

1 ABSTRACT

This research explores the development and completion of a software application that leverages live human speech to create moving media. The goal of this research is to close the gap between the development of spoken and visual content, opening up a brand-new method for producing media and conveying stories. For the purpose of converting spoken words into visually appealing and dynamic representations, the software makes use of cutting-edge voice recognition and synthesis algorithms. The research examines numerous elements and algorithms required for the precise analysis and interpretation of this task. To enable real-time processing and interpretation, deep learning models are used to extract linguistic information from the speech input. The software converts the text obtained in this way into moving media content using a modified model from image synthesis to produce moving visual media.

CONTENTS

1	Abstract	
2	Introduction	1
3	Natural Language Processing Models	3
3.1	Recurrent Neural Network (RNN).	3
3.1.1	Recurrent connections	4
3.1.2	Long Short-Term Memory (LSTM)	4
3.1.3	Gated Recurrent Units (GRUs)	5
3.1.4	Backpropagation through time (BPTT).	5
3.2	Transformer	6
3.2.1	Architecture	6
3.2.2	Tokenization	7
3.2.3	Embedding.	7
3.2.4	Positional encoding	8
3.2.5	Singe-Head Attention (Self-Attention mechanism)	8
3.2.6	Multi-Head Attention.	9
3.2.7	Masked Multi-Head Attention	10
4	Comparative analysis of speech2text models	11
5	Generative models	12
5.1	Autoencoder	12
5.1.1	Architecture	12
5.1.2	Latent space	13
5.1.3	Reconstruction Loss	13
5.2	Variational Autoencoder (VAE).	13
5.2.1	Kullback-Leibler (KL) divergence.	13
5.2.2	Evidence Lower Bound (ELBO)	14

5.3	Generative Adversarial Network (GAN)	14
5.3.1	Architecture	14
5.3.2	Adversarial process	15
5.4	Diffusion Model.	15
5.4.1	Diffusion process: Markov Chain Monte Carlo (MCMC)	15
5.4.2	Forward diffusion process	15
5.4.3	Variance schedule	16
5.4.4	Reverse diffusion process	16
5.4.5	Architecture	17
5.4.6	Training	17
5.5	Stable Diffusion Model	18
5.5.1	Contrastive Language-Image Pre-Training (CLIP)	19
6	Comparative analysis of generative models.	21
7	Development of the plan for the live-word to moving-image application	23
7.1	Methodology	23
8	Documentation for the implementation of the application	24
8.1	Software selection	24
8.1.1	Programming language	24
8.1.2	PIP packages	24
8.1.3	Operating system	25
8.2	Model selection	26
8.3	Implementation	27
8.3.1	Overview of the project's file structure	27
8.4	Performance optimisation	28
8.4.1	Hyperparameter tuning	28
8.4.2	Half-precision	29
8.4.3	Knowledge distillation	29

9	Results	32
9.1	Setup configuration	32
9.1.1	Hardware Components	32
9.1.2	Software Environment	32
9.2	Chosen hyperparameters	32
9.2.1	Audio	32
9.2.2	Video	32
9.3	Resource Utilization	33
9.3.1	Audio	33
9.3.2	Video	35
9.4	End-to-End example	36
10	Conclusion	38
10.1	Future possibilities of generative models and speech recognition.	38
10.2	Results and potential for further development of the application	38
11	References	40

2 INTRODUCTION

The ability to create intriguing and captivating content has emerged as a vital skill in a society that is becoming more and more digital and media-centric. While technological development has made it possible to produce animations and images that are aesthetically appealing, the direct incorporation of human speech into the process of producing moving media is yet uncharted ground. By concentrating on the creation and completion of a software program that can convert live human speech into graphically dynamic media, this research seeks to close this gap.

The way we interact, entertain, and deliver messages could be revolutionized by the combination of audio and visual content. Imagine a piece of software that can instantaneously translate spoken text into animated images, enhancing media output such as presentations and storytelling with a dynamic visual element that accurately represents the spoken message. A technology like this might transform the creative process by enhancing its accessibility, immersion, and engagement. This software application’s development calls for a multidisciplinary strategy that incorporates concepts from areas including speech recognition [1], [2], image synthesis [3]–[7]. The software tries to effectively analyze and interpret human speech, capturing its nuances, emotions, and linguistic components by utilizing cutting-edge approaches from these fields. The simultaneous use of computer vision techniques will produce visually appealing and contextually appropriate media by capturing and interpreting the meaning of those spoken words.

The technical difficulties of real-time speech analysis, visual synthesis, and the seamless fusion of speech and images are significant challenges in the development of the software application explored in this research. To overcome these challenges, it is essential to leverage advanced techniques and technologies. In this regard, utilizing pre-trained models proves to be a valuable strategy. The real-time extraction of phonetic and linguistic data from the speech input is one of the most important technical aspects. This job is ideally suited for deep learning models like recurrent neural networks (RNNs) [1] or Transformer-based models. [2] However, building these models from scratch might take a lot of time and money. The development process can be greatly accelerated by using pre-trained models, which have previously discovered intricate patterns and representations from enormous volumes of speech data. The pre-trained models act as a starting point, giving a strong basis on which to build and refine for the particular requirements of the software application. Additional difficulties arise from the processing requirements of real-time analysis and image generation. Real-time processing of large models necessitates a powerful computing environment. Advanced hardware accelerators, like graphics processing units (GPUs) and specialized hardware, must be integrated in this situation. These high-performance hardware platforms offer the processing power required to ef-

fectively handle complex computations. The software program can make the most of the available computational power by leveraging pre-trained models, which are frequently tailored for particular hardware configurations, in order to ensure real-time performance and responsiveness.

One of the key challenges is ensuring that the generated images should not only be visually appealing but also align with the spoken words in a coherent and meaningful manner. This can be solved by using modern architectures such as stable diffusion [7] which allows the controlled evolution of the image generation process. By tailoring such base image synthesis architecture can lead to the desired moving image effect.

3 NATURAL LANGUAGE PROCESSING MODELS

In order to close the communication gap between humans and machines, the multidisciplinary field of study known as natural language processing (NLP) [1] brings together the strengths of computer science, artificial intelligence, and linguistics. NLP has become a game-changing technology with a wide range of applications by enabling computers to comprehend, interpret, and produce human language. This involves tasks such as speech recognition, language translation, sentiment analysis, information extraction, and question answering, among others.

3.1 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to process and analyze sequential data. [1] RNNs are uniquely able to retain and use data from earlier time steps, in contrast to standard feedforward neural networks, which enables them to recognize temporal connections and patterns in sequential data. The key operation of an RNN is the update equation, which calculates the hidden state or output at each time step based on the current input and the previous state. This equation typically involves a nonlinear activation function, such as the hyperbolic tangent (tanh) or the rectified linear unit (ReLU).

The four commonly used types of Recurrent Neural Networks are: [8]

- **One-to-One:** This is not an RNN but a traditional feedforward neural network where each input corresponds to a single output. Example: Image classification
- **One-to-Many:** In this architecture, a single input is mapped to a sequence of outputs. Example: Generating image captions where a single image is used as input, and the network generates a sequence of words to describe the image.
- **Many-to-One:** Many inputs are mapped to a single output, typically used for sequence classification or sentiment analysis. Example: Sentiment analysis of movie reviews, where the network takes a sequence of words as input and predicts the sentiment of the entire review.
- **Many-to-Many:** This type of RNN takes a sequence of inputs and produces a sequence of outputs of the same length. Example: Machine translation, where the network takes a sequence of words in one language and generates a sequence of words in another language.

3.1.1 Recurrent connections

Information only moves in one direction in a typical feedforward neural network, from the input layer through hidden layers to the output layer. [9] Recurrent connections, on the other hand, inject loops into an RNN, allowing information to pass through the hidden layers from one time step to the next as well as from the input layer to the output layer. [1]

An RNN maintains an internal state or hidden representation while producing an output at each time step. The relationship between the current input and the prior hidden state determines the hidden state at a specific time step. As a result, the network is able to store information and use it later to guide predictions or judgments. Mathematically, the recurrent connection can be represented by the following equation:

$$\mathbf{h}_t = f(x_t, \mathbf{h}_{t-1})$$

3.1.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture long-term dependencies in sequential data. [1], [10] LSTMs have become a standard choice for tasks that require modeling complex sequential patterns and retaining information over extended periods. The memory cell, which acts as an information storage unit, is the main component of an LSTM. It can store data for a very long time without being impacted by changes. Three gating mechanisms are combined with the memory cell: [10]

- **Forget Gate:** Which data should be removed from the memory cell is decided by the forget gate. It generates a forget gate activation f_t that ranges from 0 to 1 for each component of the memory cell using the previous hidden state $\mathbf{h}_{t-1}$, the current input x_t , and the prior hidden state as inputs. Complete forgetting is represented by a value of 0, and complete retention by a value of 1.
- **Input Gate:** The input gate controls how new data enters the memory cell. It is made up of two parts: the candidate activation $\tilde{c}_t$ and the input gate activation i_t . While the candidate activation indicates the new candidate values for the memory cell, the input gate activation establishes the significance of the new input in updating the memory cell.
- **Output Gate:** Information transmission from the memory cell to the output is managed by the output gate. It generates an output gate activation o_t using the current

input and the prior hidden state. How much data from the memory cell should be made available to the rest of the network depends on the output gate activation.

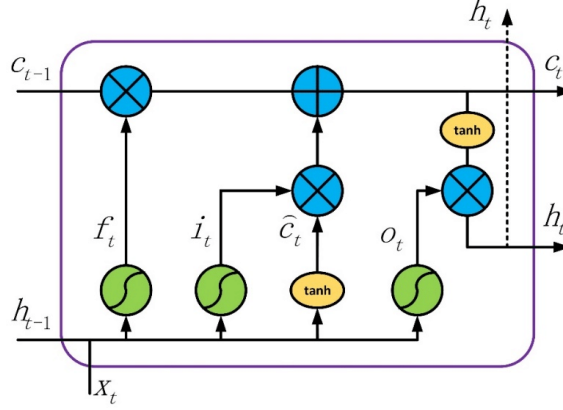


Figure 3.1: LSTM block

3.1.3 Gated Recurrent Units (GRUs)

GRUs feature an LSTM-like structure, although they have fewer gates. [11] The update gate and the reset gate are two of a GRU's most important parts. Based on the input and prior hidden state, these gates enable the GRU to selectively update and reset the hidden state.

Due to their simpler architecture than LSTM, GRUs have grown in popularity since they use fewer parameters and train and infer models more quickly. GRUs have been demonstrated to compete favorably on a number of tasks, including natural language processing, speech recognition, and machine translation, even though they might not have the same explicit memory cell as LSTM.[11]

3.1.4 Backpropagation through time (BPTT)

The way BPTT operates conceptually is by unrolling each input timestep. One copy of the network, one output, and one input timestep are all present in each timestep. Then, errors are computed and totaled for each timestep. The weights are adjusted and the network is rolled back up.

The gradient problem, where the gradients either fade or rise exponentially as they propagate through numerous time steps, can affect BPTT. It is common practice to use methods like gradient clipping and gating mechanisms (like LSTMs and GRUs) to alleviate these problems and stabilize the training process.

The RNN learns to provide predictions or sequences that better match the desired out-

puts by using BPTT iteratively, successfully capturing patterns and relationships in the sequential data. [1]

3.2 Transformer

The paper "Attention Is All You Need" [2] introduced a new kind of Sequence-to-Sequence (or Seq2Seq) architecture called the Transformer. Unlike traditional Seq2Seq models that rely on recurrent neural networks (RNNs), the Transformer architecture revolutionized machine translation and other natural language processing tasks by introducing a self-attention mechanism.

The Transformer's architecture not only improves the accuracy in variety of NLP tasks, but also allows for parallel computation, making it highly efficient for training on modern hardware. It has set new benchmarks in various natural language processing tasks, including machine translation, text summarization, question answering, and sentiment analysis. [2]

Since its introduction, the Transformer has become the de facto architecture for many state-of-the-art NLP models, and its concepts have been extended to other domains beyond language processing. The Transformer's impact on the field of deep learning underscores the importance of attention mechanisms and has paved the way for further advancements in sequence modeling and generation tasks.

3.2.1 Architecture

In many advanced neural models for sequence translation follow an encoder-decoder structure. [2] The encoder takes an input sequence of symbols and converts it into a sequence of continuous representations. These representations capture the important information from the input sequence. The decoder then takes these representations and generates an output sequence, one symbol at a time. The model works in a step-by-step manner, where it uses the previously generated symbols as additional input when generating the next symbol. This allows the model to consider the context and dependencies between symbols to produce accurate translations or predictions.

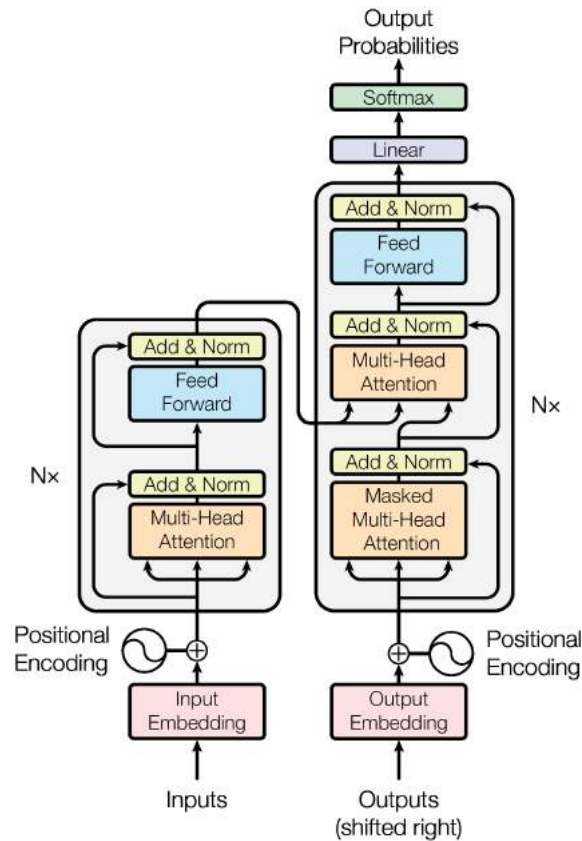


Figure 3.2: The Transformer model architecture. [2]

3.2.2 Tokenization

Tokenization is the process of encoding a string of text into transformer-readable token ID integers. Tokens can represent words, subwords, or even characters, depending on the chosen tokenization strategy. The main goal of tokenization is to convert raw text into a format that can be easily processed by the transformer model. By breaking down the input text into tokens, the model can effectively capture the sequential and contextual information present in the text. This enables the model to understand and generate meaningful representations of the input data. [2]

3.2.3 Embedding

Tokens are converted into word vectors, also known as word embeddings, using embedding layers. In the embedding space, similar words or phrases used in similar contexts frequently have comparable vector representations. As a result, the model may extrapolate and make predictions based on how closely related or similar phrases are. The embedding dimension is typically a hyperparameter that needs to be specified before training the

model. The input layer's size is being determined by the training vocabulary size, and the output layer's by the declared embedding dimension's size. The tokens are being one-hot encoded before being fed to the embedding layers. During training, the embedding layer adjusts its parameters by optimizing an objective function, typically based on the model's performance on a specific task. By converting tokens into dense vector representations, embedding layers serve a critical role in NLP models, enabling the model to comprehend and make sense of the semantic links between words. [2]

3.2.4 Positional encoding

Transformers must include positional encoding since it enables the model to distinguish between the locations of tokens in a sequence. Word position is important because it determines the sentence meaning. Transformers process tokens in parallel without a built-in sense of order, in contrast to recurrent neural networks (RNNs), which naturally collect sequential information. Positional encoding is typically represented as additional vectors added to the input embeddings of tokens. [2]

$$\text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)$$

$$\text{PE}(\text{pos}, 2i + 1) = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)$$

Figure 3.3: Positional Encoding Formulas

The sinusoidal positional encoding formula defines the values of the embeddings at each position and dimension. The values are computed using trigonometric functions and a scaling factor based on the position and the embedding dimension. The sine function is typically applied to the embeddings with even indices, while the cosine function is applied to those with odd indices. [2]

By combining the sinusoidal and cosine positional embeddings, the Transformer model gains access to a set of distinct embeddings that encode the position information in a continuous and learnable manner. This allows the model to generalize well to sequences of different lengths and capture positional relationships effectively.

3.2.5 Single-Head Attention (Self-Attention mechanism)

In case of simple attention, it selectively focuses on words based on an external query. The more determining that word for answering the query the more focus it gets. In the

context of self-attention, the term "self" refers to the fact that attention is calculated within a single sequence or context. It means that each element in the sequence attends to other elements within the same sequence. [2]

The self-attention mechanism involves three main steps:

- **Query, Key, and Value Computation:** [2] The input embeddings are transformed into three types of vectors: query vectors, key vectors, and value vectors. This transformation is typically achieved through linear projections with learned weight matrices.

For each position in the sequence, query vectors capture the information that position will use to attend to other positions, key vectors encode the information to be attended to, and value vectors hold the actual information.

- **Attention Matrix Calculation:** The attention matrix refers to a square matrix that represents the attention scores or attention values between different elements in a sequence. Attention weights, also known as attention coefficients or attention scores, are the individual values within the attention matrix.

To compute the attention weights, the dot product is taken between each query vector and the transpose of each key vector. The resulting scalar values are then scaled by the square root of the dimension of the key vectors to mitigate the effects of vanishing gradients.

A softmax function is applied to normalize the scaled dot products, resulting in attention weights that sum up to 1 for each query position.

- **Weighted Sum of Values:** Each value vector is multiplied by its corresponding attention weight and then aggregated across all positions using matrix multiplication.

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^{\top}}{\sqrt{d_k}} \right) V$$

Figure 3.4: Self-Attention formula

3.2.6 Multi-Head Attention

Multi-Head Attention is an extension of Single-Head Attention (Self-Attention) that improves the modeling capacity of the attention mechanism. It involves using multiple sets of Self-Attention mechanisms (heads) in parallel. Each attention head has its own set of learned parameters for query, key, and value transformations. The learnt parameters for query, key, and value transformations are unique to each attention head. Each head's context vectors and attention weights are calculated separately. The final representation is created by concatenating and linearly transforming the outputs of each attention head. By

paying attention to many sequence segments concurrently, multi-head attention enables the model to concentrate on various relationships or characteristics of the input sequence. It makes it possible for the model to gather many kinds of data and develop more varied and robust representations. [2]

3.2.7 Masked Multi-Head Attention

When doing autoregressive tasks, the model creates one token at a time using tokens that have already been created. When predicting the next element in a sequence, the model ideally should rely only on the information available up to that point, mimicking a real-time or left-to-right generation process. Unlike inference where we don't know the answers before hand, during the training phase the model gets provided both the source dialogues and the target dialogues. In training instead of feeding back the models own prediction the target word is being fed back. This process often being called teacher forcing. This helps the model in quantifying the difference in between the probability distribution scores across the true labels and the model's prediction. In order to prevent the model from "cheating" by paying attention to future locations and learning knowledge that it shouldn't have known, masking is added to the attention weights. This way, each token can only attend to earlier positions or itself (if Self-Attention is used). This guarantees a good procedure for the autoregressive generation. [2]

4 COMPARATIVE ANALYSIS OF SPEECH2TEXT MODELS

In this section, several speech-to-text models will be compared, namely DeepSpeech, Listen, Attend and Spell (LAS), Wave2Text, ESPnet and Whisper. By examining their advantages and disadvantages, it will aim to provide insights into the suitability of these models.

- **DeepSpeech: [12]** This model was developed by Mozilla, employs a recurrent neural network (RNN) architecture and has gained recognition for its open-source nature. One notable advantage of DeepSpeech is its competitive performance in diverse speech recognition tasks. However, it may require substantial computational resources and training data to achieve optimal results.
- **Listen, Attend and Spell (LAS): [13]** LAS models combine an encoder-decoder architecture with an attention mechanism. They excel in capturing temporal dependencies and have proven effective in automatic speech recognition (ASR) tasks. The attention mechanism enables the model to focus on relevant audio segments during transcription. However, LAS models may be complex to train and could be sensitive to background noise, requiring additional preprocessing steps.
- **Wave2Vec: [14]** Wave2Vec, a recent development by Facebook AI Research (FAIR), operates directly on raw waveforms using a combination of convolutional and transformer architectures. This approach eliminates the need for audio feature extraction and achieves state-of-the-art performance in ASR. Nonetheless, training Wave2Text models may be computationally intensive, requiring substantial resources.
- **ESPnet: [15]** ESPnet, developed by AIST, Japan, is an end-to-end speech processing toolkit that encompasses various models for speech recognition. It provides flexibility and modularity, allowing users to choose the most suitable architecture. ESPnet also offers pre-trained models, reducing the burden of training from scratch. However, it may require a deeper understanding of the toolkit and configuration for optimal performance.

–

5 GENERATIVE MODELS

By enabling robots to produce realistic and aesthetically pleasing images, generative models have transformed the field of image generation. These models create new images that mimic the original data by learning the underlying patterns and structures in a training dataset. Applications for image generation using generative models can be found in a variety of fields, such as computer vision, entertainment, and design.

Generative models have sparked remarkable progress in image generation, enabling the creation of artwork, photorealistic scenes, and even novel concepts that do not exist in the original dataset.

5.1 Autoencoder

Autoencoders are a class of neural network models used for unsupervised learning and dimensionality reduction tasks. When learning a compressed representation known as the latent space and reconstructing the original data from this representation, they excel in capturing the key characteristics and patterns of the input data. Applications for autoencoders can be found in a variety of fields, such as feature extraction, anomaly detection, and image processing.[3]

Autoencoders are trained in an unsupervised manner, meaning they do not require labeled data for training. Instead, they rely on the input data itself to learn meaningful representations. By forcing the network to learn to reconstruct the input, autoencoders extract important features and discard noise or less relevant information.

5.1.1 Architecture

The structure of an autoencoder typically consists of an encoder network, a decoder network, and a bottleneck layer representing the latent space. The encoder takes the input data and maps it to a lower-dimensional latent space representation. The decoder takes this representation and reconstructs the original input data from it.[3]

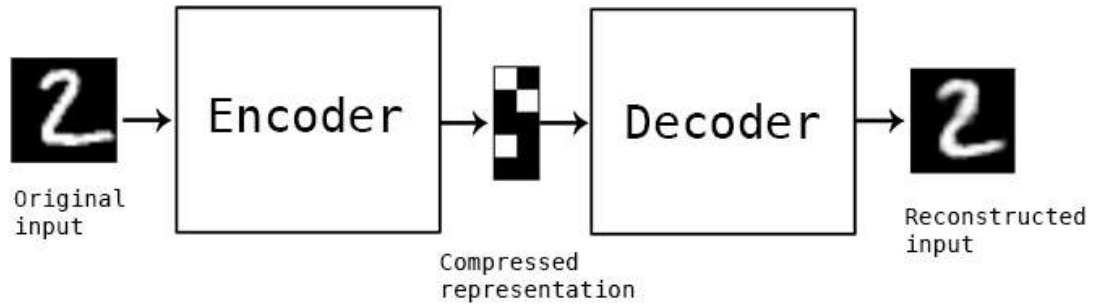


Figure 5.1: Autoencoder architecture[3]

5.1.2 Latent space

The purpose of the latent space is to capture the most important information from the input data while discarding irrelevant or noisy details. By reducing the dimensionality of the input, the autoencoder can learn a more concise and abstract representation that captures the underlying structure of the data.

The latent space itself does not have any inherent meaning or interpretability. It is a mathematical representation that captures the compressed information of the input data. However, by exploring the patterns and properties of the latent space, it is possible to gain insights into the underlying structure of the data and extract meaningful features.

5.1.3 Reconstruction Loss

MSE loss is a widely used and straightforward choice for continuous data. It calculates the average squared difference between the original input and the reconstructed output. The MSE loss is suitable when the input data follows a continuous distribution, such as in image reconstruction tasks.

5.2 Variational Autoencoder (VAE)

Variational Autoencoders (VAEs) are a type of generative model that combines the power of autoencoders with probabilistic modeling. VAEs are designed to learn a latent space that follows a specific probability distribution, typically a Gaussian distribution.[4]

5.2.1 Kullback-Leibler (KL) divergence

The Kullback-Leibler (KL) divergence is a crucial part of the loss function in variational autoencoders (VAEs). The learnt latent space is encouraged to follow a desired probability distribution using the KL divergence, which is commonly a multivariate Gaussian distribution.

A measurement of the variation between two probability distributions is the KL divergence. It quantifies the difference between the distribution of the learnt latent space and the desired probability distribution in the context of VAEs. The VAE learns to shape the latent space to fit the target distribution by reducing the KL divergence.[4]

5.2.2 Evidence Lower Bound (ELBO)

The Evidence Lower Bound (ELBO) is a fundamental idea utilized in the training and model optimization of Variational Autoencoders (VAEs). The ELBO, which acts as the objective function for training the VAE, is a lower bound on the data's log-likelihood.[16]

Application of variational inference in VAEs yields the ELBO. By using the observable data, variational inference attempts to estimate the true but frequently unreachable posterior distribution of the latent variables. To accomplish this in VAEs, an approximative posterior distribution, sometimes referred to as the variational distribution or the encoder distribution, is introduced.

$$\text{ELBO} = \mathbb{E}_{q(z|x)} [\log p(x|z)] - \text{KL} (q(z|x) || p(z))$$

The ELBO provides a lower bound on the log-likelihood of the data because the KL divergence term is always non-negative. Maximizing the ELBO during training is equivalent to maximizing the log-likelihood of the data, which is the ultimate objective in generative modeling.

5.3 Generative Adversarial Network (GAN)

An innovative development in artificial intelligence is the use of Generative Adversarial Networks (GANs). GANs, which Ian Goodfellow and his colleagues created in 2014, have transformed many fields, including picture synthesis and data production.[5]

GANs can produce artificial data that closely reflects the distributions of real data. This has important ramifications for fields like medicine or rare event prediction where gathering vast volumes of real-world data is difficult or expensive. Researchers and practitioners can supplement their datasets with GAN-generated data to create more reliable and accurate models.

5.3.1 Architecture

A generator and a discriminator are the two neural networks that make up GANs. The discriminator seeks to distinguish between authentic and fraudulent examples,

whereas the generator learns to produce realistic outputs, such as images, by creating samples from random noise. The generator and discriminator fight against one another in an iterative process, constantly honing their skills until the generator is skilled at producing believable outputs.

5.3.2 Adversarial process

The generator initially produces crude outputs, but as the discriminator provides feedback, both networks gradually refine their performance. The generator is compelled by this adversarial process to provide increasingly convincing outputs that can trick the discriminator. In order to produce high-quality outputs that closely mimic the distribution of actual data, the generator must learn the underlying patterns and properties of the training data through the use of a feedback loop.

5.4 Diffusion Model

A diffusion model or probabilistic diffusion model is a parameterized Markov chain trained using variational inference to produce samples matching the data after finite time[17]. Latent variable models can be compared to diffusion models.[6] Latent denotes a continuous feature space that is concealed. They might resemble variational autoencoders (VAEs) in this sense. Diffusion models basically work by erasing training data by adding Gaussian noise one at a time, then learning to recover the data by undoing this noise-adding process. Diffusion models can provide data that resembles the data they were trained on.

5.4.1 Diffusion process: Markov Chain Monte Carlo (MCMC)

Markov chains are mathematical models that define a system that switches between different states over time. Only the likelihood of changing the system's state can be predicted from its current condition. In other words, a system's current state contains all potential future or current states that it might take.[17]

5.4.2 Forward diffusion process

Let x_0 be a data point sampled from a real data distribution $q(x)$ ($x_0 \sim q(x)$). We define a forward diffusion process where at each step of the Markov chain we add a small amount of Gaussian noise with variance βt to x_{t-1} , producing a sequence of noisy samples $x_1, x_2, \dots, x_T$ in T steps, with each sample x_t having the distribution $q(x_t|x_{t-1})$. This diffusion process can be formulated as follows [18]:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t}x_{t-1}, \beta_t I)$$

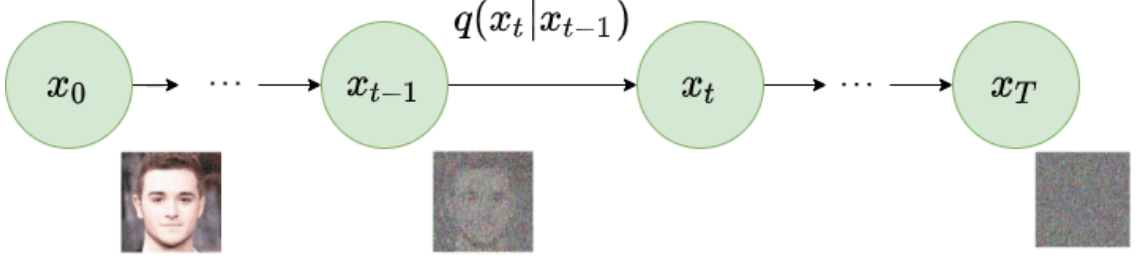


Figure 5.2: Forward diffusion process[6]

When $T \rightarrow \infty$, the final result will become a completely noisy image as if it is sampled from an isotropic Gaussian distribution. We can go from the input data x_0 to x_t in a closed form from the input data in a tractable way. Mathematically, this is the posterior probability and is defined as [18]:

$$q(x_{1:T}|x_0) = \prod_{t=1}^T q(x_t|x_{t-1})$$

5.4.3 Variance schedule

The variance parameter βt can be fixed to a constant or chosen as a schedule over the T timesteps. Variance schedule can be defined in several ways, can be linear, quadratic, cosine etc. The original DDPM (Denoising Diffusion Probabilistic Model) authors utilized a linear schedule increasing from $\beta_1 = 10^{-4}$ to $\beta_1 = 0.02$. [6] Nichol et al. 2021 showed that employing a cosine schedule works even better. [18]

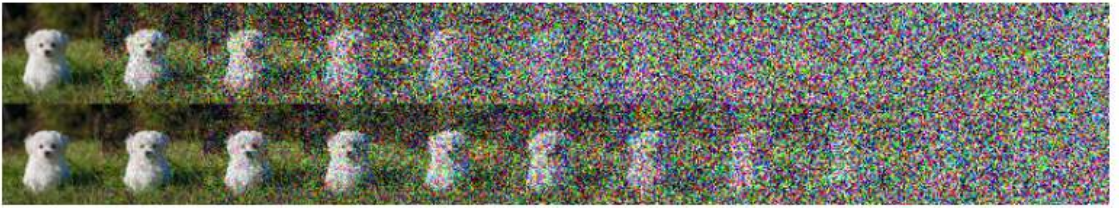


Figure 5.3: Latent samples from linear (top) and cosine (bottom) schedules respectively[18]

5.4.4 Reverse diffusion process

As $T \rightarrow \infty$, the latent x_T is nearly an isotropic Gaussian distribution. In this process the model needs to learn the reverse distribution $q(x_{t-1}|x_t)$, then it can sample x_T from $\mathcal{N}(0, I)$. The reverse process can acquire a sample from $q(x_0)$, generating a novel data point from the original data distribution.[6] The model approximate

$q(x_{t-1}|x_t)$ with a parameterized model p_0 (e.g. a neural network). Since $q(x_{t-1}|x_t)$ will also be Gaussian, for small enough β_t , p_0 can be chosen as Gaussian and just parameterize the mean and variance:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

By applying the reverse formula for all timesteps ($p_0(x_{0:T})$), also called trajectory, the model can go from x_T to the data distribution:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

5.4.5 Architecture

Ho et al. employed a U-Net neural network architecture.[6] U-Net is a symmetric architecture that uses skip connections between encoder and decoder blocks of the same feature dimension and input and output that are of the same spatial size. The input image is often downscaled, then upscaled, till it reaches its original size.[19] In the original implementation of DDPMs, the U-Net consists of Wide ResNet blocks, group normalization as well as self-attention blocks. The diffusion timestep t is specified by adding a sinusoidal position embedding into each residual block.

5.4.6 Training

A Diffusion Model is trained by finding the reverse Markov transitions that maximize the likelihood of the training data. The Kullback-Leibler (KL) divergence is a crucial step in the training of diffusion models. Diffusion steps are a common set of steps used in the training of Diffusion Models. The model gradually modifies a noise sample to resemble the desired data distribution in each diffusion phase. The difference between the intermediate distribution obtained during the diffusion stage and the actual data distribution is measured using the KL divergence.

At each step of the diffusion process, the main goal is to reduce the KL divergence between the intermediate distribution and the target distribution. To accomplish this, the model parameters are changed to direct the transformation process. The model learns to provide samples that progressively approximate the true data distribution by minimizing the KL divergence.

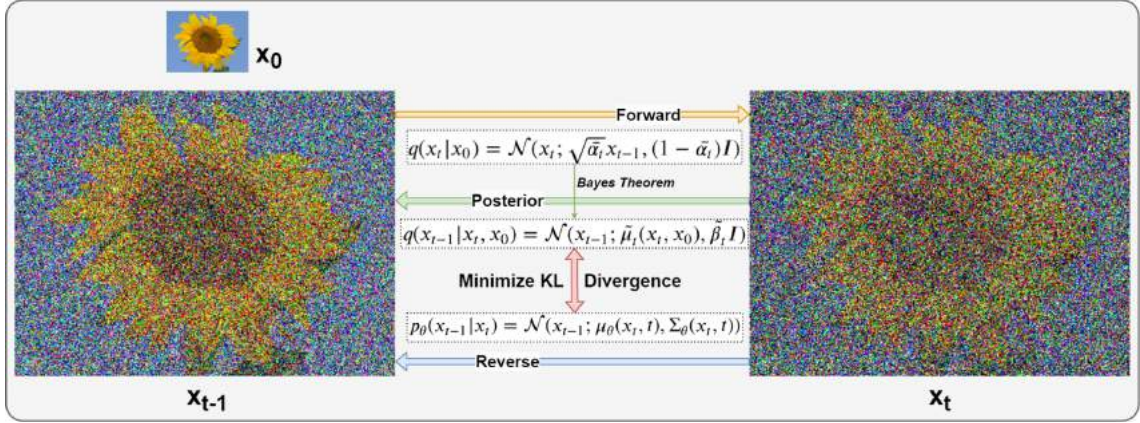


Figure 5.4: KL divergence application

In order to create a joint loss function during training, the KL divergence is often paired with a reconstruction loss or likelihood term. While the KL divergence component supports the alignment of the intermediate distribution with the data distribution, the reconstruction loss pushes the model to provide high-quality samples. The diffusion model learns to convert noise samples into realistic samples that approximate the real data distribution by improving this joint loss function.

The authors take inspiration from VAEs and reformulate the training objective using a variational lower bound (VLB), also known as “Evidence lower bound” (ELBO).[6] After some simplifications of the original ELBO formula like fixing the variance to a constant β_t , the authors settled on a simplified formula. As they have kept the variance constant, minimizing KL divergence is as simple as minimizing the difference (or distance) between means (μ) of two gaussian distributions q and p .

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t, x_0, \epsilon} [\|\epsilon - \epsilon_\theta(\sqrt{a_t}x_0 + \sqrt{1-a_t}\epsilon, t)\|^2]$$

$$L_{\text{DM}} = \mathbb{E}_{t, x_0, \epsilon} [\|\epsilon - \epsilon_\theta(x_t, t)\|^2]$$

They employed a "Mean Squared Error" between the noise added during the forward phase and the noise predicted by the model as their final loss function for training DDPMs.[6]

5.5 Stable Diffusion Model

Latent diffusion models are based on a relatively straightforward concept: we project the input into a smaller latent space and apply the diffusion there rather than directly applying the diffusion process to a high-dimensional input.

Rombach et al. suggested encoding the input into a latent representation $z_t = g(x_t)$, using an encoder network. The rationale for this choice is to process the input in a lower dimensional space, which will reduce the computational requirements of training diffusion models. A decoder network then upsamples the newly generated data using a typical diffusion model (U-Net).[7]

The typical diffusion model loss formula is being formed by adding an encoder $\mathcal{E}$ and a latent representation z . The latent diffusion model (LDM) loss is:

$$L_{\text{LDM}} = \mathbb{E}_{t, \mathcal{E}(x), \epsilon} [\|\epsilon - \epsilon_{\theta}(z_t, t)\|^2]$$

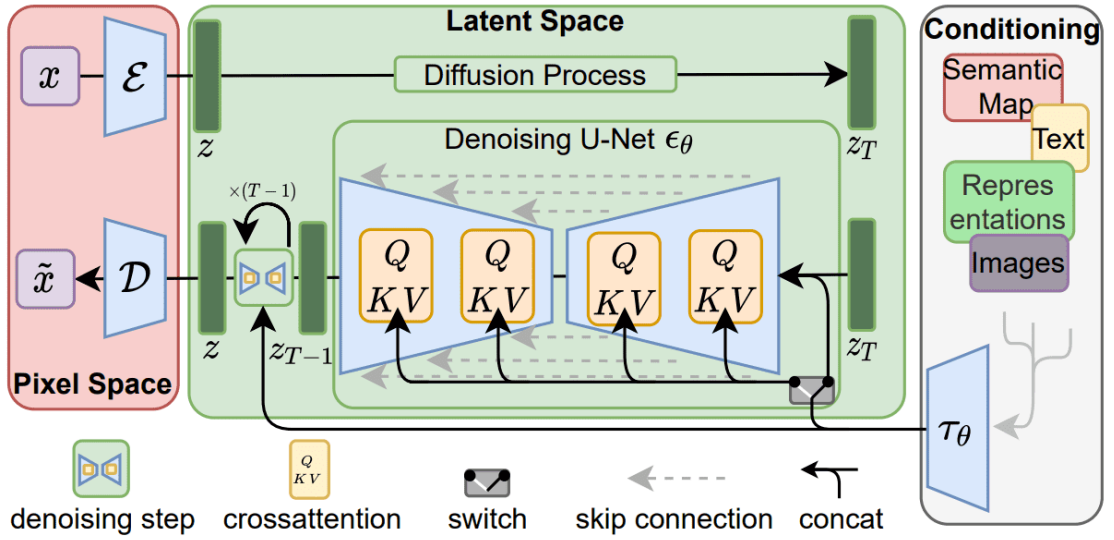


Figure 5.5: Stable Diffusion architecture[7]

5.5.1 Contrastive Language-Image Pre-Training (CLIP)

The Contrastive Language-Image Pre-Training (CLIP) architecture enables self-supervised learning of combined representations of images and text. OpenAI first presented it in 2021. By teaching a model to comprehend the relationship between written descriptions and visual content, CLIP strives to close the gap between language and vision.[20]

The main concept behind CLIP is to make use of a sizable dataset that contains pairings of photographs and the textual descriptions that go with them. The model is trained to recognize which pictures and phrases have a semantic connection and ought to be placed closer to one another in the joint embedding space. Using a contrastive learning target, this is accomplished.

By conditioning the diffusion model on relevant textual prompts, CLIP can help steer the generation process towards desired image characteristics, resulting in more controlled and coherent outputs. CLIP can be used to evaluate and score the quality of generated samples during the diffusion process. By comparing the generated samples with textual descriptions using CLIP's image-text matching capabilities.

6 COMPARATIVE ANALYSIS OF GENERATIVE MODELS

In this section, several pre-trained generative models will be compared for image synthesis, including StyleGAN, BigGAN, VQ-VAE, WhisperGAN, PixelCNN, and pre-trained Stable Diffusion Models. By examining their advantages and disadvantages, we aim to provide insights into the suitability of these models for different applications.

- **StyleGAN: [21]** StyleGAN is a pre-trained generative model known for its ability to generate high-resolution and visually impressive images. It incorporates style-based synthesis, allowing control over various aspects of image generation, such as style mixing and attribute manipulation. StyleGAN produces diverse and realistic outputs, making it well-suited for various image synthesis tasks. However, training StyleGAN from scratch can be computationally intensive and requires a large amount of data.
- **BigGAN: [22]** BigGAN is a pre-trained generative model that excels in generating high-quality and diverse images. It utilizes a conditional generative adversarial network architecture, enabling the generation of images conditioned on specific attributes or classes. BigGAN achieves state-of-the-art performance in terms of image quality and diversity. However, training BigGAN from scratch can be challenging and resource-intensive.
- **VQ-VAE: [23]** VQ-VAE is a pre-trained generative model that combines variational autoencoders with vector quantization. It learns a discrete latent representation of the input images, enabling efficient and high-quality image synthesis. VQ-VAE offers control over generated outputs by manipulating the latent variables, making it suitable for applications that require explicit control over image generation. However, VQ-VAE may produce slightly lower quality outputs compared to GAN-based models.
- **PixelCNN and PixelRNN: [24]** PixelCNN and PixelRNN are autoregressive generative models that generate images pixel by pixel. They model the conditional distribution of each pixel given the previous pixels in the image. These models have been successful in generating high-quality images but can be computationally expensive.
- **Pre-trained Stable Diffusion Models: [7]** Pre-trained Stable Diffusion Models, such as Diffusion Models and Denoising Diffusion Probabilistic Models (DDPM), are generative models that operate based on the diffusion process. These models generate high-quality images by gradually adding noise to a given image. Pre-trained Stable Diffusion Models offer stable training dynamics, controllable image synthesis, and competitive image quality. They provide the advantage of controlling the trade-off between image quality and computation by adjusting the number of diffusion steps. However, training

these models can still be computationally expensive and time-consuming. When it comes to realism and image quality, both StyleGAN and Stable Diffusion Model have established themselves as leaders in generating visually realistic and high-quality images. However, VQ-VAE, BigGan also offer impressive results, albeit with slight variations in quality. In terms of diversity, StyleGAN, Stable Diffusion models shine by producing diverse samples that encompass a wide range of image variations. Pre-trained Stable Diffusion Models can also generate diverse outputs by adjusting the number of diffusion steps. Finally, PixelCNN, with its autoregressive structure, generates images with notable quality and detail, though it may not reach the same level of realism as the more recent models. Overall, these models provide users with a spectrum of options, each with its own strengths and considerations in terms of realism, quality, and diversity.

7 DEVELOPMENT OF THE PLAN FOR THE LIVE-WORD TO MOVING-IMAGE APPLICATION

7.1 Methodology

In this research, the process commences with the capture of spoken words through a microphone, followed by the transformation of the collected audio data through a speech-to-text neural network. The resulting text is then passed to a video synthesis model, where it undergoes a transformation into a visual representation, bridging the gap between auditory input and visual output.

To generate dynamic visual content for the user, the program utilizes the initial spoken sentence as the input sentence and the subsequent sentence as the end sentence making an effort to produce a smooth transition between the two sentences. Furthermore, each new incoming sentence will be employed as the successor to the previous end sentence, forming a continuous chain of connections.

This desktop-based application is designed with a modular architecture, allowing for the flexibility to swap each component, thus enabling the customization of various styles and influencing factors, such as inference speed. The only limitation imposed is the hardware configuration.

The software's recording component must be adaptable to diverse environmental scenarios, ensuring the acquisition of optimal audio files. After being collected, the raw files are preprocessed to make them fit the speech-to-text model's format. The audio model processes the data, and the resulting output serves as the input for my video synthesis model.

Both the resolution and the quantity of transition frames are completely adjustable. There's also a requirement that the transition frames demonstrate meaningful, aesthetic, morphological, or other relevant relationships to one another in alignment with the task. For video synthesis a modified version of stable diffusion is being used in the software. Two components of this model, namely CLIP and VAE, utilize information compression. These components harness the principle of an abstract multi-dimensional space that encodes a meaningful internal representation of externally observed events. The concept involves pushing the appropriate starting and ending information through either of its encoders in the resulting space. This process enables the creation of a path between the two meanings, traversing the space and landing on points that bear a relationship with the meaning transition between the original sentences.

The hardware used to run the program and its configuration both affect how quickly inference will occur in real time.

8 DOCUMENTATION FOR THE IMPLEMENTATION OF THE APPLICATION

8.1 Software selection

8.1.1 Programming language

I have chosen to utilize Python as my main programming language for this project because of the following strong aspects:

- **Rich Ecosystem:** Python boasts a rich ecosystem of libraries and frameworks tailored to machine learning and deep learning. Libraries like TensorFlow, PyTorch, Keras, and scikit-learn provide a wide range of tools and resources for deep learning tasks.
- **Flexibility:** It is a versatile language, making it easy to experiment with different algorithms and models. It supports both object-oriented and functional programming paradigms, allowing developers to choose the approach that best suits their needs.
- **Community Support:** Deep learning enthusiasts, researchers, and developers use this language increasingly frequently these days. Help and resources may be found more easily because to the community's support, which is reflected in the wealth of tutorials, documentation, and open-source contributions.
- **Readability and Maintainability:** Python's readable and simple syntax makes code more maintainable and lowers the learning curve. For working on projects and troubleshooting intricate deep learning models, this is crucial.
- **Cross-Platform Compatibility:** Because it is cross-platform, users can create deep learning models on a range of operating systems. This adaptability is especially useful for developers and researchers who operate across various platforms.
- **Integration:** It is simple to integrate with various tools, programs, and languages. When integrating deep learning with visualization, data preprocessing, or model deployment in real-world applications, this is essential.

8.1.2 PIP packages

PIP is a command-line tool that allows user to easily install, manage, and remove Python packages and libraries. Python packages are collections of code and resources that are created and shared by the Python community to extend Python's capabilities. These are the essential packages that requires for the application to operate:

- **cv2 (OpenCV):** OpenCV is a versatile computer vision library, and it's useful in your project for tasks such as image processing and manipulation. It pro-

vides a wide range of tools for image and video analysis, making it valuable for preprocessing and handling visual data.

- **pytorch (PyTorch):** PyTorch is a popular deep learning framework with dynamic computational graphs, making it more flexible for certain deep learning tasks. Its extensive library and strong support for GPU acceleration are well-suited for developing and training deep learning models.
- **numpy:** It is fundamental for numerical computing in Python. It provides efficient data structures for arrays and matrices, which are essential for manipulating data in deep learning models and other numerical operations.
- **matplotlib:** It is a powerful library for creating visualizations and plots. It is useful for displaying data and model performance, which aids in insight-gathering and efficient result communication.
- **threading:** Tasks can be completed in parallel by using threading, which is the process of creating and managing threads within an application. This is crucial for maximizing an application's efficiency, particularly in situations involving real-time processing.
- **SpeechRecognition:** This package provides a high-level interface to capture audio from a microphone, and it offers a versatile and customizable configuration to adapt to different speech recognition requirements. The captured audio data can be processed and passed on for speech recognition or other forms of analysis in an application.
- **transformers:** Transformers provides APIs to quickly download and use those pretrained models on a given text, optimizing them on any given dataset, and then share them with the community on Hugging Face's hub. Simultaneously, every Python module that describes an architecture is completely independent and modifiable to enable quick research experiments. With a smooth integration between them, the three most widely used deep learning libraries PyTorch, TensorFlow, and Jax support Transformers. Models can be easily trained with one and then loaded for inference using the other.
- **diffusers:** For the purpose of creating pictures, sounds, and even three-dimensional structures of molecules, Diffusers is the library of choice for cutting edge pre-trained diffusion models. Whether for a simple inference solution or training own diffusion models. Diffusers is a modular toolbox that supports both.

8.1.3 Operating system

I decided to use Linux operating system for this project because it provides a stable, adaptable, and performance-focused environment that is well-suited to the requirements of deep learning development. While it may have a steeper learning curve for some users, the benefits it offers in terms of control, performance, and commu-

nity support make it a wise choice for deep learning practitioners. Here are some detailed benefits:

- **Open Source and Customization:** Because it is open-source, users can alter the system to suit their own requirements and have access to the source code. In the field of deep learning, where specific configurations and optimizations might be needed, this degree of flexibility is invaluable.
- **Performance:** Linux is known for its excellent resource management and performance. This is particularly important for training large neural networks in deep learning, where performance is the most important factor.
- **Terminal and Scripting:** It offers a robust command-line interface that is perfect for managing data, conducting experiments, and automating deep learning-related tasks. Shell commands and bash scripting are necessary for productive workflow.
- **Parallel Computing:** When working with large datasets or complex models, distributed and parallel computing is essential, and Linux is well-suited for this kind of work. In addition to supporting cluster computing, it fully utilizes multi-core CPUs.
- **Containerization:** Tools like Docker and Kubernetes work seamlessly, enabling containerization and orchestration of deep learning workloads. This simplifies deployment and scaling of models.
- **Integration with Research and Development Tools:** Deep learning tools, libraries, and frameworks are frequently created and tested on Linux, resulting in higher compatibility and performance.
- **Free and Cost-Effective:** Because Linux distributions are usually open-source and free, they're an affordable option, particularly for projects or research with limited funds.

8.2 Model selection

To create the best possible project, the selection of the best models is of utmost importance. First off, the top models provide cutting-edge performance in their specialized fields. With its Transformer design, the Whisper model has proven to be incredibly accurate and skilled at speech recognition tasks. This guarantees a highly accurate transcription of the spoken words into text. Similar to this, the stable diffusion image synthesis model, which is renowned for its capacity to produce high-quality and varied images, ensures that the outputs will be aesthetically pleasing and will meet or surpass user expectations.

Second, selecting the finest models promotes flexibility and adaptation. Whisper is one of many transformer-based models that are recognized for their capacity to manage a variety of input sequences and learn intricate patterns. This versatility en-

ables the program to take into account various speaking tenors, accents, and speech variances, ensuring reliable transcription in a variety of contexts. In a similar vein, the diffusion image synthesis model’s stability enables constant and trustworthy image production independent of the input circumstances.

8.3 Implementation

8.3.1 Overview of the project’s file structure

Here is an overview of the project’s file structure.

```
speech2video_live
├── models
│   ├── base_pipe.py
│   ├── sd_pipe.py
│   └── speech2text_pipe.py
├── notebooks
│   ├── sd_test.ipynb
│   ├── s2t_test.ipynb
│   └── vae_mnist.ipynb
├── live_run.py
└── utils.py
```

- **live_run.py** This is the main file of the project, responsible for initializing the models and creating the pipeline for the speech-to-video process. The primary hyperparameters for the project can be easily configured by passing them as parameters to the script. It initiates a thread for the speech-to-text pipeline to run concurrently with the video generation pipeline. The input speeches are gathered in a queue, which is then forwarded to the respective pipeline for preprocessing. Subsequently, a model processes the preprocessed data, and the resulting text sentences are directed to the second pipeline.
- **base_pipe.py** It’s an abstract class for the pipelines, common features are specified here.
- **sd_pipe.py** The script initializes essential components, including a tokenizer, text encoder, VAE (Variational Autoencoder), and U-Net. Additionally, it defines a function responsible for creating transitions between sentences.
- **speech2text_pipe.py** The script takes input audio signals, processes them,

and returns the corresponding text sentences.

- **utils.py** There exists a collection of utility functions within the script, serving diverse purposes such as measurements, diagnostic tools, and visualization functions. These utilities are employed in various classes across the project.
- **notebooks** This comprises a collection of notebooks primarily utilized for debugging and visualization purposes.

8.4 Performance optimisation

In this project, the primary objectives were twofold: to enable the application to run efficiently on low-end hardware, while simultaneously achieving the seamless and immediate conversion of spoken language into synthesized video content, all with a strong emphasis on real-time inference speed. Therefore, optimization is not merely a consideration but an imperative. Without it, the potential introduction of delays and latency during speech-to-text and video synthesis processes could lead to user frustration, ultimately undermining the application’s usability and overall effectiveness. Furthermore, the resource-intensive nature of the used models highlights the need for optimization. Efficiently optimized models not only reduce the computational load but also enhance the application’s adaptability to a wide array of hardware.

8.4.1 Hyperparameter tuning

Hyperparameters are configuration settings that are not learned from the data but instead are set by the machine learning engineer or researcher before the training process begins. Common hyperparameters include learning rates, batch sizes, the architecture of the model, regularization terms, and more. The goal of hyperparameter tuning is to find the ideal combination of these settings that results in the best model performance for a given task.

In the context of image generative models, the two primary hyperparameters that can significantly influence the model’s behavior are the model’s architectural size and the resulting image resolution. These hyperparameters play a central role in determining the model’s capacity, the complexity of the generated images, and the computational resources required for inference. In addition frame rate stands as another pivotal hyperparameter. The frame rate directly influences the temporal smoothness and visual quality of the generated videos, making it a critical factor in determining the overall effectiveness and user experience of the application.

This list contains a comprehensive list of the hyperparameters that were important:

- **Model sizes:** It typically refers to the number of parameters in the model, often measured in terms of the total parameter count. Generally, larger models with a greater number of parameters require more computation and memory

during inference. This results in longer inference times. The relationship between model size and inference time is often non-linear, meaning that a small increase in model size can lead to a disproportionately larger increase in inference time. Larger models place higher demands on computational resources, including CPUs, GPUs, or specialized hardware like TPUs. In many cases, this can lead to resource constraints, longer wait times, and, in extreme cases, out-of-memory errors when working with limited hardware.

- **Generated image dimensions:** Larger images contain more pixels, resulting in increased computational load during inference. Deep learning models process each pixel, and the number of operations required to analyze a larger image is significantly higher compared to a smaller one. This increased computational load translates to longer inference times. Larger images require more memory for processing. Deep learning models store intermediate activations and weights, and the memory requirements scale with the image size. When working with limited memory, loading and processing larger images may lead to memory constraints, slower access times, or even out-of-memory errors.
- **Frame rate:** Higher frame rates require the generation of more frames per second, which demands more computational resources and time for processing.
- **Sampling steps:** The number of sampling steps is the number of iterations that Stable Diffusion performs in response to the text prompt, going from random noise to a recognizable image. Generally speaking, the more sampling steps you use, the more detail you will add to your image at the expense of a longer processing time.

8.4.2 Half-precision

Half precision, often denoted as "float16" in computing, is a data format used to represent floating-point numbers with reduced precision compared to the more common "float32" format. In neural networks and deep learning, half precision is frequently used to reduce memory requirements during training and inference. Although it reduces precision, it's often sufficient for many deep learning tasks and accelerates model training and deployment, especially on GPUs and TPUs.

8.4.3 Knowledge distillation

In the upcoming section, I will explore the enhancements and optimizations that have led to increased speed in the speech-to-text model. Knowledge distillation (KD) is a compression technique in which a smaller student model is trained to reproduce the behaviour of a larger teacher one. Compared to minimising the CE

loss between the student model’s predictions and the training labels, KD allows the student model to learn from the full predictive distribution of possible next tokens in a given context, rather than just maximising the probability of the next target token in the training data.

In the following lines I will explain the concept using the Whisper model. To train the Whisper model, we assume a dataset where each example $(X_{1:T}, y_{1:N})$ is an (audio, text) pair. The model is trained using the standard cross-entropy (CE) loss, where the model is trained to predict an instance class by maximising the estimated probability of the target class labels[25]:

$$L_{CE} = - \sum_{i=1}^N P(y_i | y_{<i}, H_{1:M})$$

There are various techniques available, and here are a few of them:

- **Shrink and Fine-Tune[25]:** The most fundamental distillation technique involves reducing the teacher model to a smaller student size and training the student using the aforementioned Cross-Entropy (CE) objective. In line with the approach introduced by Shleifer & Rush (2020), we apply layer-based compression, initializing the student model by duplicating weights from strategically chosen layers of the teacher model. For instance, when creating a 2-layer student model from a 32-layer teacher model, we replicate the 1st and 32nd layers from the teacher to the student.
- **Pseudo Labelling[25]:** In the pseudo-label setting, they replace the ground truth text transcription $y_{1:N}$ with the teacher’s generation $\hat{y}_{1:N'}$ for the corresponding input audio $X_{1:T}$:

$$L_{PL} = - \sum_{i=1}^{N'} P(y_i | \hat{y}_{<i}, H_{1:M})$$

This form of distillation can be viewed as “sequence-level” KD, where knowledge is transferred from the teacher model to the student model across a sequence of generated pseudo-labels.

- **Kullback-Leibler Divergence[25]:** In the KL Divergence setting, the full probability distribution of the student model P_i is trained to match the full distribution of the teacher model Q_i by minimising the KL divergence over the entire set of next possible tokens at position i :

$$L_{KL} = \sum_{i=1}^N KL(Q_i, P_i)$$

This can be viewed as “word-level” KD, where knowledge is transferred from the teacher model to the student model via the logits over the possible tokens.

The KL Divergence is attractive since it provides information over all classes and has less variance in gradients than the CE loss.

Objective[25] The final KD training objective is a weighted sum of the KL and PL terms:

$$L_{KD} = \alpha_{KL}L_{KL} + \alpha_{PL}L_{PL}$$

where α_{KL} and α_{PL} are scalar weights for the KL and loss terms respectively, $\alpha_{KL} = 0.8$ and $\alpha_{PL} = 1.0$.

9 RESULTS

9.1 Setup configuration

In this section, I detail the setup used for my research, including key hardware components and software environment.

9.1.1 Hardware Components

- **Central Processing Unit (CPU):** Intel i5-10600K (12) @ 4.800GHz
- **Graphics Processing Unit (GPU):** NVIDIA GeForce RTX 3060 Ti, 8GB VRAM
- **Memory (RAM):** 16GB DDR4
- **Storage:** 256GB NVMe SSD, 3TB HDD
- **Microphone:** BEHRINGER Ultravoice XM-8500
- **Audio interface:** Focusrite Scarlett Solo (3rd Gen)

9.1.2 Software Environment

- **Operating System:** Ubuntu 22.04.3 LTS x86_64
- **Programming language:** Python 3.10.12
- **ML framework:** Pytorch 2.0.1
- **Other Tools:** CUDA 11.8, cuDNN 8.7

9.2 Chosen hyperparameters

9.2.1 Audio

Audio-related hyperparameters depend on both the hardware and environmental conditions, ensuring adaptability to varying acoustic environments. In my case, I've opted for a dynamic energy threshold and set the pause threshold to 0.2 seconds.

9.2.2 Video

To achieve the lowest latency, I had to identify the sampling rate that is not only the smallest but also yields the best results. After multiple trials and comparisons with different models, a sampling step of 10 appears to be the optimal choice. The visual difference beyond step 10 is so negligible that there's no justification to go beyond it.



Figure 9.1: Comparision of different generation step numbers

Given that the project involves video creation, the resolution of each frame is not a crucial factor, as it won't be prominently visible. I've determined that an image shape of 256 is sufficient for the project's needs.



Figure 9.2: Comparision of different generation image size

For achieving visually appealing transitions that are both smooth and swift, I've opted for a frame rate of 10 frames per transition. This setting strikes a balance between a pleasing visual experience and a rapid transition speed.



Figure 9.3: 10 frames/transition: a visually appealing blend of speed and smooth transitions

9.3 Resource Utilization

9.3.1 Audio

Distil-Whisper aims to condense the original Whisper model into a faster, smaller version while maintaining the model's resilience to various acoustic circumstances (speakers, speaking styles, and domains). The original writers uses a wide range

of speech recognition datasets to test Distil-Whisper’s ability to generalize across datasets and domains in order to explore this capability. In my tests, I evaluated the models using various random sentences, each with a batch size of 1. The results are in the following images.

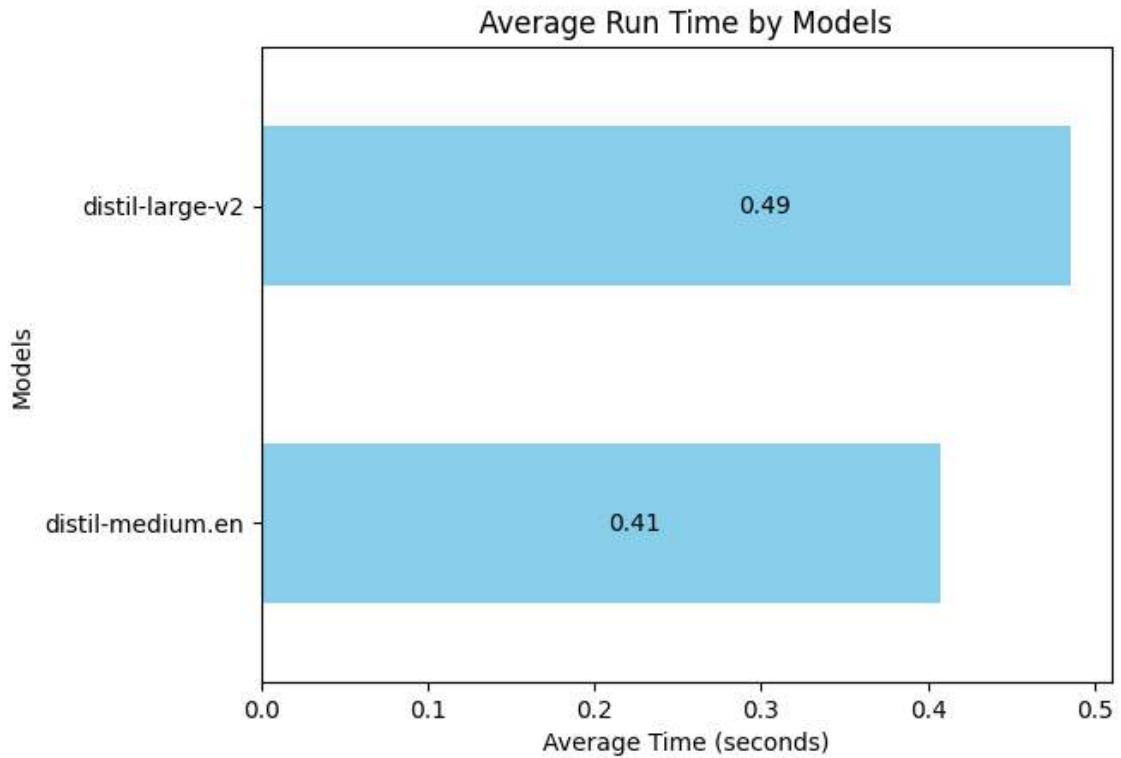


Figure 9.4: Inference speed comparsion among speech to text models

Considering the negligible speed difference depicted in the image above, I opted for the larger model. This decision stems from the importance of accurately collecting sentences, a crucial factor in the video generation process.

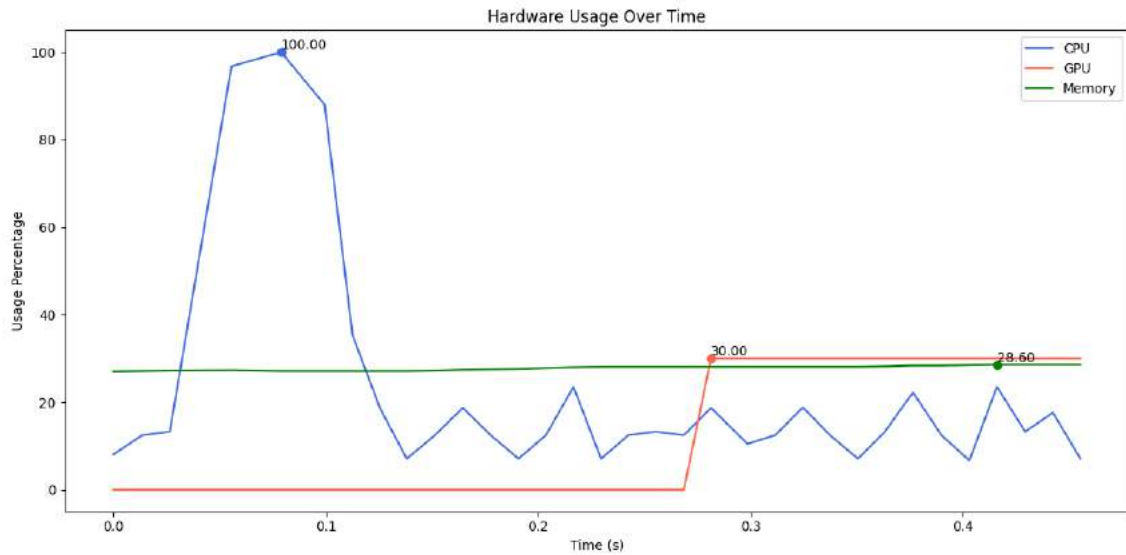


Figure 9.5: Hardware usage of distil-large-v2 model during one sentence transcription

9.3.2 Video

I've curated a selection of models that strike the right balance between image generating quality and speed for my video synthesis project. The following images depict a speed comparison among different models, employing the selected hyper-parameters.

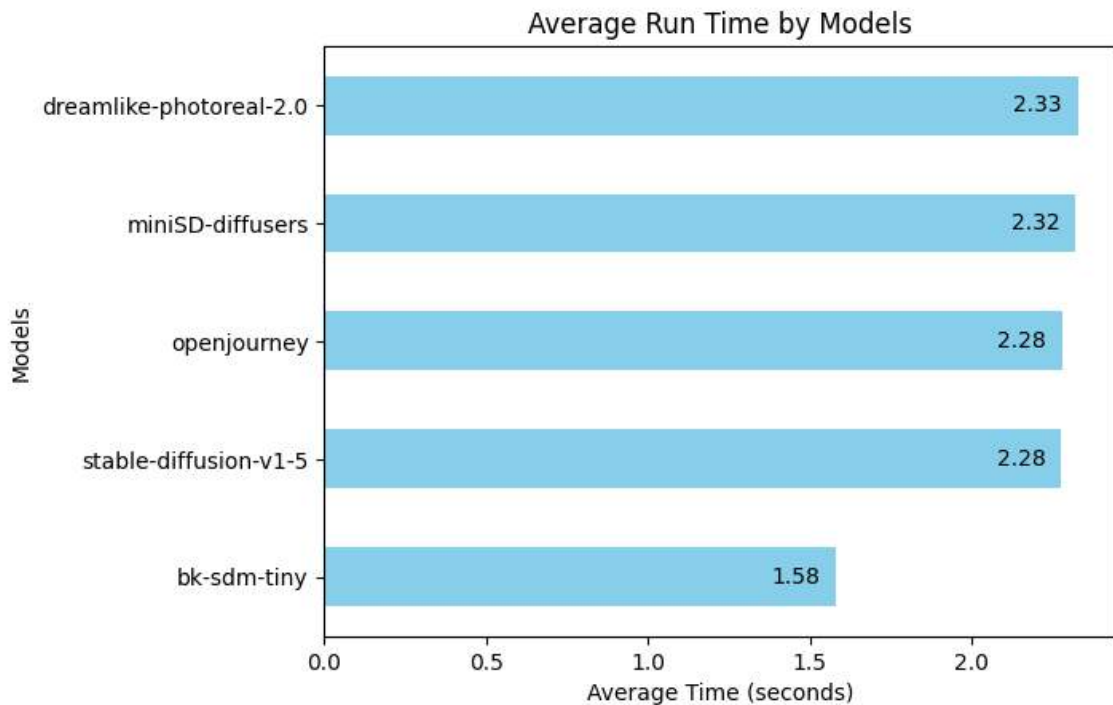


Figure 9.6: Inference speed comparsion among generative models

Given the model selection, BK-SDM-Tiny (Block-removed Knowledge-distilled Stable Diffusion Model) exhibited the best performance, prompting my decision to proceed with further work using this model. This model is built by distilling pretraining on only 0.22M LAION pairs (less than 0.1% of the entire training set) and by removing multiple residual and attention blocks from the U-Net of Stable Diffusion v1.4. Because of transferred knowledge, this compact model can mimic the original SDM even though it was trained with very few resources.

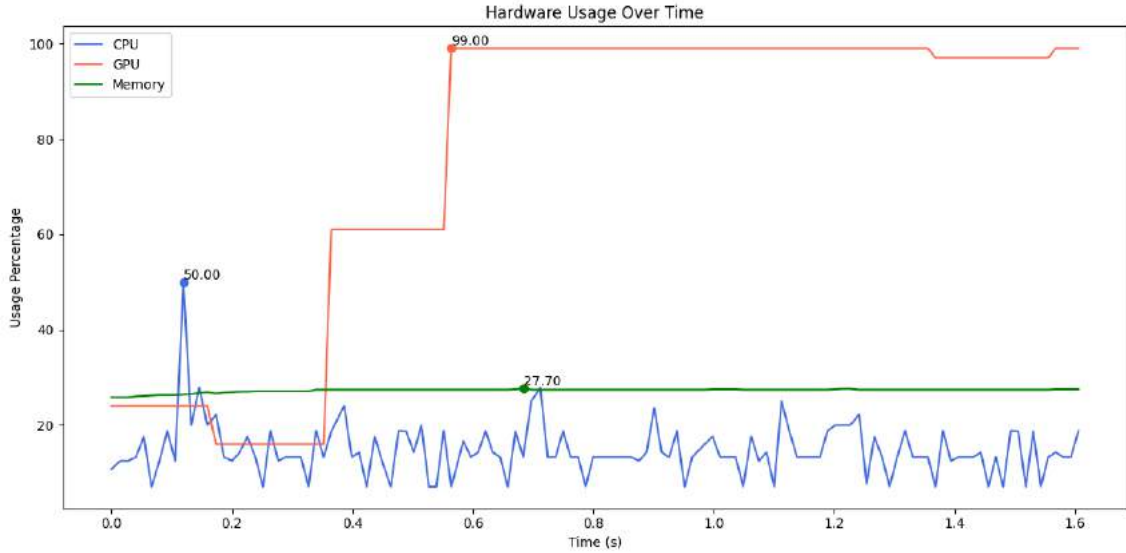


Figure 9.7: Hardware usage of BK-SDM-Tiny model during one video transition generation

9.4 End-to-End example

In this segment, I present a comprehensive demonstration featuring an illustrative narrative. The sentences are curated through a speech-to-text model, and the video synthesizer model then generates the corresponding video frames.

Story: *As the sun dipped below the horizon, fireflies painted the evening sky with tiny flickers of light. The friends gathered around a makeshift campfire, roasting marshmallows and sharing stories under the twinkling stars. One night, a gentle rain shower graced the meadow, creating a symphony of pitter-patters. The friends danced in the rain, their laughter echoing through the peaceful meadow. With each passing day, the meadow became a haven of simple joys, where friendships bloomed like wildflowers. The friends and companions cherished the magic of their enchanted meadow, forever bound by the warmth of shared moments.*

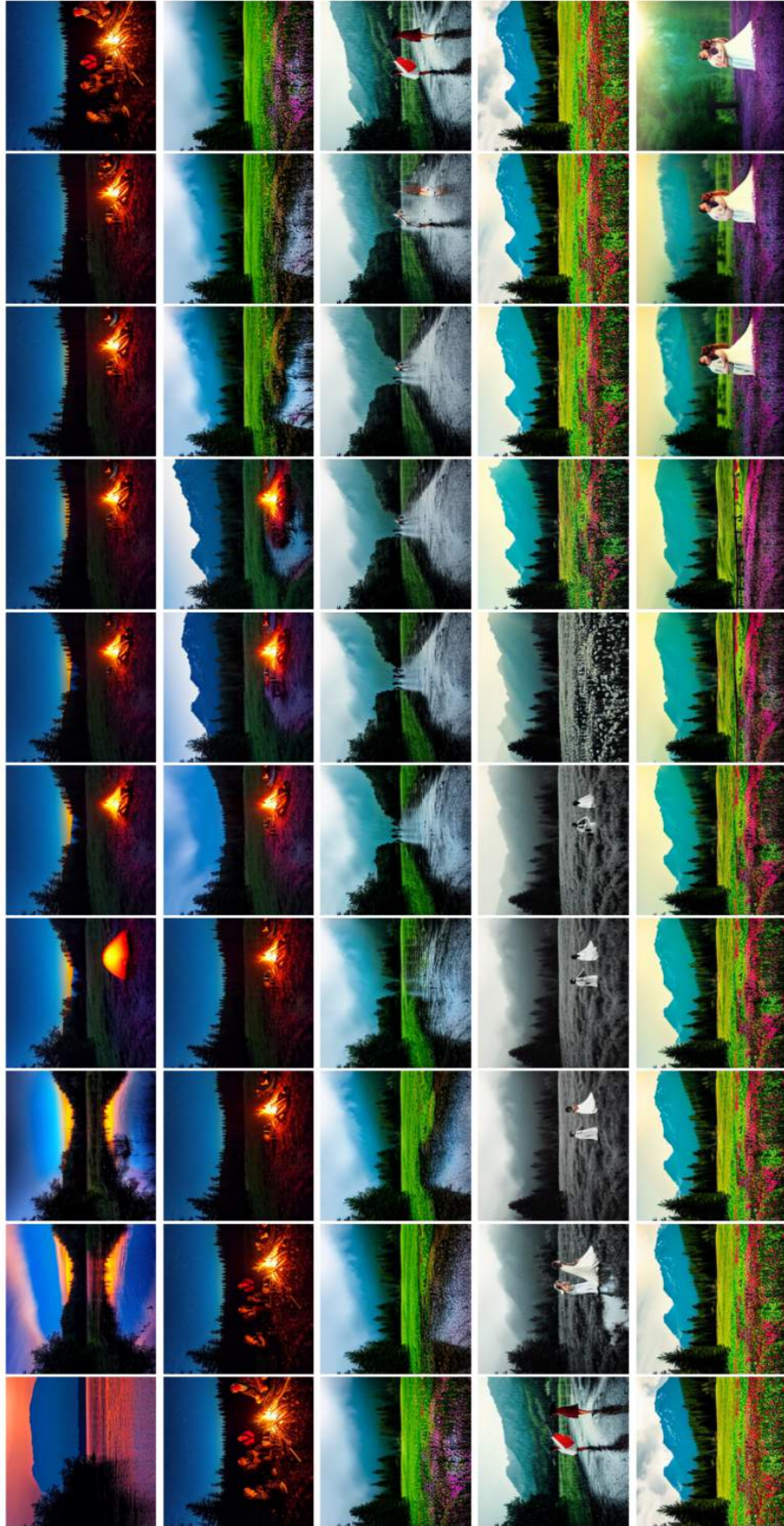


Figure 9.8: Illustration for the story generated by the video synthesis model

10 CONCLUSION

10.1 Future possibilities of generative models and speech recognition

New avenues for creative expression could be opened up by generative models, leading to completely original works of music, multimedia, and digital art. The lines separating human and machine collaboration in the arts may blur, producing unexpected and distinctive artistic results. The possibility of extremely customized user experiences might grow to unprecedented levels. These models have the potential to completely change the way people interact with digital platforms by creating virtual environments that are customized to their preferences and generating content that is unique to each user. It is possible that generative models will aid in the development of dynamic learning environments. Training scenarios, simulations, and instructional materials could be created dynamically based on the unique learner profiles of each individual, changing in real time to maximize the learning process. There may be a big change in human-AI cooperation in the future. Generative models have the potential to revolutionize human-machine collaboration in a variety of fields, including design, research, and decision-making.

Speech-to-text models have the potential to greatly improve accessibility for people with a variety of needs. Future models may be more sophisticated and flexible, allowing for the smooth translation of spoken language into text in a variety of languages and dialects. Speech-to-text models could greatly improve accessibility for people with different needs. More sophisticated and flexible models may be developed in the future, allowing for the smooth translation of spoken language into text in a variety of languages and dialects. Speech-to-text models have the potential to be essential in the development of transformative tools in education. Among the theoretical possibilities are enhanced language learning applications, personalized learning through spoken interactions, and automatic transcription of lectures.

10.2 Results and potential for further development of the application

The results of our implemented system's evaluation are encouraging. With remarkable accuracy, the speech-to-text model was able to extract spoken sentences. The efficient processing speed was discovered to guarantee the real-time translation of spoken words into text. Problems like ambient noise or other environmental factors were reduced by using the appropriately configured settings.

The extracted sentences were efficiently visualized by the video synthesis model, which produced a logical and synchronized representation of the spoken content. Visual components improved the overall quality of the synthesized output by being in line with the speech's context. The user experience was smooth and responsive with minimal latency in real-time visualization during live interactions.

Other features could be added to improve user interaction. Avatar customization, in-the-moment visual element editing, and interactive elements that react to spoken commands or particular keywords are among the available options.

Finally, it is critical to prioritize user inclusivity and accessibility. To guarantee a diverse and inclusive user base, future developments should investigate multi-lingual support, accessibility features for users with disabilities, and user-friendly interfaces.

11 REFERENCES

- [1] A. Sherstinsky, „Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network”, *Physica D: Nonlinear Phenomena*, 404th vol., p. 132–306, Mar. 2020, ISSN: 01672789. DOI: 10.1016/j.physd.2019.132306.
- [2] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, „Attention is all you need”, in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.
- [3] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, Apr. 3, 2021.
- [4] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, Dec. 10, 2022.
- [5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, *Generative adversarial networks*, Jun. 10, 2014.
- [6] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, Dec. 16, 2020.
- [7] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, Apr. 13, 2022.
- [8] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, Dec. 14, 2014.
- [9] Z. Zhang, „Pattern classification based on radial basis function neural network”, Jun. 1, 2020, pp. 213–216. DOI: 10.1109/ICSGEA51094.2020.00052.
- [10] F. Kegong, M. Jiang, J. Lu, *et al.*, „A LSTM algorithm estimating pseudo measurements for aiding INS during GNSS signal outages”, *Remote Sensing*, 12th vol., p. 256, Jan. 10, 2020. DOI: 10.3390/rs12020256.
- [11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, Dec. 11, 2014.
- [12] A. Hannun, C. Case, J. Casper, *et al.*, *Deep speech: Scaling up end-to-end speech recognition*, Dec. 19, 2014.
- [13] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, *Listen, attend and spell*, Aug. 19, 2015.
- [14] S. Schneider, A. Baevski, R. Collobert, and M. Auli, *Wav2vec: Unsupervised pre-training for speech recognition*, Sep. 11, 2019.
- [15] S. Watanabe, T. Hori, S. Karita, *et al.*, *ESPnet: End-to-end speech processing toolkit*, Mar. 30, 2018.

- [16] J. Lucas, G. Tucker, R. B. Grosse, and M. Norouzi, „Don’ t blame the ELBO! a linear VAE perspective on posterior collapse”, in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.
- [17] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, Nov. 18, 2015.
- [18] A. Nichol and P. Dhariwal, *Improved denoising diffusion probabilistic models*, Feb. 18, 2021.
- [19] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, May 18, 2015.
- [20] A. Radford, J. W. Kim, C. Hallacy, *et al.*, *Learning transferable visual models from natural language supervision*, Feb. 26, 2021.
- [21] T. Karras, S. Laine, and T. Aila, *A style-based generator architecture for generative adversarial networks*, Mar. 29, 2019.
- [22] A. Brock, J. Donahue, and K. Simonyan, *Large scale GAN training for high fidelity natural image synthesis*, version: 2, Feb. 25, 2019.
- [23] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu, *Neural discrete representation learning*, version: 2, May 30, 2018.
- [24] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, *Pixel recurrent neural networks*, Aug. 19, 2016.
- [25] S. Gandhi, P. von Platen, and A. M. Rush, *Distil-whisper: Robust knowledge distillation via large-scale pseudo labelling*, Nov. 1, 2023.