby, a new character is inserted in the final prediction when the path contains either switches to a different character or the blank symbol.

Training based on CTC aims at maximizing the log likelihoods of the annotated character sequences. Obtaining the conditional probability of a target sequence given an input sequence requires to compute the sum over all paths resulting in this labelling. The costly problem of finding all these paths can be efficiently solved using an algorithm similar to the forward-backward algorithm for hidden Markov models [Rab89]. As the details of CTC are beyond the focus of this thesis, we refer to [Gra+06] for further insights about the algorithm and the resulting loss function.

4.5.3 *Self-Training*

In our application scenario, we train the previously described HTR model on word images generated with writing styles from a target dataset to improve its recognition performance on the respective dataset. However, as generation results usually do not match the quality of real samples, we explore self-training to further adapt our model to the target dataset. The idea of self-training is to use a pretrained model

Algorithm 5 : Self-Training of an HTR model with confidence filtering [WF24]. The synthetic dataset $\mathbf{S}$ can either be generated by our HTG system or rendered from fonts.

Input : Synthetic dataset $\mathbf{S} = \{(\boldsymbol{I}^{(0)}, \boldsymbol{y}^{(0)}), \ldots, (\boldsymbol{I}^{(\mathsf{N})}, \boldsymbol{y}^{(\mathsf{N})})\}$;
Unlabeled target dataset $\mathbf{X} = \{\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(\mathsf{M})}\}$;
Models : HTR model $\mathcal{H}_\theta$; Confidence measure c_θ;
Parameters : Number of self-training cycles K; Confidence threshold τ;

$\theta_0 \leftarrow$ train $\mathcal{H}_\theta$ on $\mathbf{S}$; ▷ `Train initial model`
for $k \leftarrow 0$ **to** $K - 1$ **do**
  $\mathbf{T'}_k \leftarrow \{(\boldsymbol{X}, \mathcal{H}_{\theta_k}(\boldsymbol{X})) : \boldsymbol{X} \in \mathbf{X}\}$; ▷ `Predict pseudo-labels`
  $\mathbf{T}_k \leftarrow \{(\boldsymbol{X}, \mathcal{H}_{\theta_k}(\boldsymbol{X})) \mid c_{\theta_k}(\boldsymbol{X}) > \tau\}$; ▷ `Thresholding by confidence`
  $\theta_{k+1} \leftarrow$ train $\mathcal{H}_{\theta_k}$ on $\mathbf{T}_k$;

to predict pseudo-labels for samples from some target dataset [WF24]. The model is then fine-tuned using the pseudo-labeled samples.

A model similar to the CNN-LSTM used in this work, was adapted by self-training on unlabeled images from the test set yielding slightly improved recognition performance [RSN21]. Wolf and Fink [WF24] successfully applied self-training to retrieval and recognition models for handwritten word images and achieved impressive results without requiring any annotations for samples from the target dataset. In contrast to the work of Retsinas et al. [RSN21], the initial model was not pretrained on the original training data but on synthetic word images instead. While the synthetic word images were obtained by rendering computer fonts, we believe that this scenario is more similar to our training on generated images. Therefore, we use the self-training scheme proposed in [WF24].

The procedure is described in Algorithm 5. First, an initial model is trained on synthetic samples. Unlike the reference, we do not render images using computer fonts but generate images using our HTG model instead. With our realistic generated samples as training data, our initial model achieves a comparably strong performance on the target dataset. Therefore, it is expected to predict good pseudo-labels for self-training. At the same time, the expected performance gains are accordingly smaller compared to a model trained on rendered samples. Given the initial model, we can fine-tune it using the pseudo-labeled samples from the target dataset. These steps of pseudo-label prediction and fine-tuning are repeated for a given number of cycles.

As the pseudo-labels are expected to contain errors, we are also filtering the samples based on a confidence measure. From the two proposed strategies in [WF24], we decide to filter samples based on a defined confidence threshold as this approach resulted in better performance. Only labels with a confidence above a defined threshold are considered for fine-tuning in the respective cycle. Different from our model, the sequence-to-sequence model employed in the referenced work outputs only as many pseudo-probability distributions as characters are predicted. Therefore, confidence measures are based on the minimum or the mean of the character predictions. These measures cannot be directly transferred to our model. We adapt the idea of using the predicted pseudo-probabilities to CTC-based (i. e. non-aligned) predictions. Based on the best path decoding, we define one confidence measure as the product of probabilities along that path. Additionally, we follow Retsinas et al. [RSN21] to

use CTC loss between the output of the softmax and decoded pseudo-label as a confidence measure. However, we utilize it for a hard thresholding operation. We also explore the logarithm to smooth the CTC loss, making the approach less sensitive to the exact choice of the threshold.

5 EXPERIMENTAL EVALUATION

The generation of training data by *handwritten text generation* (HTG) is a promising approach to improve the performance of *handwritten text recognition* (HTR) models for new datasets where no labels are available. Approaching this goal raises the central question how the generation of new writing styles from a target dataset can be improved. Using our method presented in Chapter 4, we conduct a series of experiments to obtain empirical evidence for answering this question. We divide our studies into two main parts where we first focus on our method only and later explore different HTG methods in realistic settings. The first part aims at optimizing design choices for the desired application and at gaining insights into the following questions.

How can we improve the conditioning on content and style? Conditioning the generation process on a character sequence and a desired writing style is the core of our HTG approach. As both modalities are closely related, i. e. every writer has its own way to write each character, entangling of the respective features has been studied in several works. In Section 5.3.1, we explore our proposed approaches for building a combined context for conditioning the model. Furthermore, we explore *classifier-free guidance* (CFG) for training our model and show that guided sampling improves the generation results.

How can our HTG system be employed to generate training data for a given target dataset? Several works use their HTG system to augment the training data which has also been used for training the generative model itself. In contrast, this work aims at the generation of training data for new datasets which is especially valuable if no transcriptions are available. In Section 5.3.2, we evaluate the performance when using our model to directly generate samples for a target dataset. We show that performance can be considerably improved when samples from the target dataset are included in the training. For this semi-supervised training, we only require writer IDs to be annotated which we assume are easier to obtain than transcriptions.

Can we get rid of the need for any labels in our semi-supervised training? While being easier to obtain, our first attempt on semi-supervised training still relies on annotated writer labels. In Section 5.3.3, we explore our approach for including samples from the target dataset without any annotations. We show that this inclusion is beneficial compared to training the *latent diffusion model* (LDM) only on the source dataset.

Is the *masked autoencoder* (MAE) a suitable choice for our *style encoder* (SE)? The SE is a key component of our HTG system. However, due to its self-supervised training, there are no guarantees that the MAE learns to encode style related features. Therefore, we compare this choice to using a *writer identification* (WID) model as our SE instead. Our experiments in Section 5.3.4 show that both options are suitable choices with the MAE resulting in better performance for the generalization to new datasets.

How essential is the choice of the content to be generated? Training on a representative dataset with sufficient variation is essential for obtaining a robust HTR model. Especially for a new dataset where no annotations are available, the vocabulary is usually unknown. In the experiments in Section 5.3.5, we therefore evaluate the generation based on different distributions of words and writing styles. Our results underline the importance of using a word distribution close to the target dataset. Furthermore, we show that acceptable results can be obtained if only the language and the styles from the target dataset are known.

How does our approach perform for other target datasets? Since our model is designed to generate training data for new target datasets, it must be applicable to other datasets without requiring repeated adjustment of hyperparameters. We therefore employ our HTG system to two other target datasets with different properties. Our experiments in Section 5.3.6 confirm that our HTG system can also be applied to other target datasets.

Can self-training be used to improve the HTR model for the target dataset? A different approach aiming at a similar adaptation of an HTR model to a target dataset is self-training. The original method uses an initial model which is trained on word images rendered from computer fonts. In contrast, our HTR models are trained on generated images and are, therefore, expected to have a better initial performance. This raises the questions whether our models can benefit from self-training and how the obtained models compare to those initially trained on rendered images. Our experiments in Section 5.3.7 reveal that, despite having mostly a better initial performance, our models benefit less from self-training.

In the second part of our empirical evaluation, we employ our HTG system in a setting that is to be close to the practical application. As part of this study, we also employ other state-of-the-art HTG methods in the same settings and critically assess the limitations and potentials of the approaches. Our experiments aim to provide insights regarding the following questions.

How does our model compare to other HTG methods for generating training data for unseen target datasets? This aspect focuses on the main motivation of this thesis which is the generation of training samples for a target dataset. In contrast to the previous experiments, the evaluation in Section 5.4.1 is conducted with less restrictions and shows that our model compares favorably in this setting.

Do the models correctly generate the words in the desired style? By conditioning the model on a word and a writing style, we expect the generated image to contain the correctly spelled word in the desired style. We use HTR and WID models which are both trained on real samples to then assess the correctness of both modalities individually. The experiments in Section 5.4.2 show that all models mostly generate correctly recognizable word images in different settings. However, the results reveal that all models struggle with the imitation of the writing style.

In which cases is it worth to employ the HTG systems for training data generation? The training of the HTG models comes with high computational costs. At the same time, the performance of the HTR model trained on generated samples still shows a significant gap to that of a model trained on real images. In Section 5.4.3, we therefore investigate the proportions of the annotated real samples for which the generation of training data using HTG is worthwhile. The results

suggest that training data generation is mainly beneficial when no or only a few real samples with annotation are available.

In order to gain empirical evidence to answer these questions, a controlled experimental setup is required. We use one benchmark dataset as our source dataset and consider three additional ones to be our target datasets. In Section 5.1, we provide an overview of the datasets used and present details regarding their individual particularities. Besides the benchmark datasets, suitable evaluation measures are essential for a quantitative assessment of the generation results. To measure the generation performance of the models, we use three different approaches which are presented in Section 5.2. The qualitative as well as quantitative results are discussed in Section 5.3 and Section 5.4.

5.1 DATASETS

According to the problem formulation (see Section 4.1), datasets for conducting experiments for HTG must meet some requirements. First, the datasets should be written by multiple writers. Although the models are usually suitable for imitating the handwriting of only one writer, the generation of various writing styles is of particular interest. Second, like several other HTG systems, our approach works on word images, thus demanding a dataset with word-level segmentation. Third, the generation should be conditioned on the textual content as well as a specific handwriting style. Therefore, transcriptions and writer IDs need to be annotated for the word images. These demands significantly limit the choice of suitable benchmark datasets.

For our empirical evaluation, we select four publicly available benchmark datasets of handwritten word images with different characteristics. The datasets comprise samples from different languages, namely English, French and German. Furthermore, they greatly vary in the number of writers, the number of samples per writers and the overall number of word images. All datasets contain mostly upper- and lower-case letters, but also a few digits and punctuation marks. Three of them were created for research purposes and contain handwritten documents scanned in high quality and without significant distortions. In practice, however, images usually originate from a variety of sources and are often subject to distortion. Therefore, we select the fourth dataset to be a collection of handwriting images in the wild (e. g. photos of handwritten notes taken with a smartphone) which we use to explore the applicability of our approach in more challenging settings. An overview of samples from the different datasets is provided in Figure 27. To highlight the differences of real handwriting to samples rendered from computer fonts, we also include synthetic samples which we used for reference experiments on self-training [WF24].

IAM Database

The IAM database is a collection of handwritten English sentences and was proposed as a benchmark dataset for offline HTR [MB02]. It is based on texts from the Lancaster-Oslo/Bergen (LOB) corpus [JLG78], which were split into fragments of a few sentences. A number of persons were asked to write down these fragments in their own handwriting on provided forms. Starting with a total of 556 forms written by

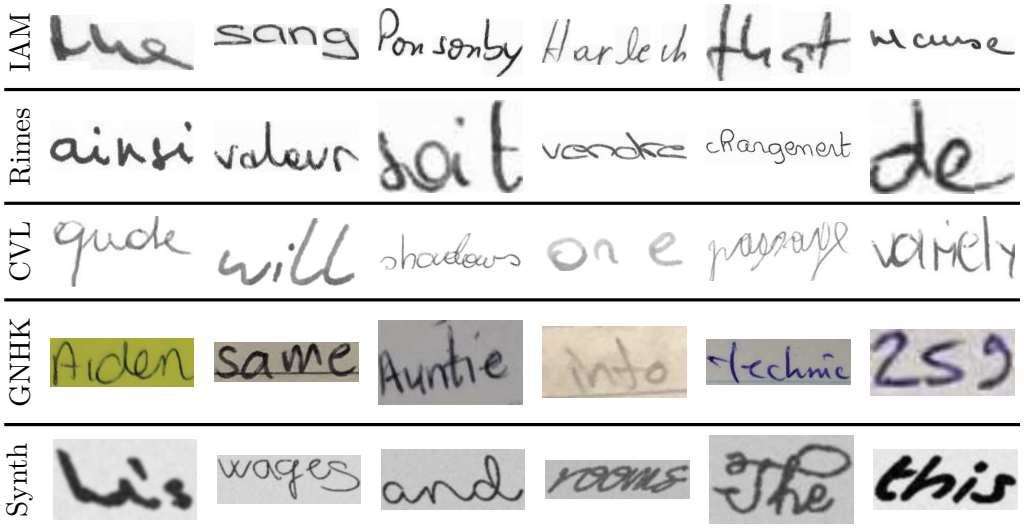


Figure 27: Examples for word images from the different benchmark datasets used in the experimental evaluation.

approximately 250 writers [MB99], the database was extended to 1,066 forms from around 400 writers [MB02]. Later, the database was further extended to include over 1,500 forms produced by more than 650 writers[1]. Since the forms were scanned with a resolution of 300 dpi, the images are of high quality with handwriting on a homogeneous and light background. Beginning with the version from 2002 [MB02], the dataset provides a word-level segmentation with case-sensitive transcriptions and annotated writer IDs. With its large variety of writing styles in combination with its size, the IAM database has become the standard benchmark dataset for HTG allowing for conditioning on text as well as style. Therefore, we employ the IAM database as our source dataset (i. e. transcriptions and writer IDs are annotated) in all our experiments.

The authors provide a split into one training, one testing, and two validation sets[1] where writers only contributed to one of the subsets. Later, a different split into one training set containing samples from 339 writers and one test with handwriting from 161 different writers has been proposed for HTG [Kan+20b]. Since this split has been adopted by several other approaches (e. g. [Mat+21; Nik+23; Van+25]), we decide to also employ these subsets for our experiments.

Rimes Database

The Rimes database is a collection of French handwritten mails which was used for different competitions [Aug+08; GA09; GE11]. Because existing mails could not be used for legal and confidentiality reasons, the dataset was created out of mails from volunteers based on fictional identities and given scenarios. In this way, over 12,000 pages written by more than 1,300 people were collected and scanned with a resolution of 300 dpi. For HTR, subsets of more than 50,000 training and around 7,000 test images were created [GE11] where each writer contributed to only one of these partitions. We use these subsets for our experiments as a target dataset. A

1 IAM Handwriting DB 3.0: https://fki.tic.heia-fr.ch/databases/iam-handwriting-database, accessed: 23.04.2025.

difficulty with this dataset lies in the fact that for most writers only a few examples (cf. Table 2) are provided. Another difficulty arises as the source dataset (i. e. IAM) only contains English words while the target dataset comprises French words. While several works applied their HTG approach to Rimes, only a few works [Kan+22; Van+25; Pip+25] explored the generation of samples from Rimes with a model trained on a different dataset. By using Rimes as a target dataset, we examine whether the model learns to generate characters independent of the word or language. However, we remove words containing characters with accents as these do not occur in the IAM database.

CVL Database

The CVL database is a collection of handwriting images which has originally been presented for writer retrieval and identification [Kle+13]. For this reason, the focus during the creation was on the variation of writing styles rather than on diverse contents. Examples were only created based on six English and one German text. However, the majority of these texts have been written by all 311 different writers. As segmentation on word-level is also provided, the dataset can be used for word spotting and, in theory, also for HTR. However, its tiny lexicon of only 292 words, coupled with the fact that the training and test partitions contain the same texts, make the dataset unsuitable as an HTR benchmark. Similar to IAM, the document images were digitized by scanning with a resolution of 300 dpi. In contrast to most other datasets, the partitioning into training and test samples is done such that the training set size is only around one-seventh of the test sets size. However, the writers of the partitions form disjoint sets. Even if the dataset is rather unsuitable for HTR, it is used in several works (e. g. [Fog+20; ZN23; Zhu+23]) as a target dataset where training data for an HTR model is generated. We therefore also use the CVL database as a target dataset in our experiments to evaluate the generalization performance to a dataset, mostly written in English.

GoodNotes Handwriting Kollection

All of the previously described datasets contain word images from mostly horizontally oriented text lines which were obtained by scanning handwritten documents. The results are mostly clean handwriting images with high resolution and only little perspective distortion in which the segmentation in bounding boxes aligns well with the horizontal orientation of the words. The GoodNotes Handwriting Kollection (GNHK) [LCL21] complements these clean datasets by a collection of unconstrained handwriting images "in the wild". Instead of flatbed-scanned images, the dataset comprises images which were captured by mobile phone cameras under unconstrained settings. This introduces perspective deformation, noise and a high variation in brightness and luminosity. Furthermore, the images depict different types of documents like shopping lists, notes or diaries resulting in a considerable variation of backgrounds. The creators of this dataset only considered English documents. However, they introduced additional variation in lexicon and style by collecting images from different continents. With the characteristics described above, this dataset presents a particularly challenging generation task. Overall, the dataset comprises

Table 2: Number of samples $|\mathbf{D}|$, number of writers $|\mathbf{W_D}|$, median number of samples per writer $\text{med}(|\mathbf{I}_w|)$ and lexicon size $|\mathbf{L_D}|$ of the benchmark datasets after filtering. Additionally, the percentage of out-of-vocabulary samples in the test set compared to the train set (OOV) and compared to IAM-train (OOV_{IAM}) are presented. The training partitions of the target dataset are marked with *.

Dataset	Split	$\lvert\mathbf{D}\rvert$	$\lvert\mathbf{W_D}\rvert$	$\text{med}(\lvert\mathbf{I}_w\rvert)$	$\lvert\mathbf{L_D}\rvert$	OOV	OOV_{IAM}
IAM	train	44 405	338	66	4 916	–	–
Rimes	train*	34 468	1 343	23	1 740	–	81.19
CVL	train*	10 067	27	373	248	–	20.72
GNHK	train*	17 346	510	26	4 423	–	24.60
IAM	test	18 436	161	70	3 332	9.42	9.42
Rimes	test	5 182	179	26	658	2.43	80.57
CVL	test	71 820	283	256	207	0.06	25.07
GNHK	test	5 552	172	26	2 009	21.34	24.05

687 document images from four continents. As no writer annotation is provided, we assume one individual writer per document image. Thereby, only a few (around 50) samples are available for most writers. Partitioning on the basis of entire documents results in each writer having contributed to either the training or the test split. Due to their unconstrained nature, some documents contain math expressions. As the synthesis of such expressions is beyond the scope of this thesis, we do not consider such examples.

Implementation and Partitioning

In our experiments, we explore different settings regarding the datasets. For the first part of the experiments, we follow previous works [Nik+23; Kan+20b][2] and limit the datasets to images which we assume to be easier to generate. We only consider words with at least two and at most seven characters. Furthermore, we only consider Latin/ASCII characters ([a-zA-Z]) and no punctuation nor digits. Despite these limitations, we do not apply any further changes to the default splits. The resulting statistics for the different partitions are presented in Table 2. We denote the share of *out-of-vocabulary* (OOV) samples in the test split with regard to the training split of the respective datasets as well as the percentage of OOV samples in each split compared to the IAM train split. As the IAM database is our source dataset, this indicates the difficulty caused by differences to the vocabulary of IAM to that of the target dataset. The OOV portion of the Rimes dataset is quite large due to the different language. However, there is still an intersection of the vocabulary with that from IAM. Furthermore, the different characteristics of the datasets can be observed, e. g., the small number of samples per writer of Rimes and GNHK.

2 While not explicitly mentioned in their paper, Kang et al. [Kan+20b] applied the same limitations in the annotation files published with their implementation.

In the second part of our evaluation, we compare our method against other state-of-the-art HTG systems and analyze their strengths and weaknesses. To create a setting closer to practical applications, we do not apply any restrictions regarding the source dataset and include samples containing digits and punctuation in these experiments. We only remove images containing only punctuation as these are usually scaled differently from the other characters and therefore might harm the performance [Van+25]. The datasets contain words with at least one and up to 21 characters since we also do not limit the number of characters per word. As we transfer our model to the target dataset, no new characters (e. g. characters with accents in French) can be introduced. We therefore restrict the target datasets to samples which only contain characters present in the source dataset (i. e. IAM).

To gain further insights into the model performances, we want to differentiate between four different settings. We distinguish between generating *in-vocabulary* (IV) and OOV words and between writing styles seen during training (S) and unseen writing styles (U). This results in the four settings IV-S, IV-U, OOV-S and OOV-U. We consider only fully labeled samples as seen words and seen styles, respectively. The generation of seen styles is therefore only possible for the source dataset. Furthermore, modifications of the partitioning into training and test splits are required to enable generation of seen styles. We aim for about 3000 samples for each setting as far as possible, since moving writer-word pairs between partitions can affect both criteria. For all target datasets, only IV-U and OOV-U settings can be evaluated where OOV is defined with regard to the source dataset. To enable the computation of our evaluation measure based on a writer identification (WID) model as discussed in the next section, we need to build an extended training set which contains samples from all writers. This set, however, will only be used for training of the auxiliary models which are required for the computation of the performance measures. The statistics of the resulting sets when omitting the limitations are presented in Table 3. Detailed information about the evaluation splits for the four scenarios are provided in the appendix in Table 15. The decline in the number of samples for IAM is caused by leaving out the IV-S, IV-U, OOV-S and OOV-U splits. For the other datasets the number of samples increases as the restrictions are omitted. Allowing longer words and extended character sets results in larger vocabularies. At the same time, the OOV shares between training and test partitions as well as between IAM and the other datasets increase.

To simplify training in batches, we resize all images to a height of 64 pixels preserving the aspect ratio, similar to related works [Nik+23; Nik+24a]. Our desired width is 256 pixels. If the resulting image is smaller than this, we pad the image to the right with white pixels for IAM, Rimes and CVL. Due to the heterogeneous backgrounds in GNHK, we decided to pad the images using their median gray value. For the first part of our experiments, images wider than 256 pixels are scaled to fit this width. While the scaling changes the aspect ratio, there are only small distortions due to the limitation to 7 characters [Nik+23]. For the second part, where our datasets contain longer words, we adopt the scaling from DiffusionPen [Nik+24a]. Therefore, we scale images which exceed the width down (preserving the aspect ratio) until they fit into the desired width of 256 pixels. In addition to the padding in the horizontal direction, we pad these images in the vertical direction to fill the height up to 64 pixels.

Table 3: Number of samples $|\mathbf{D}|$, number of writers $|\mathbf{W_D}|$, median number of samples per writer $\text{med}(|\mathbf{I}_w|)$ and lexicon size $|\mathbf{L_D}|$ of the benchmark datasets for the second part of our experiments. Additionally, the percentage of out-of-vocabulary samples in the test set compared to the train set (OOV) and compared to IAM-train (OOV_{IAM}) are presented. The training partitions of the target dataset are marked with *.

Dataset	Split	$\lvert\mathbf{D}\rvert$	$\lvert\mathbf{W_D}\rvert$	$\text{med}(\lvert\mathbf{I}_w\rvert)$	$\lvert\mathbf{L_D}\rvert$	OOV	OOV_{IAM}
IAM	train	42 434	338	63	7 384	–	–
Rimes	train*	45 867	1 344	31	3 724	–	84.17
CVL	train*	11 989	27	444	315	–	32.35
GNHK	train*	26 981	510	40	9 768	–	45.25
IAM	test	17 752	161	67	4 638	19.42	19.42
Rimes	test	6 862	179	36	1 299	4.95	83.97
CVL	test	86 282	283	308	300	0.59	37.28
GNHK	test	8 390	172	40	4 004	33.56	44.62

5.2 EVALUATION MEASURES

While the task of image generation allows for a qualitative assessment of the generated images, a comparison of different approaches is only practical if their performance can be summarized into a numerical value. In contrast to other problems like recognition or retrieval, where such an evaluation measure is defined as part of a commonly applied evaluation protocol, up to now, no standard evaluation protocol has been agreed on for HTG. There exist various evaluation measures which have been adopted by works on HTG as discussed in Section 3.2.6. These measures cover different aspects such as visual quality, readability, style imitation as well as diversity and fidelity [ZN23].

Downstream Character Error Rate

The focus of the experiments in this thesis is on the application of HTG for the generation of training data for an HTR model. Therefore, we are not explicitly interested in the visual quality. Instead, readability, diversity and fidelity are essential when creating training data. The quality of the generated dataset determines the performance of models trained with it. Therefore, we can employ the evaluation measures according to our downstream model after training it on generated samples. In the case of HTR, the standard evaluation measures are the *character error rate* (CER) and the *word error rate* (WER). These measures compare the predicted strings with the annotated transcriptions on character- and word-level for an evaluation dataset.

The CER compares two sequences of characters. The difference of these sequences is computed based on the *Levenshtein distance* [Lev66] which is often also referred to as *edit distance*. It is defined based on the number of operations required to transform

one sequence into the other by inserting, deleting or substituting elements. The CER (or *normalized edit distance*) is then defined as

$$\mathrm{CER} = \frac{I + D + S}{N}, \tag{94}$$

where I, D and S are the number of insertions, deletions and substitutions respectively. N denotes the length of the annotated sequence.

The WER is defined in analogy to the CER but based on sequences of words. As we work on images depicting single words instead of text lines, insertions and deletions cannot occur. This simplifies the WER to the ratio of incorrect predictions (substitutions S) to the total number of words N in the dataset:

$$\mathrm{WER} = \frac{S}{N}. \tag{95}$$

While prediction of completely correct words is essential when applying HTR for the automatic transcription of documents, the measure lacks granularity. All words with at least one insertion, deletion or substitution are considered incorrect independent of the exact number of wrongly predicted characters. Since we want more detailed insights into the correctness of generated character sequences, we decide to focus only on the CER.

If not stated otherwise, we replicate the training set for the training of the recognition model, i. e. we generate the same pairs of text and style like in the annotation of the original training partition. Although this means that we lose the advantage of introducing additional variation to the generated training data, we eliminate possible influences from changes in the word and style distributions. In this way, we achieve a fair comparison of results with each other as well as results of HTR models trained on the real training samples. This approach of training a downstream model on generated samples is similar to the GAN-train metric [SSA18] applied to HTG [ZN23]. It is considered to measure image quality as well as diversity.

Assessment by Recognition Models

In the second part of our experimental evaluation, we compare different HTG methods in detail. To gain further insights into the strengths and weaknesses of the individual models, we employ additional performance measures. Inspired by the GAN-test measure [SSA18] and other works on HTG (e. g. [ZN23; Mat+21; Gan+22; Van+25]), we train an HTR model on real samples and measure its performance on generated samples. This metric is then compared to the CER on the real samples. If the generative model overfits the training data, the recognition model is expected to perform better. A negative difference therefore suggests that the model overfitted the training data [SSA18]. In contrast, a large positive difference indicates that the generation quality and correctness of the content is poor. Besides the correct generation of the text with styles matching those of the desired dataset, the performance can be influenced by the generalization capabilities of the recognition model to OOV words. To minimize this impact and to allow for a fair comparison between the experiments, we again generate replicas of the respective subsets for evaluation.

This aforementioned measure, mainly captures the correctness of the generated texts, but not whether the respective word was generated in the desired style. As

long as the distribution of generated styles is similar to that of the real samples, the performance of the HTR model should not be significantly impacted. Therefore, we follow a similar scheme but use a WID model and compare its error rate on generated samples to that on real samples. This evaluation approach based on the *writer identification error rate* (WIER) was first proposed by Gan et al. [Gan+22] to evaluate the style imitation performance. Like the WER, it is computed as the number of samples F where the writer is incorrectly predicted divided by the total number of samples N:

$$\text{WIER} = \frac{F}{N}. \tag{96}$$

We found that using a ResNet-18 [He+16] as proposed by Nikolaidou et al. [Nik+23; Nik+24b] as a WID model works well for IAM and CVL. However, even on the real samples from Rimes and GNHK, the performance is poor. Therefore, we chose to employ a MobileNetV3 [How+19] with global average pooling as our WID model.

5.3 RESULTS

In this section, we present the results for a series of experiments which aim to explore different aspects in the design of our model, including training and sampling. First, we evaluate the generation quality of our model when trained with fully labeled samples in Section 5.3.1. As part of these experiments, we evaluate different choices of our proposed *style inclusions* (SIs) and the application of CFG. Furthermore, we present a qualitative evaluation. In Section 5.3.2, we use our model to generate training data for a new dataset (i. e. Rimes) and show how its generation performance can be improved by including samples from the target dataset which are only labeled with writer IDs. As the next step, we explore the parameters introduced by this training scheme in Section 5.3.3 and explore a semi-supervised training scheme without any requirements for annotations for the target dataset. An evaluation for different choices of style encoders (SEs) is presented in Section 5.3.4. After the evaluation of several aspects of the model itself, we investigate the impact of the generated content and the number of generated samples for training the HTR model in Section 5.3.5. In Section 5.3.6, we then explore the applicability of our approach for other target datasets (i. e. CVL and GNHK). A comparison to adapting an HTR model by self-training is presented in Section 5.3.7. In this context, we also investigate whether self-training can be employed to further improve HTR models trained with our generated samples.

All of the aforementioned experiments involve several random processes. On the side of the SE, we introduce randomness in the training as well as in building the style embedding cache. The LDM involves random processes in its training and during sampling. Finally, the training of the HTR model might also introduce variation. Regarding the sampling, we try to remove some variation by keeping the writer-word pairs fixed for sampling. Nonetheless, a lot of randomness in our pipeline remains which is expected to cause some variation in the results. Due to the high costs of training all models in the pipeline[3], it is infeasible to repeat all experiments several times for significance testing. For the interpretation of the empirical evaluation,

3 Training of the LDM using one Nvidia RTX 3090 took around 65 hours when using only IAM and around 120 hours when using IAM and Rimes (semi-supervised).

however, an estimate of the variance of our results is desirable. We therefore provide a reference of the variance for our most frequently used configuration of our HTG system by repeating the following experiment 15 times. We train the MAE on IAM and employ our semi-supervised training of the LDM using writer labels for Rimes as our target dataset. Using CFG during the training, we sample images with guidance scales $w_{gs} = 5$ for IAM and $w_{gs} = 2$ for Rimes. For further details regarding the experimental setup we refer to Section 5.3.2. Averaging the results from the 15 repetitions, we obtain a CER of 7.75 % with a standard deviation of 0.15 for IAM and a CER of 6.78 % for Rimes with a standard deviation of 0.37. These results indicate that the variation is relatively small considering the amount of random processes involved.

5.3.1 *Style Inclusion and Classifier-free Guidance*

Before generating training data for a partially labeled target dataset, we assess the generation performance of our model using the fully annotated source dataset (i. e. IAM) first. As part of these experiments, we explore the effect of CFG as well as our proposed SIs. First, we train an MAE (see Section 4.3) to serve as our SE. For the training, we follow the same procedure as Souibgui et al. [Sou+23]. After training, we compute embeddings for all writers by Equation 91 with $\mathsf{K} = 10$. We repeat this computation $\mathsf{N}_{se} = 100$ times per writer to introduce variance and foster the exploration of the style space. Different choices of K and N_{se} are later discussed in Section 5.3.3.

As the next step, we train different LDMs including our proposed content encoder. All of these are trained using the same cache of writer embeddings. We explore our six proposed SIs. For each SI, we train one LDM with ($p_{uc} > 0$) and one without CFG ($p_{uc} = 0$). In their experiments, Ho and Salimans [HS21] found that $p_{uc} = 0.1$ and $p_{uc} = 0.2$ achieve good results. Since we drop two conditions and introduce a large amount of unlabeled samples in later experiments, we decide to use the lower probability for unconditional training. We independently replace writer or word embeddings with the respective `[null]`-token with probability $p_{uc} = 0.1$.

After training of the LDM, we sample training images for an HTR model by generating the same pairs of writers and words as in the training set of IAM (cf. Section 5.2). For the LDMs trained with CFG, we perform unguided sampling[4] as well as sampling with different guidance scales $w_{gs} \in \{2, 3, 4, 5, 6, 7\}$. Instead of sampling with the default sampling algorithm [HJA20], we use UniPC [Zha+23]. This framework allows us to sample our LDMs using only 50 denoising steps instead of the 1000 steps used during training. While resulting in minimal degradations in the generation quality, this choice drastically reduces the computational costs for sampling (see Section A.2 for details). The HTR model is then trained only on the generated samples and evaluated on the real images from the IAM test partition. For training the recognizer, we employ the same training procedure as proposed by Retsinas et al. [Ret+22]. Additionally, we train one model on the real images from

4 In this case, the conditioning is set to the `[null]`-token with $p_{uc} = 0.1$ during training, but sampling is performed using only the conditional prediction from the model instead of a linear combintation of conditional and unconditional predictions.

Table 4: CER [%] on real IAM test images of HTR models trained on generated images from LDMs trained with different style inclusions. $p_{uc} = 0$ denotes training without CFG. For reference, the model trained on real samples from IAM train achieves a CER of 6.01 %.

Guidance	p_{uc}	TP	TPL	CP	TA	CA	TS
No CFG	0	8.68	8.57	8.74	8.48	9.06	**8.08**
Unguided	0.1	9.10	9.18	9.33	9.19	9.49	**8.85**
$w_{gs} = 2$	0.1	8.17	8.11	8.51	8.70	8.24	**7.84**
$w_{gs} = 3$	0.1	7.62	8.05	7.98	8.02	8.08	**7.54**
$w_{gs} = 4$	0.1	7.78	7.77	8.18	7.72	7.93	**7.57**
$w_{gs} = 5$	0.1	7.56	7.98	7.60	7.84	7.82	**7.39**
$w_{gs} = 6$	0.1	**7.54**	7.64	7.73	7.66	7.72	7.69
$w_{gs} = 7$	0.1	7.59	7.61	8.07	7.79	7.70	**7.56**

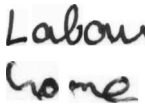
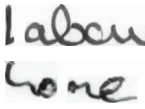
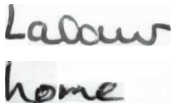
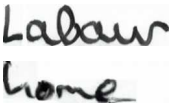
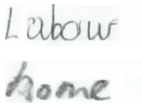
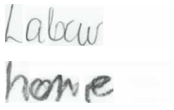
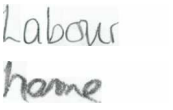
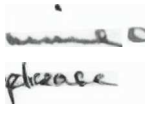
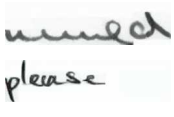
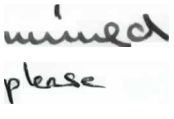
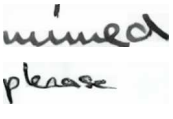
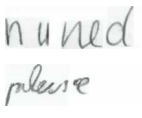
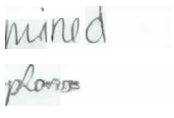
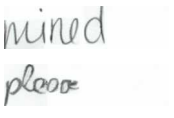
the IAM train partition to allow for a better assessment of the results. We report the results in Table 4.

First, we compare the results of models trained on generated samples with the 6.01 % CER of the model trained on real images from IAM. As expected, these performance is worse compared to training on real samples. However, we observe only a small decrease in CER which indicates that all explored configurations of our HTG system are able to generate samples which are similar to the original ones.

Good results are obtained for training with as well as without CFG. When training the models with CFG, the choice of the guidance scale w_{gs} during sampling is essential. When sampling these models without guidance, the results are inferior to those of the models trained with $p_{uc} = 0$. Increasing w_{gs}, however, results in a better performance for all SIs. The largest improvements are observed for w_{gs} up to 3. Higher guidance scales mostly result in minor improvements or even lead to a slight deterioration. These results are in line with the findings, that guidance scale trades off diversity for fidelity [HS21]. As we strongly control for diversity by conditioning the model on both textual content as well as the writing style, we assume that the decrease in diversity does not harm the performance of the downstream HTR models. At the same time, the increase in fidelity for higher guidance scales helps to generate more reliable training samples. Overall, the generation quality is not sensitive to the exact choice of w_{gs} as long as sampling is performed with guidance.

A comparison of the different SIs reveals only minor performance differences. Nevertheless, for all guidance scales except for $w_{gs} = 6$, the best results are achieved when adding the style embedding vector to the timestep embedding (TS). The difference in performance compared to the remaining SIs, however, decreases for higher guidance scales. These results suggest, that the model is able to learn the incorporation of style information in different ways. However, a global inclusion in combination with the timestep embedding seems favorable.

Table 5: Examples for the influence of CFG and different guidance scales w_{gs} for sampling. The original image from the respective writer is shown in the second column as a reference.

	Reference	No CFG	Unguided	$w_{gs} = 2$	$w_{gs} = 5$
IV-S					
IV-U					
OOV-S	N/A N/A				
OOV-U					

While we make our design choices based on the quantitative evaluation using the evaluation measures described in Section 5.2, a qualitative assessment of the results is still interesting. Therefore, we provide an overview of a few generated samples in Table 5. We use our LDMs which add the style embedding to the timestep embedding (`TS`). The focus here is on the influence of CFG and guided sampling on the visual quality and the correctness of the generated content. Besides that, we distinguish between the generation of IV and OOV words as well as styles seen during training (S) and unseen styles (U). For each case, we provide a real image of the respective word written by the respective writer as a reference. Only for the OOV-S case no reference can be provided as the partitioning of the IAM database does not contain examples which fulfill these criteria.

Throughout all four settings, a noticeable improvement in visual quality can be observed for the images generated by the model which was trained using CFG. In particular in the OOV-U case, the words generated by the model without CFG are incorrectly spelled or illegible. Further improvements can be observed, when using guided sampling. While a higher guidance scale leads to an almost continuous improvement when the samples are used for training an HTR model, this trend is not clear in the visual inspection of the generated images. In some cases the spelling of letters is better for the higher choice of guidance scale (e. g. the “b” in “Labour” in the first row). In other word images, however, the strokes are more wavy than they should be (e. g. “Labour” in the third row or “mined” in the seventh row). Word images sampled with $w_{gs} = 2$ are subjectively better generated in the latter cases.

Overall, at first glance, the writing style of the generated images mimics the one of the reference images well. In the IV-S case, this impression remains even after a more thorough inspection. When generating words from unseen writers, mainly the appearance of the strokes is imitated well. However, details of the individual handwriting like the “h” of “home” (row 4) are missed.

Table 6: CER [%] on real Rimes test images of HTR models trained on generated images. The datasets used for training the style encoder and the LDM are denoted in the rows "SE trainset" and "LDM trainset", respectively. For reference, the model trained on real samples from Rimes train achieves a CER of 2.47 %.

LDM trainset	Rimes	Both		
SE trainset	Rimes	Rimes	IAM	Both
Unguided	4.16	**3.93**	3.95	4.12
$w_{gs} = 2$	3.78	**3.52**	3.72	3.99
$w_{gs} = 3$	3.78	3.58	**3.46**	3.58
$w_{gs} = 4$	3.60	**3.43**	3.74	3.84
$w_{gs} = 5$	3.42	**3.25**	3.72	3.77

In summary, the results strongly support the use of CFG and guided sampling. However, they also reveal deficits, particularly in the imitation of handwriting specific details. At this point, it is important to point out that the given examples represent only a tiny section of the generated dataset and do not allow any general conclusions to be drawn. More examples for generated images for other datasets are presented in the appendix in Section A.5.

5.3.2 *Generation of Training Data for a Target Dataset*

After successfully generating samples for the labeled source dataset our HTG system has been trained on, we focus on our actual task of generating images for a target dataset. This scenario is particularly interesting with regard to the task of training data generation, since the generation of samples for a labeled dataset only serves as augmentation. However, generating samples based on a word and a writing style allows us to create a labeled training set with styles from the target dataset, enabling supervised training of downstream models. For this series of experiments, we stick to IAM as our source dataset with transcriptions and writer labels but introduce Rimes as our target dataset for which only writer labels are available. With a median of only 23 samples per writer, Rimes provides a realistic example for inferring pseudo-writer IDs by assuming one writer per document. In contrast to the previous experiments, the HTG has to generate writing styles and words from the target dataset which have not been seen during training. To approach this challenge, we explore a semi-supervised strategy relying only on writer labels for the target dataset (see Section 4.4).

Beyond our experimental setting, transcriptions are available for this dataset. This allows us to first conduct reference experiments which demonstrate the applicability of our HTG system to the fully labeled version of Rimes. We follow the same setup as in the previous section and train the LDM on Rimes with transcriptions and writer labels. The writer embeddings used in this experiment were computed by an SE which has been trained on examples from Rimes. Additionally, we explore the effect of training the LDM on samples from a set which is comprised of the IAM

Table 7: CER [%] on real Rimes test images of HTR models trained on generated images. The images are generated by LDMs trained only on IAM.

Guidance	p_{uc}	TP	TPL	CP	TA	CA	TS
No CFG	0	10.48	**10.30**	10.47	11.25	12.49	11.83
Unguided	0.1	8.92	**8.66**	10.07	8.73	9.11	9.30
$w_{gs} = 2$	0.1	9.24	**9.01**	11.17	9.13	10.42	9.58
$w_{gs} = 3$	0.1	9.92	9.17	10.59	**8.99**	11.31	10.19
$w_{gs} = 4$	0.1	9.96	**9.23**	11.19	9.48	11.07	9.58
$w_{gs} = 5$	0.1	9.40	**9.32**	11.73	9.56	11.55	9.86

and Rimes training partitions. One motivation for employing an MAE as our SE is the simple inclusion of samples from several data sources. Besides using writer embeddings from the SE trained only on samples from Rimes, we also train MAEs on IAM and a combination of both datasets. Since training LDMs is computationally expensive, we focus on the best performing SI from the previous experiments (i. e. TS) and employ CFG with $p_{uc} = 0.1$.

We report results for HTR models trained on samples which are generated using different guidance scales in Table 6. For reference, we also provide the CER when the recognizer has been trained on real examples of Rimes. Similar to the experiments on IAM, we observe a performance gap of HTR models trained on generated samples compared to the reference model. However, the gap is also slightly smaller in accordance with the better overall results on Rimes. Similar to our findings on IAM, increasing the guidance scale for sampling results in improved performance in almost all cases.

When comparing the different choices of training datasets for the SE and the LDM, no significant differences can be identified. An interesting observation is that the choice of the training dataset for the SE has only a minimal influence on the results. As such, training the SE only on IAM achieves competitive performance. We consider this an indication that the SE generalizes well for extracting style features from the new dataset.

Since our proposed HTG system is technically able to generalize to unseen writing styles and OOV words, we assess this generalization capability by using the LDMs trained on IAM to generate samples from Rimes. We use the images from Rimes only for computing the writer embeddings for sampling. As we assume to have annotations of the writer IDs, we can compute $\mathsf{N}_{se} = 100$ embeddings per writer based on $\mathsf{K} = 10$ random samples each. Despite this, images from Rimes are not used elsewhere in these experiments. The results in Table 7 show a considerable drop in performance compared to fully supervised training on Rimes. In contrast to the best LDM trained with annotated samples from Rimes that achieves a CER of 3.25 % (cf. Table 6), using the LDMs trained only on IAM, the best configuration has a CER of only 8.66 % (cf. Table 7). Similar to the experiments on IAM, the models trained without CFG perform worst. Although the LDMs which were trained with CFG perform better, the trend that the CER improves for higher guidance scales,

Table 8: CER [%] on real Rimes test images of HTR models trained on generated images. The images are generated by LDMs trained with fully labeled samples from IAM and samples from Rimes without transcriptions. SEs are trained on different datasets (column "SE train").

SE train	Guidance	TP	TPL	CP	TA	CA	TS
IAM	Unguided	7.52	8.65	7.03	**6.88**	7.94	7.15
	$w_{gs} = 2$	6.96	7.39	7.32	7.08	8.43	**6.61**
	$w_{gs} = 3$	7.78	**6.36**	8.23	7.41	8.48	7.29
	$w_{gs} = 4$	7.82	**6.28**	7.66	7.72	8.67	7.08
	$w_{gs} = 5$	8.21	**6.60**	7.63	7.69	8.69	6.91
Both	Unguided	9.74	9.19	**6.95**	8.53	7.24	7.17
	$w_{gs} = 2$	7.57	6.90	7.16	7.88	6.68	**6.65**
	$w_{gs} = 3$	7.05	7.42	7.23	7.99	7.17	**7.04**
	$w_{gs} = 4$	6.83	6.69	7.40	8.50	7.17	**6.63**
	$w_{gs} = 5$	6.80	**6.48**	7.12	8.60	7.04	6.96

does not hold. Instead, unguided sampling results in the best CER. This indicates, that guidance is only beneficial, when examples for the respective conditioning have been seen during training. Regarding the SI, for the transfer to the generation of Rimes, all options show comparable performances. However, in these experiments, slightly better results are obtained when the style embedding is included as a token in the conditioning sequence (i. e. `TP`, `TPL` and `TA`).

Our approach to mitigating the performance drop in this setting is to include samples from the target dataset in the training of the LDMs in a semi-supervised fashion, as described in Section 4.4. In this part of the experiments, we focus on semi-supervised training only with regard to the transcriptions of the target dataset and assume that writer labels are provided. We explore the inclusion of samples from the target dataset without any labels later in this work. Since in all prior experiments, the use of CFG with $p_{uc} = 0.1$ has been beneficial, we apply it for all further experiments and omit the configurations without CFG. In addition to the evaluations of the different SIs, another focus is on the training data of the SE. We assess, whether it is beneficial to train the SE on a combination of the source and the target dataset. The results for the different settings are presented in Table 8.

In comparison to the best results with a CER of 8.66 % for training data generated by an LDM which has only been trained on IAM (cf. Table 7), almost all results for semi-supervised training are considerably better. Furthermore, in contrast to the previous setting, guided sampling yields performance improvements in many cases. However, the results suggest that the guidance scale must be chosen carefully. Using a high guidance scale can deteriorate the results, depending on the SI used. These observations align with our previous assumption, that CFG is only beneficial if labeled samples with the respective conditioning are provided. In this scenario more

supervision is provided compared to training only on samples from IAM but less than in the fully supervised reference experiments. We assume that this behaviour is caused by the semi-supervised model being uncertain regarding the conditioning in some cases since no textual labels were provided for Rimes during training. Similar to the other settings, all SIs achieve comparable performance. For the model using embeddings from the SE trained on IAM, `TPL` results in the lowest CER for most guidance scales. When training the SE on both datasets, the the use of `TS` performs slightly better on average than the other options.

In designing the SE, we selected a model that allows for an easy inclusion of samples from multiple datasets in the training. Thereby, the model is expected to learn a better style representation for both datasets. The results, however, do not confirm this assumption, at least not with regard to the generative task considered in this thesis. Only for `CP` and `CA`, a performance improvement for all guidance scales can be observed when the SE is trained on both datasets. In contrast, results are consistently inferior for `TA` when the SE is trained on both datasets.

Since our combined dataset consists of more than 40 % samples without transcription, we expect to introduce some noise into the semi-supervised training process. As this may impact the generation performance of the models for IAM, we additionally conduct the same evaluation for IAM as for Rimes. Detailed results are reported in Table 17 in the appendix. However, we observe only a slight deterioration in the generation performance on IAM compared to the improvements on Rimes. Therefore, we conclude that semi-supervised training is an effective compromise for adapting our LDM for generating samples from a target dataset without transcriptions.

5.3.3 *Semi-supervised Training Without Writer Labels*

In the previous experiments, we have shown that including images from the target dataset without transcriptions helps for generating images for the respective dataset. However, we still rely on writer labels for the samples from the target dataset. For practical application it is preferable to not require any labels from the target dataset. Our HTG system required writer labels for the computation of writer embeddings from multiple sample images and for assigning these embeddings to samples from the respective writer. In this series of experiments, we explore the these steps in more detail and evaluate our proposed semi-supervised training without the need for writer labels as described in Section 4.4.

Several related works (e. g. [Nik+24a; Van+25; Gan+22]) randomly draw the samples for computing the writer embeddings each time a sample from the respective writer is used in training. Instead, we use a cache containing a fixed number of precomputed style embeddings per writer thereby limiting the variation in the style embeddings during training. To analyze the impact of the size of this cache, we conduct experiments with different numbers of embedding variations per writer. Because the training of the LDMs is computationally expensive, we focus on promising configurations which we identify based on the results from Section 5.3.2. As `TS` showed best performance for the generation of the source dataset and also performs well in semi-supervised training, we only use models with this SI for all following experiments. For sampling, we select the guidance scales with best performance of the respective model which is $w_{gs} = 5$ for IAM and $w_{gs} = 2$ for Rimes. To further investigate the

impact of the choice of the dataset for training the SE, we continue considering both options in our experiments. Detailed results are reported in Table 18 in the appendix. The results show, that caching style embeddings based on random samples from the writer is sufficient for a good generation performance. Therefore, we continue using $\mathsf{N}_{se} = 100$ precomputed style embeddings per writer to maintain comparability with prior experiments.

Another step that requires writer labels is collecting the K random samples for each writer embedding. This parameter controls the amount of style information that is available for computing the style embeddings. Using only one sample per writer allows for more variation and enables one-shot sampling of the respective style. At the same time, this limits the information on the writing style which can be extracted, i. e., only information about the characters present in the given samples is provided. Increasing the number of samples is expected to enrich the style embeddings and thereby lead to better generation results. However, requiring a large number of samples per writer embedding is prohibitive as there might not be enough samples available. A detailed evaluation of various choices of K is presented in Table 18 in the appendix. Improvements in the performance for higher numbers of images per embedding support the assumption that more style information can be encoded in these cases. However, our model is able to extract style information also from $\mathsf{K} = 1$ example image, only resulting a small performance drop. As high numbers of examples per writer embedding are usually not available in practice, we decide that $\mathsf{K} = 10$ offers a good trade-off between performance and the requirement of examples from a writer.

While the computation of style embeddings based on only one image is possible, writer IDs remain necessary for randomly assigning embeddings to examples during training. In order to remove this requirement, we experiment with computing the writer embedding for each training image only based on the respective image itself. We compute the embeddings using Equation 92 which introduces the hyperparameter p_r. By setting $p_r < 1$, the parameter allows to introduce variation to the embeddings despite only using one image. Setting $p_r = 1$ corresponds to using the complete image (i. e. no variation). We explore different shares of the patches for embedding computation with $p_r \in \{0.1, 0.25, 0.5, 0.75\}$. For images with a size of 64×256 pixels and a patch size of 8×8 pixels, $p_r \in \{0.1, 0.25, 0.5, 0.75\}$ correspond to 25, 64, 128 and 192 patches for the computation of the writer embeddings, respectively. As randomly selecting the patches allows for variation, we compute $\mathsf{N}_{se} = 10$ embeddings per training image. In the previous experiments, we observed that the preferable choice of guidance scales differs for different levels of supervision. Therefore we evaluate unguided sampling as well as $w_{gs} \in \{2, 5\}$. To avoid an influence of the embeddings that are used for sampling, we utilize the same embedding cache with 100 variations per writer as done before. Table 9 shows the results for the different training configurations. We use the performance of the LDM trained only on IAM as a baseline. Additionally, we report the performance of models trained in a semi-supervised fashion with writer labels for Rimes with $\mathsf{K} = 10$ and $\mathsf{K} = 1$ (both with $\mathsf{N}_{se} = 100$ for training). The main difference of the latter case to semi-supervised training without writer labels and $p_r = 1$ is that embeddings are computed on a random image from the writer instead of the training sample itself.

While training the SE on both datasets results in similar performance in semi-supervised training with writer labels, the results for semi-supervised training with-

Table 9: CER [%] on real Rimes test images of HTR models trained on generated images. The LDMs were trained with different values of p_r and compared to models with a different amount of supervision.

(a) SE training on IAM.

Model	p_r	Unguided	$w_{gs} = 2$	$w_{gs} = 5$
Semi (WL), $\mathsf{K} = 10$	–	7.15	6.61	6.91
Semi (WL), $\mathsf{K} = 1$	–	7.15	7.35	8.98
IAM only	–	9.30	9.58	9.86
Semi	0.1	10.53	9.08	8.86
	0.25	10.01	8.72	8.36
	0.5	10.20	9.02	7.90
	0.75	11.26	9.64	9.51
	1.0	10.91	10.04	9.06

(b) SE training on IAM and Rimes.

Model	p_r	Unguided	$w_{gs} = 2$	$w_{gs} = 5$
Semi (WL), $\mathsf{K} = 10$	–	7.17	6.65	6.96
Semi (WL), $\mathsf{K} = 1$	–	7.24	6.64	7.60
Semi	0.25	20.20	16.92	15.87
	1.0	25.41	24.72	19.74

out writer labels are significantly worse than those with the SE only trained on IAM. This is the case for different values of both hyperparameters p_r and w_{gs}. We therefore do not perform further experiments using this configuration. Instead, using the SE trained on IAM, we are able to achieve competitive results.

For unguided sampling, results of semi-supervised models are inferior to those of the LDM trained only on IAM. When sampling with $w_{gs} = 2$, the LDMs trained with $p_r < 1$ already produce competitive results. The best results are achieved for all tested values of p_r when images are sampled with guidance scale $w_{gs} = 5$. This confirms our expectation, that it is beneficial to adjust the guidance scale when changing the supervision during training. At the same time this observation suggests, that the choice of the guidance scale is more complex than just lowering the value according to the amount of supervision used.

Reducing p_r from 1 to 0.75 only results in a small improvement in CER for $w_{gs} = 2$. When lowering p_r further, we achieve consistent improvements compared to $p_r = 1$ and the baseline LDM trained only on IAM. This observation underlines that providing variations of the writer embeddings is beneficial for training the LDM. However, we can observe that choosing the lowest option $p_r = 0.1$ results in a small deterioration of the results. This can be explained by the low number of patches (i. e.

Table 9: CER [%] on real Rimes test images of HTR models trained on generated images. The LDMs were trained with different values of p_r using no labels for Rimes. Images were sampled with $w_{gs} = 5$ for the semi-supervised models and different numbers of embeddings, each computed using $\mathsf{K} = 10$ sample images from the respective writer. The baseline is given by an LDM trained only on samples from IAM which we sample without guidance.

N_{se} (sampling)	$p_r = 0.1$	$p_r = 0.25$	$p_r = 0.5$	$p_r = 0.75$	Baseline
100	8.86	8.36	7.90	9.51	9.30
1	8.31	8.39	9.01	9.97	9.61
2	8.96	9.17	9.33	8.55	8.78
5	8.49	8.71	8.37	9.27	9.39

25) included in the computation of the writer embedding vectors. The best performance is achieved for $p_r = 0.25$ and $p_r = 0.5$ where 64 and 128 patches are used for the computation of the writer embedding, thereby providing more information about the writing style. We conclude that the introduced hyperparameter p_r allows for trading off information per embedding vector against variations among the embeddings per image. Furthermore, the small differences in the results indicate that the model is not sensitive to the exact choice of p_r. Comparing the performance to models trained with writer labels, the difference in CER is reduced for good choices of p_r which shows the effectiveness of this approach.

In the previously conducted experiments on the choice of p_r, we always sampled with $\mathsf{N}_{se} = 100$. If no writer labels are provided, it is unrealistic to collect enough examples for this number of embeddings to contain a certain amount of variation. Instead, we explore the computation of only $\mathsf{N}_{se} \in \{1, 2, 5\}$ embeddings per writer for sampling. Since $p_r = 1$ as well as training the SE on both datasets did not achieve good results, we omit those configurations. Furthermore, we focus on sampling with $w_{gs} = 5$ except for the baseline which we sample without guidance.

The results in Table 9 show that the models behave differently when sampling with only a few embeddings per writer instead of 100. When sampling with 100 embeddings, $p_r = 0.25$ and $p_r = 0.5$ perform best. For the examined small choices of N_{se}, $p_r = 0.1$ also achieves very good results, especially for sampling with only one embedding per writer. On the contrary, $p_r = 0.75$ performs on par with the baseline, but requires significantly more training effort as samples from Rimes are included in the training. Since we consider $\mathsf{N}_{se} = 1$ for sampling the most relevant setting for practical application, the results suggest to choose $p_r = 0.1$ or $p_r = 0.25$. While both appear to be a reasonable choice, we use $p_r = 0.25$ for all following experiments.

5.3.4 *Computation of Style Embeddings*

For the computation of style embeddings, we choose to use an MAE. In contrast, many other works (e. g. [Gan+22; ZN23; Nik+24a]) use features from a writer identification (WID) model instead. There, the training objective ensures that writer

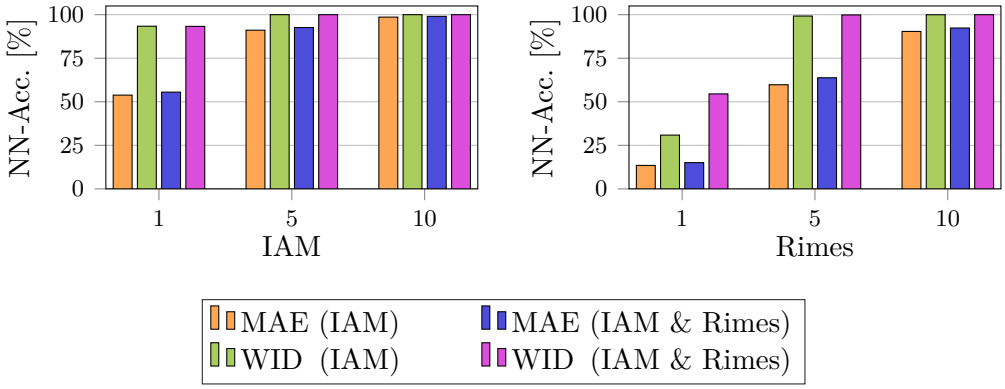


Figure 28: NN-Acc. [%] for writer retrieval based on style features computed by different models. Embeddings are averaged using $\mathsf{K} \in \{1, 5, 10\}$ images.

specific features are encoded by the models. The reconstruction loss of the MAE does not explicitly consider writer information. However, we assume that a successful reconstruction in pixel space requires textual as well as style information to be encoded in the latent features. In this section, we first explore whether writer information is contained in the embeddings computed by our SE. For comparison, we train a MobileNetV3 [How+19] for writer classification and use the features from its penultimate layer. We use either the training partition of IAM or both IAM and Rimes to train the models. As described in Section 4.3.3, we average the embeddings of K images. For every image from the training partition, we use the embedding to retrieve its nearest neighbor. We use the resulting accuracy (NN-Acc.) as a proxy measure of how much writer information has been encoded. The results for different K are visualized in Figure 28.

When computing the embedding only based on $\mathsf{K} = 1$ example, the WID model achieves significantly higher accuracy than the MAE. Furthermore, the WID shows better generalization performance when trained on IAM and evaluated on Rimes. The difference in accuracy between the models decreases when increasing K. For $\mathsf{K} = 10$, the MAE performs almost equally compared to the WID model.

The chosen proxy task, however, is closely related to the writer identification task. We therefore assume that the NN-Acc. favors the discriminative features computed by the WID model. In contrast, our application as style embeddings for conditioning our LDM is a generative task. For this task, embeddings that come from a model which has been trained with a generative objective might be more suitable. In a further experiment, we therefore compare the usage of embeddings computed by the different models for conditioning our LDM. The performance is evaluated as described in Section 5.2. We explore different scenarios as done in the previous experiments in this chapter. We train the SE and the LDM only on IAM and generate IAM and Rimes. Additionally, we evaluate the embeddings in semi-supervised training using Rimes with only writer labels and Rimes without any labels. For all experiments, we compute writer embeddings using $\mathsf{K} = 10$ samples. The results are reported in Table 10.

The performance when generating IAM is almost equal for both SEs. Using the LDM trained only on IAM to generate Rimes, the MAE-based SE shows considerably better performance. One motivation of employing an MAE as the SE was

Table 10: CER [%] for different generation scenarios with different levels of supervision (supv.) for the target dataset, i. e. Rimes.

Scenario		Rimes supv.	MAE	WID
IAM	→ IAM	–	7.39	7.28
IAM	→ Rimes	–	9.61	11.77
IAM & Rimes	→ Rimes	writer	6.65	6.68
IAM & Rimes	→ Rimes	unlabeled	9.66	9.55

that unlabeled samples from the target dataset can be included in its training. As discussed in Section 5.3.2, the inclusion in MAE training did not yield any improvements compared to training the MAE only on IAM. However, the MAE trained on both datasets performs on par with the WID-based SE. For the semi-supervised training without writer labels, we decided to not train the MAE on both datasets since this deteriorates the generation results (cf. Section 5.3.3). When training the MAE only on IAM, however, the performance is similar to that of the WID model.

In summary, the WID model learns features that are better suited for discriminative tasks. When employed as SE, the MAE and the WID model perform similar in almost all settings. While the expected improvement by including the target dataset in the MAE training has not been achieved, the MAE shows considerably better generalization to the target dataset when trained only on the source dataset.

5.3.5 *Content for Generation*

In the previous experiments, we focused on different aspects regarding the models in our HTG system and their hyperparameters. We have reproduced the instances from the training set by conditioning our model on the exact same pairs of the annotated word and writing style. While this allows for good comparability between different configurations, it is unrealistic in practical applications. For an unlabeled target dataset, neither the distribution of writing styles nor the distribution of words is known. Furthermore, the vocabulary is unknown in most cases. However, a careful choice of the word distribution is essential when training an HTR model, as the model learns an implicit language model. In the case of pretraining an HTR model on synthetic samples rendered from computer fonts, this impact has been highlighted by Wolf and Fink [WF24]. In our application, we have to choose a distribution of writing styles for generation in addition to the word distribution. The following experiments aim to evaluate the impact of word and style distributions for generation. As part of these experiments, we assess the generation performance under realistic assumptions with a minimum of knowledge about the target dataset.

To evaluate the importance of word and style distributions, we generate six different datasets. We use the reproduction of the training data from the target dataset as our baseline and refer to this setting as *annotation*. For the second dataset, we sample from the word and writer distributions from the target dataset independently. Thereby, we generate similar distributions but do not reproduce the exact pairs. To

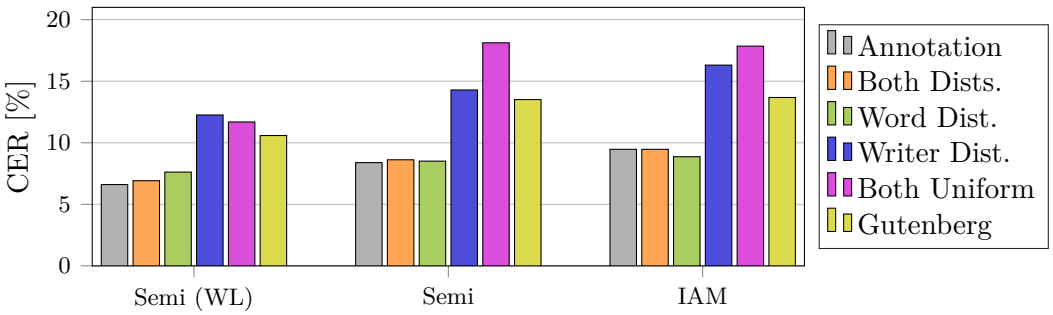


Figure 29: CER [%] on Rimes test of HTR models trained on generated datasets based on different distributions of words and writing styles.

evaluate the impact of the respective distributions individually, we generate pairs where either words or writing styles are sampled according to the original distribution. The other modality is sampled uniformly. Furthermore, we create a dataset without information about occurrences by sampling both modalities uniformly.

The last dataset is intended to represent a realistic setting in the practical application. We assume that we do not know the vocabulary of the target dataset but its language, which is French for Rimes. To obtain a word distribution similar to the one in the natural use of language, we sample words from a literature corpus [WF24]. Similar to the referenced work, we use the top 100 French ebooks from the project Gutenberg[5]. However, we apply the same restrictions (i. e. length of words, allowed characters) to the selection of words as described in Section 5.1. The filtered corpus comprises 5.2 million words from which 37 723 are unique considering capitalization (i. e. “Home” ≠ “home”). Writing styles are sampled uniformly from the target dataset. Since in a realistic setting, writer labels might not be available, we only compute one embedding vector per writer based on $\mathsf{K} = 10$ samples. Only for the model trained in a semi-supervised fashion with writer labels, we sample with 100 variations per writer.

In order to eliminate the influence of the number of samples generated, we sample the same number of pairs as provided in the annotation (34 468 for Rimes). We evaluate these settings using one model trained in a semi-supervised fashion with writer labels from Rimes, one without using these writer labels and one model trained only on IAM. The results are presented in Figure 29.

Training the HTR model on a dataset generated according to the word and writer distributions of the target dataset results in similar performance as for a dataset which is generated based on the annotation. When generating the dataset only based on the word distribution, the performance is even better for the semi-supervised model and the LDM trained only on IAM. Generating a dataset with words uniformly sampled from the vocabulary leads to significantly poorer results. This applies regardless of whether the writing styles are chosen according to the target dataset's distribution. With the generated dataset based on the Gutenberg corpus, a better CER is achieved. However, these results are considerably inferior to those using the word distribution from the target dataset. This highlights the importance of choosing an appropriate word distribution that is as close as possible to that of the target dataset.

5 https://www.gutenberg.org/, accessed: 10.01.2025.

Table 11: CER [%] on Rimes test for HTR models trained on generated training sets of different sizes based on the Gutenberg corpus. Most options are multiples of the size of the original training set with $|\mathbf{D}| = 34\,468$.

Model	Number of Generated Samples					
	$1 \cdot \lvert\mathbf{D}\rvert$	$2 \cdot \lvert\mathbf{D}\rvert$	$5 \cdot \lvert\mathbf{D}\rvert$	$10 \cdot \lvert\mathbf{D}\rvert$	$20 \cdot \lvert\mathbf{D}\rvert$	10^6
Semi (WL)	10.59	9.47	9.38	8.63	**8.31**	8.81
Semi	13.51	12.01	11.05	10.83	10.79	**10.35**
IAM only	13.68	12.66	12.15	**11.53**	12.10	11.95

Since the writers from the training and test partitions of Rimes are disjoint sets, there is no benefit to be expected by sticking to the writer distribution from the training partition. On the contrary, uniform sampling introduces more variation in the writing styles in the generated dataset. In theory, this should improve the generalization of the HTR model to different writing styles. However, such an improvement is not reflected in the results. We attribute the poor performance in these cases to the generation process. In contrast to rendering of computer fonts, our generation process is learned based on training images. The generation quality can therefore be expected to be better for writing styles which are well represented in the training dataset for the HTG model. A uniform generation of writing styles, however, requires a similar generation quality for less represented styles. Potential improvements by an increased variation are therefore expected to be partially compensated by a poorer generation quality.

Since HTG allows an arbitrary number of training samples to be generated, the question arises as to how many samples should be generated. A higher number of samples is expected to contain more variation, which is a desired property for any training dataset. However, the generation comes with computational costs. At the same time, our HTG system can be expected to have only learned an finite amount of variation in the generation process. Furthermore, the HTR model only offers a certain model capacity. Therefore, we expect the improvement by more generated training data to saturate. In addition to the generation of a dataset with the same size as the target dataset, we examine generated sets of multiples of the original size and a set of one million generated samples. To make a fair comparison between the different models, we generate the same pairs of words and writing styles for each model in each setting. We keep the number of iterations fixed to that when training on the original training set in order to only focus on the variation of the generated training sets. The results are reported in Table 11.

Increasing the number of generated samples helps for all models. However, a saturation of the improvements can be observed. Increasing the number of generated training images from $20 \cdot |\mathbf{D}|$ to one million even resulted in a slight decrease in performance for the semi-supervised model using writer labels. For the model trained only on IAM, the best performance is achieved when generating $10 \cdot |\mathbf{D}|$ samples. Only the semi-supervised model without the use of writer labels consistently improves for an increasing number of samples up to one million, albeit only slightly. Therefore, we

consider generating ten times the dataset size as the preferable choice as this number represents a good trade-off between generation effort and performance improvement.

5.3.6 *Application to Other Datasets*

So far, we always considered Rimes as our target dataset. A key question when it comes to whether the method could be used in practice is how well it can be applied to other target datasets. Therefore, we also examine the performance with CVL and GNHK as target datasets. Both datasets contain English handwriting, except for one German text in CVL. The difficulty of transferring to another language as with Rimes is not given. However, the datasets present different challenges. The training partition CVL contains significantly fewer samples (i. e. 10 067 instead of 34 468). Furthermore, the lexicon as well as the number of writers are very small. The GNHK database poses a different challenge. In contrast to the other datasets with flatbed-scanned images, it comprises unconstrained images which were captured by mobile phone cameras.

In our experiments, we consider the same settings as when using Rimes as the target dataset. We apply one model which has only been trained on the source dataset (i. e. IAM). Then we train models in a semi-supervised fashion with and without the use of writer labels. We use the models to generate the annotated word-writer pairs from the training partition of the respective target dataset. Additionally, we evaluate the more realistic setting of uniformly sampled writing styles combined with words from the Gutenberg corpus. Instead of the top 100 French ebooks, we use the English ones for CVL and GNHK and generate ten times the number of samples contained in the respective training partitions. For reference, we report two baselines. We train our HTR model using the real images from the training partition of the target dataset. In addition, we train an LDM with full supervision on the target training data in order to assess the generation performance without the challenge by the transfer to the target dataset. For training and sampling, we use the hyperparameters which we have identified to work best for Rimes. Although this choice of hyperparameters might lead to suboptimal results on the other datasets, we make this decision intentionally. We argue that there is usually no annotated validation data available to perform a parameter search.

Table 11a shows the results for applying the different models to CVL and GNHK. For comparison, we summarize the corresponding results from the previous experiments using Rimes. Besides the quantitative results in Table 11a, qualitative results for these target datasets are presented in the appendix in Section A.5. The baseline results from the recognition model trained on real samples confirm that CVL and GNHK pose new challenges compared to Rimes. Since CVL has almost no OOV words in its test set (cf. Table 11b), very good recognition performance can be expected. However, the performance is inferior compared to Rimes. We attribute this to the small number of writing styles in the training samples in comparison to the test data. For GNHK, the baseline recognition results are considerably worse which highlights the aforementioned challenge caused by the unconstrained nature of the handwriting images. The percentage decline in performance when training the HTR model on data generated by a LDM trained on the target dataset with full super-

Table 12: Results for the generation of training images for different target datasets with different levels of supervision.

(a) CER [%] on test sets from different benchmark datasets. The HTR models are trained on data generated by LDMs trained with/without transcriptions (Transcr.) and with/without writer IDs (Wr.). When words are sampled according to the Gutenberg corpus, we mark experiments with (GB).

Model	Transcr.	Wr.	Rimes	CVL	GNHK
HTR Baseline	✓		2.47	4.72	11.62
Full Sup.	✓	✓	3.42	6.80	14.61
Semi (WL)		✓	6.61	10.47	21.09
Semi			8.39	16.69	24.55
IAM only			9.61	22.53	41.91
Semi (WL, GB)		✓	8.63	21.01	19.93
Semi (GB)			10.83	27.69	23.26
IAM only (GB)			11.53	33.51	38.22

(b) Share of OOV samples [%] of the test partition compared to the training partition of the respective dataset. Additionally, the OOV with regard to a dataset with $10 \cdot |\mathbf{D}|$ words sampled from the Gutenberg corpus in the repsective language is reported.

Rimes		CVL		GNHK	
Train	Gutenberg	Train	Gutenberg	Train	Gutenberg
2.43	2.53	0.06	18.43	21.34	20.05

vision is comparable for all datasets. This demonstrates that our HTG system is generally able to generate the training images of the dataset with a good quality.

Across all models, it is evident that the generation of CVL and GNHK poses a greater challenge compared to Rimes. When training the LDMs without samples from the target dataset or with less supervision, the same observations as for the previous experiments can be made for all datasets. Generating samples with the model trained only on IAM does not yield satisfactory results. Including samples from the target dataset without any labels, however, boosts performance for CVL and GNHK. Adding writer labels to the semi-supervised training, helps to further improve the performance. These observations underline that adapting the generation to the styles in the target data set is very helpful, especially if the annotations of writer labels are provided.

The importance of the word distribution used for sampling becomes evident when comparing the results based on a reproduction of the training dataset and sampling based on the Gutenberg corpus. Especially for CVL, sampling based on the Gutenberg corpus leads to a significant decline in performance. For Rimes, performance

is only slightly worse and for GNHK the CER is even improved. The consideration of the OOV shares in Table 11b provides a possible explanation. The OOV share of the Rimes test partition with regard to its training partition and to the Gutenberg corpus is similar and performance sampling word from the Gutenberg corpus is only slightly inferior to sampling from annotation. In contrast, for CVL, the OOV share is several magnitudes higher for the Gutenberg corpus in comparison to the training partition of CVL. Furthermore, the dataset has a very small vocabulary and sampling based on the Gutenberg corpus introduces a high amount of irrelevant variation. Correspondingly, the results are considerably worse for CVL when sampling words from the Gutenberg corpus. For GNHK, however, the OOV share is a little higher when sampling from the training annotations. Accordingly, we observe a small improvement when sampling words from the Gutenberg corpus.

The comparison of the generation performance of the LDM trained only on IAM provides another interesting insight. The deterioration compared to models which use samples from the target dataset is considerable lower for Rimes than for GNHK and CVL. Unlike Rimes, the latter two datasets contain samples in the same language as IAM. This indicates, that the model generalizes well to another language with the same set of characters. It appears that the difference in the writing styles poses a greater challenge which emphasizes the benefits of incorporating styles into the training through our method.

5.3.7 *Self-Training*

Throughout all experiments, we observed a gap in the performance when the recognizer is trained on generated instead of real training samples. In order to further narrow this gap, we employ self-training as described in Section 4.5.3. Since our model as well as our datasets differ from those used in the reference [WF24], we first need to adjust the hyperparameters. We investigate the choice of the number of self-training cycles, the learning rate as well as the confidence measure and a suitable confidence threshold. We conduct the parameter search using an initial model trained on images rendered from computer fonts which is then adapted to the IAM dataset. By using our source dataset for these experiments, we can use labels for the optimization of hyperparameters and do not require annotated samples from our target datasets. On the contrary, the choice of hyperparameters might not be optimal for the application using an HTR model trained on samples for the other datasets or on generated samples. However, we consider this approach more suitable for practical applications.

The training data for the initial model is acquired similar to [WF24]. We use the same 396 fonts and render one million images using the Gutenberg corpus. However, we apply our restrictions and consider only words with 2–7 Latin characters. We select the language according to the target dataset which is French in the case of Rimes and English for the remaining datasets. The HTR model is trained for the same number of iterations as for training on the real training images of the respective dataset. Test performance is reported for the recognition of real images.

First we examine different options for the confidence measure as discussed in Section 4.5.3. In addition to the decoding confidence of the best path c_{best}, the CTC loss c_{loss} and its logarithm c_{logloss}, we consider the CER of the prediction. In practice,

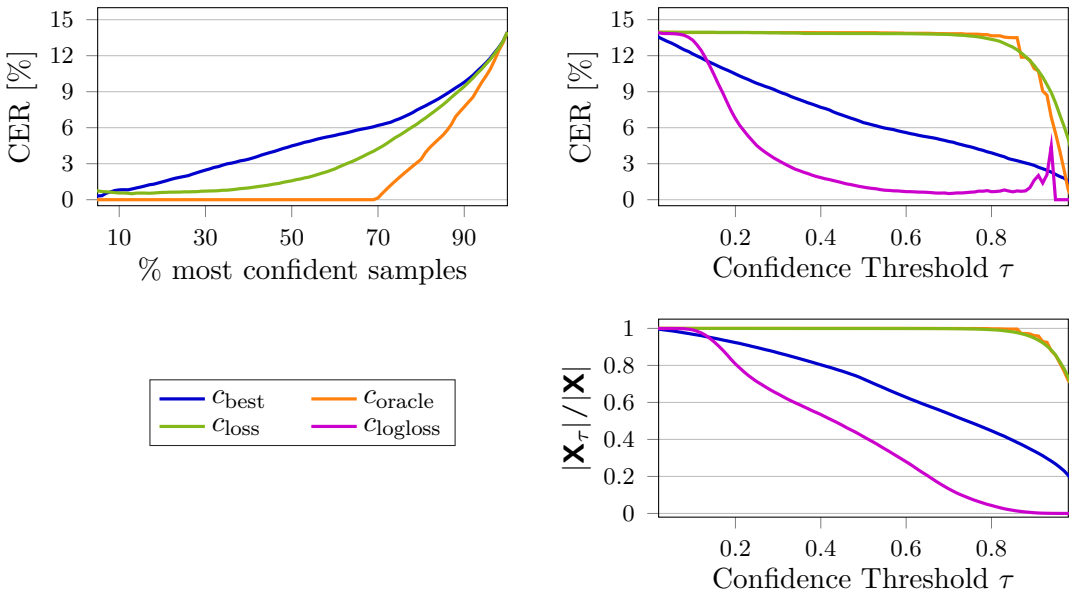


Figure 30: Effects of the different confidence measures on the CER on IAM train images. The top left plot displays the CER for different shares of the dataset when the samples are sorted according to their confidence estimates. As the logarithm is a strictly monotone transformation, the results for the CTC loss and its logarithm are identical in this case. The top right plot displays the CER when only samples with $c_\theta > \tau$ are selected. The bottom right plot shows the portion of the original dataset which remains after confidence thresholding is applied. $|\mathbf{X}|$ and $|\mathbf{X}_\tau|$ denote the size of the training dataset before and after confidence thresholding, respectively.

the latter cannot be computed as labels are unknown. However, for our experiments it serves as an oracle for the best possible selection and is therefore denoted as c_{oracle}. In order to compare the confidence measures, we normalize the values on the evaluation dataset to $[0, 1]$. Instead of directly using c_θ, we compute $1 - c_\theta$ for the loss- and CER-based measures to obtain a consistent sorting where higher values are expected to correspond to better predictions.

To determine whether the confidence measures allow a good ranking of the predictions, we use the initial model to predict labels for the IAM training samples. We order the predictions by the confidence estimates and report the CER for different portions of the dataset. The results are shown in the left plot of Figure 30. For all confidence measures, the CER increases as more samples with lower confidence are considered. This indicates that all proposed measures provide an estimate of the correctness of the predictions. However, considering all decoding paths by the *connectionist temporal classification* (CTC) loss more closely resembles the curve of the oracle confidence measure. This becomes even more apparent when measuring the CER only for samples with a confidence higher than some threshold τ. Respective results are visualized in the top right plot of Figure 30. Here, however, a problem becomes apparent when using the CTC loss as an estimate of the confidence. While closely resembling the curve of the oracle confidence, low values of the CER are only obtained for very high thresholds. At the same time, for the size of the confidence filtered datasets rapidly decreases for high thresholds (see Figure 30 bottom right).

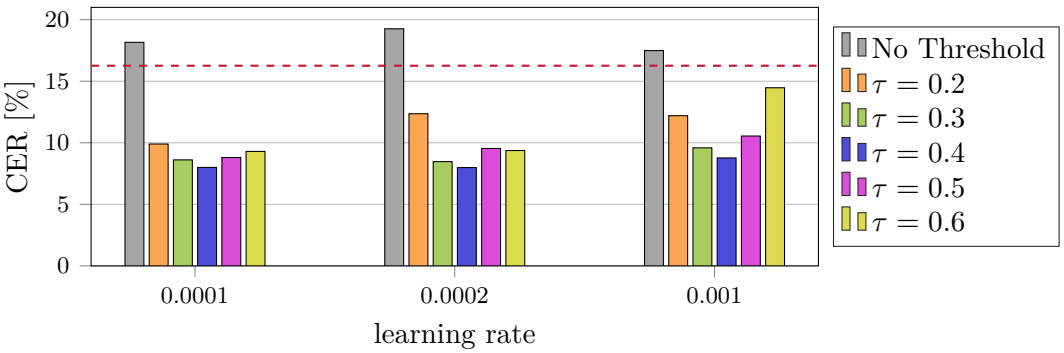


Figure 31: CER on IAM test after self-training on IAM train for 50 cycles with different learning rates and confidence thresholds τ. The red dashed line indicates the performance of the initial model before self-training.

The steep gradient for both confidence measures reveals that the results react very sensitively to the choice of the threshold. The logarithmization of the loss results in a significantly flatter slope. Good results are achieved for a much wider range of threshold values, which allows a more robust selection of the parameter. We therefore choose to use the logarithm of the CTC loss as a confidence estimate for the following experiments on self-training.

After the selection of a confidence measure for our experiments, the next step is the choice of the remaining hyperparameters. For the number of cycles, we follow Wolf and Fink [WF24] and run self-training for 50 cycles using unlabeled images from IAM train. We test the model every ten cycles on IAM test. Besides the learning rate used in our other experiments (i. e. 0.001), we explore one fifth and one tenth of this learning rate. Inspired by Retsinas et al. [RSN21], we also experiment with a learning rate of 0.005. Based on Figure 30, we explore confidence thresholds $\tau \in 0.2, 0.3, 0.4, 0.5, 0.6$. We do not consider $\tau > 0.6$, since no improvements to the CER can be observed and the remaining sample size becomes very small.

As the best results were obtained after 50 cycles of self-training when using a confidence threshold, we only report these results in Figure 31. A more detailed evaluation after every 10 cycles is presented in the appendix in Table 19. Furthermore, we omit the results for a learning rate of 0.005 as they are considerably inferior. The experiments clearly show the effectiveness of filtering the pseudo-labels based on a confidence threshold. It is important to note that self-training without confidence thresholding diverged and better results are obtained with fewer cycles. However, these results are still not competitive compared to the experiments with thresholding (cf. Table 19). All runs using thresholding lead to performance improvements after 50 cycles compared to the initial model. Comparing the different thresholds, the best results are achieved with $\tau = 0.4$ for all learning rates. The fact that the results for $\tau = 0.3$ and $\tau = 0.5$ are only slightly worse indicates the robustness of the self-training with respect to the exact value of τ. The same applies to the learning rate. The results with learning rate of 0.0002 are similar to those with learning rate 0.0001 but we assume it to be a more robust choice for further experiments. Therefore we will run self-training in the following experiments for 50 cycles with a learning rate of 0.0002 and only consider pseudo-labels with a confidence greater 0.4.

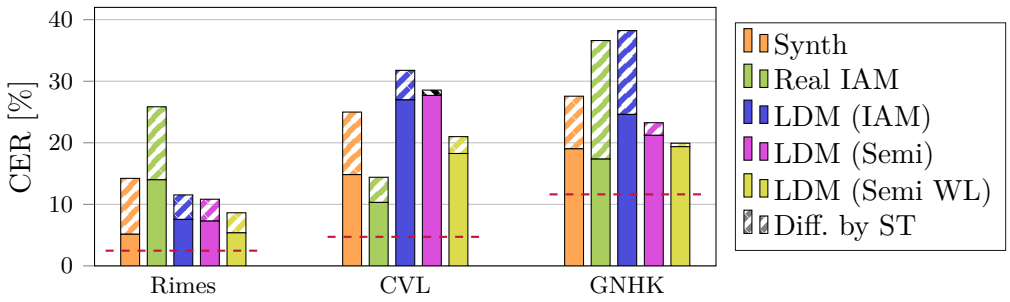


Figure 32: Improvements by self-training on different datasets. The initial models are obtained by training on word images which are rendered from computer fonts or generated by our LDMs (indicated by the color of the bars). The hatched area marks the difference in CER compared to the initial model before self-training (black marks deterioration by self-training). The red dashed lines indicate the performance of when training the HTR model on the annotated, real samples from the respective training partitions.

With the determined configuration of hyperparameters we then conduct experiments on self-training of models trained on images generated by our LDMs. For comparison, we also employ self-training on HTR models which are initially trained on images rendered from computer fonts. In all cases, we sample words according to their frequency in the Gutenberg corpus. When generating images with our LDMs, we uniformly pick the writers. For the model trained with writer labels, we use 100 variations of the embeddings for each writer during sampling. For LDMs which do not require writer labels, we only use one embedding per writer. As another baseline, we report results for an initial model which has been trained on the annotated, real training samples from IAM. The results are illustrated in Figure 32.

Regarding the performance of the initial models, we observe the same ranking of our LDMs as in Section 5.3.6. The worst performances are obtained by the model which has been trained only on IAM. Semi-supervised training improves the performance, especially if writer labels are used. In comparison to an initial model trained on rendered images, however, only the LDMs trained in a semi-supervised fashion with writer labels achieve a better CER throughout all datasets. Only for Rimes, all initial models trained on generated samples have a better CER than the model trained on rendered images. However, for GNHK, using generated images from the LDM trained only with samples from IAM results in inferior performance compared to rendering word images. For CVL this is even the case for both models, the one trained on IAM and the LDMs trained without writer labels for the target dataset. This shows, that our expectation of considerably superior performance when the initial models are trained on LDM-generated samples is not met in all cases. Again, our proposed inclusion of writer labels is essential for a substantial improvement in comparison to the rendered images. The initial model obtained by training on real samples from IAM achieves the best results for CVL and competitive results for GNHK. Only for Rimes, results are significantly inferior to the other approaches. This can be explained by the difference in language between IAM and Rimes which does not cause a problem for generated samples because we adapt our vocabulary to the matching language.

Applying self-training to the different initial models leads to considerable changes in their ranking. Despite the lower CER on Rimes of the initial models trained on generated samples, the model trained on rendered images achieves the best results after self-training. Only the model initially trained on samples generated by the LDM using writer labels achieves similar performance. An interesting observation is that better initial models have experienced a more modest enhancement through self-training, despite maintaining a discrepancy in performance compared to the fully supervised baseline. This phenomenon is especially evident when examining the results with generated images from GNHK. For CVL, we even observe an increase in CER by self-training when training the initial model on images generated by the semi-supervised LDM. Self-training for the other initial models from generated samples results only in small improvements.

For CVL and GNHK, which share the same language with IAM, the initial model trained on real images from IAM achieves the best results after self-training. While already having a good performance before self-training on CVL, the same initial model has a high CER for GNHK. The initial models trained on rendered images achieve slightly worse results after self-training. In contrast, all initial models trained on generated images achieve inferior results after self-training. Only the LDM trained with writer labels achieves performance similar to using rendered images on CVL and GNHK. In particular the results for GNHK show that better initial performance does not guarantee a better model after self-training.

One potential explanation for the enhanced performance of the initial model trained on rendered images is, that it learned more general and robust features, which are more suitable for the self-training process. Another possible reason is that our choice of hyperparameters does not generalize well to initial models trained on generated and possibly noisy samples. As the initial model is already fitted to the target dataset, it may be advantageous to adjust the learning rate, confidence threshold, and number of self-training cycles. Intermediate results during the self-training process (see Table 20 in the appendix) indicate that the number of cycles should be adapted when using models not trained on rendered images. Since the primary focus of this work is not on self-training, we leave further investigations on hyperparameters and the learning process during self-training for future work.

Overall, improvements were achieved through self-training in all but one of these experiments. Thereby, the gap to training on real, annotated images from the target dataset was significantly reduced in many cases. Despite providing good initial performance, models trained on samples generated by our LDMs benefit less from self-training. Consequently, in order to leverage the potential of HTR models trained on generated samples, it is essential to gain deeper insights into the self-training process of these models and to identify a more suitable choice of hyperparameters.

5.4 GENERATION PERFORMANCE OF STATE-OF-THE-ART HTG MODELS

In the previous experiments, we explored and evaluated our proposed HTG system in detail. We employed various configurations of our framework and applied it to generate training data for different datasets. However, we restricted the word images considered in these experiments based on the character set and word length. For practical application, it is crucial to ensure a high generation quality regardless of these

restrictions. Therefore, we conduct another series of experiments, where we employ our HTG system to generate training data without these restrictions (see Section 5.1). Besides our approach, we apply other works from the literature to address the same tasks. As part of these experiments, we assess the performance of these models in different generation scenarios to obtain insights into the strengths and limitations of state-of-the-art HTG methods. For a more comprehensive evaluation, we consider GAN-based models [Gan+22; Van+25] as well as an LDM-based model [Nik+24a]. HiGAN+ [Gan+22] and Vatr++ [Van+25] are recent approaches based on *generative adversarial networks* (GANs) which achieve state-of-the-art performance in HTG. For a comparison to a LDM-based model, we consider DiffusionPen [Nik+24a]. This model is closely related to our approach and was developed independently and concurrently with our approach as an improvement of Wordstylist [Nik+23]. However, differences in the style encoding and the training procedure (i. e. no inclusion of unlabeled samples from the target dataset) make the model an interesting candidate for comparison. For all three models, we mainly used the provided reference implementations[6] for our experiments.

In this series of experiments, we first explore the applicability of our model to training data generation for different target datasets without limitations and compare the performance to DiffusionPen, HiGAN+ and Vatr++ in Section 5.4.1. To gain a better understanding of the capabilities of these models, we assess the correctness of the generated samples for different generation tasks in Section 5.4.2. In Section 5.4.3, we use the models in a realistic application scenario. With different portions of the annotations of the datasets being given, we indentify promising application cases for using HTG for training data generation.

5.4.1 *Training Data Generation*

Given that our model has only been applied to datasets with restrictions on word length and the character set (see Section 5.1), the subsequent analysis will focus on the impact of removing these restrictions on performance. Consequently, we follow the same experimental setup as described in Section 5.3.6. We train one model only using the samples from IAM, one model including unlabeled samples from the respective target dataset in the training and another model using target samples with their corresponding writer ID. To assess the impact of changing the set of samples independently of the generation process, we report the performance of the HTR models trained on real samples. In addition to the performance on the target datasets (i. e. Rimes, CVL and GNHK), we also evaluate the performance on the source dataset (i. e. IAM). We report the results in Table 12. Although an increased complexity for recognition can be expected due to longer words and more characters, the HTR model performs better if the datasets are not restricted. The only exception is the GNHK database where the CER is slightly better when using the limited dataset.

When training the LDM only on IAM, a small improvement can be observed for all datasets except GNHK. The LDMs which have been trained in a semi-supervised

6 DiffusionPen: https://github.com/koninik/DiffusionPen at commit `059e2f2`;
HiGAN+: https://github.com/ganji15/HiGANplus at commit `5d37de7`;
Vatr++: https://github.com/EDM-Research/VATr-pp at commit `a32d6fd`.

Table 12: Comparison of CER [%] on test sets from different benchmark datasets. For training and testing both, the LDMs and the HTR model, datasets with and without limitations are considered. For reference, we report the baseline results when the HTR model is trained on real labeled samples. We do not report performance on IAM for the semi-supervised models since one model is trained for each target dataset.

Model	Limited	IAM	Rimes	CVL	GNHK
HTR Baseline	✓	6.01	2.47	4.72	11.62
		4.64	1.85	3.80	12.77
IAM only	✓	7.39	9.61	22.53	41.91
		6.64	9.01	19.72	53.36
Semi	✓	–	8.39	16.69	24.55
		–	8.90	16.71	33.29
Semi (WL)	✓	–	6.61	10.47	21.09
		–	7.08	11.23	27.80

fashion perform on par or only slightly worse when applied to the unrestricted versions of CVL and Rimes. Only when training the recognizer on generated GNHK samples, results are considerably worse. Since the increase in vocabulary size is comparable to that of Rimes, we exclude this as a reason. Instead, we assume that the unconstrained nature of the GNHK samples leads in a significant increase in variation of the word images when omitting the restrictions thereby complicating the recognition. Overall, the results demonstrate that our model is capable of generating samples in a similar quality for the full datasets as well.

As a next step, we train HiGAN+, Vatr++ and DiffusionPen on IAM and use these models to generate training images for the target datasets. In accordance with the preceding experiments, we replicate the pairs from the original training partitions for a fair comparison of the different works. The results are presented in Table 13.

For the generation of the source dataset, our approach and DiffusionPen clearly outperform both GAN-based approaches. When applied to the generation of Rimes and CVL, however, the latter methods achieve better results compared to DiffusionPen. Our model trained only on IAM achieves considerably better results for Rimes but is only slightly better than HiGAN+ when applied to CVL. When generating images from GNHK, all of the other works achieve only poor results. For GNHK, results using our model trained only on IAM are better than those of the other methods but can still be considered very poor with a CER $> 50\,\%$. Including unlabeled samples from the target datasets in the training, leads to a considerable improvement when generating images for CVL and GNHK. For Rimes, only a minimal improvement can be observed. This improvement can be amplified by including writer labels from the target dataset. However, these results are not directly comparable to the remaining models as those were trained without any labels from the target dataset.

Table 13: CER [%] on test sets from different benchmark datasets. The HTR models are trained on samples generated by different HTG models. For reference, we report the baseline results when the HTR model is trained on real labeled samples. No annotations from the target datasets are used for training the HTG models. The only exception is our semi-supervised model that requires writer labels from the target datasets (marked with *).

Model	IAM	Rimes	CVL	GNHK
HTR Baseline	4.64	1.85	3.80	12.77
HiGAN+ [Gan+22]	16.63	24.53	21.72	65.52
Vatr++ [Van+25]	17.22	21.31	24.96	69.92
DiffusionPen [Nik+24a]	7.58	28.44	34.14	62.21
Ours (IAM only)	**6.64**	9.01	19.72	53.36
Ours (Semi)	–	**8.90**	**16.71**	**33.29**
Ours (Semi WL*)	–	7.08	11.23	27.80

In contrast to the suboptimal performance observed in our experimental setup, in the respective works for both GAN-based models, samples with high quality have been presented. However, the authors employed different evaluation measures (e. g. recognition of generated word images instead using them for training). We assume that the assumption of characters having a width of half the number of pixels as their height introduces a bias which is harmful when using the generated samples for training a downstream model. A further investigation regarding the generation quality of these two models will be conducted in the following section.

5.4.2 *Correctness of Generated Content*

For a more comprehensive comparison of the different models, we use the models from the previous experiments to generate defined evaluation sets (see Section 5.1). In order to assess the generalization capabilities regarding unseen words, we differentiate between IV and OOV words. At the same time, we evaluate the correctness of the generated writing style for styles seen during training and new styles. As a manual inspection is infeasible on a larger scale, we employ handwritten text recognition (HTR) and writer identification (WID) models which have been trained on real samples. We then compare their recognition performance on the generated samples to the reference performance on real word images with the same pairs of text and style as described in Section 5.2.

Since the generative models have been trained on IAM with annotations, examples for all four cases (i. e. IV-S, OOV-S, IV-U and OOV-U) are available for IAM. For the remaining three datasets, only IV-U and OOV-U can be defined since we consider only annotated samples as “seen” during training. We report the results for IAM in Table 13a.

Table 14: Difference in percentage points in CER and WIER of HTR and WID models between real and generated word images. The models used for evaluation are trained on real samples from IAM with the samples for evaluation held out.

(a) Seen Writing Styles

Model	IV-S		OOV-S	
	CER	WIER	CER	WIER
Reference	1.56	9.28	4.41	5.94
HiGAN+ [Gan+22]	−1.13	+53.86	− 2.47	+47.86
Vatr++ [Van+25]	−1.14	+79.11	− 2.18	+79.24
DiffusionPen [Nik+24	+7.33	+50.45	+16.91	+55.15
Ours	+1.82	+11.80	+ 6.79	+22.20

(b) Unseen Writing Styles

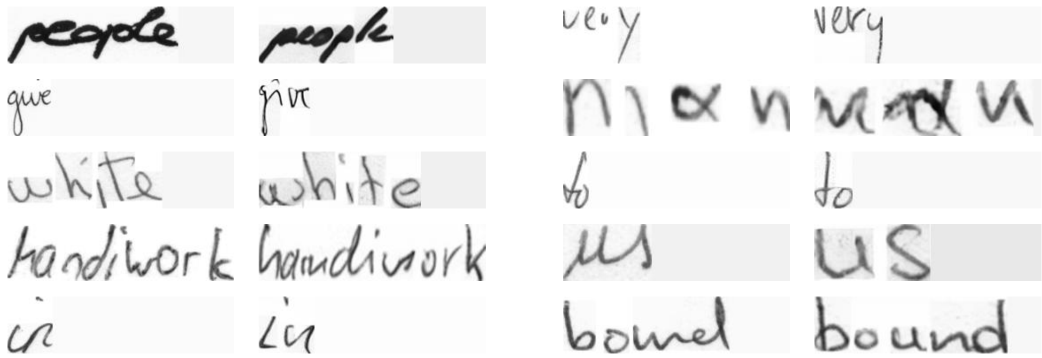
Model	IV-U		OOV-U	
	CER	WIER	CER	WIER
Reference	2.23	14.77	6.14	12.69
HiGAN+ [Gan+22]	−1.89	+65.26	− 4.16	+70.91
Vatr++ [Van+25]	−1.93	+80.53	− 3.69	+83.91
DiffusionPen [Nik+24	+3.75	+84.03	+10.96	+86.50
Ours	+0.43	+58.80	+ 4.11	+68.51

IV-S can be considered the easiest case, as the models neither have to generate new words nor unseen styles. Our model as well as HiGAN+ and Vatr++ achieve a CER close to the reference. However, the results differ in their sign. The positive difference of our model indicates that our generated samples are slightly less readable than the original samples. In contrast, the negative difference suggests that the generated word images are not as challenging for the recognizer as the real examples. This may be an indication of limited variation of the generated samples, for example, mostly generating writing styles which are simpler to recognize. In light of the suboptimal outcomes observed in the preceding experiments with HiGAN+ and Vatr++, the findings offer a potential explanation. While generating correctly spelled and well readable samples, the variation is too limited for training a robust HTR model. The differences in WIER support this assumption. Only our model produces word images where the WID model can still classify the majority of samples correctly.

The generation of samples from the OOV-S set enables the assessment of whether the models have acquired the capability to combine characters in configurations that were not observed during the training process. While the reference HTR model per-

Table 14: Comparison of real and generated IV-U examples where the writer could not be indentified corretly for the generated image.

Real	Generated	Real	Generated



forms slightly worse, the WID model achieves a lower identification error. The results for the generated images in this case show similar proportions to those observed in the IV-S scenario for both evaluation measures. However, all differences are larger which can be intuitively explained by the increased difficulty in OOV generation. Only for our model the increase in difference (around factor 3) is larger than for the other models. In the context of our experimental setup, all models can be considered capable of generating OOV samples. However, with the increased challenge in the generation task, our model is not able to maintain its good imitation performance from the IV-S setting.

Despite the generation of OOV words, a desired capability of the models is the generation and imitation of new writing styles. The CER in Table 13b shows that the generated word images are well recognizable by the HTR model. At the same time, the WID model is not able to identify the correct style in almost all cases. To get an idea whether the generative model only misses important details or completely fails to mimic the writing style, we conduct a brief qualitative evaluation. From samples where the WID model correctly identified the writer for the real samples, we randomly draw ten examples with false writer predictions for the generated images. As can be observed in Table 14, most words are written in a style which is close to the target style. Stroke width and background texture are reproduced well which creates the initial impression that the handwriting has been well imitated. However, a thorough examination reveals that the characters are written differently. In contrast to the good matching overall impression, these details are the characteristics that depend on the writer themself and not on the pen and paper used. This suggests the need for improvements in future models to more accurately replicate the visual characteristics specific for handwriting. However, it should be noted that this conclusion is initially based only on the visual inspection of a few examples, but is supported by the evaluation based on the WID model.

As a next step, we follow the same procedure to assess the performance for the generation of samples for the three target datasets considered in our experiments. The generation in these cases is more challenging in two aspects. First, only unlabeled samples from the target datasets are used during training. Second, at the same time, this means that only the more difficult cases of generating unseen writing styles can

Table 14: Difference in percentage points in CER of HTR models between real and generated word images. The HTR models used for evaluation are trained on real samples from the respective dataset with the samples for evaluation held out. The only HTG model using some annotations from the target dataset is our semi-supervised model that requires writer labels from the target dataset (marked with *).

Model	IV-U			OOV-U		
	Rimes	CVL	GNHK	Rimes	CVL	GNHK
Reference	1.20	1.28	7.40	1.34	0.57	14.75
HiGAN+ [Gan+22]	−0.43	− 1.14	− 3.93	− 0.03	− 0.47	− 0.25
Vatr++ [Van+25]	−0.40	− 0.97	− 4.87	+ 0.02	− 0.41	+ 1.71
DiffusionPen [Nik+	+1.95	+ 0.02	+ 2.47	+ 5.14	+ 1.11	+10.89
Ours	+3.18	+ 6.74	+26.94	+ 7.97	+ 7.36	+39.42
Ours (Semi)	+6.45	+28.17	+24.44	+10.93	+15.75	+31.80
Ours (Semi WL*)	+0.89	+ 0.39	+ 3.44	+ 3.87	+ 1.45	+12.71

be considered for evaluation. In analogy to the generation of unseen writing styles for IAM, extremely poor results in the WIER (i. e. $> 80\,\%$) were also achieved in these cases. Therefore, we only report the CER of the HTR models in Table 14.

Except for two cases, Vatr++ and HiGAN+ generate images resulting in a CER lower than the reference. Regarding the readability of the generated word images, a lower CER is favorable. Assuming that a CER lower than the reference value indicates an overfitting, large negative values are also not desirable. Therefore, different signs in the CER-difference might be weighted differently depending on the preferred properties of generated images. For results with the same sign a distance close to zero is clearly regarded superior. Therefore, the results of HiGAN+ and Vatr++ cannot be directly rated as better or worse than the results from DiffusionPen and our models. However when focussing on readability, HiGAN+ and Vatr++ achieve the is best results in all cases. Furthermore, as the absolute value of the respective results is very small, it can be assumed that the overall generation quality is also better compared to DiffusionPen and our model (except for Semi WL). With the same sign for the difference in the CER and a lower value, DiffusionPen also generates images that are more readable than images from our models. Only if we include examples with writer labels from the target dataset in the training of our model, we achieve a comparable performance to DiffusionPen, which does not require these labels. When comparing our semi-supervised model with the model trained only on IAM, the latter achieves considerably better results except for GNHK.

These results are in contradiction to the observed trend when the generated images are used to train an HTR model (cf. Table 13). In the case of training data generation, including examples from the target dataset has been beneficial in all cases even if no writer labels were used. These findings indicate that better readability of the generative samples (even by a model trained for the respective dataset), does not

provide any guarantees regarding the suitability of the generated samples for training an HTR model. Intuitively, this can be explained as good recognition results by a trained HTR model only require a few or even only one well readable writing style. In contrast, a good training dataset is expected to be representative and to contain as much variation from the target distribution as possible. This underlines the challenge in the evaluation of generative models and their results.

In conclusion, all models mostly generate readable and correctly spelled words when applied to IAM but miss the details of the respective handwriting styles. While achieving better performance on IAM, for the different target datasets our model produces inferior results compared to the other approaches. This difference in the ranking compared to the experiments on training data generation suggest the careful selection of an evaluation measure depending on the desired purpose of the generated samples.

5.4.3 *Application*

After the evaluation of several different aspects of the models with different approaches to evaluate their performance, we use them in a realistic setting for the task of training data generation. In contrast to the experiments conducted as part of Section 5.4.1, this means that we do not sample based on the target datasets annotations. Instead, we generate words from the Gutenberg corpus in random combination with styles from the writers of the target datasets training partition. In Section 5.3.6, we have observed that this leads to a deterioration in the results of the downstream HTR model. This raises the question of how useful the generation of training data is for a practical application. To answer this, we explore how much the performance of a recognizer trained on the source dataset can be improved by generating training data for the target dataset. We investigate the impact for different given portions of annotated data from the target dataset in order to better identify the scenarios in which the approach is worthwhile.

While the ultimate goal is to use no annotations from the target dataset, a small amount of annotated data might be available. For example, we require that a few samples per writer from the target dataset are collected in order to compute at least one embedding for that writer for sampling. Even if the additional annotation with transcriptions means a significantly higher effort, a small amount of samples might be transcribed as part of this procedure. Despite this process, a small annotated set of images from the target dataset might be available for different reasons.

As a baseline, we train an HTR model solely with the available annotated samples which is IAM and the respective small labeled subset from the target dataset. We use the same combinations of IAM and the annotated subset of the target dataset for training the HTG models. Since our semi-supervised training allows for including unlabeled samples, the training set for our models is augmented by the remaining unlabeled images from the target dataset. When training the HTR for the target dataset, we combine the real samples from IAM and the subset from the target dataset with a set of our generated samples. Following our findings from Section 5.3.5, we generate ten times the number of training samples from the respective target dataset. We train all HTR models for the same number of iterations as if exclusively training on the full target dataset. While this training setup might not yield the best

possible results, it provides a fair comparison of the different models for the different datasets. Since the investigation of various alternative training setups is beyond the focus of this work, we leave this optimization for future research.

The results are presented in Figure 33. We only explore subsets up to 25 % of the target datasets size since the baseline HTR performance is already close to the performance of the recognizer trained on 100 % of the target datasets samples. The generation of training data could also be used to augment larger annotated subsets up to 100 % of the target datasets. However, if aiming to improve upon the performance of HTR models trained on the complete target dataset, more sophisticated augmentation and training strategies are required. This includes adjusting the number of training iterations, balancing of the different data sources (i. e. source dataset, target dataset, generated samples), and the selection of content to be generated specific to the word and style distributions of the target dataset (e. g. augmentation of rare words, characters or styles). As this thesis focuses on the generation of training data for datasets where no or only very few annotated samples are available, considering larger annotated shares from the target dataset is beyond the scope of this thesis.

For Rimes, we can observe large improvements by including generated samples in HTR training when no annotated samples from Rimes are available. Vatr++, HiGAN+ and DiffusionPen achieve improvements of 5 – 6 percentage points. When using our model, the improvement is even higher with around 9 percentage points. In the setting, where 1 % of the annotated samples from Rimes are provided, performance is still improved by including generated samples. However, the improvement is smaller (around 3 percentage points). Increasing the amount of annotated samples from Rimes to 5 % further reduces the improvements to < 1 percentage point. This trend of diminishing returns continues for larger subsets of annotated samples from Rimes. Thereby, only minimal improvements can be observed by augmenting the training data for the recognizer with generated samples.

The experiments using CVL result in other findings. First, including 1 % of the annotated samples from CVL in the training of the baseline HTR model leads to a decrease in performance. We assume that the small set of samples with a different appearance and partially different language (i. e. German) introduces noise in the training of the HTR model. This is the only setting with CVL, where considerable improvements are obtained when including generated samples. When using 5 % of the samples, our model is the only approach with a minimal improvement, which is however negligible. In all remaining cases, results are better if the HTR model is only trained using IAM and the available real annotated samples from CVL. We assume, that the comparably small number of samples provided by the subsets of CVL is not sufficient for the HTG models to learn a good generation of the writing styles. At the same time, the recognizer, which was only trained on annotated samples of IAM, generalizes comparatively well to the recognition of the CVL test images and thus reduces the room for further improvement.

Like in the previous experiments, results for GNHK are significantly worse. Similar to the experiments on CVL, including only 1 % of the annotated samples deteriorates the performance of the baseline model. In this case, no samples with a different language are introduced but the difference in the writing styles is much larger. However, in both cases (0 % and 1 %), using our model for generation of training samples leads to a better CER. When no annotated samples from the target dataset are avail-

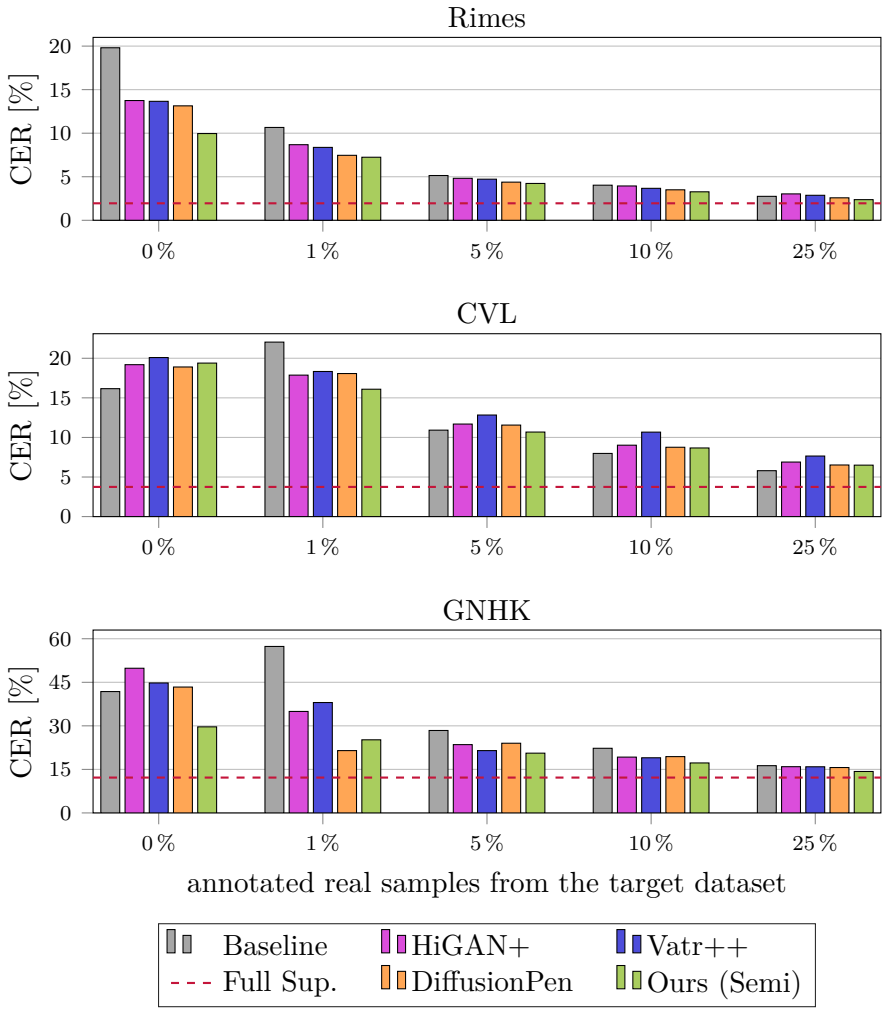


Figure 33: CER [%] on test sets from different benchmark datasets when including small annotated subsets from the target dataset in training the HTG and the HTR models.

able, using generated samples from other models results in an inferior CER. With the 1 % subset provided, generating samples with all models considerably improves the results with up to $>$ 30 percentage points in the case of DiffusionPen. The remaining results show a similar trend to that observed for Rimes. The baseline continuously improves with more annotated samples being available. At the same time, the improvements by including generated samples in the training of the HTR model shrink.

In summary, the results show that including generated samples into the training of an HTR model helps improving the recognition performance on the target dataset. Especially for no or only few provided samples with annotations from the target dataset, CER can be considerably reduced. Considering the computational costs for training the HTG models and sampling the training images, training data generation cannot be considered worthwhile in cases where larger annotated portions from the target dataset are available. Furthermore, the results on using the small CVL dataset suggest, that the application of HTG models is not straight forward and may require adaptations depending on the target dataset.

6 CONCLUSIONS

This chapter provides a summary of this thesis and an outlook on future research. In Section 6.1, the contributions of this work are summarized and the main findings from the experimental evaluation are highlighted. The limitations of the presented method are discussed in Section 6.2 and directions for future research are outlined.

6.1 SUMMARY

In this thesis, a method for training data generation based on a model for *handwritten text generation* (HTG) was proposed. The primary objective was to generate data for training a *handwritten text recognition* (HTR) model for a new target dataset without annotations. This work demonstrated that an HTG model trained with a dataset other than the target dataset can successfully be used for this purpose. Furthermore, the performance of the downstream HTR model can be considerably improved by including unlabeled samples from the target dataset following the proposed semi-supervised training schemes for the HTG model.

The basis for the generation of training data is the proposed HTG model. By choosing a latent diffusion model, the computational efficiency is improved. The model presented in this thesis extends an existing model from the literature, thereby enabling the generation of new writers, which is crucial when generating training data for new datasets. Therefore, a style encoder is presented which is implemented by a *masked autoencoder* (MAE). The MAE allows training on unlabeled data thereby not limiting the required training material to labeled samples.

Another modification made to the HTG model, is the content encoder providing various methods to entangle the text and style embeddings. In the experiments, only minor differences in the performance were observed when comparing the different choices for the style inclusion, indicating that the model is able to learn the incorporation of style information in various ways. However, a more significant impact on the performance was observed when using *classifier-free guidance* (CFG). While satisfactory outcomes were achieved without guidance, these could be enhanced by CFG. For the source dataset, the implementation of guided sampling with a guidance scale ≥ 2 is essential after training the model with CFG. When images are sampled for the target dataset, a careful adjustment of the guidance scale is necessary, depending on the inclusion of samples from the target dataset in the training of the HTG model.

For the generation of training data for a novel target dataset, different settings were considered regarding the available data and the respective annotations. It was shown that after training the HTG model only with the source dataset, the model is capable of generating samples that can successfully be used as training data for the target dataset. However, the gap in performance of an HTR model trained on the generated images compared to a model trained on real samples is larger for the target dataset than for the source dataset. Based on the assumption that the performance gap is caused by differences in writing styles between both dataset, two

semi-supervised training schemes were proposed. In order to incorporate information about the handwriting styles from the target dataset, images from that dataset only labeled with writer IDs are included in the training of the HTG model. In the experiments, a considerable increase in the performance was observed when the HTR model was trained on samples that were generated by an HTG model trained in this semi-supervised fashion. To enable the integration of examples from the target dataset without any labels, the semi-supervised training method was modified. Although the results are not competitive with results obtained when using writer IDs for the target dataset, the experiments showed that better results are achieved compared to training exclusively on the source dataset.

The generation of training data for a target dataset without available annotations gives rise to the question of which content (i. e. words and styles) should be used. In addressing this question, this thesis demonstrated that the distribution of the generated words is of particular significance. Given the absence of this information in the case of a target dataset without annotations, a text corpus of the corresponding language can be used for this purpose instead. The adoption of word frequencies in the natural language has been demonstrated to yield superior outcomes in comparison to a uniformly distributed sampling of words from the specific vocabulary of the target dataset. Furthermore, it was shown that the impact of sampling the styles uniformly or according to their frequencies in the target dataset is negligible.

In all experiments, the English IAM dataset was used as the source dataset and the French dataset Rimes served as the target dataset. In order to demonstrate the applicability of the HTG method presented in this work to other target datasets, experiments using CVL and GNHK as target datasets were conducted. Although both datasets contain mostly English samples, they comprise different writing styles, particularly with regard to the unconstrained nature of the handwriting images from GNHK. When generating training data for these three target datasets, the results underline the that differences in writing style pose a major challenge and show the benefits of the proposed semi-supervised adaptation. The generation of samples from another language, however, does not pose a problem when limiting the character set to that of the source dataset.

As there remains a gap in the performance of the HTR models trained on generated samples to those trained on real samples, self-training was implemented to narrow this gap. The self-training method used was originally presented for adapting an HTR model trained on word images rendered from computer fonts to images of real handwriting. Applying the self-training procedure to the HTR models trained on generated samples helped to significantly reduce the performance gap in many cases. However, while the HTR models trained on generated samples provided a better initial performance than the models trained on rendered images, they did benefit less from self-training.

In the last part of the experimental evaluation, the HTG method proposed in this work was compared to state-of-the-art HTG methods from the literature. GAN-based as well as DDPM-based models were considered and their performance was assessed in different ways. By using a pretrained writer identification model and a pretrained HTR model, the style imitation and the readability of generated word images were evaluated. The results indicate that all models encountered challenges in accurately replicating the style. When assessing the readability of the generated samples, the proposed HTG model is inferior to the models from the literature.

However, the primary focus of this work is the generation of training data, wherein two scenarios were considered for comparing the models. In the first scenario, the reproduction of the training images from different target datasets is used for training an HTR model. The model presented in this work considerably outperforms the other models in this task, particularly when utilizing the proposed semi-supervised training schemes. The second scenario resembles the setting in practical applications where no knowledge of the word distribution of the target dataset is present. Additionally, the incorporation of a few labeled samples from the target dataset is investigated. The experiments demonstrated that generating training data using the HTG models can considerably improve the performance of the trained HTR model, especially if no or only few labeled real samples are provided. In almost all settings, the HTG approach presented in this thesis was superior to the considered approaches from the literature.

6.2 OUTLOOK

The results presented in this thesis demonstrate that the use of HTG for training data generation is a promising approach and motivate further research. In particular, the application of this approach using other HTR models but also addressing other document analysis tasks on handwritten text are of high interest. In addition to yielding promising results, this work also underscored the difficulties and limitations of the presented HTG approach.

One considerable limitation of the presented and several other HTG approaches is the limitation to words. Thereby, training data requires a word-level segmentation which is costly to obtain. At the same time, the generation of word images limits the usage of the generated samples to downstream tasks on word-level. This work demonstrated, that removing the limitation to a certain number of characters in the generated words does not impact the generation performance. Recent studies have explored the feasibility of extending the generation to handwritten text lines (e. g. [GCK24; Din+23]) and even entire paragraphs [May+24] using *denoising diffusion probabilistic models* (DDPMs). Further investigating approaches towards the generation of complete document pages would allow for using HTG for training data generation for a variety of new tasks. However, it should be noted that the training and sampling of DDPMs is computationally expensive, even at word-level.

Modifications to the models and the self-training procedure are other promising approaches for improving the proposed HTG system. While the utilization of a pre-trained autoencoder saves the training costs for the respective model, using a model that has been trained to encode distinct features of handwriting in the latent space can be expected to enhance the generation performance. To date, only two recent works [May+24; Pip+25] explored training a handwriting specific autoencoder for their approaches, achieving promising results. Another possible modification to the proposed system is the use of a different style encoder. The experimental results demonstrated, that the employed MAE results in competitive generation performance. However, the capabilities to mimic specific styles in detail are not satisfactory and motivate for further research. Another aspect where the experiments demonstrated potential for enhancement pertains to the utilization of self-training. Almost all models examined in this work could be enhanced through self-training.

Nevertheless, a thorough examination of the parameter search reveals that better parameter choices could have been made, depending on the specific characteristics of each dataset and the level of supervision provided. However, in the case of an unlabeled target dataset, this analysis would be infeasible in practical application due to the lack of annotated validation data. This raises the necessity for a more refined approach to adapt self-training to the specific settings in these cases.

Another promising approach is the integration of synthetic data for training the HTG model itself. With the help of freely available computer fonts, almost an infinite amount of images can be rendered. While not necessarily resembling actual handwriting, mimicking a wide array of writing styles can be learned. Furthermore, the rendering from computer fonts allows for full control over the word and character frequencies. Consequently, words and characters that are infrequent in real datasets can be incorporated with greater frequency in the synthetic dataset with the objective of enhancing the generation quality for these elements. With their recently proposed autoregressive HTG model trained solely on synthetic images, Pippi et al. [Pip+25] achieve impressive results when generating training data for modern handwriting datasets. The approach has not been explored for DDPM-based models yet. A further research question pertains to the extent to which training on synthetic data is effective when the writing styles from the target dataset differ significantly from computer fonts (e. g. in historic documents). In particular, such support in the development and training of document analysis models for historical documents can facilitate the extraction of information from existing historical document collections. In such cases, the integration of unlabeled data from the target dataset, as presented in this thesis, could prove beneficial.

A ADDITIONAL RESULTS

A.1 STATISTICS OF EVALUATION SPLITS

For a detailed assessment of the generation performance for *in-vocabulary* (IV) and *out-of-vocabulary* (OOV) words and seen (S) and unseen (U) writing styles, we create partitions based on these categories as described in Section 5.1. We denote the resulting scenarios and their corresponding dataset partition as IV-S, IV-U, OOV-S and OOV-U. Table 15 provides details about the respective sets.

Table 15: Number of samples $|\mathbf{D}|$, number of writers $|\mathbf{W_D}|$, median number of samples per writer $\text{med}(|\mathbf{I}_w|)$ and lexicon size $|\mathbf{L_D}|$ of the splits used for different evaluation scenarios.

Dataset	Split	$\lvert\mathbf{D}\rvert$	$\lvert\mathbf{W_D}\rvert$	$\text{med}(\lvert\mathbf{I}_w\rvert)$	$\lvert\mathbf{L_D}\rvert$
IAM	IV-S	3 169	328	5	1 015
	IV-U	3 000	160	12	932
	OOV-S	3 333	328	5	763
	OOV-U	3 446	161	15	2 441
Rimes	IV-U	2 171	357	5	145
	OOV-U	3 000	369	7	788
CVL	IV-U	3 000	283	11	119
	OOV-U	3 000	283	11	113
GNHK	IV-U	3 000	159	18	850
	OOV-U	3 000	168	13	2 404

A.2 SAMPLING WITH UNIPC

As discussed in Section 3.1.5, the originally proposed sampling [HJA20] can be sped up by more advanced sampling frameworks. These frameworks reduce the number of denoising steps required while preserving competitive generation quality. Since each denoising step corresponds to a forward pass through the model, a significant cost reduction with regard to time and computational cost can be achieved. For our work, we employ UniPC [Zha+23] which is one of the state-of-the-art approaches for faster sampling of *diffusion models* (DMs). In their experiments, Zhao et al. [Zha+23] achieved good results for extremely low choices down to five steps. Besides low choices of only five or ten steps, we additionally explore a moderate choice of 50 steps

Table 16: *Character error rate* (CER) [%] on real IAM test images of *handwritten text recognition* (HTR) models trained on generated images from *latent diffusion models* (LDMs) trained with style inclusions `TS`. For sampling, the default sampling algorithm [HJA20] and UniPC [Zha+23] are employed with different numbers of denoising steps.

Steps	UniPC	No CFG	Unguided	$w_{gs} = 5$
1000		7.78	8.23	7.34
50	✓	8.08	8.85	7.39
10	✓	8.10	8.95	8.82
5	✓	9.18	10.03	13.51

and compare the results with sampling using the original algorithm with 1000 steps. The results in Table 16 show that sampling with 50 or ten steps using UniPC results in only slightly worse performance compared to sampling with 1000 steps. Even with only five steps, the results are competitive when sampling without guidance. However, when sampling with $w_{gs} = 5$, a small difference between sampling with 50 and ten steps can be observed. Since we found $w_{gs} = 5$ to be beneficial for sampling the source dataset, we consider this case more important than the others. Albeit further reduction of the number of sampling steps might be possible without harming the generation quality, we consider 50 steps to be a robust choice for our experiments.

A.3 TRADE-OFF IN SEMI-SUPERVISED TRAINING

We assume that the inclusion of samples without transcriptions from Rimes in the semi-supervised training might affect the generation performance of samples from IAM as well. Besides the evaluation of generated samples for Rimes, we also evaluate samples for IAM which are generated by the same models. We report the results in Table 17. The results confirm the expected deterioration of the results. However, the CER is only slightly worse for the semi-supervised training compared to training on IAM only. Given the improvements in the generation performance for Rimes, we consider this to be a good trade-off.

A.4 HYPERPARAMETERS FOR THE STYLE EMBEDDING COMPUTATION

For the computation of style embeddings, a common approach is to compute the writer embeddings based on a new random set of images each time a sample from the respective writer is used in training (e. g. [Nik+24a; Van+25; Gan+22]). This procedure, therefore, requires forward passes through the respective style encoder each iteration which is computationally expensive. Using a cache of precomputed embeddings avoids this additional effort. However, we do lack variation in these embeddings since we precompute a fixed number. Therefore, we conduct experiments with different numbers $\mathsf{N}_{se} \in \{10, 50, 100, 1000, 2500\}$ of embedding variations per writer, keeping $\mathsf{K} = 10$ fixed. We present the results in Table 18.

Table 17: CER [%] on real IAM test images of HTR models trained on generated images. The images are generated by LDMs trained fully labeled samples from IAM and samples from Rimes without transcriptions. *Style encoders* (SEs) are trained on different datasets (column "SE train"). The difference to an LDM trained and sampled with the same configuration which is only trained on IAM is indicated below each result.

SE train	Guidance	TP	TPL	CP	TA	CA	TS
IAM	Unguided	9.73 +0.63	9.42 +0.24	9.68 +0.35	9.61 +0.42	10.31 +0.82	**8.90** +0.05
	$w_{gs} = 2$	8.73 +0.56	8.37 +0.26	8.60 +0.09	8.47 −0.23	8.59 +0.35	**8.15** +0.31
	$w_{gs} = 3$	8.26 +0.64	8.17 +0.12	8.16 +0.18	8.21 +0.19	8.45 +0.37	**7.78** +0.24
	$w_{gs} = 4$	8.23 +0.45	8.18 +0.41	8.24 +0.06	8.17 +0.45	8.37 +0.44	**7.84** +0.27
	$w_{gs} = 5$	7.97 +0.41	8.03 +0.05	7.99 +0.39	7.92 +0.08	8.19 +0.37	**7.74** +0.35
Both	Unguided	9.46 +0.36	9.29 +0.11	9.50 +0.17	10.28 +1.09	9.63 +0.14	**9.15** +0.30
	$w_{gs} = 2$	8.54 +0.37	8.46 +0.35	8.60 +0.09	9.01 +0.31	8.56 +0.32	**8.12** +0.28
	$w_{gs} = 3$	8.19 +0.57	8.13 +0.08	8.29 +0.31	8.43 +0.41	8.28 +0.20	**7.96** +0.42
	$w_{gs} = 4$	8.03 +0.25	7.88 +0.11	8.25 +0.07	8.51 +0.79	8.08 +0.15	**7.68** +0.11
	$w_{gs} = 5$	8.01 +0.45	7.85 −0.13	8.18 +0.58	8.48 +0.64	8.01 +0.19	**7.71** +0.32

When comparing the different numbers of embedding variations per writer, only minimal variations of the results can be observed. In particular, not even the extreme choices of 10 or 2500 variations per writer show considerable differences. Neither for the source nor for the target dataset, any trends regarding the number of variations can be identified. As there is no improvement with an increased number of variations, we conclude that caching style embeddings based on random samples from the writer is sufficient for a good generation performance.

The other important parameter in the computation of style embeddings is the number K of examples used in the computation of each writer embedding. In the literature, common choices range from $\mathsf{K} = 1$ up to 15. Besides $\mathsf{K} = 10$ as used in the previous experiments, we explore fewer samples with $\mathsf{K} \in \{1, 5\}$ as well as more samples with $\mathsf{K} \in \{15, 25, 50\}$. It is important to note that for $\mathsf{K} = 1$ the embedding is still computed on a random sample from the respective writer. Especially for Rimes, the dataset does not provide enough samples for some writers for large K. In these cases, all sample images are included in the computation, meaning there are no variations in the cached embeddings for the respective writers. The results are reported in Table 18.

Table 18: Impact of the number of embedding variations per writer on the performance of an HTR model trained on generated samples. $\mathsf{K} = 10$ random samples from the respective writer are used per embedding.

(a) CER [%] on real IAM test images.

SE train	$\mathsf{N}_{se} = 10$	$\mathsf{N}_{se} = 50$	$\mathsf{N}_{se} = 100$	$\mathsf{N}_{se} = 1000$	$\mathsf{N}_{se} = 2500$
IAM	7.73	7.69	7.74	8.77	7.68
Both	7.58	7.59	7.71	7.59	7.84

(b) CER [%] on real Rimes test images.

SE train	$\mathsf{N}_{se} = 10$	$\mathsf{N}_{se} = 50$	$\mathsf{N}_{se} = 100$	$\mathsf{N}_{se} = 1000$	$\mathsf{N}_{se} = 2500$
IAM	6.70	6.33	6.61	6.59	6.83
Both	6.53	6.84	6.65	6.72	6.78

Except for one case (Rimes, SE trained on both datasets), computing embeddings on $\mathsf{K} = 1$ image performs worst. However, the gap in performance is small compared to models where embeddings are extracted from more images. Overall, it can be observed that performance mostly improves with an increased number of images per embedding. This observation is in line with the intuition, that more style information can be encoded in the embeddings when more example images are provided. At the same time, the relatively small performance drop for $\mathsf{K} = 1$ shows, that our *handwritten text generation* (HTG) system is capable of extracting information about the writing style based on a single image. In general, the results indicate that the model is robust against the exact choice of K. While leading to a better performance, $\mathsf{K} \in \{25, 50\}$ are not realistic in practical application (cf. $\text{med}(|\mathbf{I}_w|)$ in Table 2). Since the results for $\mathsf{K} = 1$ and $\mathsf{K} = 5$ are slightly worse, $\mathsf{K} = 10$ and $\mathsf{K} = 15$ are the most promising choices. One to two handwritten sentences per writer should be sufficient for both options to obtain the required number of word images. Because only very small differences in performance can be observed, we consider $\mathsf{K} = 10$ the preferable choice.

A.5 ADDITIONAL QUALITATIVE RESULTS

This section presents more qualitative results using HTG models trained with different levels of supervision. Besides generated examples for the target dataset Rimes, generated samples for CVL, and GNHK shown. The models that were used for generation are identical with those used in Section 5.3.6. For each target dataset, ten samples from the respective real training set were randomly picked. The same combinations of words and writing styles were used to generate samples that can directly be compared to respective reference images. Furthermore, we provide samples that were generated by HTG models that were trained on real samples from the target dataset with full supervision (i. e. styles and transcriptions). The generated images are presented in Table 18.

Table 18: Impact of the number of random samples used per embedding variation on the performance of an HTR model trained on generated samples. $\mathsf{N}_{se} = 100$ embedding variations are computed per writer.

(a) CER [%] on real IAM test images.

SE train	$\mathsf{K} = 1$	$\mathsf{K} = 5$	$\mathsf{K} = 10$	$\mathsf{K} = 15$	$\mathsf{K} = 25$	$\mathsf{K} = 50$
IAM	8.48	7.78	7.74	7.84	7.56	7.67
Both	8.65	7.90	7.71	7.82	7.56	7.49

(b) CER [%] on real Rimes test images.

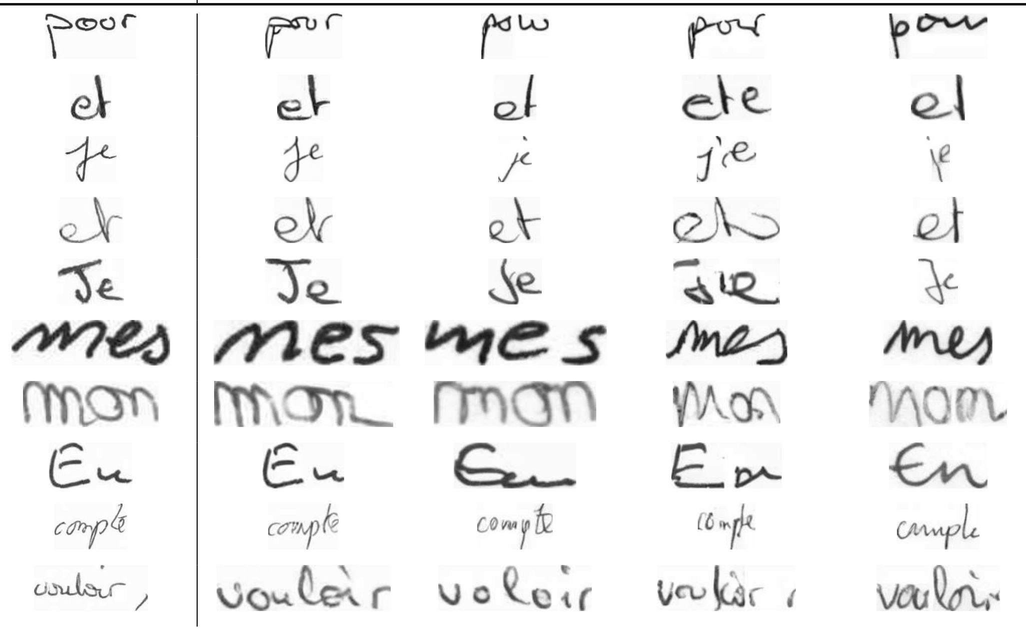
SE train	$\mathsf{K} = 1$	$\mathsf{K} = 5$	$\mathsf{K} = 10$	$\mathsf{K} = 15$	$\mathsf{K} = 25$	$\mathsf{K} = 50$
IAM	7.35	7.25	6.61	6.84	6.15	6.20
Both	6.64	6.64	6.65	6.28	6.48	6.76

For Rimes (cf. Table 18a), the fully supervised HTG model as well as the semi-supervised HTG model trained with writer IDs generate mostly well readable results. However, it can be observed, that the semi-supervised model fails to mimic details of the writing styles in some cases. The visual quality and readability of the samples generated without annotations from the target dataset (i. e. "Semi" and "IAM only") are considerably worse which is in line with the quantitative evaluation in Section 5.3.6. For the generated examples from CVL, a similar trend can be observed. In particular, the generation quality of the semi-supervised model trained without any annotations from CVL is poor. However, when used as training data the resulting HTR model still achieves a CER of 16.69 %. Furthermore, using the semi-supervised HTG model instead of the HTG model trained only on IAM for training data generation results in a considerable better HTR model. This trend is not reflected in the randomly selected examples in Table 18b underlining that a small qualitative assessment of the results is not sufficient for assessing the suitability of the generated samples to be used as training data. For GNHK the generation quality of the semi-supervised HTG model can also be considered the worst from all presented models. However, when comparing the writing styles, is becomes apparent that the HTG model trained only on IAM generated well readable examples but completely fails to mimic the styles from GNHK. This is in line with the high CER of 41.91 % from the HTR model trained on the respective generated samples and highlights the importance of style imitation in training data generation.

Table 18: Randomly selected examples from generated training images for the respective target dataset with different levels of supervision. The leftmost column provides a real image with the same content and style as the generated word for reference.

(a) Rimes

Reference	Full Sup.	Semi (WL)	Semi	IAM only



(b) CVL

Reference	Full Sup.	Semi (WL)	Semi	IAM only

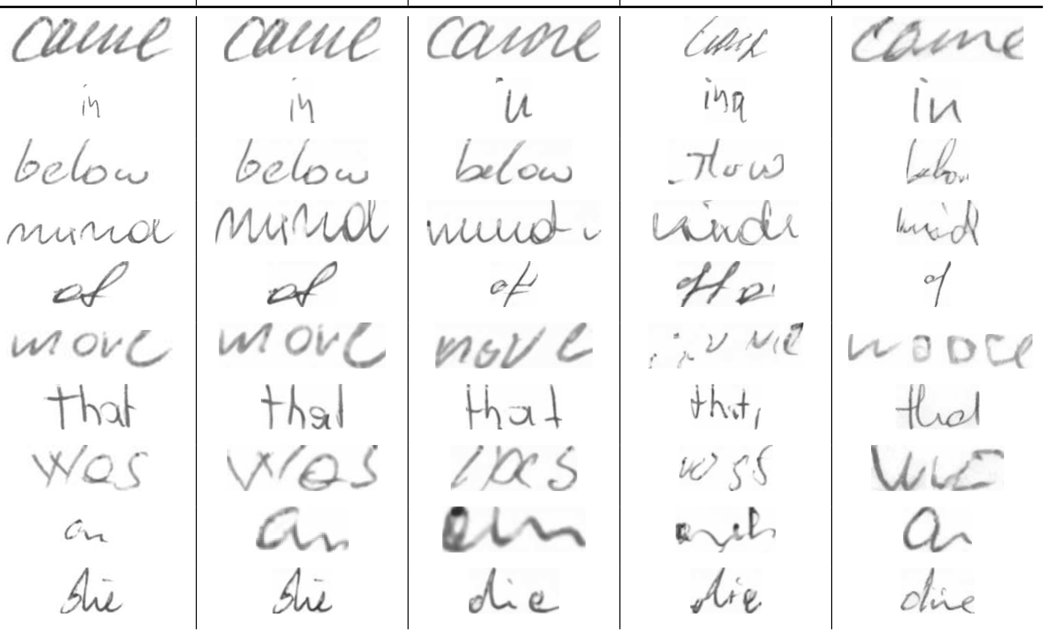


Table 18: Randomly selected examples from generated training images for the respective target dataset with different levels of supervision. The leftmost column provides a real image with the same content and style as the generated word for reference.

(c) GNHK

Reference	Full Sup.	Semi (WL)	Semi	IAM only

A.6 CONVERGENCE OF SELF-TRAINING

In order to apply self-training [WF24] using our HTR model, we first tune the hyperparameters of the approach. For the parameter search, we train our model on word images rendered using computer fonts. Then we apply self-training to adapt the model to real images without annotation from the training partition of IAM and test the performance on the test partition. Detailed results with the CER every ten cycles of self-training are reported in Table 19.

As discussed in Section 5.3.7, self-training without thresholding diverges. Using a very small threshold $\tau = 0.1$ helps but does not inhibit divergence. This indicates that self-training requires a more restrictive filtering of pseudo-labels. At the same time, high thresholds (e. g. $\tau = 0.6$) harm the performance as too few samples are considered for self-training. Furthermore, the results show that increasing the learning rate to five times the rate used in training from scratch is detrimental. In combination with the high threshold $\tau = 0.6$, the performance collapsed before finishing 30 cycles inhibiting further self-training. Another observation is that the best results are achieved after different numbers of cycles. For some runs, the best test performance was achieved before 50 cycles and further training resulted in a worse CER. However, for most configurations (e. g. $\tau \in 0.3, 0.4, 0.5$ and a small learning rate) training converges and performance only slightly varies when the training is continued.

While this experimental setup allowed for a parameter search without requiring any target dataset, we must not assume that these choices are optimal for other

Table 19: CER [%] on IAM test after self-training on IAM train for 50 cycles with different learning rates and confidence thresholds τ. The initial model was trained on images rendered from computer fonts and achieves a CER of 16.26 % before self-training.

Learning Rate	τ	Self-Training Cycles				
		10	20	30	40	50
0.0001	–	**14.31**	14.60	16.05	17.00	18.16
	0.1	**13.22**	14.74	16.21	18.24	19.52
	0.2	10.49	10.37	**9.85**	10.06	9.90
	0.3	9.50	9.09	9.00	8.86	**8.61**
	0.4	9.47	8.68	8.60	8.30	**8.00**
	0.5	9.44	8.91	8.82	8.82	**8.80**
	0.6	9.90	9.33	**9.29**	9.32	9.30
0.0002	–	**14.16**	15.27	15.81	16.39	19.26
	0.1	**13.70**	14.44	16.00	17.19	18.28
	0.2	12.15	**11.55**	12.54	12.93	12.36
	0.3	9.44	9.12	8.81	8.83	**8.47**
	0.4	9.34	8.58	8.38	8.25	**7.99**
	0.5	10.00	9.71	**9.38**	9.55	9.54
	0.6	9.88	9.50	9.52	9.74	**9.37**
0.001	–	**13.33**	14.50	14.36	15.90	17.49
	0.1	12.58	**11.82**	12.80	12.80	14.47
	0.2	**10.67**	11.40	11.59	11.85	12.20
	0.3	9.99	9.85	9.91	9.69	**9.59**
	0.4	9.20	9.05	8.93	8.81	**8.77**
	0.5	**9.88**	10.11	10.55	10.36	10.55
	0.6	**12.52**	12.94	13.87	15.89	14.47
0.005	–	**20.11**	21.20	22.19	23.94	27.33
	0.1	**18.57**	19.46	22.15	26.02	30.13
	0.2	20.69	**19.97**	21.67	25.15	26.13
	0.3	**18.86**	22.82	26.34	27.73	35.29
	0.4	**28.85**	42.52	51.48	56.80	58.08
	0.5	**43.76**	61.91	65.19	68.79	72.93
	0.6	**60.99**	83.52	N/A	N/A	N/A

Table 20: Improvements by self-training on different datasets. The CER [%] on real IAM test images is measured every ten cycles of self-training for different initial models.

Target Dataset	Pretraining	Initial Model	Self-Training Cycles 10	20	30	40	50
Rimes	Synth	14.22	6.05	5.83	5.28	5.23	**5.14**
	Real IAM	25.85	15.21	14.58	14.38	**13.97**	13.99
	LDM (IAM)	11.53	6.38	**6.00**	6.24	7.14	7.55
	LDM (Semi)	10.83	7.44	7.17	**7.16**	7.21	7.28
	LDM (WL)	8.63	5.41	5.46	5.71	5.58	**5.37**
CVL	Synth	24.99	17.44	16.12	15.80	15.17	**14.83**
	Real IAM	14.40	12.13	11.62	10.92	10.63	**10.31**
	LDM (IAM)	31.77	**24.47**	25.26	25.91	26.43	26.97
	LDM (Semi)	27.69	25.12	**24.76**	26.39	28.10	28.56
	LDM (WL)	21.00	**17.80**	17.95	17.83	17.93	18.26
GNHK	Synth	27.57	19.91	**18.94**	19.13	19.11	19.02
	Real IAM	36.61	19.84	18.48	18.06	17.52	**17.35**
	LDM (IAM)	38.23	26.05	25.30	24.86	24.72	**24.61**
	LDM (Semi)	23.26	**20.71**	21.34	20.95	21.14	21.21
	LDM (WL)	19.93	19.47	19.46	19.57	19.41	**19.37**

settings. We train initial models using generated samples. Thereby, the models are better adapted to the target dataset and provide a better initial performance. At the same time, the labels for training can be noisy. These effects might influence the progress during self-training. Therefore, we measured the CER every ten cycles when applying self-training to improve HTR model trained on our generated training data. We report detailed results in Table 20.

When using an initial model trained on rendered images or real training images from IAM, self-training results in a steady improvement except for some minor variations. Using an initial model trained on generated images, the best results are often obtained after fewer cycles. In most cases, the continuation of self-training resulted in inferior results. Especially for CVL, the best results are achieved early after only 10 – 20 cycles. After that, performance deteriorates. For the initial model trained on generated images from the semi-supervised LDM performance after all 50 cycles of self-training is even worse than the initial result. However, considering its best intermediate result after 20 cycles, self-training is able to improve the initial performance of the HTR model. These findings underline the need for a more detailed study on the choice of hyperparameters for self-training based on differently performing initial models.

BIBLIOGRAPHY

[AB17] M. Arjovsky and L. Bottou. "Towards Principled Methods for Training Generative Adversarial Networks." In: *Int. Conf. on Learning Representations*. Toulon, France, 2017.

[ACB17] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein Generative Adversarial Networks." In: *Int. Conf. on Machine Learning*. Sydney, Australia, 2017, pp. 214–223.

[AM24] A. Aswathy and P. U. Maheswari. "Generative Innovations for Paleography: Enhancing Character Image Synthesis through Unconditional Single Image Models." In: *Heritage Science* 12.258 (2024). DOI: 10.1186/s40494-024-01373-4.

[AMM19] E. Alonso, B. Moysset, and R. Messina. "Adversarial Generation of Handwritten Text Images Conditioned on Sequences." In: *Int. Conf. on Document Analysis and Recognition*. Sydney, Australia, 2019, pp. 481–486. DOI: 10.1109/ICDAR.2019.00083.

[APH18] E. Aksan, F. Pece, and O. Hilliges. "DeepWriting: Making Digital Ink Editable via Deep Generative Modeling." In: *Proc. CHI Conference on Human Factors in Computing Systems*. Montreal, Canada, 2018. DOI: 10.1145/3173574.3173779.

[Aug+08] E. Augustin, M. Carré, E. Grosicki, J.-M. Brodin, E. Geoffrois, and F. Preteux. "RIMES evaluation campaign for handwritten mail processing." In: *Int. Conf. on Frontiers in Handwriting Recognition*. La Baule, France, 2008, pp. 231–235.

[BA19] A. Baevski and M. Auli. "Adaptive Input Representations for Neural Language Modeling." In: *Int. Conf. on Learning Representations*. New Orleans, LA, USA, 2019, pp. 1–11.

[Bal+17] D. Balduzzi, M. Frean, L. Leary, J. P. Lewis, K. W.-D. Ma, and B. McWilliams. "The Shattered Gradients Problem: If resnets are the answer, then what is the question?" In: *Int. Conf. on Machine Learning*. Sydney, Australia, 2017, pp. 342–350.

[BB24] C. M. Bishop and H. Bishop. *Deep Learning: Foundations and Concepts*. Springer International Publishing, 2024. ISBN: 978-3-031-45468-4. DOI: 10.1007/978-3-031-45468-4.

[BCB15] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *Int. Conf. on Learning Representations*. San Diego, CA, USA, 2015.

[BDS19] A. Brock, J. Donahue, and K. Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis." In: *Int. Conf. on Learning Representations*. New Orleans, LA, USA, 2019, pp. 1–11.

[Bha+22] R. Bhatt, A. Rai, S. Chanda, and N. C. Krishnan. "Pho(SC)-CTC—a hybrid approach towards zero-shot word image recognition." In: *Int. Journal on Document Analysis and Recognition* 26.1 (2022), pp. 51–63. DOI: 10.1007/s10032-022-00407-6.

[Bhu+21] A. Bhunia, S. Khan, H. Cholakkal, R. Anwer, F. Khan, and M. Shah. "Handwriting Transformers." In: *IEEE/CVF Int. Conf. on Computer Vision*. Los Alamitos, CA, USA, 2021, pp. 1066–1074. DOI: 10.1109/ICCV48922.2021.00112.

[Biń+18] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton. "Demystifying MMD GANs." In: *Int. Conf. on Learning Representations*. Vancouver, Canada, 2018, pp. 1–15.

[Bis94] C. M. Bishop. *Mixture Density Networks*. Tech. rep. 1994.

[BKH16] J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer Normalization." In: *arXiv: abs/1607.06450* (2016). DOI: 10.48550/arXiv.1607.06450.

[Bon+22] S. Bond-Taylor, A. Leach, Y. Long, and C. G. Willcocks. "Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models." In: *IEEE Trans. on Pattern Analysis and Machine Intelligence* 44.11 (2022), pp. 7327–7347. DOI: 10.1109/TPAMI.2021.3116668.

[Bot10] L. Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent." In: *Proceedings of COMPSTAT'2010*. 2010, pp. 177–186. DOI: 10.1007/978-3-7908-2604-3_16.

[Bra24] K. Brandenbusch. "Semi-Supervised Adaptation of Diffusion Models for Handwritten Text Generation." In: *arXiv: abs/2412.15853* (2024). DOI: 10.48550/arXiv.2412.15853.

[BRF21] K. Brandenbusch, E. Rusakov, and G. A. Fink. "Context Aware Generation of Cuneiform Signs." In: *Proc. of the Int. Conf. on Document Analysis and Recognition*. Lausanne, Switzerland, 2021, pp. 65–79. DOI: 10.1007/978-3-030-86549-8_5.

[Bro+20] T. B. Brown *et al.* "Language Models are Few-Shot Learners." In: *arXiv: abs/2005.14165* (2020). DOI: 10.48550/arXiv.2005.14165.

[Cha+18] B. Chang, Q. Zhang, S. Pan, and L. Meng. "Generating Handwritten Chinese Characters using CycleGAN." In: *Proc. of the IEEE/CVF Winter Conf. on Applications of Computer Vision*. Lake Tahoe, NV, USA, 2018, pp. 199–207. DOI: 10.1109/wacv.2018.00028.

[Cho+14] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." In: *Proc. of the Conf. on Empirical Methods in Natural Language Processing*. Doha, Qatar, 2014. DOI: 10.3115/v1/D14-1179.

[Chu+15] J. Chung, K. Kastner, L. Dinh, K. Goel, A. Courville, and Y. Bengio. "A Recurrent Latent Variable Model for Sequential Data." In: *Advances in Neural Information Processing Systems*. Montreal, Canada, 2015.

[CLD24] M. Chen, H. Liu, and L. Dong. "P2H-GAN: An Effective Method For Generating Handwritten Expressions." In: *Int. Conf. on Artificial Neural Networks*. Lugano, Switzerlan, 2024, pp. 361–376. DOI: 10.1007/978-3-031-72338-4_25.

[CNL11] A. Coates, A. Ng, and H. Lee. "An Analysis of Single-Layer Networks in Unsupervised Feature Learning." In: *Proc. of the Int. Conf. on Artificial Intelligence and Statistics*. Fort Lauderdale, FL, USA, 2011, pp. 215–223.

[CPK23] C. C. Chang, L. P. G. Perera, and S. Khudanpur. "Crosslingual Handwritten Text Generation Using GANs." In: *Int. Conf. on Document Analysis and Recognition Workshops*. San José, CA, USA, 2023, pp. 285–301. DOI: 10.1007/978-3-031-41501-2_20.

[Dai+24] G. Dai, Y. Zhang, Q. Ke, Q. Guo, and S. Huang. "One-Shot Diffusion Mimicker for Handwritten Text Generation." In: *European Conf. on Computer Vision*. Milan, Italy, 2024, pp. 410–427. DOI: 10.1007/978-3-031-73636-0_24.

[Dau+16] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. "Language Modeling with Gated Convolutional Networks." In: *arXiv: abs/1612.08083* (2016). DOI: 10.48550/arXiv.1612.08083.

[Dav+20] B. L. Davis, C. Tensmeyer, B. L. Price, C. Wigington, B. S. Morse, and R. Jain. "Text and Style Conditioned GAN for Generation of Offline Handwriting Lines." In: *British Machine Vision Conference*. Virtual Conference, 2020.

[DB16] A. Dosovitskiy and T. Brox. "Generating Images with Perceptual Similarity Metrics based on Deep Networks." In: *Advances in Neural Information Processing Systems*. Barcelona, Spain, 2016.

[Den+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "ImageNet: A large-scale hierarchical image database." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Miami, FL, USA, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[Dev+19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics*. Minneapolis, Minnesota, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[DHS00] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. 2nd ed. Wiley, 2000. ISBN: 978-1-118-58600-6.

[DHS11] J. Duchi, E. Hazan, and Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159.

[Din+23] H. Ding, B. Luan, D. Gui, K. Chen, and Q. Huo. "Improving Handwritten OCR with Training Samples Generated by Glyph Conditional Denoising Diffusion Probabilistic Model." In: *Int. Conf. on Document Analysis and Recognition*. San José, CA, USA, 2023, pp. 20–37. DOI: 10.1007/978-3-031-41685-9_2.

[DKB14] L. Dinh, D. Krueger, and Y. Bengio. "NICE: Non-linear Independent Components Estimation." In: *arXiv: abs/1410.8516* (2014). DOI: 10.48550/arXiv.1410.8516.

[DN21] P. Dhariwal and A. Nichol. "Diffusion Models Beat GANs on Image Synthesis." In: *Advances in Neural Information Processing Systems*. Virtual Conference, 2021, pp. 8780–8794.

[Dos+21] A. Dosovitskiy *et al.* "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." In: *Int. Conf. on Learning Representations*. Vienna, Austria, 2021.

[DSB17] L. Dinh, J. Sohl-Dickstein, and S. Bengio. "Density estimation using Real NVP." In: *Int. Conf. on Learning Representations*. Toulon, France, 2017.

[DW19] B. Dai and D. Wipf. "Diagnosing and Enhancing VAE Models." In: *Int. Conf. on Learning Representations*. New Orleans, LA, USA, 2019.

[EB24] R. Elanwar and M. Betke. "Generative Adversarial Networks for Handwriting Image Generation: A Review." In: *The Visual Computer* (2024). DOI: 10.1007/s00371-024-03534-9.

[ERO21] P. Esser, R. Rombach, and B. Ommer. "Taming Transformers for High-Resolution Image Synthesis." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Virtual Conference, 2021, pp. 12868–12878. DOI: 10.1109/CVPR46437.2021.01268.

[Fog+20] S. Fogel, H. Averbuch-Elor, S. Cohen, S. Mazor, and R. Litman. "ScrabbleGAN: Semi-Supervised Varying Length Handwritten Text Generation." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Seattle, WA, USA, 2020, pp. 4323–4332. DOI: 10.1109/CVPR42600.2020.00438.

[Fua+24] M. M. Fuad, A. Faiyaz, N. M. K. Arnob, M. F. Mridha, A. K. Saha, and Z. Aung. "Okkhor-Diffusion: Class Guided Generation of Bangla Isolated Handwritten Characters Using Denoising Diffusion Probabilistic Model (DDPM)." In: *IEEE Access* 12 (2024), pp. 37521–37539. DOI: 10.1109/ACCESS.2024.3370674.

[GA09] E. Grosicki and H. E. Abed. "ICDAR 2009 Handwriting Recognition Competition." In: *Int. Conf. on Document Analysis and Recognition*. Barcelona, Spain, 2009, pp. 1398–1402. DOI: 10.1109/ICDAR.2009.184.

[Gan+22] J. Gan, W. Wang, J. Leng, and X. Gao. "HiGAN+: Handwriting Imitation GAN with Disentangled Representations." In: *ACM Trans. Graph.* 42.1 (2022), pp. 1–17. DOI: 10.1145/3550070.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[GCK24] A. Gurav, S. Chanda, and N. C. Krishnan. "HTLDIFF: Handwritten Text Line Generation - A Diffusion Model Perspective." In: *IEEE Int. Conf. on Image Processing Theory, Tools and Applications*. Rabat, Morocco, 2024, pp. 1–6. DOI: 10.1109/IPTA62886.2024.10755723.

[GD24] A. Gu and T. Dao. "Mamba: Linear-Time Sequence Modeling with Selective State Spaces." In: *Conference on Language Modeling*. Philadelphia, PA, USA, 2024.

[GE11] E. Grosicki and H. El-Abed. "ICDAR 2011 - French Handwriting Recognition Competition." In: *Int. Conf. on Document Analysis and Recognition*. Beijing, China, 2011, pp. 1459–1463. DOI: 10.1109/ICDAR.2011.290.

[GGR22] A. Gu, K. Goel, and C. Ré. "Efficiently Modeling Long Sequences with Structured State Spaces." In: *Int. Conf. on Learning Representations*. Virtual Conference, 2022.

[GKC24] A. Gurav, N. C. Krishnan, and S. Chanda. "Word-Diffusion: Diffusion-Based Handwritten Text Word Image Generation." In: *Int. Conf. on Pattern Recognition*. Kolkata, India, 2024, pp. 53–72. DOI: 10.1007/978-3-031-78495-8_4.

[Goo+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems*. Montreal, Canada, 2014.

[Gra+06] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks." In: *Int. Conf. on Machine Learning*. Pittsburgh, PA, USA, 2006, pp. 369–376. ISBN: 1595933832. DOI: 10.1145/1143844.1143891.

[Gra13] A. Graves. "Generating Sequences With Recurrent Neural Networks." In: *arXiv: abs/1308.0850* (2013). DOI: 10.48550/arXiv.1308.0850. arXiv: 1308.0850 [cs.NE].

[GSC00] F. A. Gers, J. Schmidhuber, and F. Cummins. "Learning to Forget: Continual Prediction with LSTM." In: *Neural Computation* 12.10 (2000), pp. 2451–2471. DOI: 10.1162/089976600300015015.

[Gua+20] M. Guan, H. Ding, K. Chen, and Q. Huo. "Improving Handwritten OCR with Augmented Text Line Images Synthesized from Online Handwriting Samples by Style-Conditioned GAN." In: *Int. Conf. on Frontiers in Handwriting Recognition*. Virtual Conference, 2020, pp. 151–156. DOI: 10.1109/ICFHR2020.2020.00037.

[Gui+23] D. Gui, K. Chen, H. Ding, and Q. Huo. "Zero-shot Generation of Training Data with Denoising Diffusion Probabilistic Model for Handwritten Chinese Character Recognition." In: *Int. Conf. on Document Analysis and Recognition*. San José, CA, USA, 2023, pp. 348–365. DOI: 10.1007/978-3-031-41679-8_20.

[GW21] J. Gan and W. Wang. "HiGAN: Handwriting Imitation Conditioned on Arbitrary-Length Texts and Disentangled Styles." In: *Proc. of the AAAI Conf. on Artificial Intelligence*. Virtual Conference, 2021, pp. 7484–7492. DOI: 10.1609/aaai.v35i9.16917.

[Ham+24] S. J. H. Hamdani, S. Saifullah, S. Agne, A. Dengel, and S. Ahmed. "Latent Diffusion for Guided Document Table Generation." In: *Int. Conf. on Document Analysis and Recognition*. Athens, Greece, 2024, pp. 368–383. DOI: 10.1007/978-3-031-70549-6_22.

[He+15] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *arXiv: abs/1502.01852* (2015). DOI: 10.48550/arXiv.1502.01852.

[He+16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA, 2016, pp. 770–778.

[He+22] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. "Masked Autoencoders Are Scalable Vision Learners." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. New Orleans, LA, USA, 2022, pp. 15979–15988. DOI: 10.1109/CVPR52688.2022.01553.

[Heu+17] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." In: *Advances in Neural Information Processing Systems*. Long Beach, CA, USA, 2017.

[HG16] D. Hendrycks and K. Gimpel. "Gaussian Error Linear Units (GELUs)." In: *arXiv: abs/1606.08415* (2016). DOI: 10.48550/arXiv.1606.08415.

[Hig+17] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." In: *Int. Conf. on Learning Representations*. Toulon, France, 2017.

[HJA20] J. Ho, A. Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models." In: *Advances in Neural Information Processing Systems*. Virtual Conference, 2020.

[HMB16] T. S. F. Haines, O. Mac Aodha, and G. J. Brostow. "My Text in Your Handwriting." In: *ACM Trans. on Graphics* 35.3 (2016), pp. 1–18. DOI: 10.1145/2886099.

[How+19] A. Howard *et al.* "Searching for MobileNetV3." In: *arXiv: abs/1905.02244* (2019). DOI: 10.48550/arXiv.1905.02244.

[HS21] J. Ho and T. Salimans. "Classifier-Free Diffusion Guidance." In: *NeurIPS Workshop on Deep Generative Models and Downstream Applications*. 2021. DOI: 10.48550/arXiv.2207.12598.

[HS97] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory." In: *Neural Computing* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks are Universal Approximators." In: *Neural Networks* 2.5 (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[Ing+19] R. R. Ingle, Y. Fujii, T. Deselaers, J. Baccash, and A. C. Popat. "A Scalable Handwritten Text Recognition System." In: *Int. Conf. on Document Analysis and Recognition*. Sydney, Australia, 2019, pp. 17–24. DOI: 10.1109/ICDAR.2019.00013.

[IS15] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Int. Conf. on Machine Learning*. Lille, France, 2015, pp. 448–456.

[Iso+17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-To-Image Translation With Conditional Adversarial Networks." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Honolulu, HI, USA, 2017, pp. 5967–5976. DOI: 10.1109/CVPR.2017.632.

[JAF16] J. Johnson, A. Alahi, and L. Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution." In: *European Conf. on Computer Vision*. Amsterdam, Netherlands, 2016, pp. 694–711. DOI: 10.1007/978-3-319-46475-6_43.

[JC19] B. Ji and T. Chen. "Generative Adversarial Network for Handwritten Text." In: *arXiv: abs/1907.11845* (2019). DOI: 10.48550/arXiv.1907.11845. arXiv: 1907.11845 [cs.LG].

[Jia+21] L. Jiang, B. Dai, W. Wu, and C. C. Loy. "Focal Frequency Loss for Image Reconstruction and Synthesis." In: *IEEE/CVF Int. Conf. on Computer Vision*. Virtual Conference, 2021, pp. 13919–13929.

[JLG78] S. Johansson, G. N. Leech, and H. Goodluck. *Manual of information to accompany the Lancaster-Oslo/Bergen corpus of British English, for use with digital computers*. Tech. rep. Oslo, Norway: Department of Computer Science, University of Oslo, 1978.

[Kan+20a] L. Kang, P. Riba, M. Rusiñol, A. Fornés, and M. Villegas. "Distilling Content from Style for Handwritten Word Recognition." In: *Int. Conf. on Frontiers in Handwriting Recognition*. Virtual Conference, 2020, pp. 139–144. DOI: 10.1109/ICFHR2020.2020.00035.

[Kan+20b] L. Kang, P. Riba, Y. Wang, M. Rusiñol, A. Fornés, and M. Villegas. "GANwriting: Content-Conditioned Generation of Styled Handwritten Word Images." In: *European Conf. on Computer Vision*. Glasgow, United Kingdom, 2020, pp. 273–289. DOI: 10.1007/978-3-030-58592-1_17.

[Kan+20c] L. Kang, M. Rusinol, A. Fornes, P. Riba, and M. Villegas. "Unsupervised Writer Adaptation for Synthetic-to-Real Handwritten Word Recognition." In: *Proc. of the IEEE/CVF Winter Conf. on Applications of Computer Vision*. Snowmass Village, CO, USA, 2020, pp. 3491–3500. DOI: 10.1109/WACV45572.2020.9093392.

[Kan+22] L. Kang, P. Riba, M. Rusiñol, A. Fornés, and M. Villegas. "Content and Style Aware Generation of Text-Line Images for Handwriting Recognition." In: *IEEE Trans. on Pattern Analysis and Machine Intelligence* 44.12 (2022), pp. 8846–8860. DOI: 10.1109/TPAMI.2021.3122572.

[Kar+18] T. Karras, T. Aila, S. Laine, and J. Lehtinen. "Progressive Growing of GANs for Improved Quality, Stability, and Variation." In: *Int. Conf. on Learning Representations*. Vancouver, Canada, 2018.

[KB14] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *arXiv: abs/1412.6980* (2014). DOI: 10.48550/arXiv.1412.6980.

[Kin+16] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. "Improved Variational Inference with Inverse Autoregressive Flow." In: *Advances in Neural Information Processing Systems*. Barcelona, Spain, 2016.

[KJ16] P. Krishnan and C. V. Jawahar. "Generating Synthetic Data for Text Recognition." In: *arXiv: abs/1608.04224* (2016). DOI: 10.48550/arXiv.1608.04224. arXiv: 1608.04224 [cs.CV].

[KJ19] P. Krishnan and C. V. Jawahar. "HWNet v2: an efficient word image representation for handwritten documents." In: *Int. Journal on Document Analysis and Recognition* 22.4 (2019), pp. 387–405. DOI: 10.1007/s10032-019-00336-x.

[Kle+13] F. Kleber, S. Fiel, M. Diem, and R. Sablatnig. "CVL-DataBase: An Off-Line Database for Writer Retrieval, Writer Identification and Word Spotting." In: *Int. Conf. on Document Analysis and Recognition*. Washington DC, USA, 2013, pp. 560–564. DOI: 10.1109/ICDAR.2013.117.

[KO18] V. Khrulkov and I. Oseledets. "Geometry Score: A Method For Comparing Generative Adversarial Networks." In: *Int. Conf. on Machine Learning*. Stockholm, Sweden, 2018, pp. 2621–2629.

[Kon+07] T. Konidaris, B. Gatos, K. Ntzios, I. Pratikakis, S. Theodoridis, and S. J. Perantonis. "Keyword-Guided Word Spotting in Historical Printed Documents Using Synthetic Data and User Feedback." In: *Int. Journal on Document Analysis and Recognition* 9.2–4 (Mar. 2007), pp. 167–177. ISSN: 1433-2825. DOI: 10.1007/s10032-007-0042-4.

[Kri+21] P. Krishnan, R. Kovvuri, G. Pang, B. Vassilev, and T. Hassner. "TextStyleBrush: Transfer of Text Aesthetics from a Single Example." In: *arXiv: abs/2106.08385* (2021). DOI: 10.48550/ARXIV.2106.08385. arXiv: 2106.08385 [cs.CV].

[KTT20] A. Kotani, S. Tellex, and J. Tompkin. "Generating Handwriting via Decoupled Style Descriptors." In: *European Conf. on Computer Vision*. Glasgow, United Kingdom, 2020, pp. 764–780. DOI: 10.1007/978-3-030-58610-2_45.

[KW13] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes." In: *arXiv: abs/1312.6114* (2013). DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114.

[LCL21] A. W. C. Lee, J. Chung, and M. Lee. "GNHK: A Dataset for English Handwriting in the Wild." In: *Int. Conf. on Document Analysis and Recognition*. Lausanne, Switzerland, 2021, pp. 399–412. DOI: 10.1007/978-3-030-86337-1_27.

[LeC+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition." In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[Lee+19] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks." In: *Int. Conf. on Machine Learning*. Long Beach, CA, USA, 2019, pp. 3744–3753.

[Lev66] V. Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." In: *Soviet Physics Doklady*. Vol. 10. 8. 1966, pp. 707–710.

[LH19] I. Loshchilov and F. Hutter. "Decoupled Weight Decay Regularization." In: *Int. Conf. on Learning Representations*. New Orleans, LA, USA, 2019.

[Lin+25] Y.-L. Lin, H.-C. Cheng, C.-I. Huang, C.-Y. Wang, and J.-C. Wang. "Impact of Glyph Information on Latent Space Diffusion Models for Accurate Handwritten Text Generation." In: *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*. Hyderabad, India, 2025, pp. 1–5. DOI: 10.1109/ICASSP49660.2025.10890644.

[Liu+21] X. Liu, G. Meng, S. Xiang, and C. Pan. "Handwritten Text Generation via Disentangled Representations." In: *IEEE Signal Processing Letters* 28 (2021), pp. 1838–1842. DOI: 10.1109/LSP.2021.3109541.

[Liu+24] Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, J. Jiao, and Y. Liu. "VMamba: Visual State Space Model." In: *Advances in Neural Information Processing Systems*. Vancouver, Canada, 2024, pp. 103031–103063.

[LL20] T. Luhman and E. Luhman. "Diffusion models for Handwriting Generation." In: *arXiv: abs/2011.06704* (2020). DOI: 10.48550/arXiv.2011.06704. arXiv: 2011.06704 [cs.LG].

[LM11] H. Larochelle and I. Murray. "The Neural Autoregressive Distribution Estimator." In: *Proc. of the Int. Conf. on Artificial Intelligence and Statistics*. Fort Lauderdale, FL, USA, 2011, pp. 29–37.

[LSD15] J. Long, E. Shelhamer, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA, 2015, pp. 3431–3440.

[Lu+22a] C. Lu, Y. Zhou, F. Bao, J. Chen, L. I. Chongxuan, and J. Zhu. "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps." In: *Advances in Neural Information Processing Systems*. New Orleans, LA, USA, 2022, pp. 5775–5787.

[Lu+22b] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. "DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models." In: *arXiv: abs/2211.01095* (2022). DOI: 10.48550/arXiv.2211.01095.

[Luo+23] C. Luo, Y. Zhu, L. Jin, Z. Li, and D. Peng. "SLOGAN: Handwriting Style Synthesis for Arbitrary-Length and Out-of-Vocabulary Text." In: *IEEE Trans. on Neural Networks and Learning Systems* 34.11 (2023), pp. 8503–8515. DOI: 10.1109/TNNLS.2022.3151477.

[LW07] Z. Lin and L. Wan. "Style-preserving English handwriting synthesis." In: *Pattern Recognition* 40.7 (2007), pp. 2097–2109. DOI: 10.1016/j.patcog.2006.11.024.

[LY17] J. H. Lim and J. C. Ye. "Geometric GAN." In: *arXiv: abs/1705.02894* (2017). DOI: 10.48550/arXiv.1705.02894.

[Mao+17] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. "Least Squares Generative Adversarial Networks." In: *IEEE/CVF Int. Conf. on Computer Vision*. Venice, Italy, 2017, pp. 2794–2802.

[Mat+21] A. Mattick, M. Mayr, M. Seuret, A. Maier, and V. Christlein. "SmartPatch: Improving Handwritten Word Imitation with Patch Discriminators." In: *Int. Conf. on Document Analysis and Recognition*. Ed. by J. Lladós, D. Lopresti, and S. Uchida. Lausanne, Switzerland, 2021, pp. 268–283. DOI: 10.1007/978-3-030-86549-8_18.

[May+20] M. Mayr, M. Stumpf, A. Nicolaou, M. Seuret, A. Maier, and V. Christlein. "Spatio-Temporal Handwriting Imitation." In: *European Conf. on Computer Vision Workshops*. Glasgow, United Kingdom, 2020, pp. 528–543. DOI: 10.1007/978-3-030-68238-5_38.

[May+24] M. Mayr, M. Dreier, F. Kordon, M. Seuret, J. Zöllner, F. Wu, A. Maier, and V. Christlein. "Zero-Shot Paragraph-level Handwriting Imitation with Latent Diffusion Models." In: *arXiv: abs/2409.00786* (2024). DOI: 10.48550/arXiv.2409.00786. arXiv: 2409.00786 [cs.CV].

[MB02] U.-V. Marti and H. Bunke. "The IAM-database: an English sentence database for offline handwriting recognition." In: *Int. Journal on Document Analysis and Recognition* 5.1 (2002), pp. 39–46. DOI: 10.1007/s100320200071.

[MB99] U.-V. Marti and H. Bunke. "A Full English Sentence Database for Off-Line Handwriting Recognition." In: *Int. Conf. on Document Analysis and Recognition*. Bangalore, India, 1999, pp. 705–708. DOI: 10.1109/icdar.1999.791885.

[MHN13] A. L. Maas, A. Y. Hannun, and A. Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models." In: *ICML Workshop on Deep Learning for Audio, Speech, andLanguage Processing*. Atlanta, GA, USA, 2013.

[MO14] M. Mirza and S. Osindero. "Conditional Generative Adversarial Nets." In: *arXiv: abs/1411.1784* (2014). DOI: 10.48550/arXiv.1411.1784.

[MP43] W. S. McCulloch and W. Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity." In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133. DOI: 10.1007/bf02478259.

[MP69] M. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[MTZ18] R. Mechrez, I. Talmi, and L. Zelnik-Manor. "The Contextual Loss for Image Transformation with Non-Aligned Data." In: *European Conf. on Computer Vision*. Munich, Germany, 2018, pp. 768–783.

[ND21] A. Q. Nichol and P. Dhariwal. "Improved Denoising Diffusion Probabilistic Model." In: *Int. Conf. on Machine Learning*. Virtual Conference, 2021, pp. 8162–8171.

[Nic+22] A. Q. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. Mcgrew, I. Sutskever, and M. Chen. "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models." In: *Int. Conf. on Machine Learning*. Baltimore, MD, USA, 2022, pp. 16784–16804.

[Nie03] H. Niemann. *Klassifikation von Mustern*. 2nd ed. Internet Version, Accessed: 18.06.2025. Universität Erlangen-Nürnberg, 2003.

[Nik+23] K. Nikolaidou, G. Retsinas, V. Christlein, M. Seuret, G. Sfikas, E. B. Smith, H. Mokayed, and M. Liwicki. "Wordstylist: Styled Verbatim Handwritten Text Generation with Latent Diffusion Models." In: *Int. Conf. on Document Analysis and Recognition*. San José, CA, USA, 2023, pp. 384–401. DOI: 10.1007/978-3-031-41679-8_22.

[Nik+24a] K. Nikolaidou, G. Retsinas, G. Sfikas, and M. Liwicki. "DiffusionPen: Towards Controlling the Style of Handwritten Text Generation." In: *European Conf. on Computer Vision*. Milan, Italy, 2024, pp. 417–434. DOI: 10.1007/978-3-031-73013-9_24.

[Nik+24b] K. Nikolaidou, G. Retsinas, G. Sfikas, and M. Liwicki. "Rethinking HTG Evaluation: Bridging Generation and Recognition." In: *Workshop on Critical Evaluation of Generative Models and their Impact on Society*. Milan, Italy, 2024, pp. 1–17.

[OKK16] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. "Pixel Recurrent Neural Networks." In: *Int. Conf. on Machine Learning*. New York, NY, USA, 2016, pp. 1747–1756.

[Oor+16] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu koray, O. Vinyals, and A. Graves. "Conditional Image Generation with PixelCNN Decoders." In: *Advances in Neural Information Processing Systems*. Barcelona, Spain, 2016.

[OOS17] A. Odena, C. Olah, and J. Shlens. "Conditional Image Synthesis with Auxiliary Classifier GANs." In: *Int. Conf. on Machine Learning*. Sydney, Australia, 2017, pp. 2642–2651.

[Ope+23] OpenAI *et al.* "GPT-4 Technical Report." In: *arXiv: abs/2303.08774* (2023). DOI: 10.48550/arXiv.2303.08774.

[Ope25] OpenAI. *Addendum to GPT-4o System Card: Native image generation*. Tech. rep. 2025.

[OV17] A. van den Oord, O. Vinyals, and k. kavukcuoglu koray. "Neural Discrete Representation Learning." In: *Advances in Neural Information Processing Systems*. Long Beach, CA, USA, 2017.

[Pat+16] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. "Context Encoders: Feature Learning by Inpainting." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA, 2016, pp. 2536–2544.

[PCC23] V. Pippi, S. Cascianelli, and R. Cucchiara. "Handwritten Text Generation From Visual Archetypes." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Vancouver, Canada, 2023, pp. 22458–22467.

[Pip+23] V. Pippi, F. Quattrini, S. Cascianelli, and R. Cucchiara. "HWD: A Novel Evaluation Score for Styled Handwritten Text Generation." In: *British Machine Vision Conference*. Aberdeen, UK, 2023, pp. 1–13.

[Pip+25] V. Pippi, F. Quattrini, S. Cascianelli, A. Tonioni, and R. Cucchiara. "Zero-Shot Styled Text Image Generation, but Make It Autoregressive." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Nashville, TN, USA, 2025, pp. 7910–7919.

[PPM17] G. Papamakarios, T. Pavlakou, and I. Murray. "Masked Autoregressive Flow for Density Estimation." In: *Advances in Neural Information Processing Systems*. Long Beach, CA, USA, 2017, pp. 2335–2344.

[Rab89] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition." In: *Proc. of the IEEE* 77.2 (1989), pp. 257–286. DOI: 10.1109/5.18626.

[Rad+18] A. Radford, K. Narasimhan, T. Salimansa, and I. Sutskever. *Improving Language Understanding by Generative Pre-Training*. Tech. rep. OpenAI, 2018.

[Rad+19] lec Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. *Language Models are Unsupervised Multitask Learners*. Tech. rep. OpenAI, 2019.

[Raf+20] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67.

[Ren+24] M.-S. Ren, Y.-M. Zhang, Q.-F. Wang, F. Yin, and C.-L. Liu. "DiffWriter: A Diffusion Model-Based Stylized Online Handwritten Chinese Character Generator." In: *Advances in Neural Information Processing Systems*. Vancouver, Canada, 2024, pp. 86–100.

[Ret+21a] G. Retsinas, G. Sfikas, C. Nikou, and P. Maragos. "Deformation-Invariant Networks For Handwritten Text Recognition." In: *Proc. of the IEEE Int. Conf. on Image Processing*. Anchorage, AK, USA, 2021, pp. 949–953. DOI: 10.1109/ICIP42928.2021.9506414.

[Ret+21b] G. Retsinas, G. Sfikas, C. Nikou, and P. Maragos. "From Seq2Seq Recognition to Handwritten Word Embeddings." In: *British Machine Vision Conference*. Virtual Conference, 2021, pp. 1–14.

[Ret+22] G. Retsinas, G. Sfikas, B. Gatos, and C. Nikou. "Best Practices for a Handwritten Text Recognition System." In: *Int. Workshop on Document Analysis Systems*. La Rochelle, France, 2022, pp. 247–259. DOI: 10.1007/978-3-031-06555-2_17.

[RFB15] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *Int. Conf. on Medical Image Computing and Computer-Assisted Intervention*. Munich, Germany, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors." In: *Nature* 323 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[Ria+24] N. Riaz, S. Saifullah, S. Agne, A. Dengel, and S. Ahmed. "StylusAI: Stylistic Adaptation for Robust German Handwritten Text Generation." In: *Int. Conf. on Document Analysis and Recognition*. Athens, Greece, 2024, pp. 429–444. DOI: 10.1007/978-3-031-70536-6_26.

[RM15] D. Rezende and S. Mohamed. "Variational Inference with Normalizing Flows." In: *Int. Conf. on Machine Learning*. Lille, France, 2015, pp. 1530–1538.

[RMC15] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." In: *arXiv: abs/1511.06434* (2015). DOI: 10.48550/arXiv.1511.06434. arXiv: 1511.06434 [cs.LG].

[Rom+22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. "High-Resolution Image Synthesis With Latent Diffusion Models." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. New Orleans, LA, USA, 2022, pp. 10684–10695.

[Ros58] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[RSN21] G. Retsinas, G. Sfikas, and C. Nikou. "Iterative Weighted Transductive Learning for Handwriting Recognition." In: *Int. Conf. on Document Analysis and Recognition*. Lausanne, Switzerland, 2021, pp. 587–601. ISBN: 9783030863371. DOI: 10.1007/978-3-030-86337-1_39.

[Sal+16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen. "Improved Techniques for Training GANs." In: *Advances in Neural Information Processing Systems*. Barcelona, Spain, 2016.

[San+18a] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Salt Lake City, UT, USA, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[San+18b] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. "How Does Batch Normalization Help Optimization?" In: *Advances in Neural Information Processing Systems*. Montreal, Canada, 2018.

[Sch+21] C. Schuhmann, R. Vencu, R. Beaumont, R. Kaczmarczyk, C. Mullis, A. Katta, T. Coombes, J. Jitsev, and A. Komatsuzaki. "LAION-400M: Open Dataset of CLIP-Filtered 400 Million Image-Text Pairs." In: *arXiv: abs/2111.02114* (2021). DOI: 10.48550/arXiv.2111.02114.

[SE19] Y. Song and S. Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution." In: *Advances in Neural Information Processing Systems*. Vancouver, Canada, 2019.

[Sha20] N. Shazeer. "GLU Variants Improve Transformer." In: *arXiv: abs/2002.05202* (2020). DOI: 10.48550/arXiv.2002.05202.

[SME21] J. Song, C. Meng, and S. Ermon. "Denoising Diffusion Implicit Models." In: *Int. Conf. on Learning Representations*. Vienna, Austria, 2021.

[Soh+15] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics." In: *Int. Conf. on Machine Learning*. Lille, France, 2015, pp. 2256–2265.

[Son+21] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. "Score-Based Generative Modeling through Stochastic Differential Equations." In: *Int. Conf. on Learning Representations*. Vienna, Austria, 2021.

[Sou+23] M. A. Souibgui, S. Biswas, A. Mafla, A. F. Biten, A. Fornés, Y. Kessentini, J. Lladós, L. Gomez, and D. Karatzas. "Text-DIAE: a self-supervised degradation invariant autoencoder for text recognition and document enhancement." In: *Proc. of the AAAI Conf. on Artificial Intelligence*. Washington DC, USA, 2023, pp. 2330–2338. DOI: 10.1609/aaai.v37i2.25328.

[SP97] M. Schuster and K. K. Paliwal. "Bidirectional Recurrent Neural Networks." In: *IEEE Trans. on Signal Processing* 45.11 (1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[SSA18] K. Shmelkov, C. Schmid, and K. Alahari. "How Good Is My GAN?" In: *European Conf. on Computer Vision*. Munich, Germany, 2018, pp. 218–234. DOI: 10.1007/978-3-030-01216-8_14.

[Sun+24] P. Sun, Y. Jiang, S. Chen, S. Zhang, B. Peng, P. Luo, and Z. Yuan. "Autoregressive Model Beats Diffusion: Llama for Scalable Image Generation." In: *arXiv: abs/2406.06525* (2024). DOI: 10.48550/arXiv.2406.06525.

[SVL14] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks." In: *Advances in Neural Information Processing Systems*. 2014.

[SZ15] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *Int. Conf. on Learning Representations*. San Diego, CA, USA, 2015.

[Sze+16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA, 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308.

[Sze22] R. Szeliski. *Computer Vision: Algorithms and Applications*. 2nd ed. Springer Cham, 2022. ISBN: 978-3-030-34372-9.

[TB15] L. Theis and M. Bethge. "Generative Image Modeling Using Spatial LSTMs." In: *Advances in Neural Information Processing Systems*. Montreal, Canada, 2015, pp. 1927–1935.

[TH12] T. Tieleman and G. Hinton. *Neural Networks for Machine Learning*. Lecture 6.5 - RMSProp. COURSERA, 2012.

[Tha+24] V. Thamizharasan, D. Liu, S. Agarwal, M. Fisher, M. Gharbi, O. Wang, A. Jacobson, and E. Kalogerakis. "VecFusion: Vector Font Generation with Diffusion." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Seattle, WA, USA, 2024, pp. 7943–7952. DOI: 10.1109/CVPR52733.2024.00759.

[TRG09] A. O. Thomas, A. Rusu, and V. Govindaraju. "Synthetic handwritten CAPTCHAs." In: *Pattern Recognition* 42.12 (2009), pp. 3365–3373. DOI: 10.1016/j.patcog.2008.12.018.

[TRM21] V. Tassopoulou, G. Retsinas, and P. Maragos. "Enhancing Handwritten Text Recognition with N-gram sequence decomposition and Multitask Learning." In: *Int. Conf. on Pattern Recognition*. Milan, Italy, 2021, pp. 10555–10560. DOI: 10.1109/ICPR48806.2021.9412351.

[UML13] B. Uria, I. Murray, and H. Larochelle. "RNADE: The real-valued neural autoregressivedensity-estimator." In: *Advances in Neural Information Processing Systems*. Lake Tahoe, NV, USA, 2013, pp. 2175–2183.

[Van+25] B. Vanherle, V. Pippi, S. Cascianelli, N. Michiels, F. Van Reeth, and R. Cucchiara. "VATr++: Choose Your Words Wisely for Handwritten Text Generation." In: *IEEE Trans. on Pattern Analysis and Machine Intelligence* 47.2 (2025), pp. 934–948. DOI: 10.1109/TPAMI.2024.3481154.

[Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. "Attention is All you Need." In: *Advances in Neural Information Processing Systems*. Long Beach, CA, USA, 2017.

[Vin+08] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders." In: *Int. Conf. on Machine Learning*. New York, NY, USA, 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.

[Wan+05] J. Wang, C. Wu, Y.-Q. Xu, and H.-Y. Shum. "Combining Shape and Physical Models for Online Cursive Handwriting Synthesis." In: *Int. Journal on Document Analysis and Recognition* 7.4 (2005), pp. 219–227. DOI: 10.1007/s10032-004-0131-6.

[Wan+17] P. Wang, Y. Cao, C. Shen, L. Liu, and H. T. Shen. "Temporal Pyramid Pooling-Based Convolutional Neural Network for Action Recognition." In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.12 (2017), pp. 2613–2622. DOI: 10.1109/TCSVT.2016.2576761.

[Wan+19] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao. "Learning Deep Transformer Models for Machine Translation." In: *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by A. Korhonen, D. Traum, and L. Màrquez. Florence, Italy, 2019, pp. 1810–1822. DOI: 10.18653/v1/P19-1176.

[Wan+22] Y. Wang, H. Wang, S. Sun, and H. Wei. "An Approach Based on Transformer and Deformable Convolution for Realistic Handwriting Samples Generation." In: *Int. Conf. on Pattern Recognition*. Montreal, Canada, 2022, pp. 1457–1463. DOI: 10.1109/ICPR56361.2022.9956551.

[Wan+25] Y. Wang, H. Wei, H. Wang, S. Sun, and C. He. "GL-GAN: Perceiving and Integrating Global and Local Styles for Handwritten Text Generation with Mamba." In: *Proc. of Int. Conf. on Computational Linguistics*. Abu Dhabi, UAE, 2025, pp. 2434–2444.

[WC11] A. R. Webb and K. D. Copsey. *Statistical Pattern Recognition*. Wiley, 2011. ISBN: 978-1-119-95295-4. DOI: 10.1002/9781119952954.

[WF24] F. Wolf and G. A. Fink. "Self-Training for Handwritten Word Recognition and Retrieval." In: *Int. Journal on Document Analysis and Recognition* 27.3 (2024), pp. 225–244. ISSN: 1433-2825. DOI: 10.1007/s10032-024-00484-9.

[WH18] Y. Wu and K. He. "Group Normalization." In: *European Conf. on Computer Vision*. Munich, Germany, 2018, pp. 3–19.

[WSB03] Z. Wang, E. P. Simoncelli, and A. C. Bovik. "Multiscale structural similarity for image quality assessment." In: *Asilomar Conference on Signals, Systems & Computers*. Pacific Grove, CA, USA, 2003, pp. 1398–1402. DOI: 10.1109/ACSSC.2003.1292216.

[ZC23] Q. Zhang and Y. Chen. "Fast Sampling of Diffusion Models with Exponential Integrator." In: *Int. Conf. on Learning Representations*. Kigali, Rwanda, 2023.

[ZC88] Y. T. Zhou and R. Chellappa. "Computation of Optical Flow Using a Neural Network." In: *IEEE Int. Conf. on Neural Networks*. Vol. 2. 1988, pp. 71–78. DOI: 10.1109/ICNN.1988.23914.

[ZG09] X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Springer Cham, 2009. ISBN: 978-3-031-01548-9. DOI: 10.1007/978-3-031-01548-9.

[Zha+18] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Salt Lake City, UT, USA, 2018, pp. 586–595. DOI: 10.1109/CVPR.2018.00068.

[Zha+23] W. Zhao, L. Bai, Y. Rao, J. Zhou, and J. Lu. "UniPC: A Unified Predictor-Corrector Framework for Fast Sampling of Diffusion Models." In: *Advances in Neural Information Processing Systems*. New Orleans, LA, USA, 2023.

[Zha+25] S. Zhai, R. ZHANG, P. Nakkiran, D. Berthelot, J. Gu, H. Zheng, T. Chen, M. Á. Bautista, N. Jaitly, and J. M. Susskind. "Normalizing Flows are Capable Generative Models." In: *Int. Conf. on Machine Learning*. Vancouver, Canada, 2025.

[Zhu+17] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. "Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks." In: *IEEE/CVF Int. Conf. on Computer Vision*. Venice, Italy, 2017, pp. 2223–2232.

[Zhu+23] Y. Zhu, Z. Li, T. Wang, M. He, and C. Yao. "Conditional Text Image Generation With Diffusion Models." In: *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. Vancouver, Canada, 2023, pp. 14235–14245.

[ZK16] S. Zagoruyko and N. Komodakis. "Wide Residual Networks." In: *British Machine Vision Conference*. York, UK, 2016, pp. 87.1–87.12. DOI: 10.5244/C.30.87.

[ZN21] J. Zdenek and H. Nakayama. "JokerGAN: Memory-Efficient Model for Handwritten Text Generation with Text Line Awareness." In: *ACM Int. Conf. on Multimedia*. Virtual Conference, 2021, pp. 5655–5663. DOI: 10.1145/3474085.3475713.

[ZN23] J. Zdenek and H. Nakayama. "Handwritten Text Generation with Character-Specific Encoding for Style Imitation." In: *Int. Conf. on Document Analysis and Recognition*. San José, CA, USA, 2023, pp. 313–329. DOI: 10.1007/978-3-031-41679-8_18.