Lightweight Speech-to-Image Framework for Rapid Facial Reconstruction in Forensic Settings

Lamb, Joshua

***Abstract*—Crime suspect sketches are key for suspect identification, however traditional forensic artists are rare and often unavailable in early investigations, and current software alternatives are not quick or easily accessible. By leveraging recent advancements in lightweight speech-to-text and text-to-image models, we propose a lightweight, multimodal speech-to-text-to-image framework using tuned Wav2Vec2 and Stable Diffusion v1.5 models for fast, automated generation of suspect faces from speech. Experimental results show an excellent speech-to-text Word Error Rate (WER) of 5% on testing data, and promising text-to-image semantic detail and 0.13 Mean Squared Error (MSE) on testing data, even with a limited training dataset. Additionally, this solution has tremendous impact potential, offering an offline, rapid, on-site solution that minimizes witness memory decay and enables downstream analysis, i.e. using generates images for database searching or scanning surveillance cameras within a specified radius.**

Keywords*—Artificial Intelligence, Connectionist Temporal Classification, Contrastive Language-Image Pre-Training, Diffusion, Generative Artificial Intelligence, Speech Recognition, Stable Diffusion, Transformer, Wav2Vec2

I. INTRODUCTION

A. Context and Problem Statement

Suspect sketches are a common and tested method to help identify criminals based on witness accounts. Traditional suspect sketches rely on forensic artists, however in today's digital age riddled with surveillance cameras, forensic artists are rare and often unavailable early in investigations. Software alternatives like FACES and EFIT-V exist, however they require manual input and trained operators, thus they aren't easily accessible at a crime scene [1, 2]. Since witnesses usually only provide verbal descriptions and they can quickly forget face details, rapid reconstruction is essential for suspect identification.

Recent advancements in lightweight speech-to-text and text-to-image frameworks offer the potential for fast, offline, on-site, automated generation of suspect faces from speech. Additionally, this approach enables downstream applications like using generated faces to search databases or scan surveillance cameras within a nearby radius, further assisting in rapid suspect identification. This report details the use of 2 of these architectures, pretrained Wav2Vec2 for speech-to-text and pretrained Stable Diffusion v1.5 for text-to-image, to produce a single lightweight, fast, offline speech-to-image framework for facial reconstruction in forensic settings.

The pretrained Wav2Vec2 and Stable Diffusion v1.5 architectures were tuned on subsets of LibriSpeech ASR and FaceCaption-15M respectively. LibriSpeech is a dataset of 1,000+ hours of English audiobook recordings. It was chosen due to its large size and its separation into various subsets of clean audio and noisy, more difficult audio. FaceCaption-15M is a dataset of 15 million facial images with accompanying text captions describing them. It was also chosen due to its large size, but also its clear focus on facial images specifically.

All code directly and indirectly referenced within this report may be found in the accompanying codebase. Training was done using Kaggle resources for Wav2Vec2 and Google Colab resources for Stable Diffusion v1.5, particularly using a single P100 GPU on Kaggle and a single A100 GPU on Google Colab.

B. Wav2Vec2 Overview

Wav2Vec2 is a self-supervised speech processing model developed by MetaAI, aimed at learning latent representations of speech audio. The Wav2Vec2 architecture consists of 2 main components: a Convolutional Neural Network (CNN) feature encoder followed by a Transformer encoder, both of which work to extract latent features from the input audio [3]. Because the model is self-supervised, it was pretrained using raw, unlabeled audio data using a contrastive loss, where the model learns to predict the correct audio latent representation from a list of incorrect representations [3].

Wav2Vec2ForCTC is the specific variation of Wav2Vec2 used in this report. It adds a Connectionist Temporal Classification (CTC) decoder to Wav2Vec2. Unlike a traditional Transformer decoder, the CTC decoder is non-autoregressive and alignment-free, allowing for varying input/output alignment [4]. This is critical in speech-to-text, where there is no direct correlation between a word's length and the length of the corresponding audio [4]. The CTC decoder performs classification over the tokenizer vocabulary, which consists of the alphabet instead of whole words [3], outputting a long sequence of letters before merging down to the final words. The use of a blank token allows the CTC decoder to handle repetitions and timing mismatches. For instance, if the audio is "boat", CTC may output "bb-oo-aa-aa--tt--" before merging down to "boat". CTC learns via CTCLoss, summing over all possible alignments of the un-collapsed output [3].

MetaAI also provides Wav2Vec2Processor which is used in this report. Wav2Vec2Processor is a preprocessing wrapper combining feature extraction and tokenization to convert raw audio and transcript data into a form usable by the Wav2Vec2 and Wav2Vec2ForCTC models, eliminating the need for manual preprocessing.

Between Wav2Vec2Processor and Wav2Vec2ForCTC, MetaAI provides a nearly end-to-end system for speech recognition.

C. *Stable Diffusion v1.5 Overview*

Stable Diffusion v1.5 is a lightweight text-to-image generative model developed by StabilityAI and collaborators based on the latent diffusion framework. Stable Diffusion consists of 3 main components: a Contrastive Language-Image Pretraining (CLIP)-based encoder for the text input, a denoising U-Net to produce the final image latent [5], and a Variational Autoencoder (VAE) to decode the final image latent [5]. The CLIP-based encoder leverages the power of CLIP, a multimodal model developed by OpenAI for encoding images and text into the same latent space. The denoising U-Net is a CNN that learns to progressively denoise initial random noise under the guidance of the encoded text [5]. The VAE in the case of Stable Diffusion v1.5 specifically, will decode the U-Net image latent into a final 512x512 RGB image.

Stable Diffusion is trained on a large image-caption dataset and uses denoising score matching as its learning objective [5]. During training, Gaussian noise is added to the ground truth image latent, and the U-Net learns to properly denoise the initial random noise to achieve this latent [5]. This process is reversed in inference, where the initial random noise is denoised according to the text input using the learned model parameters.

Unlike autoregressive models, diffusion models like Stable Diffusion v1.5 are iterative and non-sequential, generating the whole image at once through multiple denoising steps. Working in this latent space makes the process more memory- and compute-efficient compared to pixel-wise diffusion [5].

II. Methodology

A. *Overall Pipeline Structure*

The overall speech-to-image framework developed follows the pipeline seen below in Fig. 1.

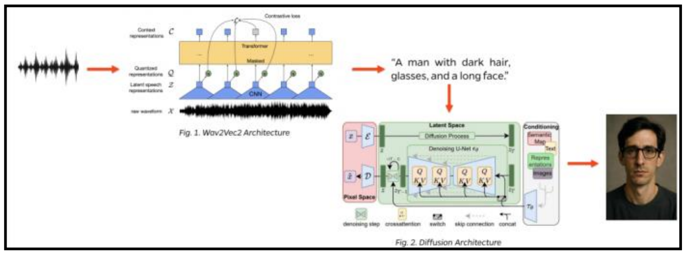

Contrastive loss
Context representations C
Transformer
Masked
Quantized representations Q
Latent speech representations Z
CNN
raw waveform X
Fig. 1. Wav2Vec2 Architecture
"A man with dark hair, glasses, and a long face."
Latent Space
Diffusion Process
Denoising U-Net ϵ_θ
Pixel Space
Conditioning
Semantic Map
Text
Representations
Images
denoising step
crossattention
switch
skip connection
concat
Fig. 2. Diffusion Architecture

Fig. 1. *Overall speech-to-image framework pipeline.*

The pipeline begins with raw audio data describing the suspect. This is fed into the Wav2Vec2 architecture, where it extracts latent features with the CNN and Transformer before using the CTC decoder to output final text. This text is fed into the Stable Diffusion v1.5 architecture, which tokenizes and embeds it using the CLIP encoder, iteratively denoises Gaussian noise under the guidance of the embedded text, and uses the VAE to produce a final, 512x512 RGB image.

B. *Data Preprocessing*

1) LibriSpeech Data: Before training either architecture, the data was loaded and preprocessed, starting with the LibriSpeech Data for Wav2Vec2. A custom "read_librispeech_dir" function was created to load in the data and return lists of the audio samples and corresponding transcripts. This function was used to read in the "dev-clean" and "test-clean" subsets, which were chosen because their clean audio does not introduce unnecessary complexity, and their small sizes facilitate faster training. They have 2,703 and 2,620 samples respectively, however due to computing and time limitations, only the first 1,000 samples of each subset were retained.

After loading in the data, audio-transcript pairs were outputted to confirm that no errors had occurred. Next, the training data and testing data were converted to PyTorch Dataset objects using a custom LibrispeechDataset class. This class leaves the data as lists instead of tensors, since tensors require equal dimensionality across all samples when not all audio clips and transcripts are the same length. Next, the training Dataset and testing Dataset were used to create PyTorch DataLoaders. A "collate_wav2vec2" function was defined and passed into the DataLoaders, which performs batch-wise normalization, preprocessing, and padding of the audio data using Wav2Vec2Processor, in addition to tokenizing and padding the transcripts using Wav2Vec2Processor.tokenizer. The batch size was set to 4 due to the mere 16GB of VRAM on Kaggle's P100 GPU, producing 250 train and test batches. To confirm successful batch-wise padding, the lengths of various training batch samples were printed, as seen in Fig. 2.

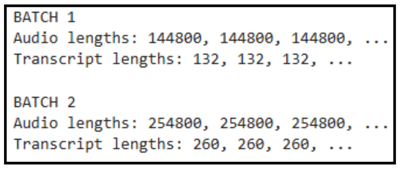
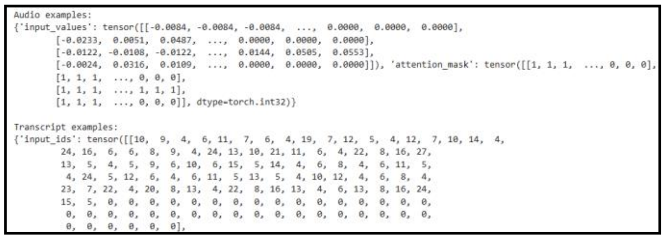

```
BATCH 1
Audio lengths: 144800, 144800, 144800, ...
Transcript lengths: 132, 132, 132, ...

BATCH 2
Audio lengths: 254800, 254800, 254800, ...
Transcript lengths: 260, 260, 260, ...
```

Fig. 2. *Data lengths of various LibriSpeech training batches.*

From Fig. 2, batch-wise padding was successful, with equal intra-batch lengths but differing inter-batch lengths.

When Wav2Vec2Processor.tokenizer generates padding, it also generates attention masks, and so now some training batch samples were printed along with their attention masks to inspect the data format, as seen in Fig. 3.



```
Audio examples:
{'input_values': tensor([[-0.0084, -0.0084, -0.0084,  ...,  0.0000,  0.0000,  0.0000],
        [-0.0233,  0.0051,  0.0487,  ...,  0.0000,  0.0000,  0.0000],
        [-0.0122, -0.0108, -0.0122,  ...,  0.0144,  0.0505,  0.0553],
        [-0.0024,  0.0316,  0.0109,  ...,  0.0000,  0.0000,  0.0000]]), 'attention_mask': tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 1, 1, 1],
        [1, 1, 1,  ..., 0, 0, 0]], dtype=torch.int32)}

Transcript examples:
{'input_ids': tensor([[10,  9,  4,  6, 11,  7,  6,  4, 19,  7, 12,  5,  4, 12,  7, 10, 14,  4,
         24, 16,  6,  6,  8,  9,  4, 24, 13, 10, 21, 11,  6,  4, 22,  8, 16, 27,
         13,  5,  4,  5,  9,  6, 10,  6, 15,  5, 14,  4,  6,  8,  4,  6, 11,  5,
          4, 24,  5, 12,  6,  4,  6, 11,  5, 13,  5,  4, 10, 12,  4,  6,  8,  4,
         23,  7, 22,  4, 20,  8, 13,  4, 22,  8, 16, 13,  4,  6, 13,  8, 16, 24,
         15,  5,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  0,  0,  0,  0],
```

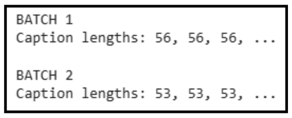
Fig. 3. Printed output of various LibriSpeech training samples.

As seen in Fig. 3, the data is now organized and ready for training as a dictionary consisting of the raw data itself and the corresponding attention masks indicating which positions are padding.

2) FaceCaption-15M Data: The FaceCaption-15M data was preprocessed for Stable Diffusion v1.5. First, the raw dataset was read in, consisting of 15 million image captions, corresponding URLs to download the images, and corresponding bounding box coordinates to extract the faces from the images. Due to computing and time limitations, only the first 1000 of the 15 million samples were retained, 800 of which were split into a training set and 200 a testing set. Basic preprocessing steps were done to pull the captions out of inner arrays they were stored in, convert the bounding box coordinates from strings to arrays of integers, and remove irregular symbols from the captions.

Next, the images themselves were loaded, preprocessed, and saved to local memory, preventing network bottlenecks down the line. Each image was cropped to the face using the bounding box coordinates, resized to 512x512 (expected by Stable Diffusion v1.5), converted to a PyTorch tensor, and normalized to the [-1, 1] range (expected by Stable Diffusion v1.5). Any image that could not be loaded due to an invalid URL was dropped from the data along with its corresponding caption and bounding box coordinates. The result was 612/800 images left in the training set and 157/200 images left in the testing set.

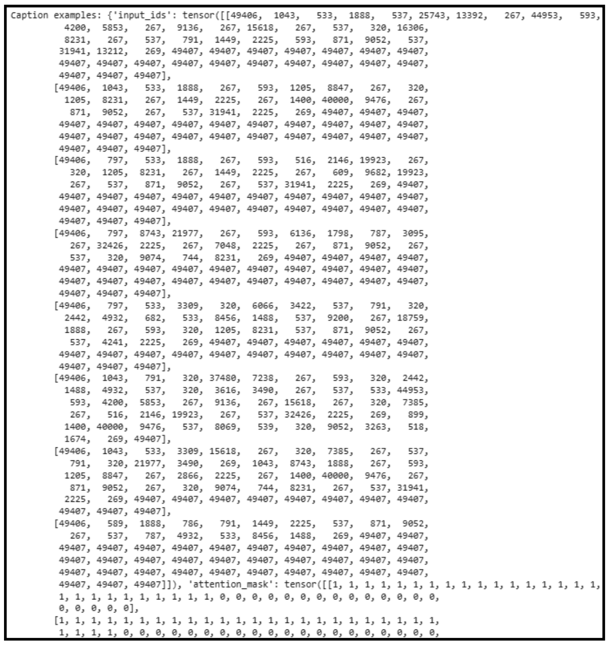
The training and testing data were then converted to PyTorch Dataset objects using a custom FaceCaption15MDataset class. This class loads the images in as they are accessed, adding computational overhead but avoiding issues with holding ~700 images on RAM at the same time. The Datasets were then converted to DataLoaders, with a "collate_fn" function passed in to perform dynamic, batch-wise tokenization and padding using the CLIP's ClipTokenizer. The batch size was set to 8 to maximize usage of the A100's 40GB of VRAM, producing 39 training batches and 10 testing batches. To confirm successful batch-wise padding, the lengths of the various tokenized training batch captions were printed, as seen in Fig. 4.



```
BATCH 1
Caption lengths: 56, 56, 56, ...

BATCH 2
Caption lengths: 53, 53, 53, ...
```

Fig. 4. Caption lengths of various FaceCaption-15M training batches.

With Fig. 4. Clearly showing the varying inter-batch caption lengths, the captions themselves were printed along with their corresponding padding attention masks to inspect the data format, as seen in Fig. 5.



```
Caption examples: {'input_ids': tensor([[49406,  1043,   533,  1888,   537, 25743, 13392,   267, 44953,   593,
          4200,  5853,   267,  9136,   267, 15618,   267,   537,   320, 16306,
          8231,   267,   537,   791,  1449,  2225,   593,   871,  9052,   537,
         31941, 13212,   269, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407],
        [49406,  1043,   533,  1888,   267,   593,  1205,  8847,   267,   320,
          1205,  8231,   267,  1449,  2225,   267,  1400, 40000,  9476,   267,
           871,  9052,   267,   537, 31941,  2225,   269, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407],
        [49406,   797,   533,  1888,   267,   593,   516,  2146, 19923,   267,
           320,  1205,  8231,   267,  1449,  2225,   267,   609,  9682, 19923,
           267,   537,   871,  9052,   267,   537, 31941,  2225,   269, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407],
        [49406,   797,  8743, 21977,   267,   593,  6136,  1798,   787,  3095,
           267, 32426,  2225,   267,  7048,  2225,   267,   871,  9052,   267,
           537,   320,  9074,   744,  8231,   269, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407],
        [49406,   797,   533,  3309,   320,  6066,  3422,   537,   791,   320,
          2442,  4932,   682,   533,  8456,  1488,   537,  9200,   267, 18759,
          1888,   267,   593,   320,  1205,  8231,   537,   871,  9052,   267,
           537,  4241,  2225,   269, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407],
        [49406,  1043,   791,   320, 37480,  7238,   267,   593,   320,  2442,
          1488,  4932,   537,   320,  3616,  3490,   267,   537,   533, 44953,
           593,  4200,  5853,   267,  9136,   267, 15618,   267,   320,  7385,
           267,   516,  2146, 19923,   267,   537, 32426,  2225,   269,   899,
          1400, 40000,  9476,   537,  8069,   539,   320,  9052,  3263,   518,
          1674,   269, 49407],
        [49406,  1043,   533,  3309, 15618,   267,   320,  7385,   267,   537,
           791,   320, 21977,  3490,   269,  1043,  8743,  1888,   267,   593,
          1205,  8847,   267,  2866,  2225,   267,  1400, 40000,  9476,   267,
           871,  9052,   267,   320,  9074,   744,  8231,   267,   537, 31941,
          2225,   269, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407],
        [49406,   589,  1888,   786,   791,  1449,  2225,   537,   871,  9052,
           267,   537,   787,  4932,   533,  8456,  1488,   269, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407, 49407,
         49407, 49407, 49407]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Fig. 5. Printed output of various FaceCaption-15M training sample captions.

As seen in Fig. 5, the captions are now a dictionary consisting of the raw captions themselves and the corresponding attention masks indicating which positions are padding.

C. Tuning

1) *Wav2Vec2:* To tune Wav2Vec2, a typical PyTorch "train" function was defined. This function moves the model to the GPU, iterates over all epochs and over all train batches on each epoch. For each batch, the raw data and attention masks are separated, forward propagation computes the model output and loss, and backward propagation updates the model weights. Additionally, the model output and true transcript are decoded back to raw, readable transcripts using Wav2Vec2Processor.batch_decode, and they are used to compute the Word Error Rate (WER). The average loss and WER for the epoch are printed. Once all epochs of training are complete, the final model is moved to the CPU and returned by the function, along with lists of the losses and WERs per epoch for plotting learning curves.

The specific hyperparameters used for training Wav2Vec2 were CTCLoss, the AdamW optimizer with a learning rate of 1e-5, and 10 epochs of training.

2) *Stable Diffusion v1.5:* To tune Stable Diffusion v1.5, a "train" function was defined. This function is like Wav2Vec2's "train" function, except it utilizes a scaler object and autocast for mixed precision memory savings, and it optionally utilizes attention slicing and gradient checkpointing for additional memory savings. The forward pass also differs from Wav2Vec2 by using all components of the diffusion model, starting by encoding the ground truth images using the VAE, generating random noise and timesteps, adding the random noise to the images, encoding the captions with CLIP, and predicting the noise using U-Net. WER is also not calculated here since it is

not applicable, so the function only returns the final model and a list of the losses per epoch for plotting a learning curve.

The specific hyperparameters used for training Stable Diffusion v1.5 were MSE loss, the AdamW optimizer with a learning rate of 1e-5 and weight decay of 0.01, and 15 epochs of training.

D. *Testing*

1) *Wav2Vec2:* To test Wav2Vec2, a “test” function was defined. This function’s structure is identical to Wav2Vec2’s “train” function except gradients are disabled, the model is set to evaluation mode, no backpropagation is performed, and there is no iteration over epochs. The function prints an overall loss and WER on all batches of the test dataset, as well as 5 examples of predicted and true transcripts.

2) *Stable Diffusion v1.5:* To test Stable Diffusion v1.5, another “test” function was defined, identical to the corresponding “train” function, except gradients are disabled, the model is set to evaluation mode, no backpropagation is performed, and there is no iteration over epochs. The function prints an overall loss on all batches of the test dataset.

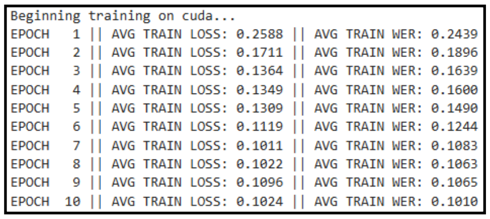
Another function “eval_truth_pretrained_tuned” was also defined to test Stable Diffusion v1.5. This function displays 5 examples of: a ground truth image and its ground truth caption, the image generated by the baseline pretrained Stable Diffusion v1.5, and the image generated by the final tuned Stable Diffusion v1.5.

3) Gradio: A Gradio interface was produced as the primary environment for users to interact with the overall framework. In this browser-based Gradio interface, a user can easily record an audio snippet and press a button to perform speech-to-text-to-image using the final tuned architectures. The intermediately generated speech and the final generated image are both displayed to the user.

III. RESULTS

A. *Wav2Vec2*

The basic training results for Wav2Vec2, including train loss and WER per epoch, can be found below in Fig. 6.



```
Beginning training on cuda...
EPOCH   1 || AVG TRAIN LOSS: 0.2588 || AVG TRAIN WER: 0.2439
EPOCH   2 || AVG TRAIN LOSS: 0.1711 || AVG TRAIN WER: 0.1896
EPOCH   3 || AVG TRAIN LOSS: 0.1364 || AVG TRAIN WER: 0.1639
EPOCH   4 || AVG TRAIN LOSS: 0.1349 || AVG TRAIN WER: 0.1600
EPOCH   5 || AVG TRAIN LOSS: 0.1309 || AVG TRAIN WER: 0.1490
EPOCH   6 || AVG TRAIN LOSS: 0.1119 || AVG TRAIN WER: 0.1244
EPOCH   7 || AVG TRAIN LOSS: 0.1011 || AVG TRAIN WER: 0.1083
EPOCH   8 || AVG TRAIN LOSS: 0.1022 || AVG TRAIN WER: 0.1063
EPOCH   9 || AVG TRAIN LOSS: 0.1096 || AVG TRAIN WER: 0.1065
EPOCH  10 || AVG TRAIN LOSS: 0.1024 || AVG TRAIN WER: 0.1010
```

Fig. 6. Wav2Vec2 training results.

From Fig. 6, the tuned Wav2Vec2 shows excellent performance after just a few epochs, reaching a training loss and train WER of ~0.10 by epoch 10. To speed up production time, early stopping was not implemented and no validation set was used, however these are important additions to consider in future work. These ideas are explained in Section IV below. The learning curves corresponding to the training loss and train WER values can be seen in Fig. 7.

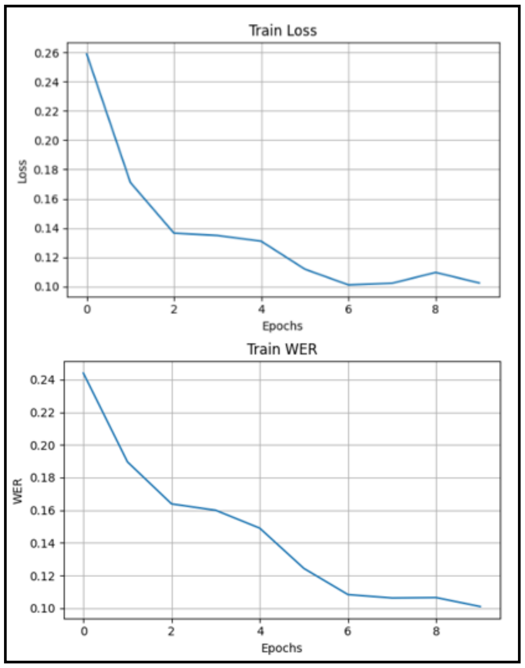
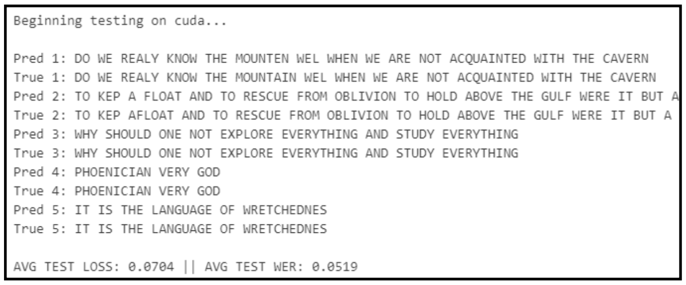


Fig. 7. Wav2Vec2 learning curves for training loss and training Word Error Rate (WER).

From the Fig. 7 learning curves, it is evident that train loss and train WER started to plateau around epochs 5-6, and overall convergence is erratic, likely due to the small batch size. Moving Wav2Vec2 tuning to the A100 GPU used for Stable Diffusion v1.5 would likely mitigate this issue. Additionally, it would be wise to use significantly more diverse samples in future work, as elaborated in Section IV below. This setup was chosen specifically for its simplicity and ease-of-implementation.

The testing results of Wav2Vec2, including testing loss, testing WER, and examples of predicted and ground truth transcripts can be seen in Fig. 8.

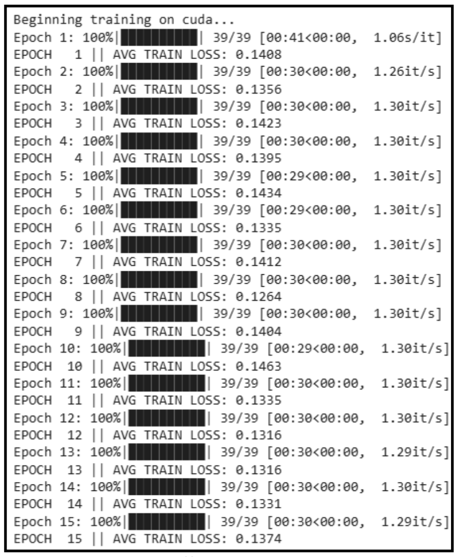


```
Beginning testing on cuda...

Pred 1: DO WE REALY KNOW THE MOUNTEN WEL WHEN WE ARE NOT ACQUAINTED WITH THE CAVERN
True 1: DO WE REALY KNOW THE MOUNTAIN WEL WHEN WE ARE NOT ACQUAINTED WITH THE CAVERN
Pred 2: TO KEP A FLOAT AND TO RESCUE FROM OBLIVION TO HOLD ABOVE THE GULF WERE IT BUT A
True 2: TO KEP AFLOAT AND TO RESCUE FROM OBLIVION TO HOLD ABOVE THE GULF WERE IT BUT A
Pred 3: WHY SHOULD ONE NOT EXPLORE EVERYTHING AND STUDY EVERYTHING
True 3: WHY SHOULD ONE NOT EXPLORE EVERYTHING AND STUDY EVERYTHING
Pred 4: PHOENICIAN VERY GOD
True 4: PHOENICIAN VERY GOD
Pred 5: IT IS THE LANGUAGE OF WRETCHEDNES
True 5: IT IS THE LANGUAGE OF WRETCHEDNES

AVG TEST LOSS: 0.0704 || AVG TEST WER: 0.0519
```

Fig. 8. Wav2Vec2 testing results.

From Fig. 8, the Wav2Vec2 test loss and test WER (~0.07 and ~0.05 respectively) are both better than the train loss and train WER (~0.10 and ~0.10 respectively). Ideally the test and train performances are equivalent, however this difference is not a cause for concern. It is likely a result of potential regularization effects in the Wav2Vec2ForCTC architecture, differences between the “dev_clean” and “test_clean” LibriSpeech subsets themselves, or information loss since the train and test datasets are only 1,000 samples each. Nonetheless, performance is exceptional and looking at the predicted transcripts and ground truth transcripts, errors are minor and result in phonetically equivalent text. For instance, “Pred 1” spells “MOUNTEN” instead of “MOUNTAIN”, and “Pred 2” spells “A FLOAT” instead of “AFLOAT”.

B. Stable Diffusion v1.5: The basic training loss results for Stable Diffusion v1.5 can be found below in Fig. 9.



```
Beginning training on cuda...
Epoch 1: 100%|██████████| 39/39 [00:41<00:00,  1.06s/it]
EPOCH   1 || AVG TRAIN LOSS: 0.1408
Epoch 2: 100%|██████████| 39/39 [00:30<00:00,  1.26it/s]
EPOCH   2 || AVG TRAIN LOSS: 0.1356
Epoch 3: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH   3 || AVG TRAIN LOSS: 0.1423
Epoch 4: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH   4 || AVG TRAIN LOSS: 0.1395
Epoch 5: 100%|██████████| 39/39 [00:29<00:00,  1.30it/s]
EPOCH   5 || AVG TRAIN LOSS: 0.1434
Epoch 6: 100%|██████████| 39/39 [00:29<00:00,  1.30it/s]
EPOCH   6 || AVG TRAIN LOSS: 0.1335
Epoch 7: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH   7 || AVG TRAIN LOSS: 0.1412
Epoch 8: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH   8 || AVG TRAIN LOSS: 0.1264
Epoch 9: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH   9 || AVG TRAIN LOSS: 0.1404
Epoch 10: 100%|██████████| 39/39 [00:29<00:00,  1.30it/s]
EPOCH  10 || AVG TRAIN LOSS: 0.1463
Epoch 11: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH  11 || AVG TRAIN LOSS: 0.1335
Epoch 12: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH  12 || AVG TRAIN LOSS: 0.1316
Epoch 13: 100%|██████████| 39/39 [00:30<00:00,  1.29it/s]
EPOCH  13 || AVG TRAIN LOSS: 0.1316
Epoch 14: 100%|██████████| 39/39 [00:30<00:00,  1.30it/s]
EPOCH  14 || AVG TRAIN LOSS: 0.1331
Epoch 15: 100%|██████████| 39/39 [00:30<00:00,  1.29it/s]
EPOCH  15 || AVG TRAIN LOSS: 0.1374
```

Fig. 9. Stable Diffusion v1.5 training results.

Looking at Fig. 9, the diffusion training is erratic and shows no indication of convergence, oscillating between ~0.13-0.14 average MSE loss. Considering the images are on a [-1, 1] scale within the “train” function, these MSE values are not inherently poor. However, the lack of improvement suggests that the pretrained Stable Diffusion v1.5 model was already competent at producing facial images, and we have likely reached a performance threshold given the specific training data being used. The learning curve corresponding to the training loss can be seen in Fig. 10.

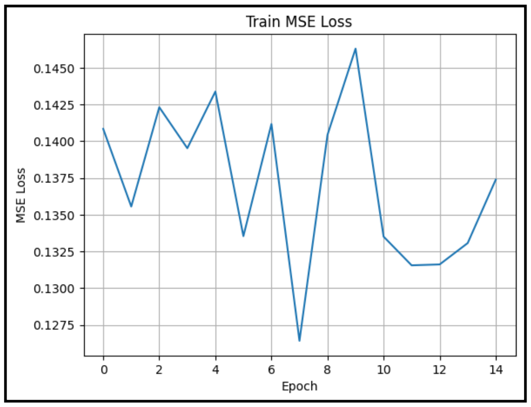


Fig. 10. Stable Diffusion v1.5 learning curve for training loss.

The learning curve supports the erratic training pattern observed in Fig. 8.

The testing results of Stable Diffusion v1.5, specifically the MSE testing loss, was found to be 0.1265. This is sensible, being slightly better than the training MSE but still within a range of ~1%.

MSE alone does not reveal whether the model is semantically aligning the generated facial features with the caption. To explore this, the results of the “eval_truth_pretrained_true” function described in section II subsection D can be seen below.

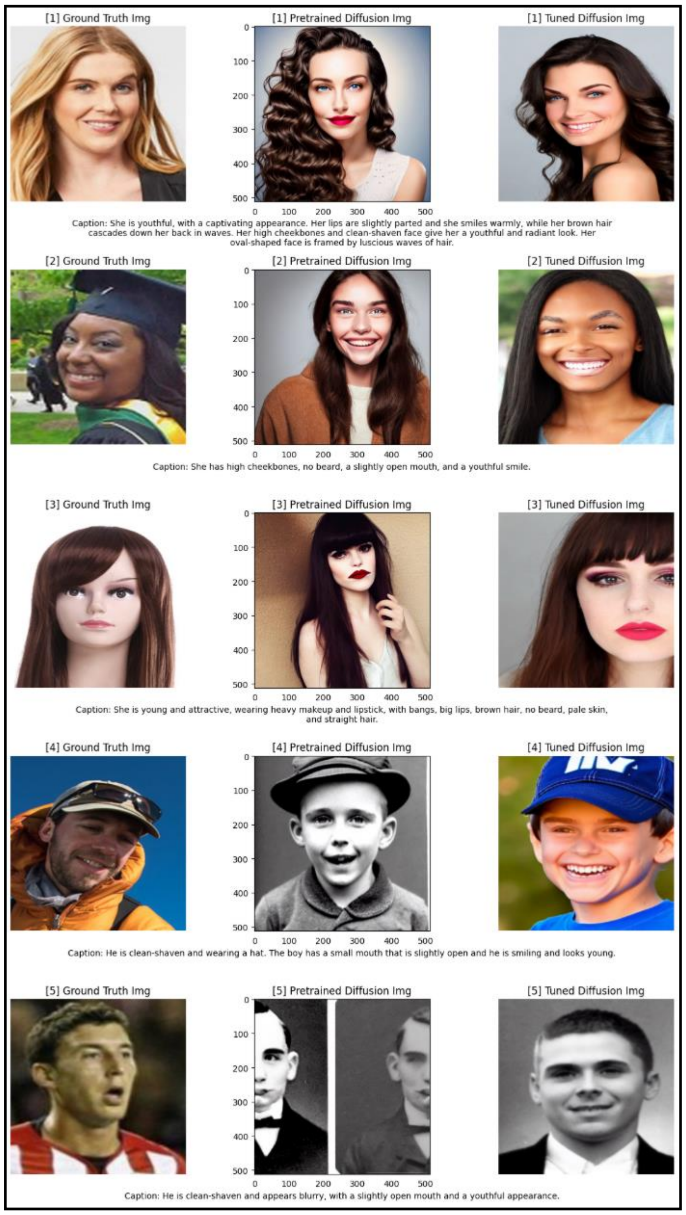
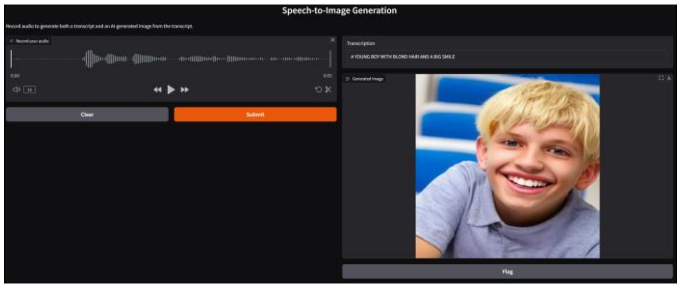

[1] Ground Truth Img
[1] Pretrained Diffusion Img
[1] Tuned Diffusion Img
Caption: She is youthful, with a captivating appearance. Her lips are slightly parted and she smiles warmly, while her brown hair cascades down her back in waves. Her high cheekbones and clean-shaven face give her a youthful and radiant look. Her oval-shaped face is framed by luscious waves of hair.
[2] Ground Truth Img
[2] Pretrained Diffusion Img
[2] Tuned Diffusion Img
Caption: She has high cheekbones, no beard, a slightly open mouth, and a youthful smile.
[3] Ground Truth Img
[3] Pretrained Diffusion Img
[3] Tuned Diffusion Img
Caption: She is young and attractive, wearing heavy makeup and lipstick, with bangs, big lips, brown hair, no beard, pale skin, and straight hair.
[4] Ground Truth Img
[4] Pretrained Diffusion Img
[4] Tuned Diffusion Img
Caption: He is clean-shaven and wearing a hat. The boy has a small mouth that is slightly open and he is smiling and looks young.
[5] Ground Truth Img
[5] Pretrained Diffusion Img
[5] Tuned Diffusion Img
Caption: He is clean-shaven and appears blurry, with a slightly open mouth and a youthful appearance.

Fig. 11. Stable Diffusion v1.5 example ground truth images, pretrained output images, and final tuned output images.

From Fig 11, it is evident that although tuning the diffusion model did not improve MSE, it did improve the model’s ability to semantically generate faces adhering to the caption. Looking at the row 1 images, only the tuned diffusion model captured the “slightly parted” lips and the hair “down her back”. Looking at the row 3 images, the pretrained diffusion model’s image is significantly less photo-realistic than the tuned diffusion model’s image. Lastly, looking at the row 5 images, the pretrained diffusion model did not even generate a single face, and the faces it did generate do not have a “slightly open mouth”, while the tuned diffusion model successfully did both.

Although the generated images all look vastly different from the ground-truth images, this can likely be attributed to poor, undescriptive captioning. For instance, the row 2 caption only mentions "high cheekbones, no beard, a slightly open mouth, and a youthful smile," all of which were successfully captured by both the pretrained and tuned diffusion models. The large differences between generated images can also be attributed to an improper setting of "guidance_scale", which controls the diffusion model's adherence to the caption. Larger values will follow the caption more strictly but have less creativity. Acquiring more descriptive training data and tuning "guidance_scale" could lead to significant performance improvements in future work. This is explained more in section IV below.

C. *Gradio:* The Gradio interface with recorded audio, transcribed text, and a final generated image can be seen below in Fig. 12.


Speech-to-Image Generation
Record audio to generate both a transcript and an AI-generated image from the transcript.
Record your audio
0:00
0:05
1x
Clear
Submit
Transcription
Generated Image
Flag

A YOUNG BOY WITH BLOND HAIR AND A BIG SMILE

Fig. 12. Gradio interface.

This interface provides a clean, simplistic way for users to utilize the speech-to-image generation system without any setup or knowledge of its inner workings.

IV. Future Work

While the current framework demonstrates promising results, there are several clear directions for future work to improve overall performance, robustness, and practical deployment.

Firstly, improving the training data for both Wav2Vec2 and Stable Diffusion v1.5. For Wav2Vec2, using other LibriSpeech subsets besides "dev-clean" would expose the model to more complex, varied pronunciations and background conditions. For Stable Diffusion v1.5, using a larger portion of the FaceCaption-15M dataset and/or manually creating better captions would improve generalization and allow the model to better capture semantic facial features.

Secondly, utilizing a validation set with early stopping. This would provide a clear indication of both models' generalization abilities, ensuring that training is stopped at the optimal time. Additionally, validation sets could be used to perform a systematic hyperparameter search. Hyperparameters such as learning rate, weight decay, and batch size were only slightly varied within this study, and a systematic search could significantly improve model convergence and final performance. Additionally, some hyperparameters were completely fixed in this study, like "guidance_scale", which controls the diffusion model's creativity and adherence to the prompt when generating images.

Lastly, there are many smaller directions to explore. Model quantization and pruning could drastically reduce model complexity with minimal performance losses, which is important for final deployment in the field. Additionally, evaluating Stable Diffusion v1.5 with CLIP-based metrics rather than MSE could better capture how well the model is generating facial features as opposed to pixel-by-pixel error, enabling more informative model training. Lastly, implementing user-informed, continuous generation could produce a better overall suspect image. In this setup, a user's description would generate multiple faces at once, the user would select the face that best matches the suspect, and variations on that face would then be generated, repeating this until the user settles on a final face. This approach is similar to EFIT-V [2] but with the benefit of starting with a detailed audio description of the witness.

V. Conclusion

This paper presented a lightweight, multimodal speech-to-text-to-image framework for rapid facial reconstruction in forensic settings, combining a tuned Wav2Vec2 architecture for speech recognition with a tuned Stable Diffusion v1.5 architecture for text-guided image generation. Experimental results demonstrated excellent speech-to-text performance, achieving a Word Error Rate (WER) of 5% on the test dataset, and promising text-to-image generation performance with a final test MSE of approximately 0.13 and a noticeably improved ability to generate facial features, despite limited training resources. The system was also packaged into a user-friendly Gradio interface, offering real-time, browser-based audio recording, transcription, and facial image generation without requiring technical expertise.

This end-to-end pipeline demonstrates significant potential for rapid, offline suspect reconstruction, minimizing witness memory decay and enabling immediate downstream analysis like database searching or scanning public surveillance cameras within a radius.

Nonetheless, several limitations were identified, such as poor training datasets, lack of a validation set, and overreliance on MSE for evaluating the diffusion model. Future work will address these gaps and explore additional avenues for improvement. Overall, this framework offers a compelling first step towards practical, AI-driven suspect sketching in time-critical forensic scenarios.

References

[1] "Faces software download: Faces 4.0: Facial composite software," Faces Software | Facial Composite Software | Faces 4.0, https://facialcomposites.com/ (accessed Apr. 29, 2025).

[2] A. M. Lech and R. A. Johnston, "Facial composite production using EFITV," *2010 International Conference on Emerging Security Technologies*, pp. 44–49, Sep. 2010. doi:10.1109/est.2010.25

[3] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations," *Facebook AI*, Oct. 2020. doi: https://doi.org/10.48550/arXiv.2006.11477

[4] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification," *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pp. 369–376, 2006. doi:10.1145/1143844.1143891

[5] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with Latent Diffusion Models," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10685, Jun. 2022. doi:10.1109/cvpr52688.2022.01042