A project report on

DiffScaler: Text Guided High Bitrate Novel Video Generation with Human face swap using Diffusion Pair Up-scale Temporal Transformer

Submitted in partial fulfillment for the award of the degree of

B. Tech Computer Science and Engineering with Specialization in Artificial Intelligence

by

GUNTAKA TRIDEV REDDY (20BCI7190)

Under the Guidance of

DR. SIBI CHAKKARAVARTHY S

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT-AP UNIVERSITY

AMARAVATI, AP, INDIA

MAY 2024

DECLARATION

I hereby declare that the thesis entitled “DiffScaler: Text Guided High Bitrate Novel Video Generation with Human face swap using Diffusion Pair Up-scale Temporal Transformer” submitted by me, for the award of the degree of B. Tech- Computer Science and Engineering with Specialization in Artificial Intelligence is a record of bonafide work carried out by me under the supervision of Dr. Sibi Chakkaravarthy S.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati

Date: **Signature of the Candidate**

CERTIFICATE

This is to certify that the Senior Design Project titled "**DIFFSCALER: TEXT GUIDED HIGH BITRATE NOVEL VIDEO GENERATION WITH HUMAN FACE SWAP USING DIFFUSION PAIR UP-SCALE TEMPORAL TRANSFORMER**" that is being submitted by **GUNTAKA TRIDEV REDDY (20BCI7190)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. SIBI CHAKKARAVARTHY S
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner 1 **Internal Examiner 2**

Approved by

DEAN
School Of Computer Science and Engineering

ABSTRACT

The generation of novel high-bitrate videos from textual prompt is one of the most challenging tasks in the current times. Adding to the complexity, the task of changing character faces in generated videos while maintaining consistent quality and data integrity is complicated. In this paper, we address this problem by presenting a solution that produces high-resolution videos and allows swapping of character faces using IMG2IMG diffusion.

We propose a Diffusion Pair Upscale Temporal Transformer integrated within a U-Net backbone in the Stable Diffusion-XL model to generate high-quality videos. This architecture is capable of swapping character faces as instructed. The experimental results shown in this paper prove to be a massive potential for filmmakers and especially the production industry, offering a path to visualize their scripts like never before

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Sibi Chakkaravarthy S, Associate Professor, School of Computer Science and Engineering, VIT-AP University, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Artificial Intelligence.

I would like to express my gratitude to Dr. G. Viswanathan (Chancellor Sir), Dr. S.V. Kota Reddy (Vice Chancellor Sir), and Dr. Ch. Pradeep Reddy (Dean, SCOPE), for providing an environment to work in and for his inspiration during the tenure of the course.

In a jubilant mood I express ingeniously my whole-hearted thanks to Dr. Reeja S. R. (HoD, Department of Artificial Intelligence and Machine Learning), all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: **Name of the student**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

S. No	In-Short	Abbreviation
1.	T2I	Text to Image
2.	T2V	Text to Video
3.	CLIP	Contrastive Language-Image Pretraining
4.	SD	Stable Diffusion
5.	SR	Super Resolution

Chapter 1
Introduction

1.1 INTRODUCTION

The emergence of advanced video generation techniques has revolutionized numerous industries, spanning from entertainment to production and beyond. One particularly intriguing advancement in this domain is the capability to produce high-quality videos based on textual annotations, facilitating the creation of personalized and intricately detailed visual content. However, generating high-bitrate videos, crucial for lifelike partnerships and authentic human facial exchanges, poses a formidable challenge.

In this paper, we introduce an innovative approach for crafting compelling high-bitrate videos featuring dynamically changing human faces triggered by input. Our methodology harnesses the combined prowess of the diffusion model and time inverters to tackle the intricate complexities of video synthesis. Central to our approach is a novel tool dubbed the DiffScaler, which plays a pivotal role in enhancing the form and fidelity of synthesized video frames, thereby ensuring temporal consistency and authenticity.

This technique enables meticulous reconstruction of facial features and expressions while guaranteeing realistic and consistent temporal changes in facial appearance. Extensive testing on diverse datasets demonstrates the efficacy of our method in generating high-definition videos aligned with input descriptions. The findings underscore notable enhancements in visual fidelity and bitrate efficiency compared to existing methodologies. Such advancements hold profound implications for applications in video editing, animation, and personalized content creation, empowering creators with a potent tool for crafting tailor-made video experiences. In essence, our research presents a groundbreaking and efficient solution to the challenges associated with high-bitrate video generation and input-driven human facial exchanges. The amalgamation of propagation models with the temporal transformers offered by DiffScaler introduces novel paradigms in the field, paving the way for fresh avenues of innovation and creativity in video synthesis.

1.2 AIM OF THE PROJECT

This project aims to create a cutting-edge system for making videos guided by text, generating high-quality, lifelike videos with the ability to switch between human faces using broadcast patterns, and introducing a new tool called DiffScaler to enhance video resolution. Alongside ensuring consistent timing, the goal is to offer a robust and user-friendly tool for both personal and professional video creation. This system will cater to diverse industries like animation, video editing, and digital content production, delivering videos that look seamless, natural, and engaging.

1.3 OBJECTIVES OF THE PROJECT

The primary goal of this study is to construct a sturdy framework for generating videos guided by text, accurately translating text descriptions into visual representations. A key aspect of this framework involves integrating advanced face editing techniques to facilitate authentic and seamless swapping of human faces within the generated videos, ensuring natural facial expressions and consistent movement. A significant focus is placed on achieving high-bitrate videos characterized by clarity and detail, adhering to established standards.

The initial phase of the project will utilize diffusion models to generate a set of basic videos. This serves as a foundation for the video system, capturing essential elements from input text descriptions. Subsequently, a new tool called DiffScaler will be developed to enhance the resolution of these initial videos while maintaining temporal coherence. This technique aims to facilitate smooth and realistic transitions between frames, with a specific emphasis on the dynamic aspects of human faces.

Moreover, the research endeavors to provide content creators with a potent tool for crafting personalized and captivating video content tailored to industries such as animation, video editing, and digital content production. Finally, the practical feasibility of the proposed method will be explored across various domains, showcasing its versatility and impact in video generation, conversion, and creation industries, as well as its potential applications in real-world scenarios.

Chapter 2

Literature Review

2.1 SURVEY ON EXISTING SYSTEM

TITLE: CogVideo: Large-scale Pre Training for Text-to-Video Generation via Transformers
YEAR: 2022
AUTHOR: Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, Jie Tang
DESCRIPTION:
Pre-trained large models have made significant strides in text generation (e.g., GPT-3) and text-to-image generation (e.g., DALL-E and CogView). However, they often come with substantial computational costs. As a result, training them from scratch becomes impractical, especially for text-to-video datasets. Their usage in video generation presents challenges due to the scarcity and inconsistency of relevant data, hampering the model's capability to interpret motion accurately.

To address these hurdles, a dedicated text-to-video converter called CogVideo was introduced, boasting 9 billion parameters. To mitigate computational expenses, a pre-trained text-to-image model, CogView2, was devised, leveraging its functionalities as a basis. This strategy significantly alleviates the demand for extensive computational resources in contrast to training a model entirely from the ground up.

TITLE: Tune-A-Video: One-Shot Tuning of Image Diffusion Models for Text-to-Video Generation
YEAR: 2023
AUTHOR: Jay Zhangjie Wu, Yixiao Ge, Xintao Wang, Stan Weixian Lei, Yuchao Gu, Yufei Shi, Wynne Hsu, Ying Shan, Xiaohu Qie, Mike Zheng Shou
To emulate the achievements seen in text-to-image (T2I) generation, recent efforts involve utilizing large video datasets to train text-to-video (T2V) generators. However, despite yielding promising outcomes, this approach proves to be computationally demanding for

such models. In light of this challenge, a novel T2V generation system called One-Shot Video Tuning has been introduced, wherein only a single text-video pair is provided.

Their model builds upon state-of-the-art text-to-image (T2I) diffusion models that were initially trained on extensive image datasets. Two primary observations are highlighted: 1) T2I models demonstrate the ability to generate vertically oriented images depicting verbs, and 2) extending the T2I model to concurrently generate multiple images showcases a comprehensive and well-rounded depiction of the content. To enhance comprehension of ongoing motion, Tune-A-Video is introduced, incorporating an optimized spatio-temporal attention mechanism along with an efficient single-shot tuning system.

Theoretically, the DDIM inversion is employed to offer structural guidance for modeling. Through comprehensive qualitative and numerical experiments, the remarkable potential of their method across various applications is unveiled. This pioneering approach notably diminishes computational expenses while upholding the production of high-quality videos, thereby pushing the boundaries of video generation capabilities beyond conventional constraints.

TITLE: Make-A-Video: Text-to-Video Generation without Text-Video Data
YEAR: 2022
AUTHOR: Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, Yaniv Taigman
DESCRIPTION:
They introduce Make-A-Video, aiming to directly transpose recent significant advancements in Text-to-Image (T2I) generation to Text-to-Video (T2V). The underlying intelligence of their platform is straightforward: understanding the world through pairs of text-image data, a concept they elucidate.

They propose a simple yet efficient method for crafting T2I models, incorporating innovative

spatio-temporal modules. Firstly, they decompose the constant U-Net and attention tensor to account for both space and time. Secondly, they devise a spatiotemporal pipeline for handling high-frame-rate videos, comprising a video decoder, interpolation model, and two super-resolution models, facilitating various applications beyond T2V.

Across spatial and temporal resolutions, faithfulness to text, and overall quality, Make-A-Video establishes a new benchmark in text-to-video generation, as evidenced by qualitative and quantitative assessments. This novel approach leverages existing T2I enhancements to significantly elevate T2V generation, offering robust solutions while addressing previous limitations.

2.2 GAPS IDENTIFIED

Despite the significant advancements in T2V models, numerous gaps and challenges persist. Pre-trained large transformers like GPT-3 for text and DALL-E for text-to-image have set prominent benchmarks, but their application in video generation remains constrained by high computational costs and the absence of high-quality text-video datasets. Particularly, the complexity of motion portrayal limits the model's understanding. Although models like CogVideo, which leverage pre-trained text-to-image models and employ multi-frame rate hierarchical training techniques, show promise, they still encounter scalability limitations and struggle to perform effectively in real-world scenarios.

Approaches like One-Shot Video Tuning aim to reduce computational expenses by utilizing text-video pairs and extending T2I patterns to generate multiple images simultaneously. However, achieving continuous motion learning remains challenging, relying heavily on DDIM inversion for system guidance, which can be intricate when dealing with complex models and abundant data. Make-A-Video presents a method for translating T2I advancements into T2V by discerning visual cues from text-image pairs and motion from

unsupervised video frames. This T2V training model is faster and eliminates the need for text-video data pairs. Nonetheless, the decomposition of temporal U-networks and attention tensors, along with the construction of spatio-temporal pipelines, require sophisticated loading algorithms that may not be viable for all applications.

2.3 PROBLEM STATEMENT

The project aims to create an advanced system for producing high-bitrate, precise video content guided by text annotations, with a specific emphasis on transforming human facial features. By harnessing the diffusion model for initial video generation and introducing a novel diffusion pair, coupled with enhancing resolution through an upscaled temporal transducer, the project endeavors to deliver high-quality videos while maintaining temporal coherence. This system addresses tasks such as animation, video editing, and digital content creation, focusing on generating appealing and natural expressions and movements through the integration of sophisticated face-swapping techniques. The overarching objective is to provide a potent, efficient, and versatile tool for both personal and professional video generation, substantiated through extensive application and comparison with current state-of-the-art methods.

Chapter 3
Requirement Analysis

3.1 SYSTEM SPECIFICATION

3.1.1 HARDWARE SPECIFICATION

• Not explicitly detailed in the document, but for faster training and inference a system with high powered GPU is preferred.

3.3.2 SOFTWARE SPECIFICATION

• Programming Languages: Python

• Frameworks: PyTorch or Tensorflow

Chapter 4

Project Description

4.1 DATASET

The WebVid 10M dataset comprises a curated collection of video samples, each meticulously optimized and organized for training purposes. Each sample within the dataset is represented as a directory containing a sequence of frames extracted from the corresponding video. Notably, the number of frames included in the dataset adheres to a standard resolution of 512x512, ensuring consistency across the data group, albeit fewer than previous iterations. Accompanying each sample is a .txt file containing textual descriptions or hints linked to the video content, furnishing contextual information to facilitate training for text-guided video generation. This structured data format serves as a valuable asset for researchers and practitioners in the fields of computer vision and multimedia, facilitating the development and evaluation of novel systems with semantic understanding capabilities.

```
+ training_samples
--- + sample_001
------- + frame_0.jpg
------- + frame_1.jpg
------- + ...
------- + frame_n.jpg
------- + prompt.txt
--- + sample_002
------- + frame_0.jpg
------- + frame_1.jpg
------- + ...
------- + frame_n.jpg
------- + prompt.txt
```

Figure 1 Dataset Hierarchy

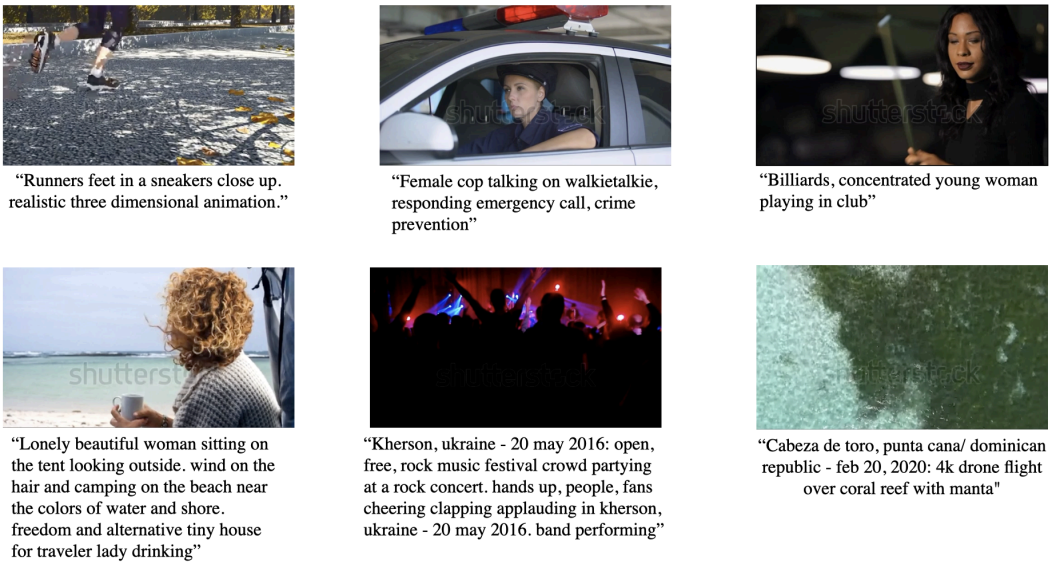

Figure 2 WebVid DATASET

4.2 CLIP ARCHITECTURE

Essentially, CLIP (Contrastive Language-Image Pretraining) comprises two primary components: a visual encoder and a text encoder. The visual encoder analyzes images and extracts comprehensive visual features, while the text encoder encodes textual descriptions, translating them into latent representations of meaning. These encoded inputs are then mapped into a shared space, where their similarity is evaluated using contrastive learning. Through optimization based on pairs of images and text with high similarity and low dissimilarity, CLIP learns to associate images with their corresponding textual contexts.

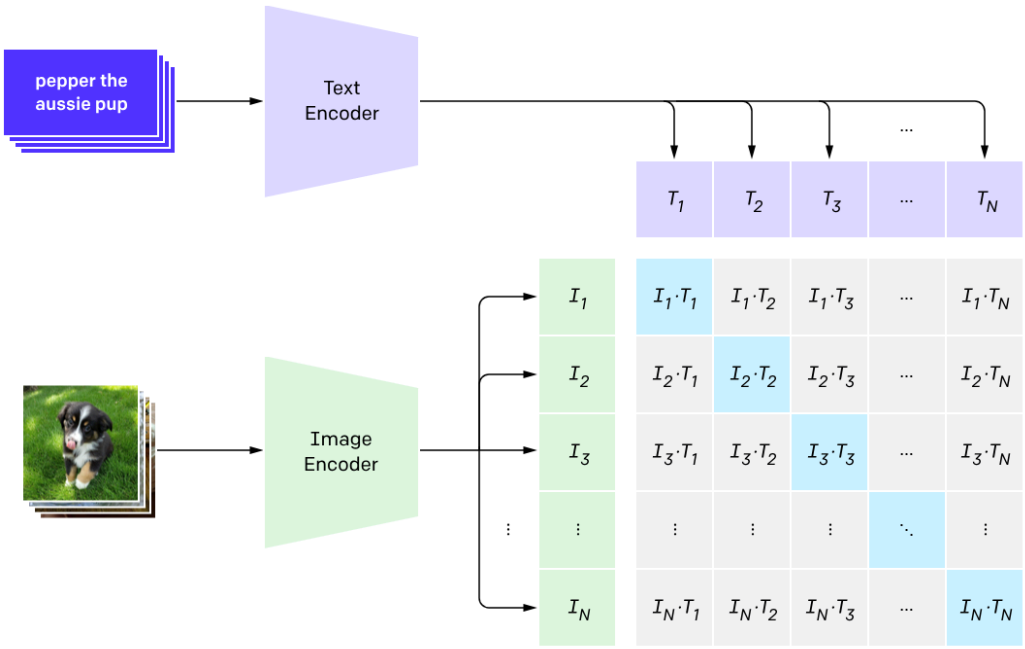

Figure 3 CLIP ARCHITECTURE

4.3 BASIC STABLE DIFFUSION ARCHITECTURE

In the text processing stage, a text encoder module is employed to convert the input text description into a numeric embedding sequence. These embeddings encapsulate the semantic logic of the textual description, facilitating downstream operations.

Subsequently, these embeddings undergo various transformations within the pipeline of the U-Net architecture. The U-Net, a convolutional neural network (CNN) commonly utilized in image classification tasks, plays a crucial role in extracting features from the embeddings, aiding in image generation. Additionally, an autoencoder, a novel neural network architecture, is utilized to reconstruct the input data. In this context, the autoencoder refines the generated image, ensuring its accuracy and alignment with the specifications in the embedding.

The scheduler module orchestrates the progression of the generation process, governing the operations of the U-net and autoencoder modules, and managing the fusion of their outputs. This control mechanism dictates the frequency of application and the synthesis of outputs from these modules. Finally, the processed image data is passed to the image decoder module, which oversees the generation of the final image. Working behind the scenes, the image decoder translates the processed data back into a visible image, thereby completing the image generation process.

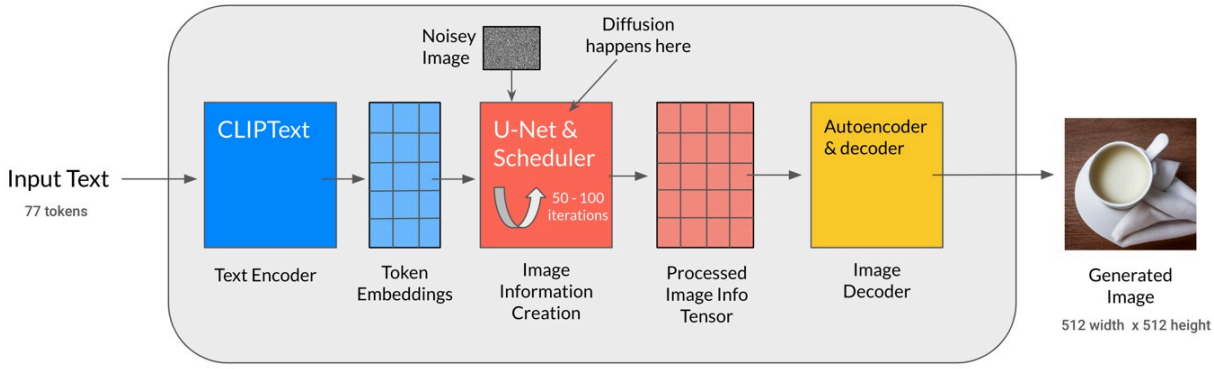

Figure 4 STABLE DIFFUSION ARCHITECTURE

4.4 HOTSHOT-XL

Hotshot-XL represents a significant leap forward in text-to-image generation, particularly in the realm of motion synthesis. Leveraging the integrated Stable Diffusion XL (SDXL) model, Hotshot-XL harnesses the capabilities of the pre-trained SDXL model to generate individual images based on textual descriptions. By feeding information into SDXL, Hotshot-XL facilitates the creation of high-quality images that faithfully depict the semantic content of the input text. Notably, Hotshot-XL introduces a groundbreaking motion-enhancing feature. This feature dynamically analyzes a sequence of textured images generated by SDXL, filtering out variations between consecutive frames. By anticipating continuity between images, Hotshot-XL imbues a sense of fluidity and natural movement, elevating the overall refinement of the output.

In its architecture, Hotshot-XL integrates a viewing window to enhance motion prediction. Typically positioned around the preceding eight frames, this viewing window serves as a temporal reference for predicting subsequent frames in the animation. Furthermore, Hotshot-XL boasts compatibility with the SDXL ControlNet implementation, further augmenting its versatility and adaptability. ControlNet empowers users to define the structure and layout of graphics, offering precise control over the final output. The synergy between Hotshot-XL and ControlNet enables users to tailor the visual aesthetics of the animation according to their specific creative vision or project requirements, enhancing the overall customization and artistic expression.

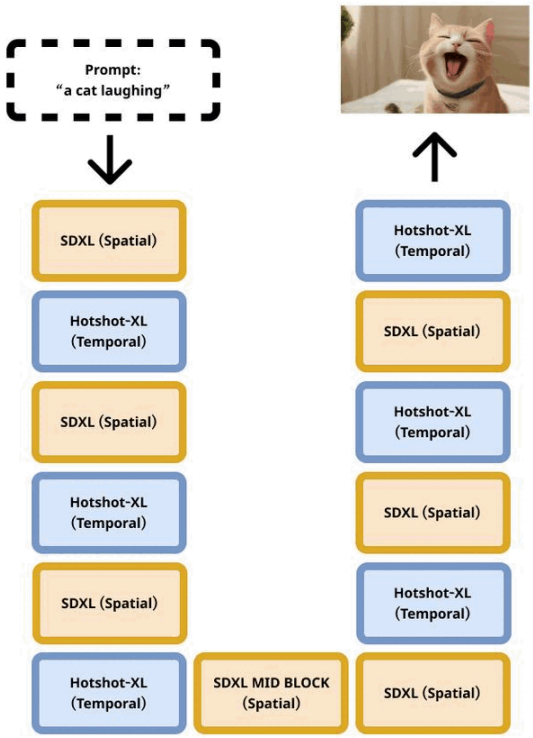

Figure 5 HOTSHOT-XL ARCHITECTURE

4.5 DIFFSCALER ARCHITECTURE

The architecture commences with retrieving the embedded text alongside an image representing a person's face. This dual-input model serves as a foundation for a comprehensive process aimed at seamlessly integrating contextual and visual information. Initially, the text content undergoes processing using a dedicated Encoder module, a critical step in converting textual descriptions into a reusable format. Subsequently, the transcripts are fed into Text2Video, a sophisticated neural network system meticulously engineered to translate textual descriptions into cohesive video sequences. This model derives contextual information from the input text, imbuing the resulting video output with semantic depth and narrative coherence.

Concurrently, the input face image embarks on its dynamic journey, entering a face transformation model where its elements serve as blueprints for facial transformation in the generated video content. Through intricate face recognition and transformation algorithms operating on a complex canvas, the facial transformation model seamlessly integrates the visual characteristics of the input facial image into each consecutive video frame, ensuring visual consistency and accuracy throughout. Following the fusion of context-sensitive video and face-object exchange, the resulting integrated video undergoes further enhancement via a superresolution model. This sophisticated deep learning algorithm plays a pivotal role in augmenting the quality of visual images or videos, enhancing resolution, sharpness, and overall visual fidelity. With the application of SuperResolution enhancement, the architecture ensures that the edited video reflects superior visual clarity and detail, catering to nuanced expressions through crisp and detailed imagery.

Finally, the processed video undergoes decoding through a dedicated decoder module, culminating in the generation of the final output. This stage marks the conclusion of the architectural process, encapsulating a seamless integration of context and visual enhancement. Leveraging sophisticated enhancement techniques that synergize textural and visual elements, the dense architecture crafts a multifaceted and immersive video experience that surpasses the sum of its individual components.

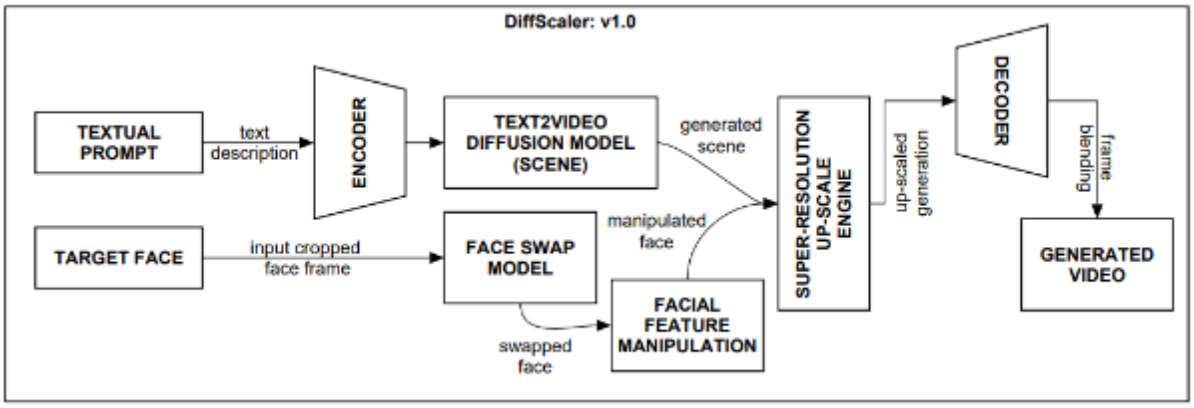

Figure 6 DIFFSCALER ARCHITECTURE

4.5.1 MODULE - 1: GENERATING VISUAL OUTPUT FROM INPUT TEXT

The Text2Video module serves as the cornerstone of the architecture, tasked with translating text annotations into cohesive video sequences. Operating through a complex interface merging natural language processing with computational methods, this module encodes input, extracts contextual understanding, and generates corresponding video content. Utilizing advanced neural networks trained on extensive text and video datasets, Text2Video bridges semantic gaps between textual and visual elements, facilitating the synthesis of dynamic video content aligned with narrative and semantic context. With its seamless integration of text logic and video synthesis capabilities, Text2Video plays a pivotal role in realizing the architecture's objective of crafting immersive and contextually rich textual descriptions of video experiences.

4.5.2 MODULE - 2: FACIAL SWAP WITH THE INPUT FACE

The face swap model stands as a pivotal component of the architecture, dedicated to seamlessly integrating facial features from an input image into video frames generated by the Text2Video module. Evolving in the realms of computer vision and image processing since the 1990s, this model has undergone significant modifications in face recognition techniques to achieve its seamless face integration capabilities. Through algorithms analyzing both the structural properties of the input face image and the contextual information of the video frames, the face swap model dynamically adjusts facial features in the video frames to align with the input face image. This synthesis ensures visual coherence and fidelity throughout the sequences, rendering the video content engaging and lifelike. Leveraging the advancements in deep learning available today, the face swap model enhances the authenticity of the produced video and elevates viewer engagement, thus contributing to the overarching goal of the architecture to create captivating and visually appealing content from textual descriptions.

4.5.3 MODULE - 3: INCREASING THE QUALITY OF THE GENERATED OUTPUT

SuperResolution (SR) plays a crucial role within the modeling framework, tasked with enhancing visual fidelity and improving video quality. Operating within the domain of image processing and development, this model utilizes advanced deep learning techniques to boost the resolution of individual frames in video sequences. Leveraging neural network architecture, the SR model effectively enhances the spatial resolution and details of each frame, resulting in sharper, clearer, and more visually appealing video content. This enhancement is particularly valuable in scenarios where video images may be lacking in detail or corrupted during the initial stages. By significantly enhancing the perceived quality of the overall video production, the SR model contributes to increased viewer satisfaction and engagement. When seamlessly integrated into the architecture, the system achieves heightened visual fidelity and authenticity, thereby enhancing the overall appeal and impact of the video content.

Chapter 5

Sample Code

```python
def main():
    global_step = 0
    min_steps_before_validation = 0

    args = parse_args()

    next_save_iter = args.save_starting_step

    if args.save_starting_step < 1:
        next_save_iter = None

    if args.report_to == "wandb":
        if not is_wandb_available():
            raise ImportError("Make sure to install wandb if you want to use it for logging during training.")

    accelerator = Accelerator(
        gradient_accumulation_steps=args.gradient_accumulation_steps,
        mixed_precision=args.mixed_precision,
        log_with=args.report_to,
        kwargs_handlers=[InitProcessGroupKwargs(timeout=timedelta(args.nccl_timeout))]
    )

    # create custom saving & loading hooks so that `accelerator.save_state(...)` serializes in a nice format
    def save_model_hook(models, weights, output_dir):
        nonlocal global_step

        for model in models:
            if isinstance(model, type(accelerator.unwrap_model(unet))):
                model.save_pretrained(os.path.join(output_dir, 'unet'))
                # make sure to pop weight so that corresponding model is not saved again
                weights.pop()

    accelerator.register_save_state_pre_hook(save_model_hook)

    set_seed(args.seed)

    # Handle the repository creation
    if accelerator.is_local_main_process:
        if args.output_dir is not None:
            os.makedirs(args.output_dir, exist_ok=True)

    # Load the tokenizer
    tokenizer = CLIPTokenizer.from_pretrained(args.pretrained_model_name_or_path, subfolder="tokenizer")
    tokenizer_2 = CLIPTokenizer.from_pretrained(args.pretrained_model_name_or_path, subfolder="tokenizer_2")
```

Figure 7 Code

```
text_encoder = CLIPTextModel.from_pretrained(args.pretrained_model_name_or_path, subfolder="text_encoder")
text_encoder_2 = CLIPTextModelWithProjection.from_pretrained(args.pretrained_model_name_or_path,
                                                             subfolder="text_encoder_2")

vae = AutoencoderKL.from_pretrained(args.pretrained_model_name_or_path, subfolder="vae")

optimizer_resume_path = None

if args.unet_resume_path:
    optimizer_fp = os.path.join(args.unet_resume_path, "optimizer.bin")

    if os.path.exists(optimizer_fp):
        optimizer_resume_path = optimizer_fp

    unet = UNet3DConditionModel.from_pretrained(args.unet_resume_path,
                                                subfolder="unet",
                                                low_cpu_mem_usage=False,
                                                device_map=None)

else:
    unet = UNet3DConditionModel.from_pretrained(args.pretrained_model_name_or_path, subfolder="unet")

if args.xformers:
    vae.set_use_memory_efficient_attention_xformers(True, None)
    unet.set_use_memory_efficient_attention_xformers(True, None)

unet_config = unet.config
unet_add_embedding = unet.add_embedding

unet.requires_grad_(False)

temporal_params = unet.temporal_parameters()

for p in temporal_params:
    p.requires_grad_(True)

vae.requires_grad_(False)
text_encoder.requires_grad_(False)
text_encoder_2.requires_grad_(False)

if args.gradient_checkpointing:
    unet.enable_gradient_checkpointing()

if args.scale_lr:
    args.learning_rate = (
            args.learning_rate * args.gradient_accumulation_steps * args.train_batch_size * accelerator.num_processes
    )
```

Figure 8 Code

```python
# Use 8-bit Adam for lower memory usage
if args.use_8bit_adam:
    try:
        import bitsandbytes as bnb
    except ImportError:
        raise ImportError(
            "To use 8-bit Adam, please install the bitsandbytes library: `pip install bitsandbytes`."
        )

    optimizer_class = bnb.optim.AdamW8bit
else:
    optimizer_class = torch.optim.AdamW

learning_rate = args.learning_rate

params_to_optimize = [
    {'params': temporal_params, "lr": learning_rate},
]

optimizer = optimizer_class(
    params_to_optimize,
    lr=args.learning_rate,
    betas=(args.adam_beta1, args.adam_beta2),
    weight_decay=args.adam_weight_decay,
    eps=args.adam_epsilon,
)

if optimizer_resume_path and not args.disable_optimizer_restore:
    logger.info("Restoring the optimizer.")
    try:

        old_optimizer_state_dict = torch.load(optimizer_resume_path)

        # Extract only the state
        old_state = old_optimizer_state_dict['state']

        # Set the state of the new optimizer
        optimizer.load_state_dict({'state': old_state, 'param_groups': optimizer.param_groups})

        del old_optimizer_state_dict
        del old_state

        torch.cuda.empty_cache()
        torch.cuda.synchronize()
        gc.collect()
```

Figure 9 Code

```
        raise

noise_scheduler = DDPMScheduler.from_pretrained(args.pretrained_model_name_or_path, subfolder="scheduler")

def compute_snr(timesteps):
    """
    Computes SNR as per https://github.com/TiankaiHang/Min-SNR-Diffusion-Training/blob/521b624bd70c67cee4bdf49225915f5945a872e3/guided_di
    """
    alphas_cumprod = noise_scheduler.alphas_cumprod
    sqrt_alphas_cumprod = alphas_cumprod ** 0.5
    sqrt_one_minus_alphas_cumprod = (1.0 - alphas_cumprod) ** 0.5

    # Expand the tensors.
    # Adapted from https://github.com/TiankaiHang/Min-SNR-Diffusion-Training/blob/521b624bd70c67cee4bdf49225915f5945a872e3/guided_diffusi
    sqrt_alphas_cumprod = sqrt_alphas_cumprod.to(device=timesteps.device)[timesteps].float()
    while len(sqrt_alphas_cumprod.shape) < len(timesteps.shape):
        sqrt_alphas_cumprod = sqrt_alphas_cumprod[..., None]
    alpha = sqrt_alphas_cumprod.expand(timesteps.shape)

    sqrt_one_minus_alphas_cumprod = sqrt_one_minus_alphas_cumprod.to(device=timesteps.device)[timesteps].float()
    while len(sqrt_one_minus_alphas_cumprod.shape) < len(timesteps.shape):
        sqrt_one_minus_alphas_cumprod = sqrt_one_minus_alphas_cumprod[..., None]
    sigma = sqrt_one_minus_alphas_cumprod.expand(timesteps.shape)

    # Compute SNR.
    snr = (alpha / sigma) ** 2
    return snr

device = torch.device('cuda')

image_transforms = transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5]),
    ]
)

def image_to_tensor(img):
    with torch.no_grad():

        if img.mode != "RGB":
            img = img.convert("RGB")

        image = image_transforms(img).to(accelerator.device)

        if image.shape[0] == 1:
            image = image.repeat(3, 1, 1)
```

Figure 10 Code

```
        image = image_transforms(img).to(accelerator.device)

        if image.shape[0] == 1:
            image = image.repeat(3, 1, 1)

        if image.shape[0] > 3:
            image = image[:3, :, :]

    return image

def make_sample(sample):

    nonlocal unet_config
    nonlocal unet_add_embedding

    images = [Image.open(img) for img in sample['image_fps']]

    og_size = images[0].size

    for i, im in enumerate(images):
        if im.mode != "RGB":
            images[i] = im.convert("RGB")

    aspect_ratio_map = res_to_aspect_map[args.resolution]

    required_size = tuple(aspect_ratio_map[args.aspect_ratio])

    if required_size != og_size:

        def resize_image(x):
            img_size = x.size
            if img_size == required_size:
                return x.resize(required_size, Image.LANCZOS)

            return scale_aspect_fill(x, required_size[0], required_size[1])

        with ThreadPoolExecutor(max_workers=len(images)) as executor:
            images = list(executor.map(resize_image, images))

    frames = torch.stack([image_to_tensor(x) for x in images])

    l, u, *_ = get_crop_coordinates(og_size, images[0].size)
    crop_coords = (l, u)

    additional_time_ids = add_time_ids(
        unet_config,
```

Figure 11 Code

```
        unet_config,
        unet_add_embedding,
        text_encoder_2,
        og_size,
        crop_coords,
        (required_size[0], required_size[1]),
        dtype=torch.float32
    ).to(device)

    input_ids_0 = tokenizer(
        sample['prompt'],
        padding="do_not_pad",
        truncation=True,
        max_length=tokenizer.model_max_length,
    ).input_ids

    input_ids_1 = tokenizer_2(
        sample['prompt'],
        padding="do_not_pad",
        truncation=True,
        max_length=tokenizer.model_max_length,
    ).input_ids

    return {
        "frames": frames,
        "input_ids_0": input_ids_0,
        "input_ids_1": input_ids_1,
        "additional_time_ids": additional_time_ids,
    }

def collate_fn(examples: list) -> dict:

    # Two Text encoders
    # First Text Encoder  -> Penultimate Layer
    # Second Text Encoder -> Pooled Layer

    input_ids_0 = [example['input_ids_0'] for example in examples]
    input_ids_0 = tokenizer.pad({"input_ids": input_ids_0}, padding="max_length",
                                max_length=tokenizer.model_max_length, return_tensors="pt").input_ids

    prompt_embeds_0 = text_encoder(
        input_ids_0.to(device),
        output_hidden_states=True,
    )

    # we take penultimate embeddings from the first text encoder
```

Figure 12 Code

```
    input_ids_1 = [example['input_ids_1'] for example in examples]
    input_ids_1 = tokenizer_2.pad({"input_ids": input_ids_1}, padding="max_length",
                                  max_length=tokenizer.model_max_length, return_tensors="pt").input_ids

    # We are only ALWAYS interested in the pooled output of the final text encoder
    prompt_embeds = text_encoder_2(
        input_ids_1.to(device),
        output_hidden_states=True
    )

    pooled_prompt_embeds = prompt_embeds[0]
    prompt_embeds_1 = prompt_embeds.hidden_states[-2]

    prompt_embeds = torch.concat([prompt_embeds_0, prompt_embeds_1], dim=-1)

    *_, h, w = examples[0]['frames'].shape

    return {
        "frames": torch.stack([x['frames'] for x in examples]).to(memory_format=torch.contiguous_format).float(),
        "prompt_embeds": prompt_embeds.to(memory_format=torch.contiguous_format).float(),
        "pooled_prompt_embeds": pooled_prompt_embeds,
        "additional_time_ids": torch.stack([x['additional_time_ids'] for x in examples]),
    }

# Region - Dataloaders
dataset = HotshotXLDataset(args.data_dir, make_sample)
dataloader = DataLoader(dataset, args.train_batch_size, shuffle=True, collate_fn=collate_fn)

# Scheduler and math around the number of training steps.
overrode_max_train_steps = False
num_update_steps_per_epoch = math.ceil(len(dataloader) / args.gradient_accumulation_steps)

if args.max_train_steps is None:
    args.max_train_steps = args.num_train_epochs * num_update_steps_per_epoch
    overrode_max_train_steps = True

lr_scheduler = get_scheduler(
    args.lr_scheduler,
    optimizer=optimizer,
    num_warmup_steps=args.lr_warmup_steps * args.gradient_accumulation_steps,
    num_training_steps=args.max_train_steps * args.gradient_accumulation_steps,
)

unet, optimizer, lr_scheduler, dataloader = accelerator.prepare(
    unet, optimizer, lr_scheduler, dataloader
)
```

Figure 13 Code

```
def to_images(video_frames: torch.Tensor):
    import torchvision.transforms as transforms
    to_pil = transforms.ToPILImage()
    video_frames = rearrange(video_frames, "b c f w h -> b f c w h")
    bsz = video_frames.shape[0]
    images = []
    for i in range(bsz):
        video = video_frames[i]
        for j in range(video.shape[0]):
            image = to_pil(video[j])
            images.append(image)
    return images

def to_video_frames(images: list) -> np.ndarray:
    x = np.stack([np.asarray(img) for img in images])
    return np.transpose(x, (0, 3, 1, 2))

def run_validation(step=0, node_index=0):

    nonlocal global_step
    nonlocal accelerator

    if args.test_prompts:
        prompts = args.test_prompts.split("|")
    else:
        prompts = [
            "a woman is lifting weights in a gym",
            "a group of people are dancing at a party",
            "a teddy bear doing the front crawl"
        ]

    torch.cuda.empty_cache()
    gc.collect()

    logger.info(f"Running inference to test model at {step} steps")
    with torch.no_grad():

        pipe = HotshotXLPipeline.from_pretrained(
            args.pretrained_model_name_or_path,
            unet=accelerator.unwrap_model(unet),
            text_encoder=text_encoder,
            text_encoder_2=text_encoder_2,
            vae=vae,
        )

        videos = []
```

Figure 14 Code

```
                    width=w,
                    height=h,
                    original_size=(1920, 1080),  # todo - pass in as args?
                    target_size=(args.resolution, args.resolution),
                    num_inference_steps=30,
                    video_length=8,
                    output_type="tensor",
                    generator=torch.Generator().manual_seed(111)).videos

            videos.append(to_images(video))

        for tracker in accelerator.trackers:

            if tracker.name == "wandb":
                tracker.log(
                    {
                        "validation": [wandb.Video(to_video_frames(video), fps=8, format='mp4') for video in
                                       videos],
                    }, step=global_step
                )

        del pipe

    return

# Move text_encode and vae to gpu.
vae.to(accelerator.device, dtype=torch.bfloat16 if args.vae_b16 else torch.float32)
text_encoder.to(accelerator.device)
text_encoder_2.to(accelerator.device)

# We need to recalculate our total training steps as the size of the training dataloader may have changed.

num_update_steps_per_epoch = math.ceil(len(dataloader) / args.gradient_accumulation_steps)
if overrode_max_train_steps:
    args.max_train_steps = args.num_train_epochs * num_update_steps_per_epoch
# Afterward we recalculate our number of training epochs
args.num_train_epochs = math.ceil(args.max_train_steps / num_update_steps_per_epoch)

# We need to initialize the trackers we use, and also store our configuration.
# The trackers initialize automatically on the main process.

if accelerator.is_main_process:
    accelerator.init_trackers(args.project_name)

def bar(prg):
    br = '|' + '' * prg + ' ' * (25 - prg) + '|'
```

Figure 15 Code

```python
# Train!
total_batch_size = args.train_batch_size * accelerator.num_processes * args.gradient_accumulation_steps

if accelerator.is_main_process:
    logger.info("***** Running training *****")
    logger.info(f"  Num examples = {len(dataset)}")
    logger.info(f"  Num Epochs = {args.num_train_epochs}")
    logger.info(f"  Instantaneous batch size per device = {args.train_batch_size}")
    logger.info(f"  Total train batch size (w. parallel, distributed & accumulation) = {total_batch_size}")
    logger.info(f"  Gradient Accumulation steps = {args.gradient_accumulation_steps}")
    logger.info(f"  Total optimization steps = {args.max_train_steps}")

# Only show the progress bar once on each machine.
progress_bar = tqdm(range(args.max_train_steps), disable=not accelerator.is_local_main_process)

latents_scaler = vae.config.scaling_factor

def save_checkpoint():
    save_dir = Path(args.output_dir)
    save_dir = str(save_dir)
    save_dir = save_dir.replace(" ", "_")
    if not os.path.exists(save_dir):
        os.makedirs(save_dir, exist_ok=True)
    accelerator.save_state(save_dir)

def save_checkpoint_and_wait():
    if accelerator.is_main_process:
        save_checkpoint()
    accelerator.wait_for_everyone()

def save_model_and_wait():
    if accelerator.is_main_process:
        HotshotXLPipeline.from_pretrained(
            args.pretrained_model_name_or_path,
            unet=accelerator.unwrap_model(unet),
            text_encoder=text_encoder,
            text_encoder_2=text_encoder_2,
            vae=vae,
        ).save_pretrained(args.output_dir, safe_serialization=True)
    accelerator.wait_for_everyone()

def compute_loss_from_batch(batch: dict):
    frames = batch["frames"]
    bsz, number_of_frames, c, w, h = frames.shape

    # Convert images to latent space
```

Figure 16 Code

```
with torch.no_grad():

    if args.max_vae_encode:
        latents = []

        x = rearrange(frames, "bs nf c h w -> (bs nf) c h w")

        for latent_index in range(0, x.shape[0], args.max_vae_encode):
            sample = x[latent_index: latent_index + args.max_vae_encode]

            latent = vae.encode(sample.to(dtype=vae.dtype)).latent_dist.sample().float()
            if len(latent.shape) == 3:
                latent = latent.unsqueeze(0)

            latents.append(latent)
            torch.cuda.empty_cache()

        latents = torch.cat(latents, dim=0)
    else:

        # convert the latents from 5d -> 4d, so we can run it though the vae encoder
        x = rearrange(frames, "bs nf c h w -> (bs nf) c h w")

        del frames

        torch.cuda.empty_cache()

        latents = vae.encode(x.to(dtype=vae.dtype)).latent_dist.sample().float()

    if args.latent_nan_checking and torch.any(torch.isnan(latents)):
        accelerator.print("NaN found in latents, replacing with zeros")
        latents = torch.where(torch.isnan(latents), torch.zeros_like(latents), latents)

    latents = rearrange(latents, "(b f) c h w -> b c f h w", b=bsz)

    torch.cuda.empty_cache()

    noise = torch.randn_like(latents, device=latents.device)

    if args.noise_offset:
        # https://www.crosslabs.org//blog/diffusion-with-offset-noise
        noise += args.noise_offset * torch.randn(
            (latents.shape[0], latents.shape[1], 1, 1, 1), device=latents.device
        )

    # Sample a random timestep for each image
    timesteps = torch.randint(0, noise_scheduler.config.num_train_timesteps, (bsz,), device=latents.device)
```

Figure 17 Code

```
        latents = latents * latents_scaler

        # Add noise to the latents according to the noise magnitude at each timestep
        # (this is the forward diffusion process)

        prompt_embeds = batch['prompt_embeds']
        add_text_embeds = batch['pooled_prompt_embeds']

        additional_time_ids = batch['additional_time_ids']  # .repeat_interleave(number_of_frames, dim=0)

        added_cond_kwargs = {"text_embeds": add_text_embeds, "time_ids": additional_time_ids}

        noisy_latents = noise_scheduler.add_noise(latents, noise, timesteps)

        if noise_scheduler.config.prediction_type == "epsilon":
            target = noise
        elif noise_scheduler.config.prediction_type == "v_prediction":
            target = noise_scheduler.get_velocity(latents, noise, timesteps)
        else:
            raise ValueError(f"Unknown prediction type {noise_scheduler.config.prediction_type}")

    noisy_latents.requires_grad = True

    model_pred = unet(noisy_latents,
                      timesteps,
                      cross_attention_kwargs=None,
                      encoder_hidden_states=prompt_embeds,
                      added_cond_kwargs=added_cond_kwargs,
                      return_dict=False,
                      )[0]

    if args.snr_gamma:
        # Compute loss-weights as per Section 3.4 of https://arxiv.org/abs/2303.09556.
        # Since we predict the noise instead of x_0, the original formulation is slightly changed.
        # This is discussed in Section 4.2 of the same paper.
        snr = compute_snr(timesteps)
        mse_loss_weights = (
                torch.stack([snr, args.snr_gamma * torch.ones_like(timesteps)], dim=1).min(dim=1)[0] / snr
        )
        # We first calculate the original loss. Then we mean over the non-batch dimensions and
        # rebalance the sample-wise losses with their respective loss weights.
        # Finally, we take the mean of the rebalanced loss.
        loss = F.mse_loss(model_pred.float(), target.float(), reduction="none")

        loss = loss.mean(dim=list(range(1, len(loss.shape)))) * mse_loss_weights
        return loss.mean()
```

Figure 18 Code

```
    fll = round((global_step * 100) / args.max_train_steps)
    fll = round(fll / 4)
    pr = bar(fll)

    logs = {"loss": loss.detach().item(), "lr": lr_scheduler.get_last_lr()[0], "loss_time": (time.time() - now)}

    if args.validate_every_steps is not None and global_step > min_steps_before_validation and global_step % args.validate_every_steps ==
        if accelerator.is_main_process:
            run_validation(step=global_step, node_index=accelerator.process_index // 8)

        accelerator.wait_for_everyone()

    for key, val in logging_data.items():
        logs[key] = val

    progress_bar.set_postfix(**logs)
    progress_bar.set_description_str("Progress:" + pr)
    accelerator.log(logs, step=global_step)

    if accelerator.is_main_process \
            and next_save_iter is not None \
            and global_step < args.max_train_steps \
            and global_step + 1 == next_save_iter:
        save_checkpoint()

        torch.cuda.empty_cache()
        gc.collect()

        next_save_iter += args.save_n_steps

for epoch in range(args.num_train_epochs):
    unet.train()

    for step, batch in enumerate(dataloader):
        process_batch(batch)

        if global_step >= args.max_train_steps:
            break

    if global_step >= args.max_train_steps:
        logger.info("Max train steps reached. Breaking while loop")
        break

    accelerator.wait_for_everyone()
```

Figure 19 Code

Chapter 6

Results Observed

Prompt: A camel drinking water from a lake

Output:

Figure 20 Camel Drinking output

Prompt: A dog using laptop

Output:

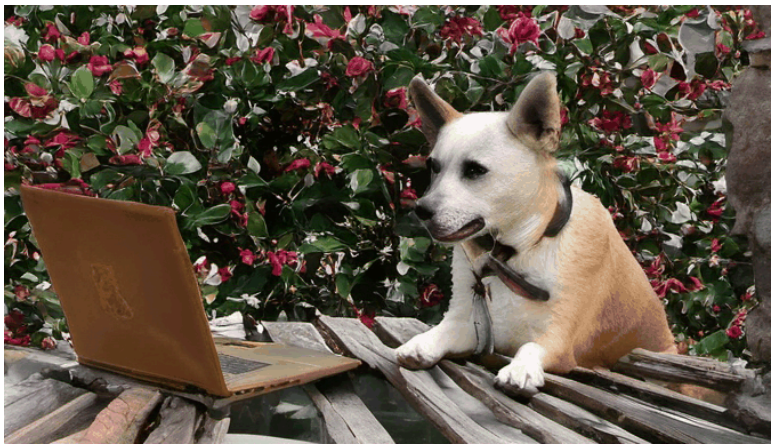

Figure 21 DOG USING LAPTOP OUTPUT

Prompt: A cat with umbrella in a beach

Output:

Figure 22 CAT WITH UMBRELLA OUTPUT

Prompt: A helicopter in middle of a road

Output:

Figure 23 HELICOPTER ON ROAD OUTPUT

Prompt: A cat sleeping on a road

Output:

Figure 24 CAT SLEEPING ON ROAD OUTPUT

Conclusion & Future Work

In conclusion, the development of the Diffusion Pair Up-scale Temporal Transformer marks a significant advancement in information-guided human face transformation within video generation. Through a multi-frame rate hierarchical training approach and additional spatio-temporal modules, DiffScaler showcases remarkable capabilities in producing high-bitrate, realistic videos while preserving facial expressions through angle swapping and maintaining temporal synchrony.

However, there are still numerous avenues for further exploration and growth for DiffScaler in the future. Firstly, ongoing research to enhance computational performance and model refinement will be crucial for real-world applications. Exploring methods to reduce training time and memory requirements without compromising video quality is paramount.

Furthermore, improving the definition and controllability of videos generated by DiffScaler remains a vital area for future development. Enhancing understanding and manipulation of artifacts, such as monitoring specific facial expression features or integrating data processing into the generation process, can offer utility and usability in various contexts. Additionally, integrating DiffScaler into interactive storytelling platforms, virtual reality experiences, and augmented reality applications presents an exciting opportunity to broaden its impact and application. By incorporating user interaction and dynamic content, DiffScaler can contribute to creating immersive and engaging digital experiences across diverse environments.

REFERENCES

[1] Hong, Wenyi, et al. "Cogvideo: Large-scale pretraining for text-to-video generation via transformers." arXiv preprint arXiv:2205.15868 (2022).

[2] Wu, Jay Zhangjie, et al. "Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023.

[3] Singer, Uriel, et al. "Make-a-video: Text-to-video generation without text-video data." arXiv preprint arXiv:2209.14792 (2022).

[4] Hong, Wenyi, et al. "Cogvideo: Large-scale pretraining for text-to-video generation via transformers." arXiv preprint arXiv:2205.15868 (2022).

[5] Yang, Shuai, et al. "Rerender a video: Zero-shot text-guided video-to-video translation." SIGGRAPH Asia 2023 Conference Papers. 2023.

[6] Wang, Jiuniu, et al. "Modelscope text-to-video technical report." arXiv preprint arXiv:2308.06571 (2023).

[7] Wang, Jiuniu, et al. "Modelscope text-to-video technical report." arXiv preprint arXiv:2308.06571 (2023).

[8] Wolf, Thomas, et al. "Transformers: State-of-the-art natural language processing." Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. 2020.

[9] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[10] Waswani, A., et al. "Attention is all you need." NIPS. 2017.