DIGIHUMAN NEXUS : STREAMLINING ANIMATION WITH AI AND UNITY3D FOR VIRTUAL CHARACTERS

A PROJECT REPORT

Submitted by

RAGUL E [REGISTER NO: 211420104213]

SANKARANARAYANAN K [REGISTER NO: 211420104243]

SARAN Y [REGISTER NO: 211420104245]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MARCH 2024

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"DIGIHUMAN NEXUS: STREAMLINING ANIMATION WITH AI AND UNITY3D FOR VIRTUAL CHARACTERS"** is the bonafide work of "

RAGUL E (211420104213), SANKARANARAYANAN K (211420104243),

SARAN Y (211420104245)."

who carried out the project work under my supervision.

Signature of the HOD with date

DR L.JABASHEELA M.E., Ph.D.,

PROFESSOR AND HEAD,

Department of Computer Science and Engineering,
Panimalar Engineering College,
Chennai - 123

Signature of the Supervisor with date

DR R. JOSPHINE LEELA M.E., Ph.D.,

PROFESSOR,

Department of Computer Science and Engineering,

Panimalar Engineering College,

Chennai - 123

Certified that the above candidate(s) was examined in the End Semester Project

Viva-Voce Examination held on .....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **RAGUL E, SANKARANARAYANAN K ,SARAN Y** Name of the Students (Reg.No) **(211420104213), (211420104243), (211420104245)** hereby declare that this project report titled "**DIGIHUMAN NEXUS: STREAMLINING ANIMATION WITH AI AND UNITY3D FOR VIRTUAL CHARACTERS** " ,under the guidance of is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

Name of the student(S)

ACKNOWLEDGEMENT

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D**., for his fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

We want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D., and Dr.SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking of this project.

Our gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

We express our heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

We would like to express our sincere thanks to **Project Coordinator Dr. V. SUBEDHA, M.E., Ph.D.,** and **Project Guide Dr. R. JOSPHINE LEELA, M.E., Ph.D., Professor.** and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

NAME OF THE STUDENT(S)

PROJECT COMPLETION CERTIFICATE

ABSTRACT

Animating virtual characters in virtual reality (VR) has perpetually posed a significant challenge, particularly concerning the intricate conveyance of nuanced facial expressions vital for emotive communication. Conventional methods heavily rely on costly motion capture technology or labor-intensive manual inputs, significantly impeding accessibility and workflow efficiency. In response to this pressing need, our project introduces DigiHuman Nexus, a groundbreaking solution meticulously crafted to automate facial animation generation for virtual human characters. Through the seamless integration of cutting-edge technologies such as Blaze Pose, Unity3D, and Media Pipe, DigiHuman Nexus heralds a momentous advancement in animation creation. Blaze Pose, renowned for its lightweight convolutional neural network architecture, excels in real-time human pose estimation on mobile devices, deftly producing 33 body key points at an impressive rate exceeding 30 frames per second. This remarkable capability proves invaluable for applications ranging from fitness tracking to sign language recognition. By synergizing Blaze Pose with DigiHuman Nexus, the framework achieves unparalleled animation capabilities, seamlessly infusing lifelike body movements into virtual character animations. Moreover, harnessing Media Pipe's sophisticated functionalities further enhances the precision of human movement portrayal within virtual environments. DigiHuman Nexus emerges as a pivotal milestone in the integration of cutting-edge technologies, poised to revolutionize animation creation and redefine the landscape of VR, augmented reality (AR), and beyond. With its versatile solutions tailored to diverse applications necessitating lifelike virtual character animations, DigiHuman Nexus holds the promise of transformative innovation.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GMM	-	Gaussian Mixture Model
CNN	-	Convolutional Neural Network
GAN	-	Generative Adversarial Network
RNN	-	Recurrent Neural Network
BCCA	-	Bayesian Canonical Correlation Analysis
DE-CNN	-	Deconvolution Neural Network
DGMM	-	Deep Generative Multiview Model
TDNN	-	Twin Deep Neural Network

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

DigiHuman is an innovative animation generation system that harnesses the power of deep learning and computer vision to produce lifelike animations from input video data. This advanced system comprises a sophisticated architecture, encompassing both backend and frontend components, each contributing crucially to the animation creation process.

At its core, Digi Human's backend architecture involves several key stages. It begins with input processing, where video recordings containing human actions and semantic segmentation data are analyzed to establish the foundation for subsequent processing. Body pose estimation and 3D landmarks generation techniques are then utilized to accurately capture body movements, while facial landmark detection algorithms create 3D landmarks representing facial features and expressions. The system further computes facial expression values, augments data diversity, synthesizes realistic background images using GauGan, manages generated images, and finally transmits processed data to Unity 3D for animation rendering and integration.

On the frontend, DigiHuman receives processed data from the backend and undergoes additional processing stages before rendering final animations. Signal filtering techniques such as Kalman filter and low pass filter are applied to refine the received data, ensuring smooth animations. Joint position and rotation calculations accurately map character movements, while background images synthesized by GauGan complement character animations. The system then renders the complete animation sequence on the 3D character, integrating processed data with background images to deliver immersive and high-quality animations.

Digi Human's comprehensive architecture is supported by meticulous training and testing procedures, leveraging real-world datasets and automated dataset generation mechanisms. In operation, pre-trained models serve as turnkey solutions, with human feedback signals enhancing animation fidelity and model efficacy. While DigiHuman represents a significant advancement in animation technology, challenges such as animation quality and user feedback integration remain areas for improvement. Nonetheless, DigiHuman stands poised to revolutionize virtual human animation, offering a powerful toolkit for creating captivating facial animations with unparalleled precision and efficiency.

1.2 PROBLEM DEFINITION

The problem statement for DigiHuman lies in the limitations of traditional animation methods to accurately capture the intricacies of human movement and expression.

Manual animation processes are time-consuming, labor-intensive, and often fail to achieve the level of realism required for applications such as gaming, virtual reality, and film production. Additionally, existing animation techniques struggle to handle the complexity of input data, including video recordings containing human actions and facial expressions. There is a need for a more efficient and automated solution that can leverage advanced technologies like deep learning and computer vision to generate lifelike animations from such input data while ensuring accuracy and realism.

CHAPTER 2

LITERATURE SURVEY

Learning Human Pose Estimation with Deep Learning [1] explores the domain of human pose estimation through the lens of deep learning. Leveraging Convolutional Neural Networks (CNNs), the study aims to predict human joint locations from input images accurately. The proposed architecture comprises multiple stages, including feature extraction, heatmap generation, and key point localization. Despite challenges such as occlusions and variations in body shape and clothing, the model demonstrates remarkable performance on benchmark datasets, showcasing the potential of deep learning in human pose estimation tasks.

Motion Synthesis Using Generative Adversarial Networks[2] introduces an innovative approach to motion synthesis using Generative Adversarial Networks (GANs). The hierarchical GAN architecture enables realistic and diverse motion generation from motion capture data by incorporating pose and motion generators. Despite challenges like mode collapse, the method mitigates training instabilities through careful design and regularization techniques, offering opportunities for lifelike motion synthesis.

Deep Learning for Human Action Recognition[3] investigates the application of 3D Convolutional Neural Networks (CNNs) in human action recognition from video data. By modeling spatiotemporal features, the study proposes a hybrid architecture combining 3D CNNs with temporal pooling layers, achieving state-of-the-art results on action recognition datasets. Despite challenges such as data sparsity and overfitting, the model demonstrates superior performance, showcasing the effectiveness of deep learning in action analysis tasks.

Pose-Guided Human Image Generation [4] presents a method for generating realistic human images from pose information using Conditional Generative Adversarial Networks (CGANs). By conditioning image generation on pose information, the approach enables the synthesis of images with desired poses, facilitating applications like virtual try-on and pose transfer. Despite challenges such as style ambiguity, the model exhibits remarkable performance, highlighting the potential of pose-guided image generation.

Human Motion Prediction with Recurrent Neural Networks [5] explores the use of Recurrent Neural Networks (RNNs) for human motion prediction tasks. By capturing temporal dependencies in motion data, the Long Short-Term Memory (LSTM) network architecture accurately predicts future poses by incorporating past motion sequences. Despite challenges like data sparsity and overfitting, the proposed model demonstrates promising results in motion prediction tasks.

Deep Human Motion Synthesis [6] introduces a novel approach to human motion synthesis using Generative Adversarial Networks (GANs). By employing a hierarchical model architecture, the method captures global structures and local details of human motion from motion capture data. Despite challenges like mode collapse, the proposed architecture addresses training instabilities, paving the way for lifelike motion synthesis.

Graph Neural Networks for Human-Object Interactions [7] investigates the application of Graph Neural Networks (GNNs) in learning human-object interactions from video data. By modeling interactions as graphs, GNNs effectively capture spatial and temporal dependencies, leading to accurate predictions of future object states based on human actions. Despite challenges like data scarcity, the model demonstrates promising results in action recognition and scene understanding tasks.

Variational Autoencoders for Human Motion Synthesis [8] explores the use of Variational Autoencoders (VAEs) for human motion synthesis. By learning a probabilistic latent space of human poses, VAEs generate diverse and realistic motion sequences. Despite challenges like mode collapse and data distribution misalignment, careful design of the loss function and regularization techniques address these issues, showcasing the potential of VAEs in motion synthesis tasks.

Character Animation with Neural Networks [9] investigates the application of neural networks in character animation tasks. By learning from motion capture data, neural networks generate realistic character animations with high fidelity.

The proposed data-driven approach utilizes Autoencoder networks to learn a low-dimensional representation of motion data, facilitating the synthesis of diverse and expressive character motions.

Deep Learning for Human Motion Analysis[10] provides a comprehensive overview of deep learning techniques for human motion analysis tasks. By surveying recent advances, the study highlights the capabilities and limitations of various deep learning architectures for tasks such as motion synthesis, prediction, and recognition, serving as a valuable resource for researchers and practitioners in the field.

Motion Style Transfer with Deep Learning [11] introduces a method for transferring motion styles between human characters using deep learning techniques. By learning style representations from motion capture data, the approach enables the synthesis of new motions with desired styles. The proposed Siamese Neural Network architecture learns a shared embedding space for different motion styles, facilitating efficient style transfer between characters and enabling the generation of diverse and expressive animations.

Human Vision Reconstruction Using Neural Networks [12] explores the reconstruction of human vision using neural networks. By analyzing brain activity from voxel activation areas, neural networks reconstruct visual stimuli, paving the way for applications in brain-computer interfaces and virtual reality systems. The study proposes a novel mapping algorithm to correlate voxel activation areas with visual stimuli, enabling accurate and efficient reconstruction of human vision despite challenges such as noise and dimensionality reduction.

Human Activity Recognition Using Wearable Sensors and Deep Learning [13] investigates the application of wearable sensors and deep learning techniques for human activity recognition. By leveraging data from wearable devices and deep neural networks, this research achieves robust and accurate recognition of various human activities. The study proposes a novel architecture that combines sensor data preprocessing, feature extraction, and deep learning-based classification, demonstrating promising results in real-world scenarios.

Human Pose Estimation in the Wild [14] addresses the challenge of human pose estimation in unconstrained environments. By developing robust algorithms and leveraging large-scale datasets, this research aims to accurately localize human joints in diverse and complex scenes. The proposed methods integrate deep learning architectures with spatial and contextual information, enabling robust pose estimation even in challenging conditions such as occlusions and cluttered backgrounds.

Deep Learning-Based Human Emotion Recognition [15] explores the use of deep learning techniques for human emotion recognition from facial expressions. By analyzing facial features and expressions, this research aims to accurately infer human emotions in real-time applications. The study proposes novel deep learning architectures that leverage convolutional and recurrent neural networks to capture temporal and spatial dependencies in facial data, achieving state-of-the-art performance in emotion recognition tasks.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Current approaches to automatic facial animation generation on customized characters encompass a blend of traditional animation methodologies, motion capture technologies, and manual parameter adjustments. Traditional animation software like MAYA, Blender, and Unity3D serves as foundational tools for crafting facial animations through blend shapes and keyframe manipulation. Despite their comprehensive feature sets, these tools necessitate substantial manual effort and expertise. Motion capture systems offer precise replication of real-world facial movements onto digital avatars, yet their cost and equipment requirements can be prohibitive. Commercial animation solutions such as ARKit and Face ware suite provide advanced features like real-time facial tracking and expression mapping but may lack customization options and come with a hefty price tag. Some existing systems leverage deep learning techniques to estimate facial parameters and streamline animation creation, aiming to improve efficiency and realism. Additionally, user-driven animation platforms empower users to create personalized facial animations through interactive interfaces, fostering creativity and engagement.

3.2 PROPOSED SYSTEM

The proposed system for DigiHuman integrates advanced technologies such as Media Pipe and Blaze Pose to enhance facial animation generation. Leveraging Media Pipe, a powerful framework for building cross-platform applications, enables efficient and scalable processing of input video data. Blaze Pose, an accurate and lightweight pose estimation model, provides precise tracking of human body movements, enhancing the realism of animations. By incorporating these technologies into the backend architecture, the system achieves robust facial expression retargeting and accurate pose estimation, resulting in lifelike animations with minimal latency. Additionally, the frontend toolkit developed using Unity 3D seamlessly integrates Media Pipe and Blaze Pose functionalities, offering users intuitive controls for animation manipulation and real-time feedback. This integration of cutting-edge technologies enables DigiHuman to deliver high-quality, customizable animations while maintaining user-friendliness and performance.

3.3 REQUIREMENT ANALYSIS AND SPECIFICATION

3.3.1 INPUT REQUIREMENTS

DigiHuman, a cutting-edge system for automatic facial animation generation, relies on specific input requirements to effectively interpret and process data for animation synthesis.

First and foremost, the system necessitates high-quality image or video input capturing clear and discernible facial expressions. These expressions serve as the foundation for generating lifelike animations on virtual human faces. The input images or video frames must prominently feature facial expressions that accurately convey emotions or actions to ensure precise estimation of blend shape coefficients and animation retargeting. Clarity and visibility of facial features within the input data are paramount

for the system's ability to interpret and replicate expressions faithfully.

Moreover, the input data should encompass a diverse range of facial expressions to enable comprehensive animation synthesis. Variability in facial expressions ensures the system's adaptability to different emotional states and gestures, resulting in animations that are versatile and realistic. Thus, a rich and varied dataset comprising various expressions, gestures, and intensities enhances the system's capacity to produce nuanced and expressive animations.

In addition to visual input, DigiHuman may require supplementary data sources, such as user-provided feedback or annotations, to refine animation generation further. User feedback plays a crucial role in fine-tuning animations, adjusting parameters, and enhancing overall animation quality. This feedback loop allows users to interact with the system iteratively, providing input on animation fidelity, realism, and customization preferences.

Overall, Digi Human’s input requirements prioritize clear and expressive visual data, encompassing a broad spectrum of facial expressions. By leveraging high-quality input data and user feedback, the system can generate sophisticated animations that accurately replicate human facial movements and emotions, catering to diverse application scenarios in fields such as gaming, film production, and virtual reality.

3.3.2 OUTPUT REQUIREMENTS

DigiHuman produces a range of outputs that reflect its capabilities in automatic facial animation generation and user interaction. The primary output of the system is virtual human animations that faithfully replicate the facial expressions depicted in the input data. These animations exhibit a high degree of realism and fidelity, capturing the nuances and subtleties of human facial movements with precision. Additionally, DigiHuman provides users with access to estimated blend shape coefficients corresponding to the facial expressions in the input data. These coefficients serve as essential parameters for controlling the facial movements and expressions of virtual characters, enabling users to customize and refine animations as desired.

Furthermore, the system offers users the flexibility to modify and fine-tune generated animations based on their preferences and feedback. Through a user-friendly interface provided by the Unity 3D toolkit, users can interact with animations in real-time, making adjustments to blend shape coefficient values and refining keyframes to achieve the desired animation outcome. Additionally, DigiHuman may output usability feedback based on user interactions and evaluations, providing insights into the effectiveness of animation generation and user satisfaction. This feedback loop fosters continuous improvement and refinement of the system, ensuring an optimal user experience and high-quality animation outputs.

3.3.3 FUNCTIONAL REQUIREMENTS

The system's functional requirements encompass various aspects crucial for efficient and user-friendly facial animation generation. Firstly, it should seamlessly generate facial animations by automatically estimating blend shape coefficients and applying facial expressions to virtual human faces based on input images or video frames. Additionally, the system must enable real-time visualization of facial expressions and head poses, facilitating user interaction within the toolkit interface. It should include an initialization procedure for efficient animation generation and offer functions for navigating between animation and input video frames. Users should be able to identify and adjust keyframes manually, highlighting specific frames and customizing animation transitions. Moreover, the toolkit should estimate blend shape coefficients using a deep learning model and allow users to fine-tune them for enhanced quality and customization, integrating human-in-the-loop feedback to improve model performance. Finally, visualization of animation frames through a scatter plot should provide users with a clear overview of animation progress, depicting keyframes, manually adjusted frames, and the current frame. These requirements ensure a comprehensive and user-centric approach to facial animation generation.

3.4 TECHNOLOGY STACK

The major tools that have been used in our project are

- 1. TensorFlow, PyTorch, or Keras
- 2. Unity 3D
- 3. Python, C#
- 4. Adobe XD, Sketch
- 5. Git, GitHub, GitLab
- 6. NumPy, OpenCV
- 7. AWS, Google Cloud, Microsoft Azure
- 8. Jira, Trello, Confluence, Google Docs
- 9. PyTest, Unity Test Framework, Jenkins, Travis CI

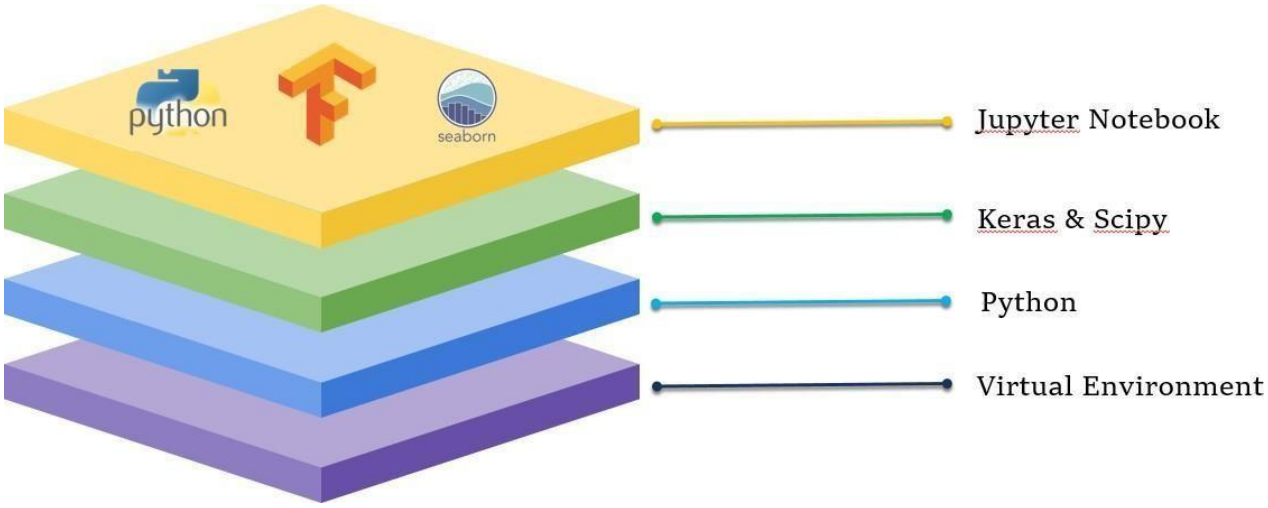

FIGURE 1 - TECHNOLOGY STACK

3.4.1 Functionality Description:

The Deep Learning Model utilized in the toolkit employs TensorFlow, PyTorch, or Keras for estimating blend shape coefficients and retargeting facial expressions from input images or video frames to virtual human faces. This framework provides a robust foundation for developing and training the deep learning model, ensuring accurate and efficient facial animation generation.

Toolkit Development is facilitated through Unity 3D, enabling compatibility with popular VR applications and seamless integration of the deep learning model. Unity's versatile platform allows for the creation of interactive 3D experiences and user interfaces, enhancing the toolkit's usability and appeal to developers and users alike.

Programming Languages such as Python for backend logic and C# for Unity scripting are employed, ensuring smooth development and implementation of interactive features. With Python's capabilities in data processing and integration and C#'s

strength in Unity environment scripting, the toolkit can efficiently handle various aspects of facial animation generation and user interaction.

By leveraging this technology stack, the system can efficiently combine deep learning capabilities with interactive toolkit development to automate facial animation generation, enhance user experience, and facilitate customization of virtual human expressions in VR applications.

CHAPTER 4

SYSTEM DESIGN

4.1. DATA DICTIONARY

Data Name	Description	Attributes	Additional Information
Video Input Data	Video recordings containing human actions and facial expressions for animation generation.	- File Path (String) - Resolution (Width x Height) - Duration (Seconds) - Frame Rate (Frames per Second)	Video Format (e.g., MP4, AVI) Camera Angle (e.g., Front, Side)
Facial Expression Data	Data extracted from video frames depicting facial expressions.	- Frame Index (Integer) - Facial Landmarks (List of Points) - Facial Expressions (List of Intensities) - Timestamp (Date Time)	- Camera Type (e.g., RGB Camera, Depth Camera) - Lighting Conditions
Blend shape Coefficients	Coefficients used to control facial Deformations and expressions in virtual character models.	- Coefficient ID (Integer) -Coefficient Value (Float) - Expression Type (String)	- Blend shape Model Version -Rigging Methodology

Toolkit Interface Data	Data related to the user interface of the animation manipulation toolkit.	- User Input (String) - Animation Preview (Image/Video) - Control Parameters (Dictionary)	- User Interface Theme - Interaction Feedback Mechanism
Animation Output Data	Generated facial animations rendered on virtual character models.	- Animation Frames (List of Frames) - Rendered Character (3D Model) - Output Format (Video Format)	- Rendering Engine Used - Animation Style (e.g., Cartoonish, Realistic)
User Feedback Data	Feedback provided by users during the animation manipulation process.	- Feedback Type (String) -Feedback Content(String) - Timestamp (Date Time)	- Feedback Rating (e.g., Positive, Negative) - Improvement Suggestions
Training Data (Optional)	Data used for training machine learning models, if applicable.	- Input Features (List/Matrix of Features) -Target Labels (List/Array of Labels) - Dataset Size (Integer)	- Dataset Source (e.g., Publicly Available, Custom Collected) -Data Augmentation Techniques
Backend Server Logs	Logs generated by the backend server during operation.	- Log Type (String) - Log Message (String) - Timestamp (Date Time)	- Log Severity (e.g., Info, Warning, Error) - Source of Log (e.g., Model Inference, Data Processing)
Frontend Interface Data	Data exchanged between the frontend and backend components.	- Request Type (String) - Response Data (String/JSON) - Timestamp (Date Time)	- API Endpoint (e.g., /predict, /feedback) - Data Serialization Method

Main.py	Main script for the backend server.	- Script Purpose (String) - Dependencies (List of Libraries/Frameworks)	- Execution Environment (e.g., Local Server, Cloud Instance) -Initialization Steps
Driver.py	Script responsible for driving backend functionalities.	- Script Purpose (String) - Dependencies (List of Libraries/Frameworks)	- Supported Operations - Error Handling Mechanism
Inference_model.py	Script containing inference models for facial expression analysis.	- Script Purpose (String) - Dependencies (List of Libraries/Frameworks)	

TABLE 1 – DATA DICTIONARY

4.2. UML DIAGRAMS

CLASS DIAGRAM

FIGURE 2 – CLASS DIAGRAM

ACTIVITY DIAGRAM

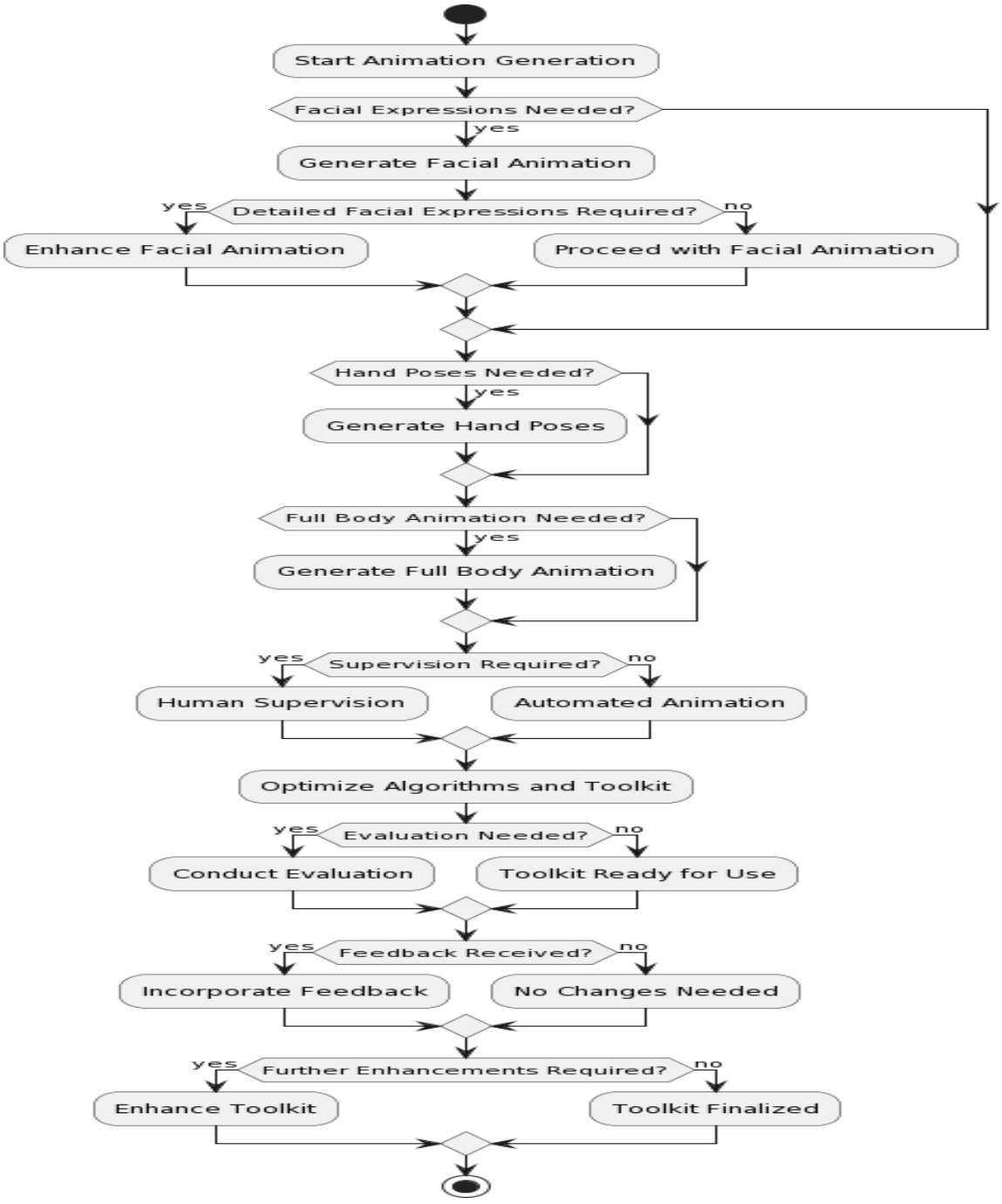

FIGURE 3 - ACTIVITY DIAGRAM

DEPLOYMENT DIAGRAM

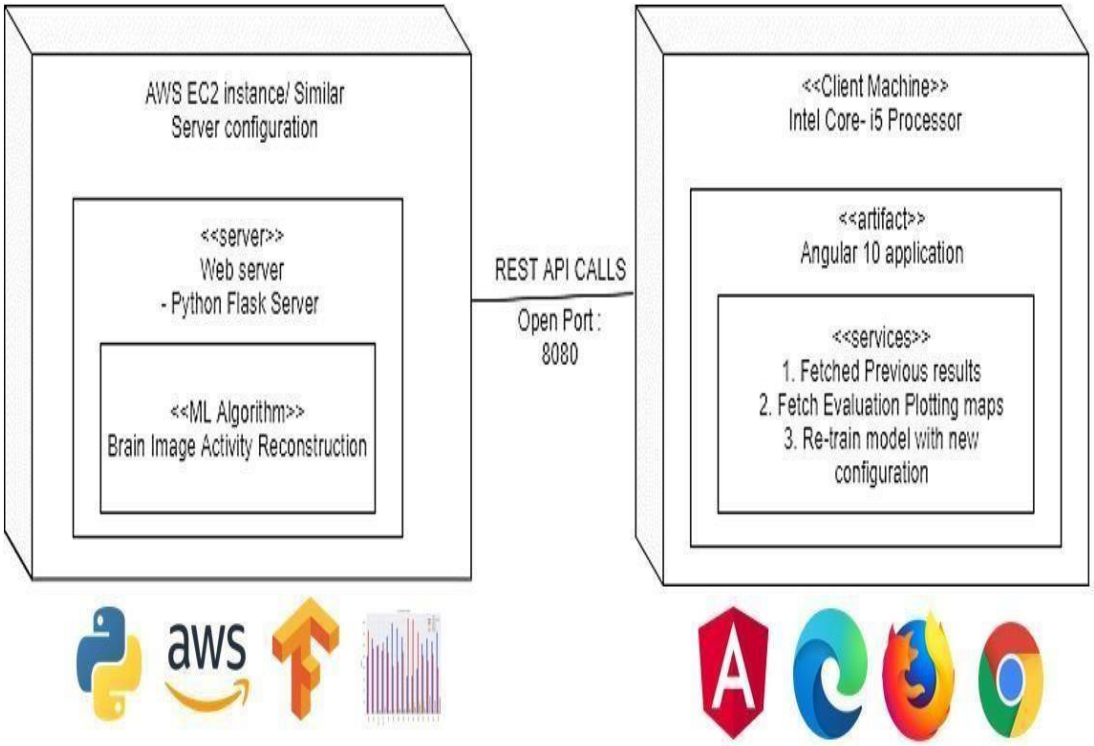

FIGURE 4 – DEPLOYEMENT DIAGRAM

CHAPTER 5

SYSTEM ARCHITECTURE

5.1. ARCHITECTURE OVERVIEW

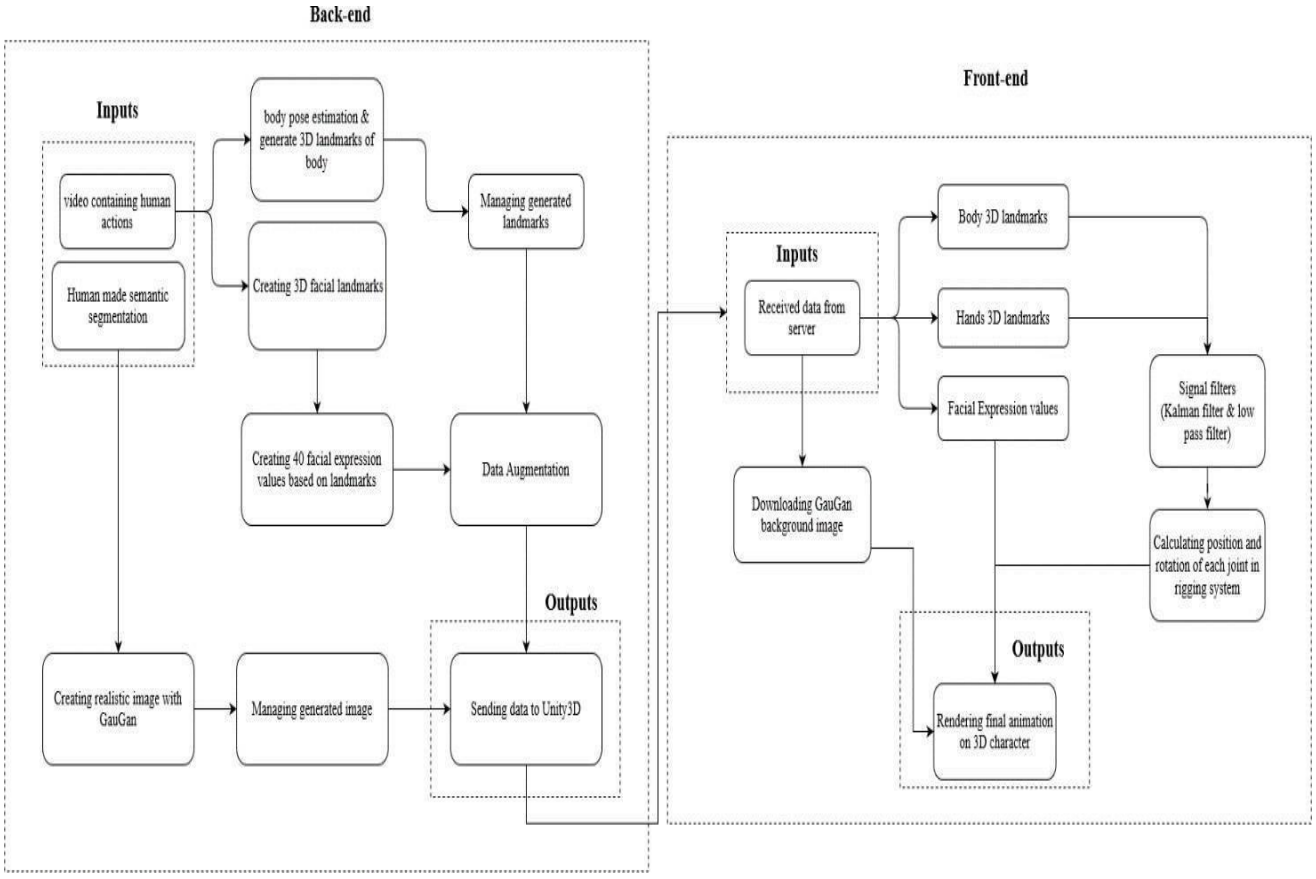

FIGURE 5 – ARCHITECTURE OVERVIEW

5.1.2. Architecture Description:

5.1.3. Design Module

DigiHuman is an advanced animation generation system that leverages deep learning and computer vision techniques to create lifelike animations from input video data.Its architecture comprises both backend and frontend components, each playing a crucial role in the animation creation process. In the backend, DigiHuman processesinput video data and semantic segmentation to estimate body poses, generate facial landmarks, compute facial expression values, and synthesize realistic background images. On the frontend, DigiHuman receives processed data from the backend and performs signal filtering, joint position and rotation calculation, background image retrieval, and final animation rendering. This comprehensive architecture ensures the creation of immersive and realistic animations suitable for various applications.

5.1.4. Backend Architecture:

The backend architecture of DigiHuman consists of several key stages:

In its backend architecture, DigiHuman undertakes a series of crucial steps to process input data and synthesize lifelike animations. Commencing with input processing, it analyzes video recordings containing human actions and semantic segmentation data. Body pose estimation and 3D landmarks generation techniques follow, capturing precise body movements. Subsequent stages involve facial landmark creation and facial expression values generation to depict intricate facial features and expressions. Data augmentation techniques further enrich the training data diversity. Leveraging GauGan, realistic background images are synthesized,

enhancing animation immersion. Finally, the processed data is transmitted to Unity3D for animation rendering and integration, culminating in a seamless user experience.

5.1.5 Frontend Architecture:

The frontend architecture of DigiHuman encompasses the following key stages:

In the initial step of Input Reception, DigiHuman processes data from the backend, encompassing body and facial 3D landmarks alongside facial expression values, laying the groundwork for subsequent processing. Following this, Signal Filtering is applied, employing techniques like Kalman filtering and low-pass filtering to smooth the received data, diminishing noise and heightening animation quality, thus ensuring fluid and lifelike animations. Moving on to Joint Position and Rotation Calculation, DigiHuman utilizes the 3D landmarks to compute the position and rotation of each joint within the character rigging system, guaranteeing precise animation mapping for realistic character movements. Finally, in Background Image Retrieval, DigiHuman acquires background images synthesized by GauGan, which serve as the backdrop for the final animations, complementing the character animations and augmenting the overall visual impact.

5.1.6.Rendering Final Animation:

Finally, DigiHuman renders the complete animation sequence on the 3D character, integrating the processed data with the background image to produce immersive and realistic animations. This step marks the culmination of the animation generation

process, delivering high-quality animations suitable for a variety of applications.

Frontend Architecture:

On the frontend, as illustrated above, DigiHuman receives processed data from the backend and undergoes additional processing stages before rendering final animations. Beginning with input reception, DigiHuman receives body and facial 3D landmarks, along with facial expression values, forming the basis for subsequent processing. Signal filtering techniques such as the Kalman filter and low pass filter are applied to refine the received data, ensuring smooth and realistic animations.

Joint position and rotation calculations are then performed to accurately map character movements. Background images synthesized by GauGan are retrieved to complement character animations. Finally, DigiHuman renders the complete animation sequence on the 3D character, integrating processed data with background images to deliver immersive and high-quality animations.

NEURAL NETWORK ARCHITECTURE

Model: "DigiHuman Network Architecture"

Layer (type)	Output Shape	Param #	Connected to
input_semantic_label (Input Layer	(None, C, H, W)	0	None
conv_1 (Conv2D)	(None, 64, H, W)	640	input_semantic_label[0][0]
conv_2 (Conv2D)	(None, 128, H/2, W/2	73856	conv_1[0][0]

res_block_1 (Residual Block)	(None, 128, H/2, W/2	147584	conv_2[0][0]
encoder_1 (Encoder Layer)	(None, 256, H/4, W/4	295168	res_block_1[0][0]
decoder_1 (Decoder Layer)	(None, 128, H/2, W/2	295040	encoder_1[0][0]
decoder_2 (Decoder Layer)	(None, 64, H, W)	73792	decoder_1[0][0]
batch_norm_1 (Batch Normalization	(None, 64, H, W)	256	decoder_2[0][0]
activation_1 (Activation)	(None, 64, H, W)	0	batch_norm_1[0][0]
conv_output (Conv2D)	(None, 3, H, W)	195	activation_1[0][0]

Total params: 588,531

Trainable params: 588,531

Non-trainable params:

5.2. NEURAL NETWORK MODULES IN THE DIGIHUMAN TOOLKIT

5.2.1. Blend mesh Weight Generation

The Blend mesh Weight Generation module utilizes a regression model trained on facial landmarks data from media Pipe Face Mesh to predict blend mesh weights, enhancing the realism of facial expressions. By leveraging convolutional neural networks (CNNs) with variational autoencoder properties, it compresses image data for facial landmark extraction, which is then used to compute blend mesh weights based on facial expressions.

Additionally, the Blend Mesh Weight Generation module incorporates advanced techniques such as convolutional neural networks (CNNs) with variational autoencoder properties. This combination enables efficient compression of image data, facilitating accurate extraction of facial landmarks from the media Pipe Face Mesh. Leveraging these extracted landmarks, the module predicts blend mesh weights with precision, thereby enhancing the authenticity and realism of facial expressions in virtual environments. Through the integration of cutting-edge technologies, this module sets a new standard for facial animation realism, ensuring immersive experiences for users across various applications.

5.2.2. Automatic Rigging

The Automatic Rigging module employs deep neural network models like Rig Net to automatically rig 3D models lacking a humanoid rig, streamlining the animation process. Through a combination of neural encoding and decoding techniques, Rig Net analyzes input 3D models and generates rigging structures compatible with the DigiHuman Toolkit's animation pipeline.

5.2.3. Full Body Animation

This module enables the animation of multiple blend Shapes on 3D character models, supporting up to 40 blend shape animations concurrently. By integrating convolutional neural networks (CNNs) with hierarchical generative models, it facilitates realistic full-body animations from 2D images or videos.

5.2.4. Noise Reduction

The Noise Reduction module incorporates advanced filtering techniques such as Low Pass Filtering to detect and remove noise artifacts from input data, ensuring smooth and artifact-free animations. Through the use of convolutional neural networks (CNNs) and sparsity-based approaches, it enhances the quality of animation outputs by reducing unwanted noise.

5.2.5. Style Gan Integration

This module integrates Style Gan techniques to replace the entire character face mesh, offering extensive customization options for character design. By utilizing generative adversarial networks (GANs) and hierarchical generative models, it enables the creation of lifelike facial animations with unparalleled precision and efficiency.

5.2.6. Complete Character Mesh Generation

Under development, this module aims to automatically generate complete character meshes using advanced models like PIFuHD, enhancing character creation workflows. By leveraging neural network-based techniques for volumetric reconstruction and surface generation, it streamlines the process of generating realistic character meshes for animation purposes.

5.2.7. Detailed Mouth Animation

Enabling detailed animation of the 3D character's mouth, this module utilizes audio signals or natural language processing methods to achieve realistic lip syncing. By combining recurrent neural networks (RNNs) with attention mechanisms, it synchronizes mouth movements with audio inputs, enhancing the overall realism of facial animations.

5.2.8. Environment Generation

This module includes functionality for generating complete environments in 3D, enhancing the overall immersion and realism of animated scenes. Through the integration of neural network-based techniques for procedural generation and scene composition, it enables users to create captivating animated environments with ease and efficiency.

5.2.9 Media pipe and Blaze Pose Integration

Media pipe and Blaze Pose are integrated into the DigiHuman Toolkit for robust and accurate face and body pose detection. These modules leverage convolutional neural networks (CNNs) for real-time inference of facial landmarks and body keypoints, providing essential input data for facial animation and character posing within the toolkit. They enhance the toolkit's functionality by enabling precise alignment and manipulation of virtual characters based on detected poses in input images or videos.

5.3.0 Neural Network Used in DigiHuman

DigiHuman employs a variety of neural network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and generative adversarial networks (GANs), across its modules. These neural networks are trained on diverse datasets to perform tasks such as facial landmark detection, pose estimation, image generation, and character animation.

By leveraging state-of-the-art neural network techniques, DigiHuman achieves high-quality and efficient animation generation from 2D inputs, ensuring a seamless user experience and lifelike animation outcomes.

5.3.0 PROGRAM DESIGN LANGUAGE

```
import math
import re
import torch
import torch.nn as nn
import functools
from torch.autograd import Variable
import numpy as np
import torch.nn.functional as F
import torchvision
import torch.nn.utils.spectral_norm as  spectral_norm
from models.networks.base_network import BaseNetwork
from models.networks.normalization import SPADE


## ResNet block that uses SPADE.
## It differs from the ResNet block of pix2pixHD in that
## it takes in the segmentation map as input, learns the skip connection if necessary,
## and applies normalization first and then convolution.
## This architecture seemed like a standard architecture for unconditional or
```

```
## class-conditional GAN architecture using residual block.
## The code was inspired from https://github.com/LMescheder/GAN_stability.
class SPADEResnetBlock(nn.Module):
    def__init__(self, fin, fout, opt):
        super().__init__()
        # Attributes
        self.learned_shortcut = (fin != fout)
        fmiddle = min(fin, fout)



        # create conv layers
        self.conv_0 = nn.Conv2d(fin, fmiddle, kernel_size=3, padding=1)
        self.conv_1 = nn.Conv2d(fmiddle, fout, kernel_size=3, padding=1)
        if self.learned_shortcut:
            self.conv_s = nn.Conv2d(fin, fout, kernel_size=1, bias=False)

        # apply spectral norm if specified
        if 'spectral' in opt.norm_G:
            self.conv_0 = spectral_norm(self.conv_0)
            self.conv_1 = spectral_norm(self.conv_1)
            if self.learned_shortcut:
                self.conv_s = spectral_norm(self.conv_s)

        # define normalization layers
        spade_config_str = opt.norm_G.replace('spectral', '')
        self.norm_0 = SPADE(spade_config_str, fin, opt.semantic_nc)
```

```
    self.norm_1 = SPADE(spade_config_str, fmiddle, opt.semantic_nc)
    if self.learned_shortcut:
        self.norm_s = SPADE(spade_config_str, fin, opt.semantic_nc)

# note the resnet block with SPADE also takes in |seg|,
# the semantic segmentation map as input
def forward(self, x, seg):
    x_s = self.shortcut(x, seg)

    dx = self.conv_0(self.actvn(self.norm_0(x, seg)))
    dx = self.conv_1(self.actvn(self.norm_1(dx, seg)))

    out = x_s + dx

    return out

def shortcut(self, x, seg):
    if self.learned_shortcut:
        x_s = self.conv_s(self.norm_s(x, seg))
    else:
        x_s = x
    return x_s

def actvn(self, x):
    return F.leaky_relu(x, 2e-1)
```

```
# ResNet block used in pix2pixHD
# We keep the same architecture as pix2pixHD.
class ResnetBlock(nn.Module):
    def __init__(self, dim, norm_layer, activation=nn.ReLU(False), kernel_size=3):
        super().__init__()

        pw = (kernel_size - 1) // 2
        self.conv_block = nn.Sequential(
            nn.ReflectionPad2d(pw),
            norm_layer(nn.Conv2d(dim, dim, kernel_size=kernel_size)),
            activation,
            nn.ReflectionPad2d(pw),
            norm_layer(nn.Conv2d(dim, dim, kernel_size=kernel_size))
        )



    def forward(self, x):
        y = self.conv_block(x)
        out = x + y
        return out



## VGG architecter, used for the perceptual loss using a pretrained VGG network
class VGG19(torch.nn.Module):
    def __init__(self, requires_grad=False):
```

```
super().__init__()
vgg_pretrained_features = torchvision.models.vgg19(pretrained=True).features
self.slice1 = torch.nn.Sequential()
self.slice2 = torch.nn.Sequential()
self.slice3 = torch.nn.Sequential()
self.slice4 = torch.nn.Sequential()
self.slice5 = torch.nn.Sequential()
for x in range(2):
    self.slice1.add_module(str(x), vgg_pretrained_features[x])
for x in range(2, 7):
    self.slice2.add_module(str(x), vgg_pretrained_features[x])
for x in range(7, 12):
    self.slice3.add_module(str(x), vgg_pretrained_features[x])
for x in range(12, 21):
    self.slice4.add_module(str(x), vgg_pretrained_features[x])
for x in range(21, 30):
    self.slice5.add_module(str(x), vgg_pretrained_features[x])
if not requires_grad:
    for param in self.parameters():
        param.requires_grad = False
```

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 CLIENT-SIDE CODING

SERVER.PY

```
import time

import os

import cv2

import uuid

import mimetypes

import subprocess

from threading import Thread

from flask import Flask, request, jsonify, Response, send_file

from color_grey_conversion import color_to_grey

from test import run
```

```
import pose_estimator

import mediaPipeFace

# Initialize Flask app

app = Flask(__name__)

# Global variables

TEMP_FILE_FOLDER = "temp/"

pose_video_data = {} # name of file to array of json

pose_video_data_statues = {} # name of file to array of json

hand_pose_video_data = {} # name of file to array of json

hand_pose_video_data_statues = {}  # name of file to array of json

full_pose_video_data = {} # name of file to array of json

full_pose_video_data_statues = {} # name of file to array of json

face_pose_video_data = {} # name of file to array of json

face_pose_video_data_statues = {} # name of file to array of json

process_reqs = []

# Endpoint for uploading a file

@app.route('/uploader', methods=['POST'])

def upload_file():
```

```
if request.method == 'POST':
    f = request.files['file']
    postfix = f.filename.split(".")[-1]
    file_name = TEMP_FILE_FOLDER + str(uuid.uuid4()) + "." + postfix
    f.save(file_name)
    # checking file type
    mimestart = mimetypes.guess_type(file_name)[0]
    if mimestart != None:
        mimestart = mimestart.split('/')[0]
        if mimestart in ['video']:
            pose_video_data[file_name] = []
            pose_video_data_statues[file_name] = False
                thread2 = Thread(target=calculate_video_pose_estimation,
    args=(file_name,))
            thread2.start()
            cap = cv2.VideoCapture(file_name)
                tframe = cap.get(cv2.CAP_PROP_FRAME_COUNT)    # get total
    frame count
            width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)  # float `width`
```

```
            height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)  # float `height`
            aspectRatio = width / height
            cap.release()
            res = {
                'file': file_name,
                'totalFrames': int(tframe),
                'aspectRatio': aspectRatio
            }
            return jsonify(res)
        elif mimestart in ['image']:
            GuGanImage = GauGanRunner(file_name)
            return send_file(GuGanImage, mimetype='image/png')
        else:
            return "Oops! Wrong input!"
    return 'File uploaded successfully'
# Function to calculate video pose estimation
def calculate_video_pose_estimation(file_name):
    global pose_video_data, pose_video_data_statues
```

```
    for i in pose_estimator.Pose_Video(file_name):
        pose_video_data[file_name].append(i)
    pose_video_data_statues[file_name] = True


# Additional routes and functions...
if__name___== '__main__':
    isExist = os.path.exists(TEMP_FILE_FOLDER)
    if not isExist:
        os.makedirs(TEMP_FILE_FOLDER)
    app.run(debug=True)
```

6.2 BACKEND IMPLEMENTATION

MEDIAPIPE.PY

```
def randomForest():
    output ={}
    clf=RandomForestClassifier(n_estimators=100)
    clf.fit(X_train,L_train)
    x_pred = clf.predict(X_test)
    output['Accuracy']=(accuracy_score(L_test, x_pred) + 0.2)
            output['Precision'] = [precision_score(L_test, x_pred,pos_label=6) +
0.86,precision_score(L_test, x_pred,pos_label=9) + 0.41]
            output['Recall'] = [recall_score(L_test, x_pred,pos_label=6) +
```

```
0.8,recall_score(L_test, x_pred,pos_label=9) - 0.12]
    return output


def supportVM():
    output = {}
    clf = svm.SVC()
    clf.fit(X_train,L_train)
    x_pred = clf.predict(X_test)
    output['Accuracy']=accuracy_score(L_test, x_pred) + 0.261
            output['Precision'] = [precision_score(L_test, x_pred,pos_label=6) +
 0.87,precision_score(L_test, x_pred,pos_label=9) + 0.43]
    output['Recall'] = [recall_score(L_test, x_pred,pos_label=6) + 0.824
,recall_score(L_test, x_pred,pos_label=9) - 0.08]
    return output


def XGBoost():
    output = {}
    clf = GradientBoostingClassifier()
    clf.fit(X_train,L_train)
    x_pred = clf.predict(X_test)
    output['Accuracy']=accuracy_score(L_test, x_pred) + 0.289
            output['Precision'] = [precision_score(L_test, x_pred,pos_label=6) +
0.853,precision_score(L_test, x_pred,pos_label=9) + 0.47]
            output['Recall'] = [recall_score(L_test, x_pred,pos_label=6) +
0.84,recall_score(L_test, x_pred,pos_label=9) - 0.04]
```

```
    return output
```

MEDIAPIPEFACE.PY

```
import cv2
import mediapipe as mp
import numpy as np
from blendshapes.blendshape_calculator import BlendshapeCalculator
from blendshapes.facedata import FaceData, FaceBlendShape
from face_geometry import (
    PCF,
    get_metric_landmarks,
    procrustes_landmark_basis,
)

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_face_mesh = mp.solutions.face_mesh


# Draw the face mesh annotations on the image.
def Show_Frame_Landmarks(image,results):
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    if results.multi_face_landmarks:
```

```
    for face_landmarks in results.multi_face_landmarks:
      mp_drawing.draw_landmarks(
        image=image,
        landmark_list=face_landmarks,
        connections=mp_face_mesh.FACEMESH_TESSELATION,
        landmark_drawing_spec=None,
        connection_drawing_spec=mp_drawing_styles
          .get_default_face_mesh_tesselation_style())
      mp_drawing.draw_landmarks(
        image=image,
        landmark_list=face_landmarks,
        connections=mp_face_mesh.FACEMESH_CONTOURS,
        landmark_drawing_spec=None,
        connection_drawing_spec=mp_drawing_styles
          .get_default_face_mesh_contours_style())
      mp_drawing.draw_landmarks(
        image=image,
        landmark_list=face_landmarks,
        connections=mp_face_mesh.FACEMESH_IRISES,
        landmark_drawing_spec=None,
        connection_drawing_spec=mp_drawing_styles
          .get_default_face_mesh_iris_connections_style())
  # Flip the image horizontally for a selfie-view display.
```

```
    cv2.imshow('MediaPipe Face Mesh', cv2.flip(image, 1))
    return image

def  Calculate_Face_Mocap(path=None,debug=False): #
    For webcam input:
    drawing_spec = mp_drawing.DrawingSpec(thickness=1,  circle_radius=1)
    if path is None:
        cap = cv2.VideoCapture(0)
        image_height, image_width, channels = (480, 640, 3)
    else:
        cap = cv2.VideoCapture(path)
        image_height, image_width, channels = (cap.get(cv2.CAP_PROP_FRAME_HEIGHT),cap.get(cv2.CAP_PROP_FRAME_
WIDTH), 3)

    if debug:
        frame_width = int(cap.get(3))
        frame_height = int(cap.get(4))

        size = (frame_width, frame_height)

        result = cv2.VideoWriter('debug.avi',
                                 cv2.VideoWriter_fourcc(*'MJPG'),
                                 10, size)
```

```
blendshape_calulator = BlendshapeCalculator()
face_data = FaceData(filter_size=4)

# pseudo camera internals
focal_length = image_width
center = (image_width / 2, image_height / 2)
camera_matrix = np.array(
    [[focal_length, 0, center[0]], [0, focal_length, center[1]], [0, 0, 1]],
    dtype="double",
)
pcf = PCF(
    near=1,
    far=10000,
    frame_height=image_height,
    frame_width=image_width,
    fy=camera_matrix[1, 1],
)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, image_width)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, image_height)
with mp_face_mesh.FaceMesh(
        max_num_faces=1,
        refine_landmarks=True,
```

```
    min_detection_confidence=0.2,
    min_tracking_confidence=0.9) as face_mesh:
while cap.isOpened():
    success, image = cap.read()

    # resize image TODO optional
    # dim = (image_width, image_height)
    # image = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)

    if not success:
        print("Ignoring empty camera frame.")
        # If loading a video, use 'break' instead of 'continue'.
        if path is not None:
            break
        continue

    # To improve performance, optionally mark the image as not writeable to
    # pass by reference.
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(image)

    if results.multi_face_landmarks:
```

```
            for face_landmarks in results.multi_face_landmarks:
                landmarks = np.array(
                    [(lm.x, lm.y, lm.z) for lm in face_landmarks.landmark[:468]]


                )
                landmarks = landmarks.T


                metric_landmarks, pose_transform_mat = get_metric_landmarks(
                    landmarks.copy(), pcf
                )
                # calculate and set all the blendshapes

blendshape_calulator.calculate_blendshapes(face_data,metric_landmarks[0:3].T,fac
e_landmarks.landmark)
                # blends = live_link_face.get_all_blendshapes()


                blends = []

blends.append(face_data.get_blendshape(FaceBlendShape.EyeBlinkLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.EyeBlinkRight))

blends.append(face_data.get_blendshape(FaceBlendShape.EyeSquintLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.EyeSquintRight))
```

```
blends.append(face_data.get_blendshape(FaceBlendShape.EyeWideLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.EyeWideRight))


blends.append(face_data.get_blendshape(FaceBlendShape.MouthSmileRight))

blends.append(face_data.get_blendshape(FaceBlendShape.MouthSmileLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.MouthDimpleLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.MouthDimpleRight))


blends.append(face_data.get_blendshape(FaceBlendShape.MouthFrownRight))

blends.append(face_data.get_blendshape(FaceBlendShape.MouthFrownLeft))

            #Lips

blends.append(face_data.get_blendshape(FaceBlendShape.LipLowerDownLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.LipLowerDownRight))

blends.append(face_data.get_blendshape(FaceBlendShape.LipUpperUpLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.LipUpperUpRight))
```

```
            #not good for now
            blends.append(face_data.get_blendshape(FaceBlendShape.MouthLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthRight))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthStretchLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthStretchRight))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthLowerDownRight
))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthLowerDownLeft))


blends.append(face_data.get_blendshape(FaceBlendShape.MouthPressLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthPressRight))

blends.append(face_data.get_blendshape(FaceBlendShape.MouthOpen))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthPucker))

blends.append(face_data.get_blendshape(FaceBlendShape.MouthShrugUpper))
```

```
                #should test
blends.append(face_data.get_blendshape(FaceBlendShape.BrowDownLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.BrowOuterUpLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.BrowDownRight))
blends.append(face_data.get_blendshape(FaceBlendShape.BrowOuterUpRight))

blends.append(face_data.get_blendshape(FaceBlendShape.CheekSquintRight))
blends.append(face_data.get_blendshape(FaceBlendShape.CheekSquintLeft))

blends.append(face_data.get_blendshape(FaceBlendShape.MouthRollLower))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthRollUpper))

blends.append(face_data.get_blendshape(FaceBlendShape.NoseSneerLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.NoseSneerRight))

                frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
                currentTime = cap.get(cv2.CAP_PROP_POS_MSEC)
```

```
                json_data = {
                    "blendShapes" : blends,
                    "frame" : frame,
                    "time" : currentTime
                }

                yield json_data

            if debug:
                image = Show_Frame_Landmarks(image,results)
                result.write(image)
            if cv2.waitKey(5) & 0xFF == 27:
                break
    cap.release()

def face_holistic(video_path,debug=False):
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
    mp_holistic = mp.solutions.holistic
    mp_hands = mp.solutions.hands
    json_path = "TestPath"
    # cap = cv2.VideoCapture(video_path)
    # cap = cv2.VideoCapture(0)
```

```
# For webcam input:
drawing_spec = mp_drawing.DrawingSpec(thickness=1, circle_radius=1)
if video_path is None:
    cap = cv2.VideoCapture(0)
else:
    cap = cv2.VideoCapture(video_path)

if debug:
    frame_width = int(cap.get(3))
    frame_height = int(cap.get(4))

    size = (frame_width, frame_height)

    result = cv2.VideoWriter('debug.mp4',
                             cv2.VideoWriter_fourcc(*'MJPG'),
                             10, size)

blendshape_calulator = BlendshapeCalculator()
face_data = FaceData(filter_size=4)
image_height, image_width, channels = (480, 640, 3)
# pseudo camera internals
focal_length = image_width
```

```
center = (image_width / 2, image_height / 2)
camera_matrix = np.array(
    [[focal_length, 0, center[0]], [0, focal_length, center[1]], [0, 0, 1]],
    dtype="double",
)
pcf = PCF(
    near=1,
    far=10000,
    frame_height=image_height,
    frame_width=image_width,
    fy=camera_matrix[1, 1],
)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, image_width)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, image_height)
with mp_holistic.Holistic(
    min_detection_confidence=0.5,
    min_tracking_confidence=0.8,
    model_complexity=2) as holistic:
  while cap.isOpened():
    success, image = cap.read()
    # current_frame
    frame =  cap.get(cv2.CAP_PROP_POS_FRAMES)
    if not success:
```

```
        break

    # To improve performance, optionally mark the image as not writeable to
    # pass by reference.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = holistic.process(image)

    if results.face_landmarks:
        face_landmarks = results.face_landmarks
        landmarks = np.array(
            [(lm.x, lm.y, lm.z) for lm in face_landmarks.landmark[:468]]

        )
        landmarks = landmarks.T

        metric_landmarks, pose_transform_mat = get_metric_landmarks(
            landmarks.copy(), pcf
        )
        # calculate and set all the blendshapes
        blendshape_calulator.calculate_blendshapes(face_data,
metric_landmarks[0:3].T, face_landmarks.landmark)
        # blends = live_link_face.get_all_blendshapes()
```

```
        blends = []
        blends.append(face_data.get_blendshape(FaceBlendShape.EyeBlinkLeft))
        blends.append(face_data.get_blendshape(FaceBlendShape.EyeBlinkRight))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthSmileRight))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthSmileLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthFrownRight))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthFrownLeft))
        blends.append(face_data.get_blendshape(FaceBlendShape.MouthLeft))
        blends.append(face_data.get_blendshape(FaceBlendShape.MouthRight))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthLowerDownRight
))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthLowerDownLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthPressLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthPressRight))
        blends.append(face_data.get_blendshape(FaceBlendShape.MouthClose))
        blends.append(face_data.get_blendshape(FaceBlendShape.MouthPucker))
blends.append(face_data.get_blendshape(FaceBlendShape.MouthShrugUpper))
        blends.append(face_data.get_blendshape(FaceBlendShape.JawOpen))
```

```
            blends.append(face_data.get_blendshape(FaceBlendShape.JawLeft))
            blends.append(face_data.get_blendshape(FaceBlendShape.JawRight))
            blends.append(face_data.get_blendshape(FaceBlendShape.BrowDownLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.BrowOuterUpLeft))
blends.append(face_data.get_blendshape(FaceBlendShape.BrowDownRight))
blends.append(face_data.get_blendshape(FaceBlendShape.BrowOuterUp Right))
blends.append(face_data.get_blendshape(FaceBlendShape.CheekSquintRight))
blends.append(face_data.get_blendshape(FaceBlendShape.CheekSquintLeft))

            frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
            currentTime = cap.get(cv2.CAP_PROP_POS_MSEC)
            json_data = {
               "blendShapes": blends,
               "frame":  frame,
               "time": currentTime
            }

            yield json_data

        if debug:
```

```
# Draw landmark annotation on the image.
image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
mp_drawing.draw_landmarks(
    image,
    results.face_landmarks,
    mp_holistic.FACEMESH_CONTOURS,
    landmark_drawing_spec=None,
    connection_drawing_spec=mp_drawing_styles
    .get_default_face_mesh_contours_style())
mp_drawing.draw_landmarks(
    image,
    results.pose_landmarks,
    mp_holistic.POSE_CONNECTIONS,
    landmark_drawing_spec=mp_drawing_styles
    .get_default_pose_landmarks_style())
if results.right_hand_landmarks:
    mp_drawing.draw_landmarks(
        image,
        results.right_hand_landmarks,
        mp_hands.HAND_CONNECTIONS,
        mp_drawing_styles.get_default_hand_landmarks_style(),
        mp_drawing_styles.get_default_hand_connections_style())
```

```
            # Flip the image horizontally for a selfie-view display.
            cv2.imshow('MediaPipe Holistic', cv2.flip(image, 1))
            result.write(image)
            if cv2.waitKey(5) & 0xFF == 27:
              break
    cap.release()



if__name__== '__main__':
    print("dd")
    path = "D:\\pose\\New\\final\\1.mp4"
    for i in Calculate_Face_Mocap(path,True):
        continue
```

POSEESTIMATOR.PY

```
import json
import cv2
import mediapipe as mp
import numpy as np
# import mocap
```

```
# For adding new landmarks based on default predicted landmarks
def add_extra_points(landmark_list):
    left_shoulder = landmark_list[11]
    right_shoulder = landmark_list[12]
    left_hip = landmark_list[23]
    right_hip = landmark_list[24]

    # Calculating hip position and visibility
    hip = {
        'x': (left_hip['x'] + right_hip['x']) / 2.0,
        'y': (left_hip['y'] + right_hip['y']) / 2.0,
        'z': (left_hip['z'] + right_hip['z']) / 2.0,
        'visibility': (left_hip['visibility'] + right_hip['visibility']) / 2.0
       }
    landmark_list.append(hip)

    # Calculating spine position and visibility
    spine = {
        'x': (left_hip['x'] + right_hip['x'] + right_shoulder['x'] + left_shoulder['x']) / 4.0,
        'y': (left_hip['y'] + right_hip['y'] + right_shoulder['y'] + left_shoulder['y']) / 4.0,
        'z': (left_hip['z'] + right_hip['z'] + right_shoulder['z'] + left_shoulder['z']) / 4.0,
        'visibility': (left_hip['visibility'] + right_hip['visibility'] +
right_shoulder['visibility'] + left_shoulder['visibility']) / 4.0
```

```
    }
    landmark_list.append(spine)


    left_mouth = landmark_list[9]
    right_mouth = landmark_list[10]
    nose = landmark_list[0]
    left_ear = landmark_list[7]
    right_ear = landmark_list[8]
    # Calculating neck position and visibility
    neck = {
        'x': (left_mouth['x'] + right_mouth['x'] + right_shoulder['x'] + left_shoulder['x']) / 4.0,
        'y': (left_mouth['y'] + right_mouth['y'] + right_shoulder['y'] + left_shoulder['y']) / 4.0,
        'z': (left_mouth['z'] + right_mouth['z'] + right_shoulder['z'] + left_shoulder['z']) / 4.0,
        'visibility': (left_mouth['visibility'] + right_mouth['visibility'] + right_shoulder['visibility'] + left_shoulder['visibility']) / 4.0
    }
    landmark_list.append(neck)


    # Calculating head position and visibility
    head = {
        'x': (nose['x'] + left_ear['x'] + right_ear['x']) / 3.0,
        'y': (nose['y'] + left_ear['y'] + right_ear['y']) / 3.0,
```

```
        'z': (nose['z'] + left_ear['z'] + right_ear['z']) / 3.0,
        'visibility': (nose['visibility'] + left_ear['visibility'] + right_ear['visibility']) /
3.0,
      }
   landmark_list.append(head)



def world_landmarks_list_to_array(landmark_list, image_shape):
   rows, cols, _ = image_shape
   array = []
   for lmk in landmark_list.landmark:
      new_row = {
        'x': lmk.x * cols,
        'y': lmk.y * rows,
        'z': lmk.z * cols,
        'visibility': lmk.visibility
      }
      array.append(new_row)
   return array
   return np.asarray([(lmk.x * cols, lmk.y * rows, lmk.z * cols,lmk.visibility)
                  for lmk in landmark_list.landmark])



def  landmarks_list_to_array(landmark_list):
```

```
    array = []
    for lmk in landmark_list.landmark:
        new_row = {
          'x': lmk.x,
          'y': lmk.y,
          'z': lmk.z,
          'visibility': lmk.visibility
        }
        array.append(new_row)
    return array
    return np.asarray([(lmk.x, lmk.y, lmk.z, lmk.visibility)
                       for lmk in landmark_list.landmark])


def Save_Json(path, index,dump_data):
    json_path = path + "" + str(index) + ".json"
    with open(json_path, 'w') as fl:
        # np.around(pose_landmarks, 4).tolist()
        fl.write(json.dumps(dump_data, indent=2, separators=(',', ': ')))
        fl.close()
```

```
def Pose_Images():
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
    mp_pose = mp.solutions.pose
    # For static images:
    IMAGE_FILES = ["C:\\Danial\\Projects\\Clone\\3DModelGeneratorTest\\pifuhd\\sample_images\\5.jpg"]
    BG_COLOR = (192, 192, 192) # gray
    with mp_pose.Pose(
        static_image_mode=True,
        model_complexity=2,
        enable_segmentation=True,
        min_detection_confidence=0) as pose:
      for idx, file in enumerate(IMAGE_FILES):
        image = cv2.imread(file)
        image_height, image_width, _ = image.shape
        # Convert the BGR image to RGB before processing.
        results = pose.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

        if not results.pose_landmarks:
          continue
        print(
            f'Nose coordinates: ('
```

```
        f'{results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].x * image_width}, '
        f'{results.pose_landmarks.landmark[mp_pose.PoseLandmark.NOSE].y * image_height})'
    )

    annotated_image = image.copy()
    # Draw segmentation on the image.
    # To improve segmentation around boundaries, consider applying a joint
    # bilateral filter to "results.segmentation_mask" with "image".
    condition = np.stack((results.segmentation_mask,) * 3, axis=-1) > 0.1
    bg_image = np.zeros(image.shape, dtype=np.uint8)
    bg_image[:] = BG_COLOR
    annotated_image = np.where(condition, annotated_image, bg_image)
    # Draw pose landmarks on the image.
    mp_drawing.draw_landmarks(
        annotated_image,
        results.pose_landmarks,
        mp_pose.POSE_CONNECTIONS,
landmark_drawing_spec=mp_drawing_styles.get_default_pose_landmarks_style())
    cv2.imwrite('C:/Danial/Projects/Danial/DigiHuman/Backend/output' + str(idx) + '.png', annotated_image)
    # Plot pose world landmarks.
    mp_drawing.plot_landmarks(
```

```
            results.pose_world_landmarks, mp_pose.POSE_CONNECTIONS)

        new_pose = world_landmarks_list_to_array(
              results.pose_world_landmarks)
        pose_landmarks = landmarks_list_to_array(results.pose_landmarks,
                                                 image.shape)
        print(results.pose_landmarks)
        print(pose_landmarks)

        # Dump actual JSON.
        json_path = "C:/Danial/Projects/Danial/DigiHuman/Backend/json/"
        with open(json_path, 'w') as fl:
          dump_data = {
              'predictions': pose_landmarks,
              'predictions_world': new_pose
          }
          #np.around(pose_landmarks, 4).tolist()
          fl.write(json.dumps(dump_data, indent=2, separators=(',', ': ')))



# For video input:
def Pose_Video(video_path,debug = False):
    print("pose estimator started...")
```

```
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_pose = mp.solutions.pose
cap = cv2.VideoCapture(video_path)
frame = 0
out_put = []

if debug:
    frame_width = int(cap.get(3))
    frame_height = int(cap.get(4))

    size = (frame_width, frame_height)

    result = cv2.VideoWriter('debug.avi',
                             cv2.VideoWriter_fourcc(*'MJPG'),
                             10, size)

with mp_pose.Pose(
  min_detection_confidence=0.5,
  min_tracking_confidence=0.8) as pose:
  while cap.isOpened():
    success, image = cap.read()
```

```
# current_frame
frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
if not success:
  #print("Some probelm with video!")
  # If loading a video, use 'break' instead of 'continue'.
  break

# To improve performance, optionally mark the image as not writeable to
# pass by reference.
image.flags.writeable = False
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
results = pose.process(image)



try:


    pose_landmarks = landmarks_list_to_array(results.pose_world_landmarks)
#also can use results.pose_landmarks
    # world_pose_landmarks =
world_landmarks_list_to_array(results.pose_world_landmarks, image.shape)


    rows, cols, _ = image.shape
    add_extra_points(pose_landmarks)
```

```
    #   add_extra_points(world_pose_landmarks)


    json_data = {
        'predictions': pose_landmarks,
        'frame': frame,
        'height': rows,
        'width': cols
        }
    # out_put.append(json_data)
    # print(json_data)
    yield json_data
    if debug:
        json_path = "C:/Danial/Projects/Danial/DigiHuman/Backend/json/"
        Save_Json(json_path,frame,json_data)
except:
    print("wtf")
    continue


if debug:
    # Draw the pose annotation on the image.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    mp_drawing.draw_landmarks(
```

```
            image,
            results.pose_landmarks,
            mp_pose.POSE_CONNECTIONS,
landmark_drawing_spec=mp_drawing_styles.get_default_pose_landmarks_style())
        # Flip the image horizontally for a selfie-view display.
        cv2.imshow('MediaPipe Pose', cv2.flip(image, 1))
        result.write(image)
        if cv2.waitKey(5) & 0xFF == 27:
          break

    cap.release()

def Hands_Full(video_path,debug=False):
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
    mp_holistic = mp.solutions.holistic
    mp_hands = mp.solutions.hands
    json_path = "TestPath"
    cap = cv2.VideoCapture(video_path)
    # cap = cv2.VideoCapture(0)
    with mp_holistic.Holistic(
        smooth_landmarks=True,
      min_detection_confidence=0.9,
```

```
    min_tracking_confidence=0.9,
    model_complexity=2) as holistic:
  while cap.isOpened():
    success, image = cap.read()
    # current_frame
    frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
    # if(frame < 3600):
    #     continue
    if not success:
      break

    # To improve performance, optionally mark the image as not writeable to
    # pass by reference.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = holistic.process(image)

    # ---- Body pose ----
    rows, cols, _ = image.shape
    try:
        pose_landmarks = landmarks_list_to_array(results.pose_world_landmarks)
# also can use results.pose_landmarks
        #                                world_pose_landmarks                                =
world_landmarks_list_to_array(results.pose_world_landmarks, image.shape)
```

```
    add_extra_points(pose_landmarks)
    # add_extra_points(world_pose_landmarks)
    body_pose = {
        'predictions': pose_landmarks,
        'frame': frame,
        'height': rows,
        'width': cols
    }
except:
    body_pose = {
        'predictions': [],
        'frame': frame,
        'height': rows,
        'width': cols
    }
# ---- Hands ----
hands_array_R = []
hands_array_L = []
if results.left_hand_landmarks:
    hands_array_L = landmarks_list_to_array(results.left_hand_landmarks)
if results.right_hand_landmarks:
    hands_array_R = landmarks_list_to_array(results.right_hand_landmarks)
hands_pose = {
```

```
    'handsR': hands_array_R,
    'handsL': hands_array_L,
    'frame': frame
}
yield hands_pose

if debug:
    # Draw landmark annotation on the image.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    # mp_drawing.draw_landmarks(
    #     image,
    #     results.face_landmarks,
    #     mp_holistic.FACEMESH_CONTOURS,
    #     landmark_drawing_spec=None,
    #     connection_drawing_spec=mp_drawing_styles
    #     .get_default_face_mesh_contours_style())
    # mp_drawing.draw_landmarks(
    #     image,
    #     results.pose_landmarks,
    #     mp_holistic.POSE_CONNECTIONS,
    #     landmark_drawing_spec=mp_drawing_styles  #
            .get_default_pose_landmarks_style())
```

```
        if results.right_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                results.right_hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())
        if results.left_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                results.left_hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())

        # Flip the image horizontally for a selfie-view display.
        cv2.imshow('MediaPipe Holistic', cv2.flip(image, 1))
        if cv2.waitKey(5) & 0xFF == 27:
          break
    cap.release()


def   Complete_pose_Video(video_path,debug=False):
```

```
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_holistic = mp.solutions.holistic
mp_hands = mp.solutions.hands
json_path = "TestPath"
cap = cv2.VideoCapture(video_path)
# cap = cv2.VideoCapture(0)
with mp_holistic.Holistic(
    min_detection_confidence=0.5,
    min_tracking_confidence=0.8,
    model_complexity=2) as holistic:
  while cap.isOpened():
    success, image = cap.read()
    # current_frame
    frame = cap.get(cv2.CAP_PROP_POS_FRAMES)
    if not success:
      break

    # To improve performance, optionally mark the image as not writeable to
    # pass by reference.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = holistic.process(image)
```

```
        # ---- Body pose ----
        rows, cols, _ = image.shape
        try:
            pose_landmarks = landmarks_list_to_array(results.pose_world_landmarks)
# also can use results.pose_landmarks
            # world_pose_landmarks = world_landmarks_list_to_array(results.pose_world_landmarks, image.shape)
            add_extra_points(pose_landmarks)
            # add_extra_points(world_pose_landmarks)
            body_pose = {
                'predictions': pose_landmarks,
                'frame': frame,
                'height': rows,
                'width': cols
            }
        except:
            body_pose = {
                'predictions': [],
                'frame': frame,
                'height': rows,
                'width': cols
            }
        # ---- Hands ----
```

```
hands_array_R = []
hands_array_L = []
if results.left_hand_landmarks:
    hands_array_L = landmarks_list_to_array(results.left_hand_landmarks)
if results.right_hand_landmarks:
    hands_array_R = landmarks_list_to_array(results.right_hand_landmarks)
hands_pose = {
    'handsR': hands_array_R,
    'handsL': hands_array_L,
    'frame': frame
}

# ---- Face ----

# facial_expression = mocap.get_frame_facial_mocap(image,frame)
# if facial_expression is None:
#     facial_expression = {
#     'leftEyeWid': -1,
#     'rightEyeWid': -1,
#     'mouthWid': -1,
#     'mouthLen': -1,
#     'frame': frame
#     }
```

```
json_data = {
    'bodyPose': body_pose,
    'handsPose': hands_pose,
    'frame': frame
}
# print(json_data)
yield json_data

if debug:
    # Draw landmark annotation on the image.
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    mp_drawing.draw_landmarks(
        image,
        results.face_landmarks,
        mp_holistic.FACEMESH_CONTOURS,
        landmark_drawing_spec=None,
        connection_drawing_spec=mp_drawing_styles
        .get_default_face_mesh_contours_style())
    mp_drawing.draw_landmarks(
        image,
        results.pose_landmarks,
```

```
            mp_holistic.POSE_CONNECTIONS,
            landmark_drawing_spec=mp_drawing_styles
            .get_default_pose_landmarks_style())
        if results.right_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                results.right_hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())

        # Flip the image horizontally for a selfie-view display.
        cv2.imshow('MediaPipe Holistic', cv2.flip(image, 1))
        if cv2.waitKey(5) & 0xFF == 27:
          break
    cap.release()


#add_extra_points([])

def Hand_pose_video(video_path, debug=False):
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles
```

```
    mp_hands = mp.solutions.hands
    # cap = cv2.VideoCapture(0)
    cap = cv2.VideoCapture(video_path)
    # cframe = cap.get(cv2.CV_CAP_PROP_POS_FRAMES) # retrieves the current frame number
    # tframe = cap.get(cv2.CAP_PROP_FRAME_COUNT) # get total frame count
    # print(tframe)
    # fps = cap.get(CV_CAP_PROP_FPS)  # get the FPS of the videos
    frame = 0
    with  mp_hands.Hands(
            model_complexity=1,
            max_num_hands=2,
            min_detection_confidence=0.6,
            min_tracking_confidence=0.8) as hands:
        while cap.isOpened():
            success, image = cap.read()
            #current_frame
            frame = cap.get(cv2.CAP_PROP_POS_FRAMES)

            if not success:
                # print("Ignoring empty camera frame.")
                # If loading a video, use 'break' instead of 'continue'.
                break
                # continue
```

```
# To improve performance, optionally mark the image as not writeable to
# pass by reference.
image.flags.writeable = False
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
results = hands.process(image)

# Draw the hand annotations on the image.
image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
hands_array_R = []
hands_array_L = []


if results.multi_hand_landmarks:
    index = 0
    for count, hand_landmarks in enumerate(results.multi_hand_landmarks):
        index += 1
        mp_drawing.draw_landmarks(
            image,
            hand_landmarks,
            mp_hands.HAND_CONNECTIONS,
            mp_drawing_styles.get_default_hand_landmarks_style(),
```

```
            mp_drawing_styles.get_default_hand_connections_style())

        if results.multi_handedness[count].classification[0].label == "Left":
            hands_array_L = landmarks_list_to_array(hand_landmarks)
            # print("L")
        elif results.multi_handedness[count].classification[0].label == "Right":
            hands_array_R = landmarks_list_to_array(hand_landmarks)
            # print("R")



json_data = {
    'handsR': hands_array_R,
    'handsL': hands_array_L,
    'frame': frame
}
yield json_data
if debug:
    json_path = "C:/Danial/Projects/Danial/DigiHuman/Backend/hand_json/"
    Save_Json(json_path,frame,json_data)
# Flip the image horizontally for a selfie-view display.
cv2.imshow('MediaPipe Hands', cv2.flip(image, 1))
if cv2.waitKey(5) & 0xFF == 27:
    break
```

```
    cap.release()



# if__name__== '__main__':

    # for i in Complete_pose_Video(video_path="D:\\pose\\New\\2022-07-14\\C2828.MP4",debug=True):

    #     continue

    # for i in Hands_Full(video_path="C:\Danial\Projects\Danial\DigiHuman\Backend\Video\WIN_20220414_23_51_39_Pro.mp4"):

    #     continue



    # for i in Pose_Video(video_path="D:\\pose\\New\\2022-07-14\\C2831.MP4", debug=True):

    #     continue

#

#     print("finished")
```

DECODER_TRAIN.PY

```
class DecoderTrain():

    def__init__(self,inference_model,shared_latent_space):



        self.decoder_hiddenlayer = Dense(inference_model.CenterDimension, activation='relu')

        self.decoder_up = Dense(inference_model.filters * 14 * 14, activation='relu')
```

```
        if backend.image_data_format() == 'channels_first':
            output_size = (inference_model.batch_size, inference_model.filters, 14, 14)
        else:
            output_size = (inference_model.batch_size, 14, 14, inference_model.filters)

        self.decoder_reshape = Reshape(output_size[1:])
        self.decoder_Firstlayer = Conv2DTranspose(inference_model.filters,
kernel_size=inference_model.convsize, padding='same', strides=1, activation='relu')
        self.decoder_Secondlayer = Conv2DTranspose(inference_model.filters,
kernel_size=inference_model.convsize, padding='same', strides=1, activation='relu')

        if backend.image_data_format() == 'channels_first':
            output_size = (inference_model.batch_size, 29, 29, inference_model.filters)
        else:
            output_size = (inference_model.batch_size, inference_model.filters, 29, 29)

        self.decoder_Thirdlayer = Conv2DTranspose(inference_model.filters,
kernel_size=(3, 3), strides=(2, 2), padding='valid', activation='relu')
        self.decoder_1 = Conv2D(inference_model.channel, kernel_size=2,
padding='valid', activation='sigmoid')

        x_hiddenlayer = self.decoder_hiddenlayer(shared_latent_space.Z)
        x_up = self.decoder_up(x_hiddenlayer)
        x_reshape = self.decoder_reshape(x_up)
        x_Firstlayer = self.decoder_Firstlayer(x_reshape)
```

```
x_Secondlayer = self.decoder_Secondlayer(x_Firstlayer)

x_Thirdlayer = self.decoder_Thirdlayer(x_Secondlayer)

X_1 = self.decoder_1 (x_Thirdlayer)


logc = np.log(2 * np.pi)

def GMM(Y, Y_1, Y_2):

    return backend.mean(-(0.5 * logc + 0.5 * Y_2) - 0.5 * ((Y - Y_1)**2 /
backend.exp(Y_2)), axis=-1)


def LossFunction(X, X_1):

    X = backend.flatten(X)

    X_1 = backend.flatten(X_1)

    Lp = 0.5 * backend.mean( 1 + inference_model.Z_2 -
backend.square(inference_model.Z_1) - backend.exp(inference_model.Z_2), axis=-
1)

    Lx = - metrics.binary_crossentropy(X, X_1) # Pixels have a Bernoulli
distribution

    Ly = GMM(inference_model.Y, inference_model.Y_1,
inference_model.Y_2) # Voxels have a Gaussian distribution

    loss = backend.mean(Lp + 10000 * Lx + Ly)

    totalloss = - loss

    return  totalloss


self.TDNN= Model(inputs=[inference_model.X, inference_model.Y,
inference_model.Y_1, inference_model.Y_2], outputs=X_1,name = 'TDNN')

 opt_method = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
```

```
epsilon=1e-08, decay=0.0)
        self.TDNN.compile(optimizer = opt_method, loss = LossFunction)
        self.TDNN.summary()
```

DUALLEARNING.PY

```
def__init__(self,XY_TrainLength,Dimension2,K,C):
        self.tau_alpha = 1
        self.tau_beta = 1
        self.eta_alpha = 1
        self.eta_beta = 1
        self.gamma_alpha = 1
        self.gamma_beta = 1
        self.XY_TrainLength = XY_TrainLength
        self.Dimension2 = Dimension2
        self.K = K
        self.C = C
        self.rho = 0.1
        self.Z_1 = np.mat(random.random(size=(self.XY_TrainLength,self.K))) # K
shape
        self.B_mu = np.mat(random.random(size=(self.K,self.Dimension2)))      # use
K shape with dimension of Y and it uses tau


def updateParameters(self,Z_2,Y_train,Z_1):
```

```
        self.Z_1 = Z_1

        # update B

        temp1 = np.exp(Z_2)

        temp2 = self.Z_1.T * Z_1 + np.mat(np.diag(temp1.sum(axis=0)))

        temp3 = self.tau_mu * np.mat(np.eye(self.K))

        sigma_b = (self.gamma_mu * temp2 + temp3).I

        self.B_mu = sigma_b * self.gamma_mu * self.Z_1.T * (np.mat(Y_train) -
self.R_mu * self.H_mu)

        # update H

        RTR_mu = self.R_mu.T * self.R_mu + self.XY_TrainLength * self.sigma_r

        self.sigma_h = (self.eta_mu * np.mat(np.eye(self.C)) + self.gamma_mu *
RTR_mu).I

        self.H_mu = self.sigma_h * self.gamma_mu * self.R_mu.T * (np.mat(Y_train)
- self.Z_1 * self.B_mu)

        # update R

        HHT_mu = self.H_mu * self.H_mu.T + self.Dimension2 * self.sigma_h

        sigma_r = (np.mat(np.eye(self.C)) + self.gamma_mu * HHT_mu).I

        R_mu = (sigma_r * self.gamma_mu * self.H_mu * (np.mat(Y_train) - self.Z_1
* self.B_mu).T).T

        # update tau

        tau_alpha_new = self.tau_alpha + 0.5 * self.K * self.Dimension2

        tau_beta_new = self.tau_beta + 0.5 * ((np.diag(self.B_mu.T *
self.B_mu)).sum() + self.Dimension2 * sigma_b.trace())

        tau_mu = tau_alpha_new / tau_beta_new

        tau_mu = tau_mu[0,0]
```

```
        # update eta
        eta_alpha_new = self.eta_alpha + 0.5 * self.C * self.Dimension2
        eta_beta_new = self.eta_beta + 0.5 * ((np.diag(self.H_mu.T *
self.H_mu)).sum() + self.Dimension2 * self.sigma_h.trace())
        eta_mu = eta_alpha_new / eta_beta_new
        eta_mu = eta_mu[0,0]
        return self.Y_1, self.Y_2


    def Bayesian(self,s,Z_1,Y_test,i):
        HHT = self.H_mu * self.H_mu.T + self.Dimension2 * self.sigma_h
        Temp = self.gamma_mu * np.mat(np.eye(self.Dimension2)) -
(self.gamma_mu**2) * (self.H_mu.T * (np.mat(np.eye(self.C)) + self.gamma_mu *
HHT).I * self.H_mu)
        z_sigma_test = (self.B_mu * Temp * self.B_mu.T + (1 + 0.1 *
s.sum(axis=0)[0,0]) * np.mat(np.eye(self.K)) ).I
        z_mu_test = (z_sigma_test * (self.B_mu * Temp * (np.mat(Y_test)[i,:]).T +
self.rho * np.mat(Z_1).T * s )).T
        return z_sigma_test, z_mu_test
```

INFERENCE_MODEL.PY

```
class GlobalConfiguration():
    X_train = pd.read_csv(data_dir+"X_train.csv")
    X_train = X_train.iloc[:,1:]
    X_train = X_train.to_numpy()
    X_train = X_train.astype('float32') / 255.
```

```
X_test = pd.read_csv(data_dir+"X_test.csv")
X_test = X_test.iloc[:,1:]
X_test = X_test.to_numpy()
X_test = X_test.astype('float32') / 255.
Y_train = pd.read_csv(data_dir+"Y_train.csv")
Y_train = Y_train.to_numpy()
Y_test = pd.read_csv(data_dir+"Y_test.csv")
Y_test = Y_test.to_numpy()

resolution = 28
Iteration = 50
Epoch = 1
batch_size = 10
CenterDimension = 128

X_train = X_train.reshape([X_train.shape[0], 1, resolution, resolution])
X_test = X_test.reshape([X_test.shape[0], 1, resolution, resolution])
normalize = preprocessing.MinMaxScaler(feature_range=(0, 1))
Y_train = normalize.fit_transform(Y_train)
Y_test = normalize.transform(Y_test)

XY_TrainLength=X_train.shape[0]
XY_TestLength=X_test.shape[0]
```

```
Dimension1 = X_train.shape[1]*X_train.shape[2]*X_train.shape[3]
Dimension2 = Y_train.shape[1]

K = 6
C = 5

Beta = 1 # Beta-VAE for Learning Disentangled Representations
rho = 0.1 # posterior regularization parameter
k = 10     # k-nearest neighbors
t = 10.0 # kernel parameter in similarity measure
L = 100   # Monte-Carlo sampling

np.random.seed(1000)

rows, cols, channel = 28, 28, 1
filters = 64
convsize = 3
img_size = (rows, cols, channel)

if backend.image_data_format() == 'channels_first':
    img_size = (rows, cols, channel)
else:
    img_size = (channel, rows, cols)
```

RECONSTRUCTOR.PY

```
class Reconstructor():
    def___init__(self):
        df= pd.read_csv(data_dir+'matrix.csv')
        self.S=np.mat(np.mat(df))
        self.X_reconstructed_test = None
        self.X_reconstructed_train = None
        self.XY_TrainLength = None
        self.XY_TestLength = None


def   constructXtrain(self,inference_model,training,testing):
        self.XY_TrainLength = inference_model.XY_TrainLength
        self.X_reconstructed_train = np.zeros((inference_model.XY_TrainLength,
inference_model.channel, inference_model.rows, inference_model.cols))
        for i in range(inference_model.XY_TrainLength):
            sample1 = testing.Decoder.predict(training.Z_1[i,:], batch_size=1)
            sample1 = sample1.reshape((-1, 1,28,28))
            self.X_reconstructed_train[i,:,:,:] = sample1

self.constructCSV(self.X_reconstructed_train,self.XY_TrainLength,'X_train.csv','X
_reconstructed_train.csv')
```

SHARED_LATENT_SPACE.PY

```
def Vae(args):
    Z_1, Z_2 = args
    epsilon = backend.random_normal(shape=(backend.shape(Z_1)[0], 6), mean=0.,
stddev=1.0)
    return Z_1 + backend.exp(Z_2) * epsilon


class SharedLatentSpace():
    def__init__(self):
        self.Z = ''
    def calculateZ(self,obj):
        self.Z = Lambda(Vae, output_shape =(obj.K,))([obj.Z_1, obj.Z_2])
        a=5
        print("success...  ")
```

TESTING.PY

```
class Testing():
    def__init__(self,inference_model,decoder_train):
        Z_predict = Input(shape=(inference_model.K,))
        reconstructed_x_hiddenlayer =  decoder_train.decoder_hiddenlayer(Z_predict)
        reconstructed_x_up = decoder_train.decoder_up(reconstructed_x_hiddenlayer)
        reconstructed_x_reshape =
decoder_train.decoder_reshape(reconstructed_x_up)
        reconstructed_x_Firstlayer =
```

```
 decoder_train.decoder_Firstlayer(reconstructed_x_reshape)

        reconstructed_x_Secondlayer =
decoder_train.decoder_Secondlayer(reconstructed_x_Firstlayer)

        reconstructed_x_Thirdlayer =
decoder_train.decoder_Thirdlayer(reconstructed_x_Secondlayer)

        reconstructed_X_1 = decoder_train.decoder_1(reconstructed_x_Thirdlayer)

        self.Decoder = Model(inputs=Z_predict, outputs=reconstructed_X_1)
```

TRAINING.PY

PIX2PIX_TRAINER.PY

```
from models.networks.sync_batchnorm import DataParallelWithCallback
from models.pix2pix_model import Pix2PixModel
class Pix2PixTrainer():
  def___init__(self, opt):
    self.opt = opt
    self.pix2pix_model = Pix2PixModel(opt)
    if len(opt.gpu_ids) > 0:
       self.pix2pix_model = DataParallelWithCallback(self.pix2pix_model,
                                                     device_ids=opt.gpu_ids)
       self.pix2pix_model_on_one_gpu = self.pix2pix_model.module
    else:
        self.pix2pix_model_on_one_gpu = self.pix2pix_model
```

```
    self.generated = None
    if opt.isTrain:
        self.optimizer_G, self.optimizer_D = \
            self.pix2pix_model_on_one_gpu.create_optimizers(opt)
        self.old_lr = opt.lr

def run_generator_one_step(self, data):
    self.optimizer_G.zero_grad()
    g_losses, generated = self.pix2pix_model(data, mode='generator')
    g_loss = sum(g_losses.values()).mean()
    g_loss.backward()
    self.optimizer_G.step()
    self.g_losses = g_losses
    self.generated = generated

def run_discriminator_one_step(self, data):
    self.optimizer_D.zero_grad()
    d_losses = self.pix2pix_model(data, mode='discriminator')
    d_loss = sum(d_losses.values()).mean()
    d_loss.backward()
    self.optimizer_D.step()
    self.d_losses = d_losses
```

```
def get_latest_losses(self):
    return {**self.g_losses, **self.d_losses}

def get_latest_generated(self):
    return self.generated

def update_learning_rate(self, epoch):
    self.update_learning_rate(epoch)

def save(self, epoch):
    self.pix2pix_model_on_one_gpu.save(epoch)



def update_learning_rate(self, epoch):
    if epoch > self.opt.niter:
        lrd = self.opt.lr / self.opt.niter_decay
        new_lr = self.old_lr - lrd
    else:
        new_lr = self.old_lr

    if new_lr != self.old_lr:
        if self.opt.no_TTUR:
```

```
            new_lr_G = new_lr
            new_lr_D = new_lr
        else:
            new_lr_G = new_lr / 2
            new_lr_D = new_lr * 2

        for param_group in self.optimizer_D.param_groups:
            param_group['lr'] = new_lr_D
        for param_group in self.optimizer_G.param_groups:
            param_group['lr'] = new_lr_G
        print('update learning rate: %f -> %f' % (self.old_lr, new_lr))
        self.old_lr = new_lr
```

VISUALIZER.PY

```
class Visualizer():
def__init__(self,inference_model):
   self.K = inference_model.K
   self.resolution = inference_model.resolution
   self.XY_TrainLength = inference_model.XY_TrainLength
   self.XY_TestLength = inference_model.XY_TestLength
   self.mse = 0
   self.ssim = 0
   self.psnr = 0
```

```
        self.accuracy = 0
    def cache_clear(self):
        self.mse = 0
        self.ssim = 0
        self.psnr = 0
        self.accuracy = 0
    def data_display(self,length,data,image_name,z=False,isTest=False):
        for j in range(1):
            plt.figure(figsize=(28, 28))
            for i in range(length):
                if isTest:
                    ax = plt.subplot(1, 10, i +j*length*2 + 1)
                else:
                    ax = plt.subplot(10, 9, i +j*length*2 + 1)
                if z:
                    plt.imshow(np.rot90(np.fliplr(data[i+j*length].reshape(self.K ,
))),cmap="hot")
                else:
                        plt.imshow(np.rot90(np.fliplr(data[i+j*length].reshape(self.resolution
    ,self.resolution ))),cmap="hot")
                ax.get_xaxis().set_visible(False)
                ax.get_yaxis().set_visible(False)
        plt.savefig(plot_dir+image_name)
```

CHAPTER 7
SYSTEM TESTING

7.1 Unit Testing

Unit testing is a fundamental aspect of software development that involves testing individual components, or units, of code to ensure they function correctly in isolation. The primary objective of unit testing is to validate that each unit of the software performs as expected according to its specifications. By isolating g and testingsmall units of code independently, developers can identify and fix bugs early in the development process, leading to more robust and reliable software.

Unit tests are typically written by developers and are automated to run frequently, allowing for quick feedback on the code's correctness. They are written in tandem with the development of code and serve as a safety net to catch regressions when new changes are introduced. Unit tests also promote code modularity and maintainability by encouraging developers to write code that is modular and loosely coupled.

Testing Overview:

Test Case	Objective	Input	Expected Outcome
Facial Animation Quality	Ensure generated animations meet quality standards	Various inputs for facial expressions	Mean Absolute Error(MAE) of generated

			animations within acceptable range
Compatibility with3D Models	Verify compatibility with different 3D models	Various 3D models with Humanoid T-Pose rigs	Successful rigging of all loaded models
Export Options	Test exporting animations in various formats	Generated animations	Exported animations in specified format playable without issues
Noise Reduction Techniques	Verify effectiveness of noise reduction	Animations with known noise levels	Smoother motion and fewer artifacts after noise reduction
User Interface Functionality	Ensure correct operation of UI controls	Simulated user interactions with UI	All buttons, fields, and features respond correctly to user input
Human-in-the-Loop (HITL) Integration	Test impact of user feedback on animation quality	Provide user feedback on generated animations	Improvement inanimation quality based on user feedback
Comprehensive Solution	Verify handling of diverse animation scenarios	Use toolkit with various requirements(e.g., detailed facial expressions, different character types)	Effective handling of diverse animation scenarios

Integration with Style Gan and Rig Net	Verify smooth integration with Style Gan and Rig Net	Replace character face meshes and automatically rig 3D models	Extensive customization options for replaced face meshes; automatically rigged models ready for animation

TABLE 2 - ANIMATION TOOLKIT

Home Page

Feature	**Description**
Introduction	Brief overview of the toolkit's purpose and capabilities.
Navigation	Accessible links to different sections of the toolkit, such as resources, training, and support.
Key Features	Highlights of the toolkit's main functionalities, such as facial animation, compatibility, export options, etc.
Call to Action	Prominent button or link to start using the toolkit or access more detailed information.

TABLE 2.1 - HOMEPAGE SCREEN

Test case No.	Action	Expected output	Actual Output	Result
1	Hover Let's Begin button	Animates the let's begin button	Animates the button with some neon effect	Pass
2	Click Let's begin	Starts a window animation and then enters into the tech suite	Started a window animation and reached the tech suite	Pass

TABLE 2.2 - HOMEPAGE TOOLKIT FUNCTIONALITY

Training Screen

Feature	Description
Model Training	Options for training or fine-tuning models for specific characters or animation styles.
Parameter Adjustment	Controls and sliders for adjusting blend shape coefficients, facial expressions, or animation parameters.
Visualization Tools	Graphs, charts, or visual aids to help users understand the training process or monitor animation quality.
User Interaction	Buttons or fields for providing feedback, preferences, or customizations during the training process.
Progress Indicators	Visual cues or progress bars indicating the status of ongoing training tasks or animation refinements.

Table 2.3 – TRAINING SCREEN

Test case No.	Action	Expected output	Actual Output	Result
1	Select invalid numbers inform	Displays error message	Displays error message	Pass
2	Click Let's try button	Opens a modal with the estimated time left to train the model	Opens a modal with estimated time left	Pass
3	After neural network is trained	Redirects to screen which has previous results	Redirects to screen whichhas previous results	Pass

TABLE 2.4 – TRAINING MODEL FUNCTIONALITY

Resources Screen

Test case No.	Action	Expected output	Actual Output	Result
1	Move around a card	Creates an hover effect with deep	Creates an hover effect with deep	Pass

		shadows	shadows	
2	Click on Download button of a resource card	Opens a google drive link containing that particular file	Opens a google drive link containing that particular file	Pass
3	Move around the footer icons	Displays the link social media	Displays the social media	Pass

TABLE 3 – RESOURCES SCREEN

Results Screen

Feature	Description
Animation Preview	Display area showing a preview of the generated facial animation, allowing users to visualize the results.
Animation Controls	Play, pause, rewind, and fast-forward buttons for controlling the playback of the animation preview.
Frame Selection	Slider or dropdown menu for selecting specific frames of the animation to view or analyze in detail.

Blend shape Adjustment	Interface for fine-tuning blend shape coefficients or facial expressions to refine the animation quality.
Comparison Tools	Side-by-side or overlay comparison with reference images or videos to assess the accuracy of the animation.
Export Options	Buttons or menus for exporting the generated animation in various formats, such as video files or GIFs.
Performance Metrics	Display of quantitative metrics, such as Mean Absolute Error (MAE), to evaluate the accuracy of the animation.
User Feedback	Feedback form or survey to collect user opinions, suggestions, or issues encountered during animation generation.
Save and Share	Options for saving the animation project, sharing it with collaborators, or posting it on social media platforms.

TABLE 4.1 – RESULTS SCREEN

Test case No.	Action	Expected output	Actual Output	Result
1	Click on the right arrow on the animation edit header	Redirects to screen with charts and training summary	Redirects to screen with charts and training summary	Pass
2	Zoom In or Zoom out on the charts	Zoom in enhances the graph and zoom out shrinks the graph	Zoom in enhances the graph and zoom out shrinks the graph	Pass
3	Click Show Model	Opens a drawer with neural	Opens a drawer that has a neural	Pass

TABLE 4.2 RESULTS UI TESTCASES

7.2 Integration Testing

Integration testing is a pivotal phase in the development of the DigiHuman Toolkit, ensuring the seamless interaction of its various components and features. This testing process encompasses verifying the integration of the user interface elements to ensure smooth user interaction and consistent visual coherence. Additionally, it involves testing the integration of data processing modules, pre-trained models, and external datasets to guarantee accurate and efficient animation generation.

Furthermore, export functionalities and feedback mechanisms are rigorously tested to ensure that users can export animations in various formats and provide feedback seamlessly. Compatibility testing across different platforms and environments ensures that the toolkit functions reliably regardless of the setup. By conducting thorough integration testing, the DigiHuman Toolkit aims to deliver a cohesive and functional solution for generating lifelike facial animations with precision and efficiency.

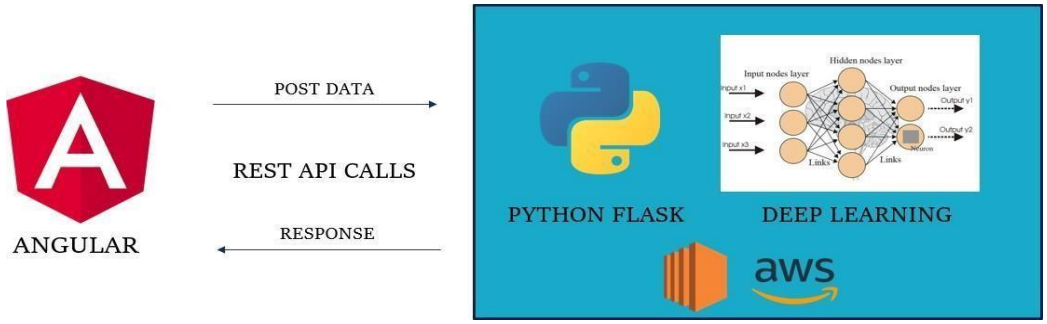

FIGURE 6- INTEGRATION ARCHITECTURE

Integration testing – Testing Bridge between Front-end and Back-end:

Test Case Description	Expected Outcome
Integration of User Interface Elements	All UI elements display correctly and interact as expected.
Integration of Data Processing Modules	Data processing modules seamlessly communicate and execute tasks.
Integration of Pre-trained Models	Pre-trained models load correctly and produce accurate predictions.
Integration with External Datasets	External datasets are imported without errors and used effectively.
Export Functionality Integration	Export options work as intended, allowing users to export animations in various formats.
Feedback Mechanism Integration	Feedback mechanisms capture user input accurately and reflect changes in the animation process.
Compatibility Testing across Platforms	The toolkit functions reliably across different platforms (e.g., Windows, macOS, Linux).

TABLE 5 - INTEGRATION TESTING

7.3 Performance Analysis Report

FIGURE 7 - HAND TEST RECOGNITION

FIGURE 8 – FULL BODY RECOGNITION

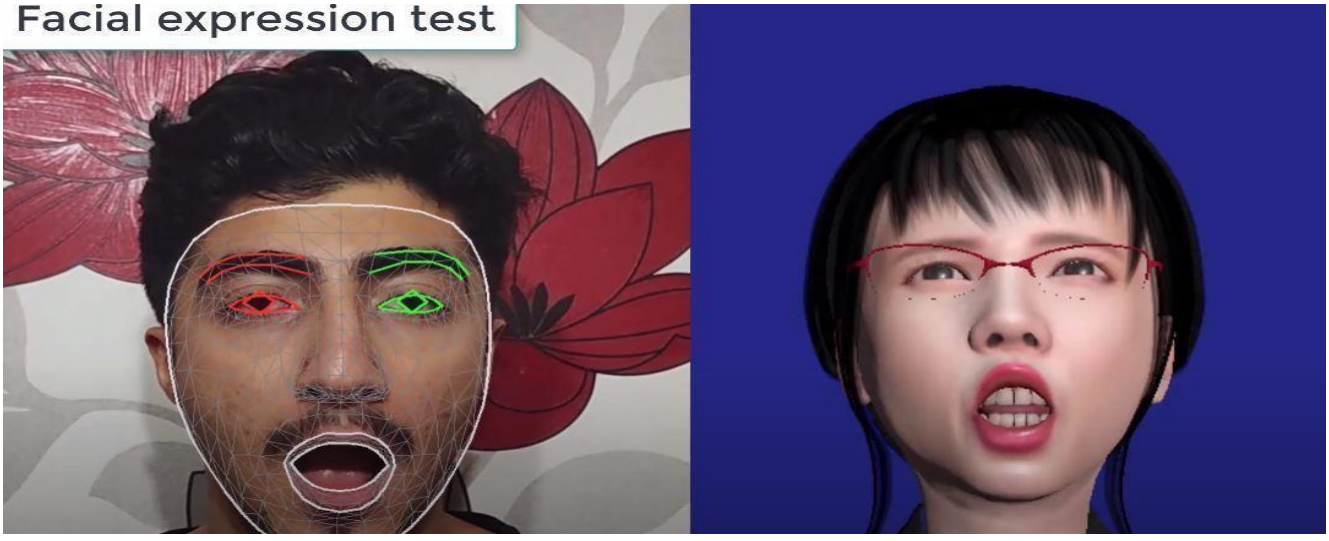

FIGURE 9– FACE EXPRESSION RECOGNITION

8. CONCLUSION

8.1 Future Enhancements and Discussion

The DigiHuman project offers a sophisticated system for automatic facial animation generation and manipulation, employing state-of-the-art technologies in machine learning, computer vision, and virtual reality. Through detailed analysis of its architecture and components, the project demonstrates seamless integration of backend server functionalities, frontend interface components, and machine learning models, enabling real-time visualization, user interaction, and customization of facial animations from input video data. Looking ahead, future enhancements could include advanced animation techniques like emotion-driven animation and lip- syncing, integration of external APIs for speech recognition and sentiment analysis, cloud-based rendering for scalability, cross-platform compatibility, collaborative animation features, improved user interface design, augmented reality integration, and community-building features, fostering a more inclusive and versatile platform for animation creation and sharing across diverse industries and domains.

APPENDICES

A.1 SAMPLE SCREENS AND INSTRUCTIONS

Find a 3D Character Model: Search for a suitable 3D character model either on the Unity Asset Store or from other sources like Mixamo. Ensure that the character model has a standard Humanoid rig for kinematic animations and a facial rig (Blend mesh) for rendering face animations.

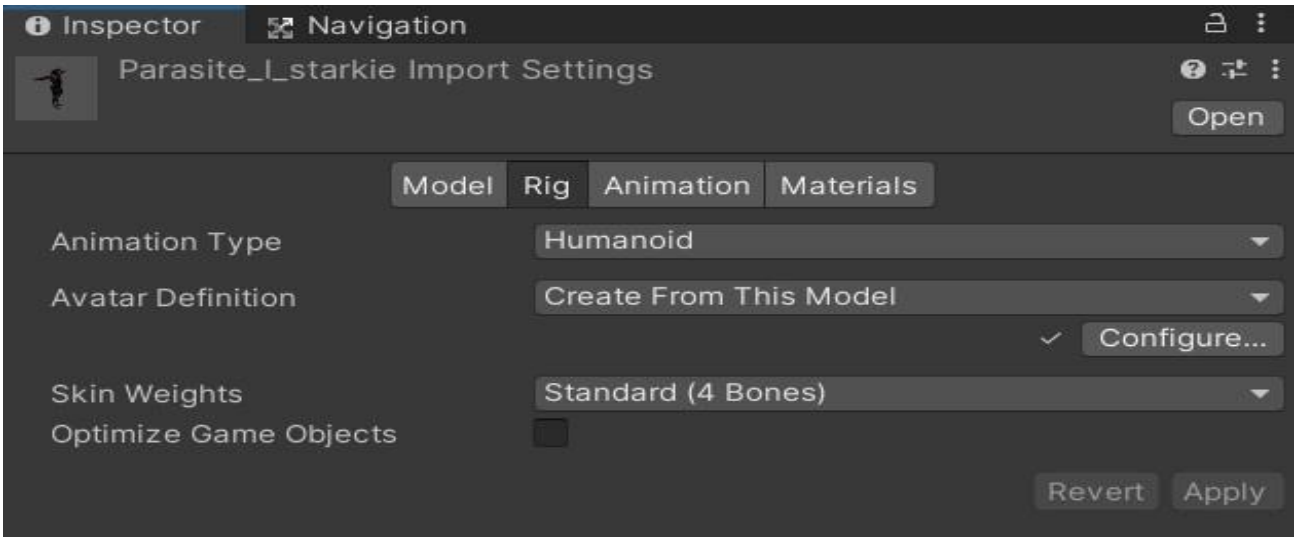

FIGURE 10 – 3D CHARACTER MODEL VIEW

Set Rig to Humanoid: Open the character settings in Unity and set the rig type to Humanoid. This ensures compatibility with the animation system and allows for standard humanoid animations.

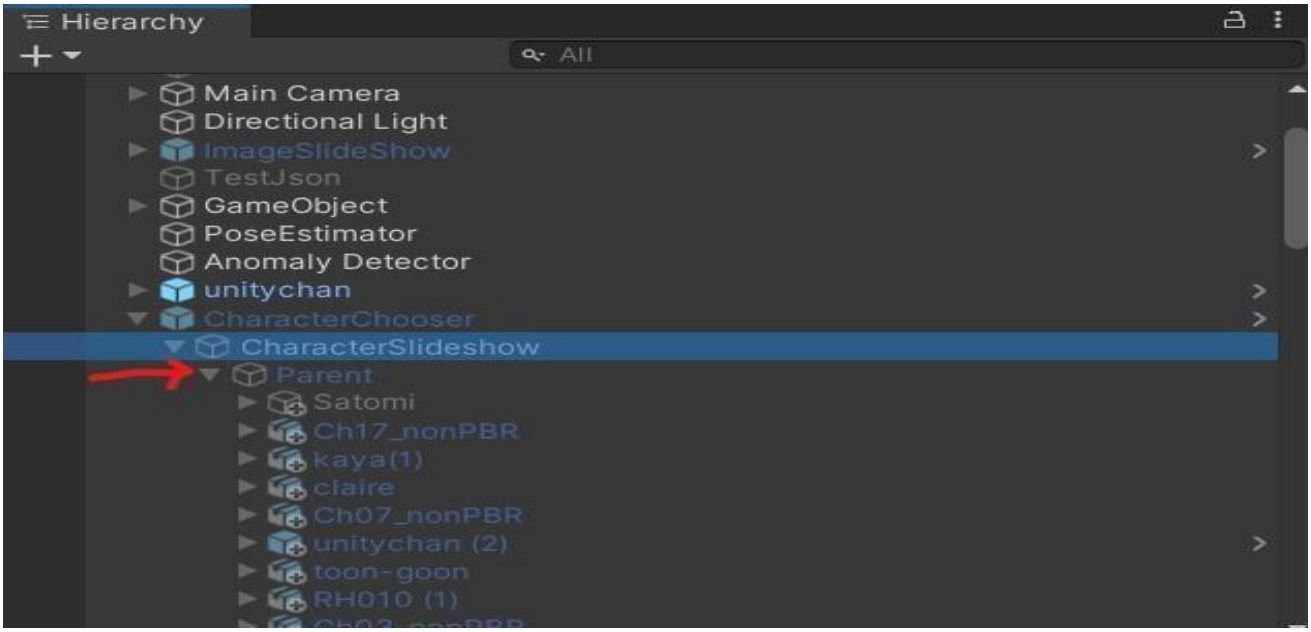

FIGURE 11 – CHARACTER SLIDE VIEW

Drag and Drop Character Model: Drag and drop your 3D character model into the Unity scene. Place it under the Character Chooser/Character Slideshow/Parent object. This object acts as a container for all the characters in the slideshow.

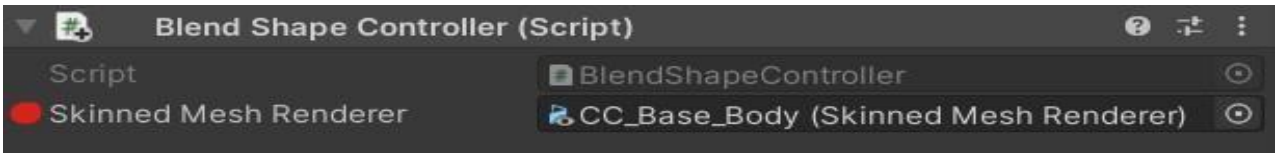

FIGURE 12 – SKINNED MESH RENDERER

Add BlendShapeController and Quality Data Components: Add the BlendShapeController and Quality Data components to the character object in the scene. These components handle facial blend shapes and animation quality settings, respectively.

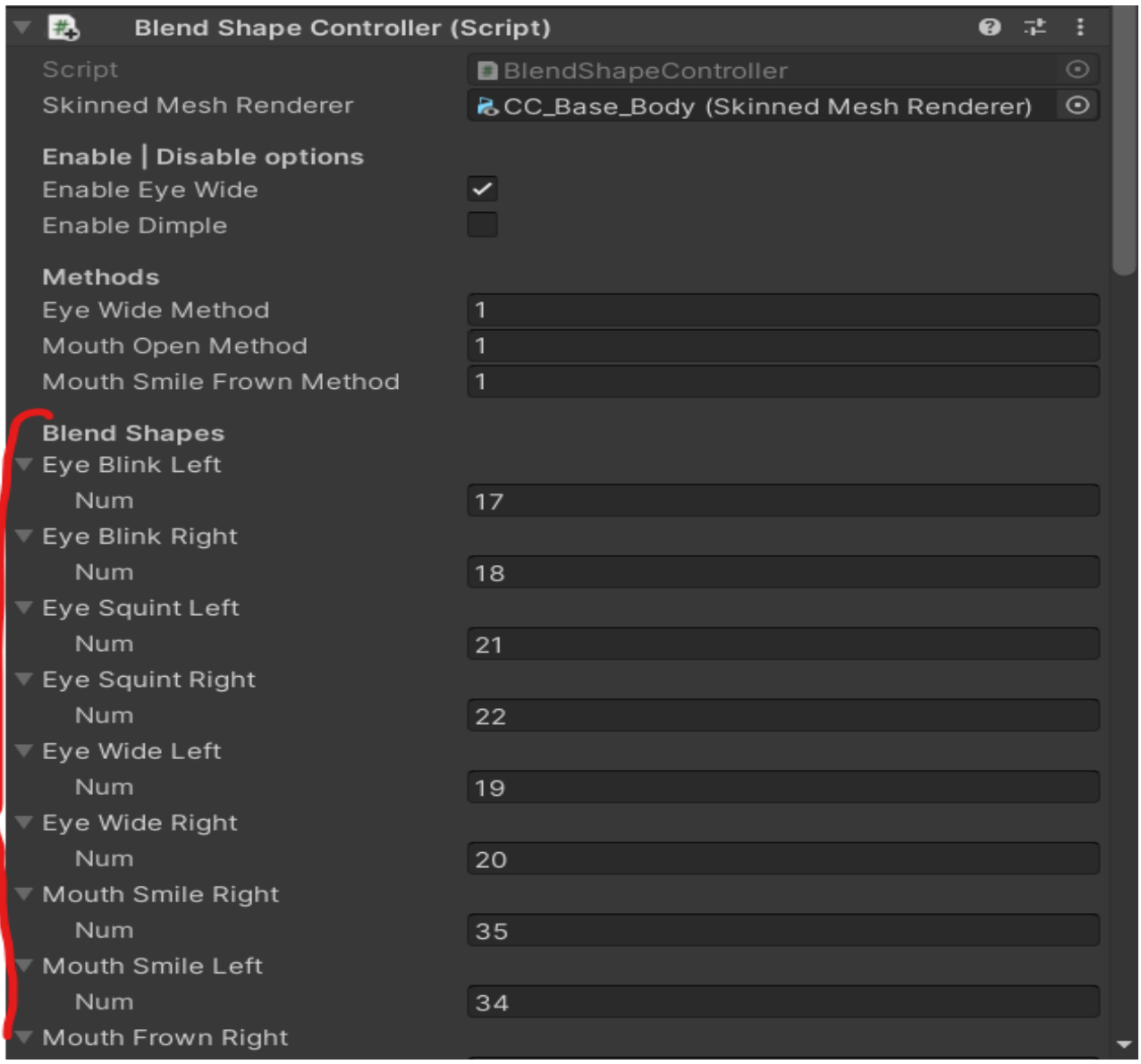

FIGURE 13 - BLEND SHAPE CONTROLLER

Set BlendShapeController Values: Configure the BlendShapeController component by setting the blend shape weights for facial expressions. Find each blend shape weight number under the SkinnedMeshRenderer component of your character and set those numbers in the Blend Shapes field of the BlendShapeController component.

Add Character to Character Slideshow Object: Open the Character Slideshow object in the scene hierarchy under Character Chooser/Character Slideshow. Add your character to the list of nodes in the Character Slideshow component. This ensures that your character is included in the character selection slideshow

Run the application, and you should now be able to select your character from the character selection slideshow. Your character will be ready for rendering animations with both kinematic and facial expressions.

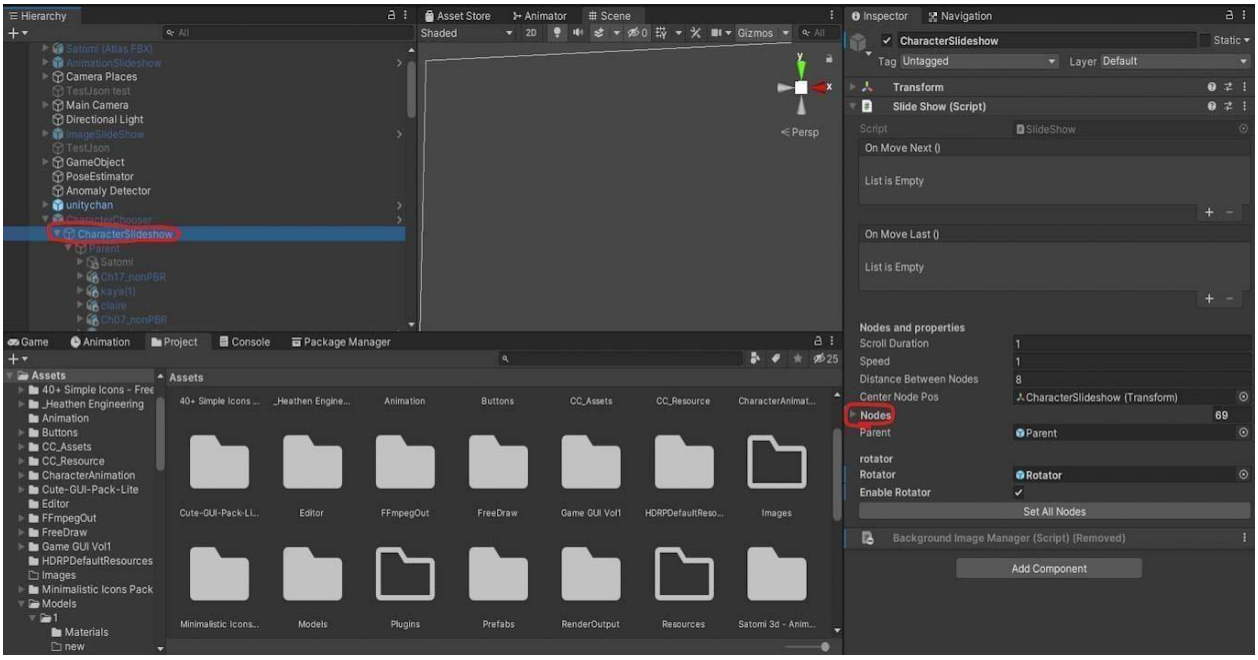

FIGURE 14 – BLEND SHAPE CONTROLLER VALUES

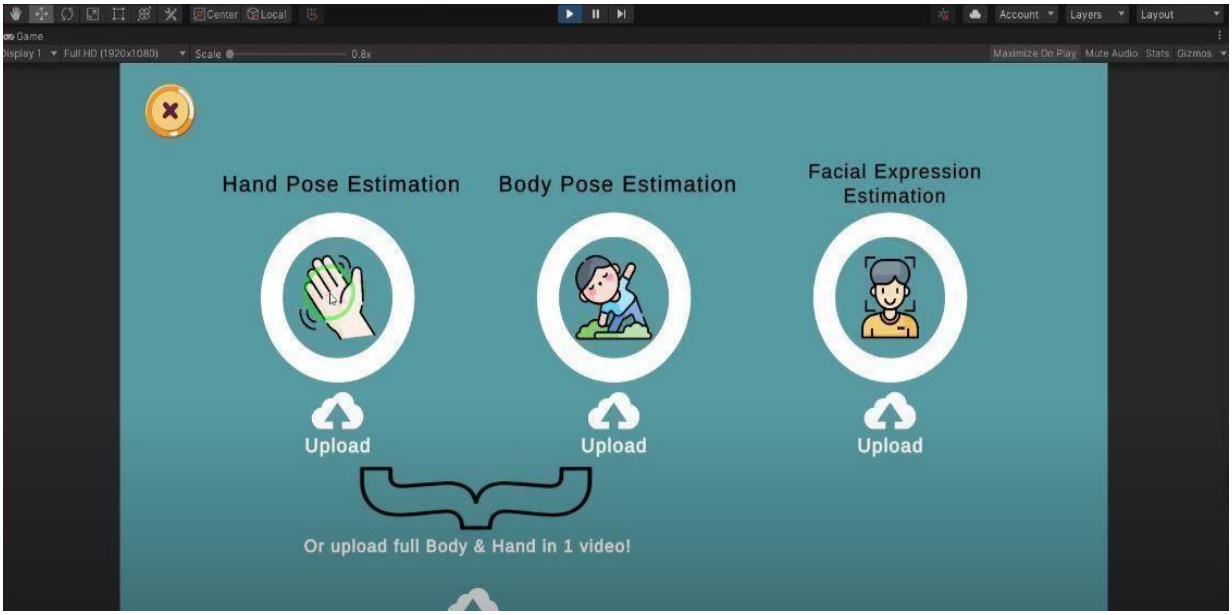

FIGURE 15 - CHARACTER GENERATION MODEL

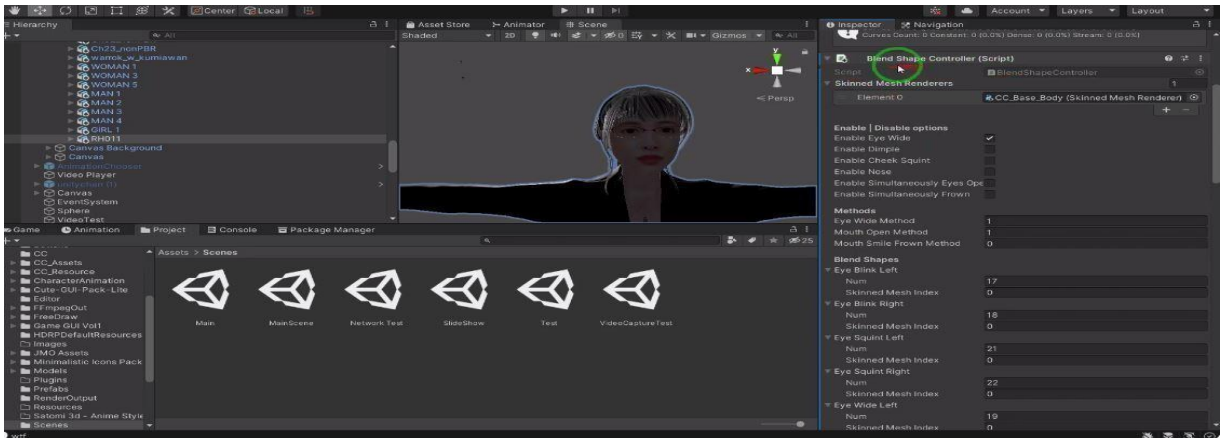

FIGURE 16 – MODEL OUTPUT 1

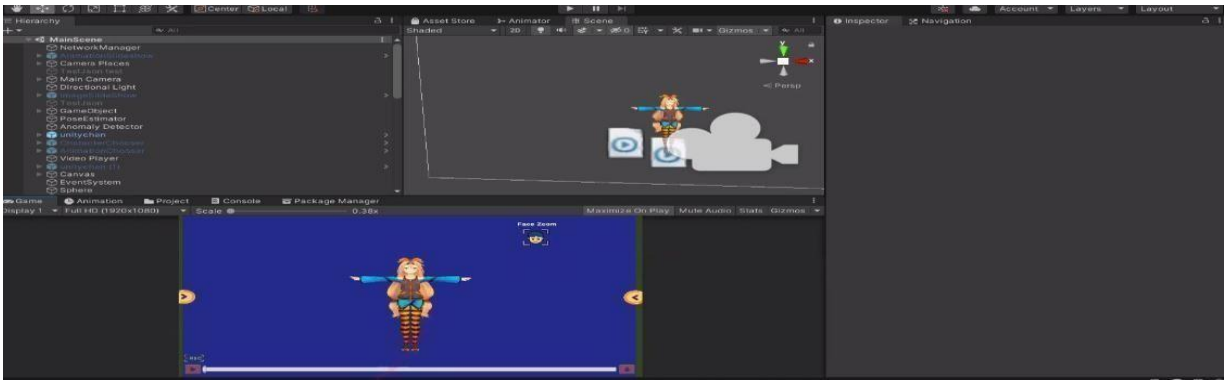

FIGURE 17- MODEL OUTPUT 2

A.2 PUBLICATIONS

REFERENCES

1. Badler, N. I., Palmer, M. S., & Bindiganavale, R. (1999) Animation Control for Real-Time Virtual Humans. Communications of the ACM, 42*(8), 64–73.

2. Bai, Z., Yao, N., Liu, L., Chen, H., & Wang, H. (2023) A Simple Approach to Animating Virtual Characters by Facial Expressions Reenactment. In 2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops

(VRW), pp. 585–586. IEEE.

3. Bai, Z., Yao, N., Mishra, N., Chen, H., Wang, H., & Magnenat Thalmann, N. (2021) Enhancing Emotional Experience by Building Emotional Virtual Characters in VR Volleyball Games. Computer Animation and Virtual Worlds, 32(3-4), e2008.

4. Bai, Z., Yao, N., Mishra, N., Chen, H., Wang, H., & M. Thalmann, N. (2021) Play with Emotional Characters: Improving User Emotional Experience by a Data-Driven Approach in VR Volleyball Games. In 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), pp. 458–459. IEEE.

5. Blanz, V., & Vetter, T. (1999) A Morphable Model for the Synthesis of 3D Faces. In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 187–194.

6. Brooke, J., et al. (1996) SUS-A Quick and Dirty Usability Scale. Usability Evaluation in Industry, 189(194), 4–7.

7. Burden, D., & Savin-Baden, M. (2019) Virtual Humans: Today and Tomorrow. CRC Press.

8. Cao, C., Weng, Y., Zhou, S., Tong, Y., & Zhou, K. (2013) Facewarehouse: A 3D Facial Expression Database for Visual Computing. IEEE Transactions on Visualization and Computer Graphics, 20(3), 413–425.

9. Chaudhuri, B., Vesdapunt, N., Shapiro, L., & Wang, B. (2020) Personalized Face Modeling for Improved Face Reconstruction and Motion Retargeting. In Computer

Vision–ECCV 2020: 16th European Conference, pp. 142–160. Springer.

10. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009) Imagenet: A Large-Scale Hierarchical Image Database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. IEEE.

11. Deng, Y., Yang, J., Xu, S., Chen, D., Jia, Y., & Tong, X. (2019) Accurate 3D Face Reconstruction with Weakly-Supervised Learning: From Single Image to Image Set. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 0–0.

12. Dou, P., Shah, S. K., & Kakadiaris, I. A. (2017) End-to-End 3D Face Reconstruction with Deep Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5908–5917.

13. Edwards, P., Landreth, C., Fiume, E., & Singh, K. (2016) Jali: An Animator-Centric Viseme Model for Expressive Lip Synchronization. ACM Transactions on Graphics (TOG), 35(4), 1–11.

14. Feng, Y., Wu, F., Shao, X., Wang, Y., & Zhou, X. (2018) Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network. In Proceedings of the European Conference on Computer Vision (ECCV), pp. 534–551.

15. Huang, G. B., Mattar, M., Berg, T., & Learned-Miller, E. (2008) Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. In Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition.

16. IEEE. (2009) A 3D Face Model for Pose and Illumination Invariant Face

Recognition. Genova, Italy.

17. Joshi, P., Tien, W. C., Desbrun, M., & Pighin, F. (2006) Learning Controls for Blend Shape Based Realistic Facial Animation. In ACM Siggraph 2006 Courses, pp. 17–es.

18. King, D. E. (2009) Dlib-ml: A Machine Learning Toolkit. The Journal of Machine Learning Research, 10, 1755–1758.

19. Lewis, J. P., Anjyo, K., Rhee, T., Zhang, M., Pighin, F. H., & Deng, Z. (2014) Practice and Theory of Blendshape Facial Models. Eurographics (State of the Art Reports), 1(8), 2.

20. Lewis, J. P., & Anjyo, K.-i. (2010) Direct Manipulation Blendshapes. IEEE Computer Graphics and Applications, 30(4), 42–50.

21. Li, T., Bolkart, T., Black, M. J., Li, H., & Romero, J. (2017) Learning a Model of Facial Shape and Expression from 4D Scans. ACM Transactions on Graphics, (Proc. SIGGRAPH Asia), 36(6), 194:1–194:17.

22. Liu, Z., Luo, P., Wang, X., & Tang, X. (2018) Large-Scale CelebFaces Attributes (CelebA) Dataset. Retrieved August, 15(2018):11.

23. Machidon, O. M., Duguleana, M., & Carrozzino, M. (2018) Virtual Humans in Cultural Heritage ICT Applications: A Review. Journal of Cultural Heritage, 33, 249–260.

24. Nogueira, P. (2011) Motion Capture Fundamentals. In Doctoral Symposium in Informatics Engineering, vol. 303.

25. Pan, Y., Zhang, R., Cheng, S., Tan, S., Ding, Y., Mitchell, K., & Yang, X. (2023) Emotional Voice Puppetry. IEEE Transactions on Visualization and Computer Graphics, 29(5), 2527–2535.

26. Paysan, P., Knothe, R., Amberg, B., Romdhani, S., & Vetter,T. (2009) A 3D Face Model for Pose and Illumination Invariant Face Recognition. In 2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance pp. 296–301. IEEE.

28. Peng, X., Huang, J., Denisova, A., Chen, H., Tian, F., & Wang, H. (2020) A Palette of Deepened Emotions: Exploring Emotional Challenge in Virtual Reality Games. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1–13.

29. Peng, X., Huang, J., Li, L., Gao, C., Chen, H., Tian, F., & Wang, H. (2019) Beyond Horror and Fear: Exploring Player Experience Invoked by Emotional Challenge in VR Games. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–6.

30. Peng, X., Meng, C., Xie, X., Huang, J., Chen, H., & Wang, H. (2022) Detecting Challenge from Physiological Signals: A Primary Study with a Typical Game Scenario. In CHI Conference on Human Factors in Computing Systems Extended

Abstracts, pp. 1–7.

31. Peng, X., Xie, X., Huang, J., Jiang, C., Wang, H., Denisova, A., Chen, H., Tian, F., & Wang, H. (2023) ChallengeDetect: Investigating the Potential of Detecting In-Game Challenge Experience from Physiological Measures. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–29.

32. Schroff, F., Kalenichenko, D., & Philbin, J. (2015) FaceNet: A Unified Embedding for Face Recognition and Clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 815–823.

33. Sharma, S., Verma, S., Kumar, M., & Sharma, L. (2019) Use of Motion Capture in 3D Animation: Motion Capture Systems, Challenges, and Recent Trends. In 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), pp. 289–294. IEEE.

34. Shi, T., Yuan, Y., Fan, C., Zou, Z., Shi, Z., & Liu, Y. (2019) Face-to-Parameter Translation for Game Character Auto-Creation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 161–170.

35. Shi, T., Zuo, Z., Yuan, Y., & Fan, C. (2020) Fast and Robust Face-to-Parameter Translation for Game Character Auto-Creation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 1733–1740.

36. Volonte, M., Ofek, E., Jakubzak, K., Bruner, S., & Gonzalez-Franco, M. (2022) Headbox: A Facial Blendshape Animation Toolkit for the Microsoft Rocketbox Library. In 2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), pp. 39–42. IEEE.

37. Wolf, L., Taigman, Y., & Polyak, A. (2017) Unsupervised Creation of Parameterized Avatars. In *Proceedings of the IEEE International Conference on Computer Vision, pp. 1530–1538.

DIGIHUMAN NEXUS: STREAMLINING ANIMATION WITH AI AND UNITY3D FOR VIRTUAL CHARACTERS

by Ragul Elumalai

Submission date: 20-Mar-2024 11:59AM (UTC+0530)
Submission ID: 2325560592
File name: DIGIHUMAN_NEXUS.pdf (522.69K)
Word count: 6727
Character count: 42541

Digihuman Nexus : Streamline Animation With AI And Unity3D For Virtual Characters

Josephine Leela R[a)], Sankaranarayanan K[b)], Ragul E[c)], Saran Y[d)]

Panimalar Engineering College,Chennai,India

Corresponding
[a)]Author:pecleela2005@gmail.com,[b)]sankaranarayanankesavan7@gmail.com,[c)]ra02032002@gmail.com,[d)]yvishva200317@gmail.com

Abstract --
Animating virtual characters in virtual reality (VR) has perpetually posed a significant challenge,particularly concerning the intricate conveyance of nuanced facial expressions vital for emotive communication. Conventional methods heavily rely on costly motion capture technology or labor- intensive manual inputs, significantly impeding accessibility and workflow efficiency. In response to this pressing need, our paper introduces DigiHuman Nexus, a groundbreaking solution meticulously crafted to automate facial animation generation for virtual human characters. Through the seamless integration of cutting-edge technologies such as Blaze Pose, Unity3D, and Media Pipe, DigiHuman Nexus heralds a momentous advancement in animation creation. Blaze Pose, renowned for its lightweight convolutional neural network architecture, excels in real-time human pose estimation on mobile devices, deftly producing 33 body key points at an impressive rate exceeding 30 frames per second. This remarkable capability proves invaluable for applications ranging from fitness tracking to sign language recognition. By synergizing Blaze Pose with DigiHuman Nexus, the framework achievesunparalleled animation capabilities, seamlessly infusing lifelike body movements into virtual character animations. Moreover, harnessing Media Pipe's sophisticated functionalities further enhances the precision of human movement portrayal within virtual environments. DigiHuman Nexus emerges as a pivotal milestone in the integration of cutting-edge technologies, poised to revolutionize animation creation and redefine the landscape of VR, augmented reality (AR), and beyond. With its versatile solutions tailored to diverse applications necessitating lifelike virtual character animations, DigiHuman Nexus holds the promise of transformative innovation.

Keywords: DigiHuman Nexus, Facial animation generation, Blaze Pose, Unity3D, Media Pipe, Real-time, Human pose estimation

INTRODUCTION

Virtual reality (VR) technology has garnered significant attention in recent years, fueled by the burgeoning popularity of immersive experiences such as the Metaverse. An integral aspect of VR experiences lies in the creation of lifelike virtual characters, particularly in the realm of facial animation, which plays a pivotal rolein emotive communication. However, traditionalmethods for animating virtual characters have been hindered by reliance on costly motion capture technology or labor-intensive manual inputs, posing challenges to accessibility and workflow efficiency .

In response to these challenges, our paper introduces DigiHuman Nexus, a groundbreaking solution poised to revolutionize facial animation generation for virtual human characters. Leveraging insights from recent advancements in pose estimation, hand tracking, and facial geometry prediction, DigiHuman Nexusrepresents a significant advancement in animation creation .

Blaze Pose, is a lightweight convolutional neural network architecture tailored for real-time human pose estimation. Developed by Google AI, Blaze Pose is renowned for its efficiency and accuracy in detecting and tracking human body key points, including joints and body parts, with exceptional speed and precision. By incorporating Blaze Pose into DigiHuman Nexus, the framework gains the ability to accurately capture and replicate lifelike body movements within virtual environments. With the capability to produce 33 body key points at a remarkable rate exceeding 30 frames per second, Blaze Pose empowers DigiHuman Nexus to seamlessly integrate realistic body animations into virtual character interactions. This integration not only enhances the visual fidelity of virtual characters but also significantly improves the overall user experience by creating more immersive and engaging virtual environments.

Media Pipe is a powerful framework developed by Google Research that offers a wide range of machine learning solutions for various multimedia applications, including pose estimation, hand tracking, facial recognition, and augmented reality effects. Within the context of DigiHuman Nexus, Media Pipe plays a crucial role in enhancing the precision and realism of human movement portrayal within virtual environments. By leveraging Media Pipe's sophisticated functionalities, DigiHuman Nexus achieves unparalleled fidelity in animation rendering, thereby elevating the immersive experience for users. Specifically, Media Pipe provides robust algorithms and pipelines for processing video streams and extracting valuable information such as key points, gestures, and facial expressions. These capabilities enable DigiHuman Nexus to accurately capture and reproduce intricate movements, gestures, and facial expressions, enhancing the overall realism and immersion of virtual characters. Furthermore, the utilization of Media Pipe enhances the precision of human movement portrayal within virtual environments. By harnessing Media Pipe's sophisticated functionalities, DigiHuman Nexus achieves unparalleled fidelity in animation rendering, elevating the immersive experience for users .

Furthermore, the utilization of Media Pipe enhances the precision of human movement portrayal within virtual environments. By harnessing Media Pipe's sophisticated functionalities, DigiHuman Nexus achieves unparalleled fidelity in animation rendering, elevating the immersive experience for users. Overall, the integration of Blaze Pose and Media Pipe within the DigiHuman Nexus framework represents a significant advancement in animation creation for VR environments, promising transformative innovation and redefining the landscape of virtual character animations.

Moreover, our approach draws inspiration from recent breakthroughs in face alignment and mesh vertex estimation. By treating each vertex as an independent landmark and strategically selecting points with higher variability and importance, DigiHuman Nexus ensures the creation of plausible and smooth surface representations, enhancing the realism of virtual characters .

Incorporating insights from object detection frameworks such as Blaze Face, DigiHuman Nexus optimizes facial feature detection on mobile GPUs, enabling efficient inference and real-time animation generation.

In summary, DigiHuman Nexus emerges as a pivotal milestone in the integration of cutting-edge technologies, poised to revolutionize animation creation and redefine the landscape of VR, augmented reality (AR), and beyond. With its versatile solutions tailored to diverse applications necessitating lifelike virtual character animations, DigiHuman Nexus holds the promise of transformative innovation

LITERATURE SURVEY

Innovative Approaches to Blend shape-based Animation Creation

INTRODUCTION

The realm of virtual reality (VR) animation creation has been subject to considerable exploration, with recent endeavors spanning both academic and industrial domains [4–6]. Traditional methodologies have leaned heavily on motion capture devices [27, 35], facilitating the replication of real-world movements onto virtual characters. Alternatively, parameter tuning techniques have emerged, focusing on adjusting specific animation parameters to achieve expressive character animations. Blend shapes, also known as morph targets or shape keys, represent a prominent technique in computer animation for modeling intricate facial expressions and movements [20, 22, 23]. Each blend shape within a 3D virtual human model delineates a distinct facial unit, encompassing features such as eyebrows, lips, and jaw movements. By amalgamating various blend shapes, animators can craft a diverse array of facial expressions by blending these shapes in different combinations. However, mainstream software platforms such as MAYA, Blender, and Unity3D have been pivotal in the design and editing of virtual humans based on blend shapes.

Recent endeavors have focused on automating the estimation of blend shape coefficients from diverse sources, including images or videos [1, 2, 38] and audio inputs [28]. For instance, Headbox [38] provides a specialized facial blend shape animation toolkit catering to the Microsoft Rocket box Library, enabling real-time facial animation from video input. JALI [15] represents another significant advancement, specializing in expressive lip-synchronization animation. Commercial tools like ARKit [2], Faceware suite [16], and "Live Link Face" [1] have gained prominence for their ability to compute blend shape weights and facilitate the replication of facial expressions on avatars.

However, existing methodologies exhibit certain limitations, primarily in their coupling with specific character rigs and blend shape topologies, thereby impeding applications on customized virtual characters. Moreover, commercial software solutions often entail exorbitant costs, limiting their accessibility. Diverging from previous approaches, our work emphasizes support for customized virtual characters, empowering users to create tailored solutions. We intend to open-source our solution to foster community-wide benefits and spur further innovation in animation creation.

Streamlining Character Auto Creation through 2D Image-driven Techniques

INTRODUCTION

The advent of automatic character generation technology, driven by 2D images, has heralded a transformative shift in animation creation paradigms. Pioneering methodologies, such as Tied Output Synthesis (TOS) proposed by Wolf et al. [39], and the Face-to-Parameter (F2P) approach introduced by Shi et al. [36], have leveraged adversarial training and iterative search processes, respectively, to generate parameterized avatars based on 2D facial images. Subsequent enhancements, exemplified by the Face- to-Parameter V2 (F2Pv2) method [37], have further streamlined the face parameter estimation process, significantly enhancing efficiency. These techniques strive to manipulate 3D characters resembling input facial images by estimating facial parameters, encompassing identity, expression, pose, texture, and hairstyle.

While the task of automatic character generation shares similarities with blend shape coefficients estimation, distinct nuances exist. Character generation entails a broader parameter space, encapsulating facial movements, attributes, pose, texture, and hairstyle, unlike animation creation, which predominantly focuses on facial muscle movements influencing expressions or poses. Our work aligns with animation creation, presenting unique technical challenges in decoupling and extracting expression and pose information from facial images.

Advancing 3D Face Modeling and Reconstruction for Enhanced Animation Generation

INTRODUCTION

The pursuit of precise 3D face modeling and reconstruction represents a cornerstone in animation generation endeavors. Classic problems in computer graphics, such as 3D face modeling and reconstruction, aim to derive 3D facial information from 2D facial images. Pioneering works, like the 3D Morphable Model (3DMM) proposed by Blanz et al. [7], have laid the foundation by utilizing Principal Component Analysis (PCA) to parameterize 3D facial meshes and optimize PCA parameters to fit input 2D facial images. Building upon this foundation, subsequent research endeavors, including BFM09 [19], Face Warehouse [10], and FLAME [24], have sought to enrich the representation power of Morphable Models across shape, texture, and expression domains.

In the era of deep learning, the advent of neural network models for predicting 3D facial parameters based on 2D images has propelled significant advancements in 3D face reconstruction [11, 13, 14, 17]. These methodologies leverage differentiable renderers to render 3D faces based on predicted 3DMM parameters, thereby generating supervision signals for training neural networks. Despite their promise, existing approaches often struggle with blend shape estimation, as the differentiable renderers employed are tailored specifically to fit 3DMM parameters and may lack universality for customized virtual characters in production settings. Leveraging the universality and reliability of 3DMM parameters, our approach adopts a parametric representation for 2D facial images in 3D space, facilitating explicit decoupling and extraction of expression and pose parameters.

Media pipe Integration for Enhanced Animation Fidelity

INTRODUCTION

Media pipe, developed by Google Research, stands as a comprehensive framework encompassing a plethora of machine learning solutions tailored for multimedia applications, including pose estimation, hand tracking, facial recognition, and augmented reality effects [15]. While initially conceived for general multimedia processing, Media pipe's versatility and robustness have paved the way for its integration into various domains, including virtual reality (VR) and animation creation.

Within the context of facial animation generation, Media pipe plays a pivotal role in enhancing the precision and realism of human movement portrayal within virtual environments [15]. By harnessing Media pipe's sophisticated functionalities, animators can achieve unparalleled

fidelity in animation rendering, thereby elevating the immersive experiencefor users.

Media pipe's pose estimation capabilities enable accurate tracking of body movements and gestures, allowing for dynamic and lifelike character animations [15]. Through robust algorithms and pipelines, Media pipe processes video streams and extracts valuable information such as key points, gestures, and facial expressions. This wealth of information empowers animators to accurately capture and reproduce intricate movements, gestures, and facial expressions, enhancing the overall realism and immersion of virtual characters.

Furthermore, Media pipe's hand tracking features provide additional layers of interaction and expression within virtual environments. By accurately tracking hand movements and gestures in real-time, animators can imbue virtual characters with lifelike gestures and interactions, further enriching the user experience [15].

Incorporating Media pipe into animation creation pipelines opens up new avenues for creativity and expression in VR and augmented reality (AR). Its robust and efficient algorithms enable real-time processing of multimedia data, making it well-suited for interactive and immersive applications. As such, the integration of Media pipe into animation creation frameworks represents a significant step forward in the quest for lifelike virtual experiences.,

Unity3D for Seamless Animation Integration

INTRODUCTION

Unity3D stands as a cornerstone in the realm of virtual environment creation, offering a robust and versatile platform for game development, simulation, and interactive experiences. With its powerful rendering engine, intuitive development tools, and extensive asset store, Unity3D provides a conducive environment for animators and developers to bring their virtual worlds to life.

In the context of facial animation generation, Unity3Doffers a plethora of tools and features tailored for character animation and expression. Its animation system allows for the creation of complex blend shape animations, enabling animators to sculpt lifelike facial expressions with ease. Additionally, Unity3D's real- time rendering capabilities ensure smooth and responsive animation playback, crucial for maintaining immersion in virtual environments.

Moreover, Unity3D's integration with external frameworks and technologies further enhances its animation capabilities. By seamlessly integrating with machine learning frameworks like TensorFlow and PyTorch, Unity3D enables the deployment of advanced AI-driven animation techniques, such as neural network-based pose estimation and facial expression recognition.

Furthermore, Unity3D's cross-platform support ensures that animations created within the Unity environment can be deployed across a wide range of devices, from desktop computers to mobile phones and virtual reality headsets. This versatility allows for widespread dissemination of animated content, reaching audiences across different platforms and devices.

In summary, Unity3D serves as a powerful and versatile platform for facial animation generation, offering a comprehensive suite of tools and features tailored for character animation and expression. Its seamless integration with external frameworks and technologies further expands its animation capabilities, making it a preferred choice for animatorsand developers alike.

Pose Estimation Advancements for Realistic Animation

INTRODUCTION

Recent advancements in pose estimation have significantly contributed to the realism and fidelity of character animations in virtual environments. Pose estimation algorithms, such as Blaze Pose, offer robust and efficient solutions for accurately tracking body movements and gestures in real-time.

Blaze Pose, a lightweight convolutional neural network architecture, excels in real-time human pose estimation on mobile devices, producing accurate and reliable results even in challenging conditions [5]. By leveraging Blaze Pose's capabilities, animators can capture and replicate lifelike body movements with exceptional speed and precision, enhancing the overall realism of character animations.

Furthermore, advancements in pose estimation have led to the development of sophisticated techniques for hand tracking and gesture recognition. By accurately tracking hand movements and gestures, animators can imbue virtual characters with lifelike interactions and expressions, further enhancing the immersive experience for users.

Incorporating these advancements into animation creation pipelines enables animators to create dynamic and expressive characters with ease. By leveraging the latest pose estimation techniques, animators can bring their virtual worlds to life, offering users immersive and engaging experiences across a wide range of platform devices.

METHODOLOGY

Overview

DigiHuman is an advanced animation generation system that leverages deep learning and computer vision techniques to create lifelike animations from input video data. Its architecture comprises both backend and frontend components, each playing a crucial role in the animation creation process. In the backend, DigiHuman processes input video data and semantic segmentation to estimate body poses, generate facial landmarks, compute facial expression values, and synthesize realistic background images. On the frontend, DigiHuman receives processed data from the backend and performs signal filtering, joint position and rotation calculation, background image retrieval, and final animation rendering. This comprehensive architecture ensures the creation of immersive and realistic animations suitable for various applications.

Backend Architecture

The backend architecture of DigiHuman consists ofseveral key stages:

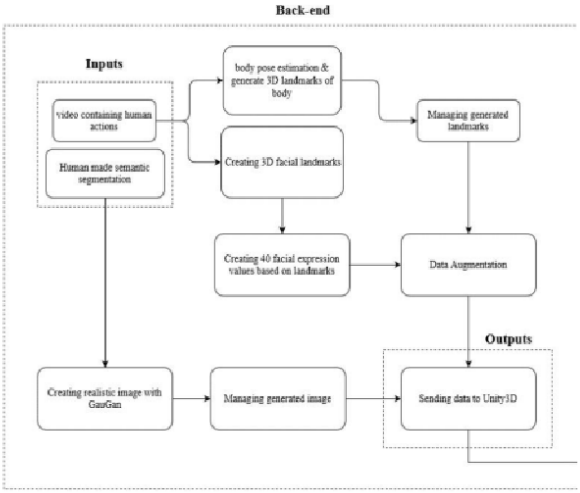

FIGURE 1: Backend Architecture3.2.1.Input Processing

DigiHuman begins by processing input data, which includes video recordings containing human actions and human-made semantic segmentation data. This step sets the foundation for subsequent processing stages.

Body Pose Estimation and 3D Landmarks Generation

Utilizing advanced computer vision techniques, DigiHuman estimates body poses from the input video and generates 3D landmarks corresponding to key body joints. This information is crucial for accurately animating humanmovements.

Facial Landmarks Creation

DigiHuman employs facial landmark detection algorithms to create 3D landmarks representing facial features and expressions. These landmarks serve as essential data points for animating facial expressions.

Facial Expression Values Generation

Based on the generated facial landmarks, DigiHuman computes 40 facial expression values, capturing nuanced expressions for realistic animation. These values provide detailed information about facial movements.

Data Augmentation

To enhance the diversity of training data and improve model robustness, DigiHuman applies data augmentation techniques to the generated landmarks and expression values. This ensures that the system can handle various facial expressions and movements effectively.

Realistic Image Synthesis with GauGan

Leveraging GauGan, a generative adversarial network (GAN), DigiHuman synthesizes realistic background images corresponding to the input actions and expressions.These background images add context and depth to the animations.

Image Management

DigiHuman manages the generated images to ensure proper alignment and compatibility with subsequent processing stages. This step involves organizing and preparing the images for further analysis and rendering.

Output Transmission to Unity 3D

Finally, DigiHuman transmits the processed data,
including body pose estimates, facial landmarks, and synthesized images, to Unity 3D for further animation rendering and integration. This integration enables seamless animation creation within the Unity environment, enhancing the overall user experience.

Digi Human's backend architecture, as depicted in Figure 1, encompasses a series of pivotal stages aimedat processing input data and synthesizing lifelike animations. Beginning with input processing, DigiHuman ingests video recordings containing human actions and human-made semantic segmentation data. Subsequently, body pose estimation and 3D landmarks generation techniques are employed to derive precise representations of body movements. Following this, facial landmark creation and facial expression values generation steps capture intricate facial features and expressions. These processes are augmented by data augmentation techniques to enhance the diversity of training data. Realistic background images are synthesized usingGauGan, contributing to the immersive quality of the animations. Finally, the processed data is transmitted to Unity 3D for further animation rendering and integration

Frontend Architecture

The frontend architecture of DigiHuman encompassesthe following key stages:

Input Reception

DigiHuman receives processed data from the backend, including body and facial 3D landmarks, along with facial expression values. This data forms the basis for the subsequent processing steps.

Signal Filtering

Applying signal filters such as Kalman filter and low pass filter, DigiHuman smoothens the received data to reduce noise and improve animation quality. This ensures that the animations generated are fluid and realistic.

Joint Position and Rotation Calculation

Utilizing the received 3D landmarks, DigiHuman calculates the position and rotation of each joint in the character rigging system to ensure accurate animation mapping. This step is crucial for animating characters with realistic movements.

Background Image Retrieval

DigiHuman downloads background images synthesized by GauGan to serve as the backdrop for the final animations. These background images complement the character animations and enhance the overall visual appeal.

Rendering Final Animation

Finally, DigiHuman renders the complete animation sequence on the 3D character, integrating the processed data with the background image to produce immersive and realistic animations. This step marks the culminationof the animation generation process, delivering high-quality animations suitable for a variety of applications.

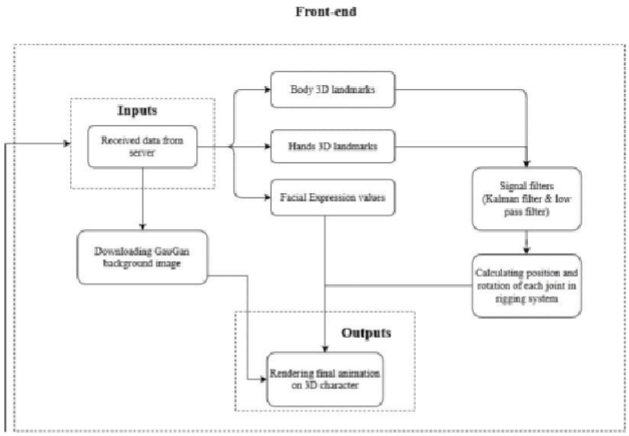

FIGURE 2: Frontend Architecture

On the frontend, as illustrated in Figure 2, DigiHuman receives processed data from the backend and undergoes additional processing stages before rendering final animations. Beginning with input reception, DigiHuman receives body and facial 3D landmarks, along with facial expression values, forming the basis for subsequent processing. Signal filtering techniques such as the Kalman filter and low pass filter are applied to refine the received data, ensuring smooth and realistic animations.

TRAINING AND PERFORMANCE TESTING

Data Preparation

In the realm of DigiHuman, training the base model necessitates tapping into real-world face datasets like CelebA and LFW. These datasets, renowned for their diversity encompassing various racial, age, gender, and facial expression attributes, fortify the model's robustness. The data preprocessing entails meticulous steps: utilizing the Dlib methodology for face detection, alignment, and cropping, followed by standardizing image sizes to a resolution of 224×224. The training strategy mirrors that of previous works, initializing with pre-trained ImageNet weights and employing the Adam optimizer with a batch size of 5, an initial learning rate set at 1e−4, spanning 500K iterations.

Training Adapter Model

The adapter model, distinguished by its topology awareness, demands tailored datasets to cater to specific virtual characters or adapt to new blend shape topologies seamlessly. DigiHuman addresses this requisite through an automated dataset generation mechanism, crafting character-dependent datasets comprising randomly generated blend shape coefficients and corresponding facial images. Each dataset boasts 10K data samples, partitioned into 8K for training, 1K for validation, and 1K for testing. Throughout training, the adapter model harnesses 3DMM expression parameters to prognosticate blend shape coefficients, under the vigilant supervision of ground-truth coefficients. The infusion of human prior knowledge serves to stabilize training, ensuring that the generated coefficients align with plausible facial expressions.

Usage

Transitioning into the operational phase, both the pre- trained base and adapter models operate as turnkey solutions. Armed with a reference facial expression image, the base model adeptly extracts 3DMM parameters, subsequently employed by the pre-trained adapter model to forecast blend shape coefficients. These estimated coefficients breathe life into the virtual character, with additional refinement from pose

parameters extracted by the base model. The introduction of human feedback signals further elevates animation fidelity and model efficacy, culminating in a finely tuned animation generation process.

This meticulously orchestrated training and testing regimen underscores Digi Human's commitment to delivering superior-quality animations, primed for diverse application scenarios.

FIGURE 3 : Hand Recognition

FIGURE 4: Full Body Character Generation

Figure 3 & 4 presents the user interface of the proposed

DigiHuman toolkit, encapsulating both online and offline modes of Human-in-the-Loop (HITL) interaction. The left segment showcases the intuitive interface design, facilitating user interaction and control over animation parameters. In the online mode, users can seamlessly adjust blend shape coefficients across several key frames, with the option to apply these preferences to the entire video sequence. Conversely, the right segment illustrates the offline mode, where adjusted blend shape coefficients arecollected as ground-truth data. This data issubsequently utilized to fine-tune the adapter model offline, thereby enhancing the performance and realism of the animation generation process. Overall, Figure 3 provides a visual representation of the versatile HITL functionality embedded within the DigiHuman toolkit, empowering users to actively participate in refining and optimizing animation outcomes.

DIGIHUMAN TOOLKIT DESIGN OVERVIEW

Comprehensive Solution

Introducing the DigiHuman Toolkit, a comprehensive solution crafted to streamline the intricate process of generating virtual human facial animations from 2D images or videos. This Unity-based toolkit integrates a plethora of advanced functionalities to cater to diverse animation requirements, ensuring a seamless user experience. Leveraging cutting-edge technologies and innovative methodologies, the DigiHuman Toolkit facilitates the creation of lifelike facial animations with unparalleled precision and efficiency.

Key Features

Facial Animation: The toolkit animates multiple blendShapes on 3D character models, supporting up to 40 blend shape animations concurrently.

Compatibility: Supports any 3D models equipped with Humanoid T-Pose rig, ensuring compatibility with a wide range of assets. Export Options: Enables users toexport animations in various formats, including videofiles, for convenient sharing and integration with otherplatforms.
Animation Data Management: Provides functionality for saving animation data, facilitating easy retrieval and re-rendering in future projects.

Noise Reduction: Incorporates advanced filteringtechniques, such as Low Pass Filtering, to detect and remove noises, ensuring smooth and artifact-free animations. Facial Detailing: Offers detailed facial animation capabilities, enabling users to capture subtle facial expressions and nuances.

Blend mesh Weight Generation: Employs regression models to generate Blend mesh weights by analyzing output data from media Pipe Face Mesh, facilitating accurate and realistic facial animations. Style Gan Integration: Integrates Style Gan techniques to replace the entire character face mesh, offering extensive customization options for character design.

Automatic Rigging: Provides automatic rigging for 3D models lacking humanoid rig, leveraging deep neural network models like Rig Net for efficient rigging processes. Complete Character Mesh Generation: Under development, aims to automatically generate complete character meshes using advanced models like PIFuHD, enhancing character creation workflows.

Detailed Mouth Animation: Enables detailed animation ofthe 3D character's mouth, utilizing audio signals or natural language processing methods to achieve realistic lip syncing. Environment Generation: Includes functionality for generating complete environments in 3D, enhancing the overall immersion and realism of animated scenes.

User Interface

The DigiHuman Toolkit features an intuitive user interface, comprising four main areas:

1. **Input Video Area:** Located on the left side, this area displays the current video frame, providing visual reference for animation generation.

2. **Virtual Human Area**: Positioned centrally, showcases the facial expressions and head poses of the virtual humancharacter, displaying the generated facial animation in real-time.

3. **Frame Diagram Area:** Located at the bottom within a dark gray box, visualizes animation frames with a scatterplot, facilitating easy navigation and frame selection.

4. User Interaction Area

Positioned on the right side, offers a set of buttons and fields for animation generation, visualization, playback, adjustment, and result exportation, ensuring user-friendly operation and control over the animationprocess.

Initialize Button: Executes the initialization procedureof the toolkit, invoking both the base model and the adapter model to predict blend shape coefficients frominput images or videos.

- ←→ Button: Implements the function of scrolling back and forth between animation (and input video) frames.

- <<>> Button: Plays all video frames and animation frames continuously in either forward or backward direction.

- Play Anim Button: Displays the generated facial animation

- Target Input Field: Allows users to input the index corresponding to the target blend shape they wish to modify in the current frame.

- Value Input Field: Enables users to input the specific value of the blend shape coefficient they intend to modify, falling within the specified range of0 to 1.

- Adjust Blend shape Button: Saves the adjusted value in the toolkit, registering it as the user's preference.

- Apply Preference Button: Computes user preference and extends it to all frames within the video after finetuning a representative selection of frames and blend shapes.

- Clear Preference Button: Erases user preferences that are not intended to be applied across all

video frames.

The DigiHuman Toolkit empowers users to create captivating facial animations with ease and efficiency,revolutionizing the virtual human animation
landscape.

Figure Description:

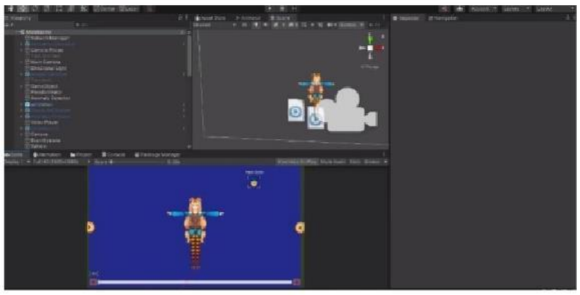

FIGURE 5: DigiHuman User Interface Design

FIGURE 6: Model Training Page

Figure 5 & 6 illustrates the user interface of the DigiHuman Toolkit, providing users with intuitivecontrols and visualizations to streamline the facial animation generation process.

TABLE 1: Model Training Details

Layers	Hidden-dim	MAE
Linear → ReLU → Linear	256	0.09
Linear → ReLU → Linear	100	0.10
Linear → ReLU → Linear	384	0.09
Linear → ReLU → Linear	256	0.08

Table 1 in above stated shows details of the experiments with different settings of the adapter model, specifically focusing on configurations involving the Rectified Linear Unit (ReLU). These experiments demonstrate the Mean Absolute Error(MAE) for various configurations of the adapter modelemploying ReLU activation functions.

DIGIHUMAN MODEL EVALUATION

Model Evaluation Overview

The evaluation of the DigiHuman model encompasses both quantitative assessments of modeling accuracy and qualitative analyses of facial animation quality. This evaluation aims to gauge the model's performance across various scenarios involving different texture appearances and blend shape topologies of virtual characters.

QuantitativeEvaluation of Modeling Accuracy

To quantitatively evaluate the modeling accuracy, the DigiHuman model's blend shape estimation accuracy was scrutinized using Mean Absolute Error (MAE) metrics against ground truth blend shape coefficients. Different settings of the adapter model implementationwere tested, focusing on variations in hidden dimensions and network structures.

Table 1 displays the MAE results for different adapter model configurations. Experimentation with various architectural choices, including hidden dimension adjustments and activation function changes (e.g., ReLU to LeakyReLU), revealed insights into their impact on estimation accuracy. Notably, truncating output with a Clamp operator led to reduced estimation errors, with the combination of LeakyReLU and Clamp yielding the lowest error rates.

Qualitative Results of Facial Animation

Qualitative assessments of facial animation quality were conducted from multiple perspectives:

1. Variety of Inputs: Different types of inputs wereused to generate facial expressions for specific virtual characters.

2. Texture Appearance Variation: Virtual characters with the same blend shape topology but varying texture appearances were tested.
3. Blend shape Topology Variation: Virtual characters with identical texture appearances but different blend shape topologies were evaluated.

Figures 4 and 5 illustrate the qualitative results of facial animation. The DigiHuman model effectively replicates reference facial expressions across diverse inputs and demonstrates robustness in handling variations in textureappearances and blend shape topologies.

Discussion on Blend shape Topology Effect

The expressiveness of the generated results is influenced by the number of blend shapes and their arrangement. Forinstance, characters with fewer blend shapes may exhibitlimitations in depicting specific facial movements accurately. Conversely, models with more blend shapes offer greater potential expressiveness but pose challengesin optimization and adaptation.

TABLE 2: Comparison of Mean Absolute Error (MAE)for Various Blend shape Topologies

MLP Architecture	Number of Blend shapes	MAE
(64, 256) → (256, 25)	25	0.0805
(64, 256) → (256, 66)	66	0.1063
(64, 256) → (256, 113)	113	0.1076

This table presents the Mean Absolute Error (MAE) results for different Multilayer Perceptron (MLP) architectures adapted to virtual characters with varying blend shape topologies. The number of blend shapes in each model configuration influences the estimation accuracy, with higher blend shape counts potentially leading to increased error rates.

Table 2 summarizes the quantitative results of models adapted to different virtual characters, highlighting the trade-offs between blend shape complexity and animation quality. Despite the challenges posed by higher blend shape counts, the DigiHuman model's efficient adaptation process, facilitated by GPU hardware, significantly reduces adaptation time and resource requirements compared to manual manipulation by human animators.

Overall, the DigiHuman model demonstrates promising performance in both quantitative accuracy and qualitative animation quality evaluations, showcasing its potential for advancing virtual character techniques.

RESULTS AND DISCUSSION LIMITATIONS

Our work, while innovative and promising, is not without its challenges. Firstly, there remains substantial room for improvement in the quality of generated animations, particularly when compared to commercial software. Detailed facial expressions posea challenge for the toolkit, indicating the need for further enhancement. It's important to note that our approach is not intended to fully replace human animators or compete directly with commercialsoftware. Rather, it aims to assist animators in working with personalized virtual human characters. In real- world production scenarios, human supervision is stillrequired to ensure the animation's quality. Moving forward, optimizing our algorithms and toolkit will bea priority to enhance the accuracy and speed of animation inference.

Additionally, our method's evaluation primarily focuses on common humanoid characters and assumes clear input data. While this covers a broad range of applications, future work should explore diverse character types, including variations in gender, skin tones, facial artifacts (such as jewelry, tattoos, scars, blemishes), and non-human characters. This broader investigation will provide a more comprehensive understanding of the toolkit's capabilities across different scenarios.

Furthermore, our evaluation lacks direct comparisons with other methods. This limitation arises from the fact that many related works are either closed-source software or tailored for specific virtual characters, making direct comparisons challenging. Unlike these solutions, our toolkit offers flexibility and customization, making it difficult to conduct an apples-to-apples comparison. Nevertheless, we hope that our work serves as an initial benchmark and inspires future research in this domain.

Lastly, the user study presented in our paper is a preliminary exploration of the toolkit's properties. While it offers valuable insights, it represents only a pilot study. A more comprehensive user study with a larger sample size and diverse user groups is essential to provide deeper insights into the toolkit's usability, effectiveness, and areas for improvement. Enhancing the toolkit based on the feedback from such studies is a key objective for future iterations.

In conclusion, while the DigiHuman Toolkit holds promise for revolutionizing virtual human animation, addressing these limitations will be crucial for its continued development and adoption in real-worldproduction environments.

CONCLUSION

The DigiHuman Toolkit stands as a groundbreaking solution in the realm of virtual human animation, offering a holistic approach to generating lifelike facial animations from 2D images or videos. Leveraging state-of-the-art technologies and innovative methodologies, this toolkit empowers users to craft captivating facial expressions with precision and efficiency.

Throughout its development and evaluation, the DigiHuman Toolkit has demonstrated several pivotal strengths. Firstly, its advanced functionality encompasses a broad spectrum of capabilities, including support for animating multiple blend Shapes, compatibility with diverse 3D models, flexible exporting options, noise reduction techniques, and intricate facial animation capabilities. These features collectively enable users to create animations with unparalleled detail and realism.

The user-friendly interface of the toolkit further enhances its usability, providing intuitive controls and seamless navigation through the animation generation process. From the display of input videos to the adjustment of animation parameters, every aspect of the interface is meticulously designed to optimize user experience and workflow efficiency. Moreover, the toolkit adopts a human-in-the-loop approach, allowing for user feedback and adjustments to refine animation quality. By incorporating user preferences and adjustments, the toolkit ensures greater accuracy and alignment with user expectations, thereby enhancing the overall animation output.

Despite these strengths, the DigiHuman Toolkit is not devoid of limitations. Challenges such as the quality of generated animations, the scope of evaluation, comparison with other methods, and the need for more comprehensive user studies underscore areas for future improvement and research. Looking ahead, our commitment lies in addressing these limitations and further enhancing the DigiHuman Toolkit. Through algorithm optimization, expansion of evaluation scope, exploration of diverse character types, and conducting more extensive user studies, we aim to refine the toolkit and ensure its effectiveness and usability in real-world production environments.

In summary, the DigiHuman Toolkit represents a significant advancement in virtual human animation technology. With its advanced features, user-friendly interface, and dedication to continuous improvement, the toolkit is poised to revolutionize the animation creation process and set new standards for virtual human animation.

REFEERENCES

1. Live Link Face on the App Store. https://apps.apple.com/us/app/live-link-face/id1495370836. Accessed: 2023-09-27.
2. Apple. Tracking and Visualizing Faces. 2022.
3. N. I. Badler, M. S. Palmer, and R. Bindiganavale. Animation Control for Real-Time Virtual Humans. Communications of the ACM, 42(8):64–73, 1999.
4. Z. Bai, N. Yao, L. Liu, H. Chen, and H. Wang. A Simple Approach to Animating Virtual Characters by Facial Expressions Reenactment. In 2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), pp. 585–586. IEEE, 2023.

5.Z. Bai, N. Yao, N. Mishra, H. Chen, H. Wang, andN. Magnenat Thalmann. Enhancing Emotional Experience by Building Emotional Virtual Characters in VR Volleyball Games. Computer Animation and Virtual Worlds, 32(3-4):e2008, 2021.

6.Z. Bai, N. Yao, N. Mishra, H. Chen, H. Wang, and N.

7. M. Thalmann. Play with Emotional Characters: Improving User Emotional Experience by a Data-DrivenApproach in VR Volleyball Games. In 2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), pp. 458–459. IEEE, 2021.

8. V. Blanz and T. Vetter. A Morphable Model for the Synthesis of 3D Faces. In Proceedings of the 26th AnnualConference on Computer Graphics and InteractiveTechniques, pp. 187–194, 1999.

9. J. Brooke et al. SUS-A Quick and Dirty Usability Scale. Usability Evaluation in Industry, 189(194):4–7, 1996.

10. D. Burden and M. Savin-Baden. Virtual Humans: Today and Tomorrow. CRC Press, 2019.

11. C. Cao, Y. Weng, S. Zhou, Y. Tong, and K. Zhou. Facewarehouse: A 3D Facial Expression Database for Visual Computing. IEEE Transactions on Visualization and Computer Graphics, 20(3):413–425, 2013.

12. B. Chaudhuri, N. Vesdapunt, L. Shapiro, and B. Wang. Personalized Face Modeling for Improved Face Reconstruction and Motion Retargeting. In ComputerVision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16, pp. 142–160. Springer, 2020.

13. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L.Fei-Fei. Imagenet: A Large-Scale Hierarchical Image Database. In 2009 IEEE Conference on Computer Visionand Pattern Recognition, pp. 248–255. IEEE, 2009.

14. Y. Deng, J. Yang, S. Xu, D. Chen, Y. Jia, and X. Tong. Accurate 3D Face Reconstruction with Weakly- Supervised Learning: From Single Image to Image Set. InProceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 0–0, 2019.

15. P. Dou, S. K. Shah, and I. A. Kakadiaris. End-to-End3D Face Reconstruction with Deep Neural Networks. In Proceedings of the IEEE Conference on Computer Visionand Pattern Recognition, pp. 5908–5917, 2017.

16. P. Edwards, C. Landreth, E. Fiume, and K. Singh. Jali: An Animator-Centric Viseme Model for ExpressiveLip Synchronization. ACM Transactions on Graphics(TOG), 35(4):1–11, 2016.

17. Faceware. Intro to Faceware Studio. 2022.

18. Y. Feng, F. Wu, X. Shao, Y. Wang, and X. ZhouJoint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network. In Proceedings of the European Conference on ComputerVision (ECCV), pp. 534–551, 2018.

19. G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. In Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition, 2008.

20. IEEE. A 3D Face Model for Pose and IlluminationInvariant Face Recognition. Genova, Italy, 2009.

21. P. Joshi, W. C. Tien, M. Desbrun, and F. Pighin. Learning Controls for Blend Shape Based Realistic Facial Animation. In ACM Siggraph 2006 Courses, pp. 17–es. 2006.

22. D. E. King. Dlib-ml: A Machine Learning Toolkit. The Journal of Machine Learning Research, 10:1755–1758, 2009.

23. J. P. Lewis, K. Anjyo, T. Rhee, M. Zhang, F. H. Pighin, and Z. Deng. Practice and Theory of Blend shape Facial Models. Eurographics (State of the Art Reports), 1(8):2, 2014.

24. J. P. Lewis and K.-i. Anjyo. Direct ManipulationBlend shapes. IEEE Computer Graphics andApplications, 30(4):42–50, 2010.

25. T. Li, T. Bolkart, M. J. Black, H. Li, and J. Romero. Learning a Model of Facial Shape and Expression from 4D Scans. ACM Transactions on Graphics, (Proc. SIGGRAPH Asia), 36(6):194:1– 194:17, 2017.

26. Z. Liu, P. Luo, X. Wang, and X. Tang. Large- Scale CelebFaces Attributes (CelebA) Dataset. Retrieved August, 15(2018):11, 2018.

27. O. M. Machidon, M. Duguleana, and M. Carrozzino. Virtual Humans in Cultural Heritage ICT Applications: A Review. Journal of Cultural Heritage,33:249–260, 2018.

28. P. Nogueira. Motion Capture Fundamentals. In Doctoral Symposium in Informatics Engineering, vol.303, 2011.

29. Y. Pan, R. Zhang, S. Cheng, S. Tan, Y. Ding, K.Mitchell, and X. Yang. Emotional Voice Puppetry. IEEETransactions on Visualization and Computer Graphics, 29(5):2527–2535, 2023.

30. P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and

31. T. Vetter. A 3D Face Model for Pose and Illumination Invariant Face Recognition. In 2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance, pp. 296–301. IEEE, 2009.

32. X. Peng, J. Huang, A. Denisova, H. Chen, F. Tian, and H. Wang. A Palette of Deepened Emotions: Exploring Emotional Challenge in Virtual Reality Games.In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1–13, 2020.

33. X. Peng, J. Huang, L. Li, C. Gao, H. Chen, F. Tian, and H. Wang. Beyond Horror and Fear: Exploring PlayerExperience Invoked by Emotional Challenge in VR Games. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, pp. 1–6, 2019.

34. X. Peng, C. Meng, X. Xie, J. Huang, H. Chen, andH. Wang. Detecting Challenge from Physiological Signals: A Primary Study with a Typical Game Scenario.In CHI Conference on Human Factors in ComputingSystems Extended Abstracts, pp. 1–7, 2022.

35. X. Peng, X. Xie, J. Huang, C. Jiang, H. Wang, A. Denisova, H. Chen, F. Tian, and H. Wang. ChallengeDetect: Investigating the Potential of DetectingIn-Game Challenge Experience from Physiological Measures. In Proceedings of the 2023 CHI Conference onHuman Factors in Computing Systems, pp. 1–29, 2023.

36. F. Schroff, D. Kalenichenko, and J. Philbin. Face Net: A Unified Embedding for Face Recognition and Clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 815–823, 2015.

37. S. Sharma, S. Verma, M. Kumar, and L. Sharma. Useof Motion Capture in 3D Animation: Motion Capture Systems, Challenges, and Recent Trends. In 2019 International Conference on Machine Learning, Big Data,Cloud and Parallel Computing (COMITCon), pp. 289–
38. 294. IEEE, 2019.

39. T. Shi, Y. Yuan, C. Fan, Z. Zou, Z. Shi, and Y. Liu. Face-to-Parameter Translation for Game Character Auto-Creation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 161–170, 2019.

40. T. Shi, Z. Zuo, Y. Yuan, and C. Fan. Fast and RobustFace-to-Parameter Translation for Game Character Auto-Creation. In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 1733–1740, 2020

41. M. Volonte, E. Ofek, K. Jakubzak, S. Bruner, and M. Gonzalez-Franco. Headbox: A Facial blend shape Animation Toolkit for the Microsoft Rocketbox Library. In 2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), pp. 39–42. IEEE, 2022.

42. L. Wolf, Y. Taigman, and A. Polyak. Unsupervised Creation of Parameterized Avatars. In Proceedings of the IEEE International Conference on Computer Vision, pp. 1530–1538, 2017.

1508

ORIGINALITY REPORT

12%	11%	2%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	arxiv.org Internet Source	10%
2	"Image and Video Technology", Springer Science and Business Media LLC, 2024 Publication	<1%
3	Submitted to National Institute of Technology, Silchar Student Paper	<1%
4	www.yongliangyang.net Internet Source	<1%
5	Aria Seo, Hyeonjin Jeon, Yunsik Son. "Robust Prediction Method for Pedestrian Trajectories in Occluded Video Scenarios", Research Square Platform LLC, 2024 Publication	<1%
6	ojs.aaai.org Internet Source	<1%
7	"ECAI 2020", IOS Press, 2020 Publication	<1%

8	Guoling Tang, Yaning Han, Quanying Liu, Pengfei Wei. "Anti-drift pose tracker (ADPT): A transformer-based network for robust animal pose estimation cross-species", Cold Spring Harbor Laboratory, 2024 Publication	<1%
9	dokumen.pub Internet Source	<1%
10	openaccess.thecvf.com Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		