TransLingualVisionary

Authors: Kavish Purani, Neeraj Ramesh, Sandra Serbu
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—The first iteration of a system capable of detecting, interpreting, and displaying live ASL speech to English text. TransLingualVisionary is a novel system that enables hearing impaired ASL users to better participate in virtual streaming environments. Using a live video stream, MediaPipe's human pose estimation, a SPOTER word classification model, and a LLM to perform sign language translation, TLV contributes to developing sign language processing systems using machine recognition and translation.**

***Index Terms*— ASL, Gesture Recognition, HPE, LLM, NVIDIA Jetson Nano, MediaPipe, NLP, RNN, SLT, SPOTER, STGCN, TLV, Video Processing**

1 INTRODUCTION

For many deaf or hard of hearing (HOH) individuals, sign language is a faster and more efficient way to communicate than written text. Written text can be inaccessible to deaf individuals due to its speed and the difficulty of learning a written language without it's phonetic component. For many hearing impaired, lack of widespread knowledge of American Sign Language (ASL) often requires them to rely on assistance from translators to communicate with non-ASL users.

Many digital environments today feature video/audio communication platforms that create this same challenge for anyone who uses ASL as their primary method of communication. Deaf individuals participating in live digital environments, such as online meetings and live streams, don't have the autonomy to engage in any digital environment without the aid of a translator. Our project seeks to be a step towards bridging that communication gap closer together through the development of "TransLingualVisionary" (TLV), a real-time ASL to English text translation platform.

TLV addresses these issues head-on by offering a user-friendly web application that enables live translation of ASL into text, promoting inclusivity and reducing digital isolation. We plan to do this using computer vision and machine learning to transcribe a live video feed into written English text in real time. A MediaPipe Human Pose Estimation (HPE) and word classification models detect ASL from a video feed input and pass signed words to a Large Language Model (LLM) that interprets ASL-syntax sentences into natural English text. The output text will then be displayed on a web application, enabling efficient two-way communication on audio/video streaming platforms.

Any hearing impaired individual who prefers communicating via ASL but interacts with non-ASL users virtually could benefit from TLV.

2 USE-CASE REQUIREMENTS

The primary objective of this project is twofold: to increase accessibility to community involvement by expanding the range of virtual communication ASL users have access to and to reduce digital isolation felt by those within deaf and HOH communities. By achieving these objectives, TLV aims to set a new standard for inclusivity in digital communication platforms. These are the use-case requirements to guarantee the best user experience:

- Speed – The user will need their signed speech translated at an approximately real-time pace to provide fluid translation in a conversational setting. Thus the product must have minimal latency; we aim to detect signs at speeds of 120 words per minute at minimum [14], which matches the average speed at which a fluent ASL speaker signs.
- Accuracy – For our product to be functional, it must reliably translate the majority of a user's signed speech accurately. A user should not need to re-sign or type out words/phrases to effectively communicate. Thus, we aim for ∼40% BLEU score[7] for sentence translation (see section 7.2 for metric details) for sentence translation on a dataset of 2000 words.
- Recognition Distance – Our design must work within a distance range that any user of a digital environment may want to use. The system must successfully recognize full-upper body signs captured at a distance up to 4-5 feet away from the camera.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

Our architecture breaks down this ASL translation task into 3 main components:

- Human pose estimation model: generates human skeleton landmarks of significant body points given a video input.
- Classification model: classifies a sequence of skeletal landmarks to a certain gesture class.
- LLM model: performs sign language translation (SLT) by interpreting a sequence of transcribed ASL words with high lexical similarity to spoken English but low syntactic similarity.

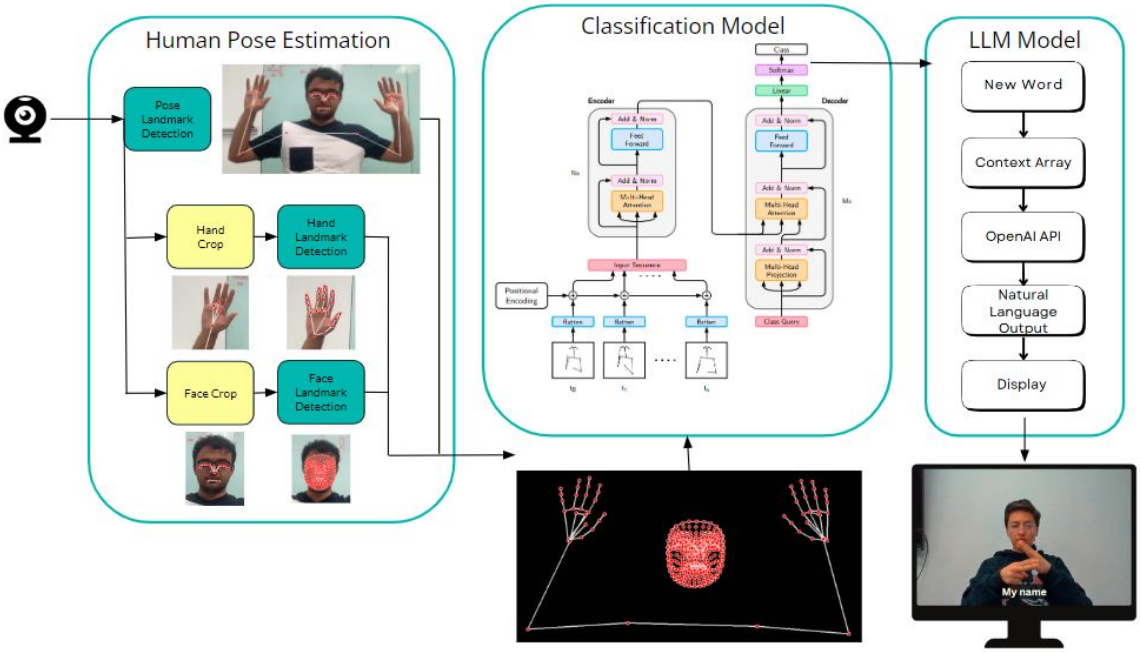

Figure 1: System diagram. Each component and its overarching tasks within the system's flow of information. User video input is captured by an external camera that's sent to human pose estimation. HPE vectors are sent to the word classification model to generate an English word or phrase. Generated text is used by an LLM to perform SLT and generate a natural language English sentence that can be sent to the application display to be seen by users.

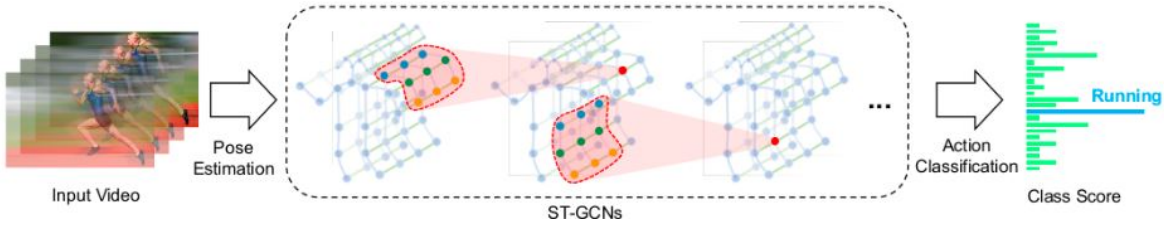

Figure 2: A closer look at the Word Classification model through a block diagram. The word classification model utilizes HPE landmarks to detect and predict it's most-confident word classification.

Overall system information flow can be seen in Figure 1, which displays how components interact within the system. The system architecture integrates three main computational modules using a Logitech C920S camera and an NVIDIA Jetson Nano Developer Kit.

Since the system will all be running on a single device, we need to ensure that each component is efficient and can run in real-time. The HPE model processes each video frame to extract the skeletal landmarks. These landmarks are then filtered and processed to be fed into the classification model, which will output the recognized ASL words. The sequence of words will then be passed to an LLM API call, which performs SLT and generates a meaningfully analogous English sentence. This sentence will feed into a web application that displays translated text.

One of the core engineering principles employed in our ASL translation system is the divide-and-conquer approach. We break down the complex problem of translating ASL videos into English text into smaller, more manageable sub-problems: pose estimation, gesture classification, and language translation. Each of these sub-problems is tackled by a specialized model tailored for that specific task. This modular design not only simplifies the overall system but also allows for easier optimization, maintenance, and potential upgrades to individual components without affecting the entire pipeline. Another engineering principle we leveraged is abstracting low-level data into higher-level representations. The pose estimation model abstracts the raw video data into a sequence of skeletal landmarks, which serves as a more compact and informative representation for the downstream gesture classification model. Similarly, the classified gestures are further abstracted into a sequence of ASL glosses or sign labels, which the language model can interpret more effectively.

The human pose estimation component of our system is grounded in principles of computer vision. Techniques like convolutional neural networks (CNNs) and object detection algorithms are used to locate and track body landmarks across video frames. These methods leverage concepts like feature extraction, edge detection, and pattern recognition

to identify human forms and extract meaningful skeletal representations. The language translation component utilizes principles from the field of natural language processing (NLP). While we did utilize a general LLM API, which is not inherently trained for NLP tasks, the size of these models and their datasets, along with a transformer architecture and self-attention mechanisms allows them to understand the structure and context of ASL sign sequences and generate coherent English translations [11].

Mathematically, linear algebra plays a crucial role in various components of our system. The neural network models, including the pose estimation, gesture classification, and language models, rely heavily on linear algebraic operations like matrix multiplications, convolutions, and transformations. These operations are fundamental to the propagation of data through the network layers and the learning of weight parameters during training. Additionally, our classification model is trained using principles of probability and statistics. Optimization objective functions, such as cross-entropy loss or mean squared error, are grounded in statistical concepts and techniques like back propagation, stochastic gradient descent, and regularization methods used to adjust model parameters are based on the principles of probability and statistical inference. Both our human pose estimation and gesture classification model also employ principles from signal processing. Techniques like filtering, normalization, and temporal modeling are used to preprocess and analyze the spatio-temporal patterns in the landmark data.

4 DESIGN REQUIREMENTS

The design specifications established below will ensure our product's engineering design solution meets all use-case requirements defined in Section 2's Use-Case Requirements.

4.1 Speed

To ensure overall latency will be approximately real-time, we aim to present visual feed and translation on web UI within 3 seconds. The average English sentence is between 15–20 words long and, 3 seconds would provide enough time to group words into an approximation of sentence-length clusters for an average-paced ASL user. The text generation LLM can use context past a 3-second window to re-format sentences in the light of new information, like an eyebrow raise which could indicate a question or confusion, but some estimation of sentence content should be displayed in this time frame.

The 3-second timing requirement will need to be distributed across all system components. The distribution of component latencies necessary to meet this timing requirement is described in Section 7.1.

4.2 Accuracy

To achieve a sentence translation BLEU[7] score of 40% minimum, we've created 3 quantitative design requirements to meet. Our system will need to recognize when a user is signing, correctly identify ASL words, and correctly interpret ASL semantics. To gauge these metrics, our system will meet these 3 requirements across 3 separate components:

- ∼95% sign recognition rate. That is, the classification model will be able to detect when a user is signing at least 95% of the time. Airing on the side of caution, our system may allow for false positives if a user motions in a way that is similar to signed speech.
- We originally aimed to recognize 2000 signed words at ∼85% accuracy. Given the limitations and scope of our project, we ultimately found that 2000 word classifications was unrealistic as seen by current research [5]. As a result, our final model was trained on 100 words. Word classification must be able to correctly classify all words included in the training dataset with about 80% accuracy. Classifications will be detected and modified in the LLM text generation process, allowing for a slight margin of error in our classification model.
- Translate identified clusters of words into full english sentences with a BLEU[7] score of ∼40%. Given syntactically ASL word sequences, the LLM text generation must produce a natural english sentence with a BLEU[7] score of ∼40%.

4.3 Classification Distance

To ensure all users are guaranteed a useful product, our system must recognize and retain the speed and accuracy metrics of the classification model for users up to 4-5 feet away from the camera input. The HPE model should be able to pass information that is equally useful for the word classification model.

5 DESIGN TRADE STUDIES

5.1 Human Pose Estimation

5.1.1 Mediapipe vs OpenPose

OpenPose and MediaPipe are both open-source libraries for detecting 2D and 3D human pose estimation from image and video data. However, there are a few key differences that make MediaPipe a more suitable solution for American Sign Language (ASL) translation purposes.

A core distinction is that MediaPipe has been optimized for real-time performance on mobile and edge devices, enabling faster and more responsive pose tracking. This allows MediaPipe models to pick up rapid transitions

between ASL signs more accurately compared to OpenPose, which is built for maximixing precision while sacrificing speed. We can see this in the research paper by Kien Nguyen Phan et al., in which they have researched and detailed the comparisons between the two HPE models [16]. Additionally, MediaPipe natively supports tracking of hand landmarks and face mesh - both crucial components to identify individual signs and facial expressions in ASL. The hand and face modeling in MediaPipe captures more fine-grained details like finger curls and eye gaze direction. Finally, MediaPipe's model-building process is more customizable to target the specific use case of ASL translation by tuning appropriate semantic thresholds and heavier loss weights. The optimized models use less computing for inference - an advantage over OpenPose when deploying translation apps on low-power devices.

We also had the choice of decidign between various Mediapipe models, namely between the hand-only estimation model [12] and the holistic estimation model [13]. On one hand, just landmarking the hand gives us fewer outputs, which could make our pipeline infer faster and take in fewer vectors as the input for our classification model. However, this has downsides in that it only captures the hand's position relative to the image frame, meaning that this could potentially cause a distance issue or could miss out on important facial expressions that could indicate important ASL semantics. As such, we made the decision to utilise the holistic model and limit what landmarks we do use in order to minizmize the number of parameters that we need. Since the holistic model is comprised of Mediapipe's hand, face mesh, and pose estimation models, we decided to use the the hand landmarks and a subset of the pose landmarks. This not only provides useful classification indicators, but also acts as points of normalization for our other points.

5.1.2 FPGA v. Jetson v. Computer

The main consideration between these two devices is which one would hold the Human Pose Estimation model and how that would impact the latency of the overall pipeline. FPGAs provide a parallel architecture that can be optimized for deep learning inference workloads, enabling extremely low latency predictions even on larger models. As per Farhad Fallahlalehzari, who quoted Xlinx research comparing FPGAs and GPUs, "FPGAs result in significantly higher computer capability", while also being far less power-hungry, ensuring more cost-efficient and stable operation when deployed [8]. Additionally, by running pose estimation directly on the FPGA, the landmark location vectors can be transferred to the main application processor without needing intermediate serialization. This reduces overall system latency and allows for faster processing of the pose data, important for real-time requirements. There is also the consideration of space. Both of the models we are considering for human pose estimation can be fit on an FPGA and a Jetson, but the Jetson will also be holding our classification model as well. As such, running human pose estimation on the Jetson might mean that we would have to reduce the size of our classification model, decreasing the number of parameters that we have and risking lower accuracy numbers or classify on a smaller overall class set.

However, the FPGA is only worth it if we are able to quantize the human pose estimation model on the FPGA because otherwise, we are just running human pose estimation on the ARM processor of the FPGA board and we would not get any of the benefits of FPGA optimization. In this case, a Jetson would be far superior considering the processing difference between both the Jetson and the FPGA. Unfortunately, Mediapipe's holistic model was not able to be quantized and thus could not necessarily be optimized on the FPGA given our resources. As such, we had decided to use the Jetson for our overall system. Ultimately, the Jetson Nano Developer kit is no longer supported and is in the process of being discontinued. All Jetson modules and developer kits are supported by JetPack SDK, but due to the Jetson Nano's EOL status, the latest JetPack image that's compatible with our model has deprecated dependencies that are not compatible with the dependencies required to run the Mediapipe version that we wanted. Additionally, we did not have ethernet access to internet so it was difficult showcase our working LLM API on the jetson. As a result, we needed to use a computer's CPU to present the work that we out into our project for the final demo.

5.2 Classification Model

A crucial component of our ASL-to-English translation pipeline is the gesture classification model, which takes the sequence of skeletal landmarks from the pose estimation stage and classifies them into individual ASL signs or gestures. For this task, we evaluated three different deep learning architectures: Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNNs), Transformers (specifically the SPOTER model), and Spatial-Temporal Graph Convolutional Networks (ST-GCN).

5.2.1 LSTM RNN

LSTM RNNs have been widely used for sequence classification tasks, including gesture recognition. They process the input sequence of skeletal landmarks sequentially, while maintaining an internal hidden state that captures the temporal context from previous time steps. This recurrent nature allows LSTMs to model the dynamics and dependencies within the gesture sequences effectively. One advantage of using LSTMs for gesture classification is their ability to handle variable-length input sequences, which is crucial as different ASL signs may have varying durations. Additionally, LSTMs can capture relatively long-term dependencies within the gesture sequences in comparison to their Gated Recurrent Unit (GRU) RNN counterparts. This is especially important for recognizing complex signs that involve multiple movements or transitions. However, LSTMs suffer from some limitations that led us to not choose an LSTM model. Difficulty in parallelization results in slower training

and inference times, which can be problematic for real-time applications like our ASL translation system. Furthermore, LSTMs are susceptible to the exploding gradient problem, which occurs when the gradients become extremely large during backpropagation, causing the weights to update in an unstable manner, leading to poor convergence or divergence. Standard GRU RNNs also suffer from the vanishing gradient problem, which is when the gradients become increasingly smaller as they are backpropagated through many layers, making it difficult for the model to learn long-term dependencies effectively. The vanishing gradient problem arises due to the repeated multiplication of small values during backpropagation, causing the gradients to shrink exponentially over long sequences. While the architectural difference between an LSTM and a GRU model handles this, an LSTM is still susceptible to the exploding gradient problem where the gradient grows exponentially over time, which would heavily hinder performance.

5.2.2 Transformer: SPOTER

Transformer models are a powerful alternative to RNNs like LSTMs for sequence modeling tasks. Instead of relying on recurrent computations, transformers utilize self-attention mechanisms to capture long-range dependencies within the input sequence. The self-attention mechanism allows each element in the sequence to attend to all other elements, enabling the model to capture dependencies regardless of their distance in the sequence. Compared to LSTMs, Transformers offer several advantages for gesture classification:

- Parallelization: Transformers are parallelizable, as the computations for different positions in the sequence can be performed independently. This parallelization capability allows for faster training and inference times, which is crucial for real-time applications like our ASL translation system.
- Long-term Dependencies: The self-attention mechanism in transformers allows them to capture long-term dependencies more effectively than LSTMs, without suffering from the vanishing/exploding gradient problem. This property is especially beneficial for recognizing complex ASL gestures that involve multiple movements or transitions over an extended duration.
- Structural Modeling: While traditional transformers do not explicitly model the structural relationships between body joints, they can be adapted to incorporate such information through techniques like spatial or graph-based attention mechanisms.

The model that we ended up using for this was the SPOTER model [2]. The SPOTER (Sign POse-based TransformER) is a transformer architecture proposed for word-level sign language recognition from video. It takes sequences of 2D skeletal landmarks extracted by pose estimation as input. A novel normalization scheme projects these landmarks onto the "signing space" in front of the signer to focus on relevant spatial relationships. Augmentations like rotations and perspective transforms are applied to the skeletal data for better generalization. At its core is a transformer encoder-decoder, where the decoder has a single "class query" vector that attends to the encoded landmark sequence to predict the sign language gesture class. SPOTER achieves state-of-the-art performance on datasets like WLASL and LSA64 for the pose-based recognition setting. Being skeletal-based, it is computationally efficient compared to models using full image/video inputs. It can achieve good accuracy even from limited training data, benefiting from the informative pose representation and augmentations tailored for sign language.

SPOTER was one one of the two models that we explored and tested for our pipeline, as it worked as a step forward progression from an LSTM. Both this paper and an embeddings exploration of the SPOTER model [10] had relatively high accuracy numbers for the WLASL dataset [6] compared to other architectures. When training the SPOTER model, we were able to achieve a validation accuracy of about 73% for a 100 class dataset and a validation accuracy of about 91% for a 10 class dataset. However, during inference time, we were only able to reach an accuracy of approximately 55% for our 10 class dataset.

5.2.3 ST-GCN

The final model that we considered and ended up using for our demo was the ST-GCN model. ST-GCN (Spatial-Temporal Graph Convolutional Networks) is a graph-based approach that leverages the structural information of the human body to learn spatial and temporal patterns for gesture recognition. It represents the human skeleton as a spatial-temporal graph, where nodes correspond to joints, and edges encode the physical connections between them. ST-GCN applies graph convolutional operations to capture the spatial and temporal dependencies within this graph representation, allowing for gesture classification. The key advantage of ST-GCN for ASL gesture recognition is its ability to explicitly model the anatomical constraints and relationships between body joints, which is useful for recognizing hand shapes and precise hand-body interactions. Additionally, ST-GCNs have demonstrated superior performance on datasets tailored for sign language recognition, outperforming other methods like LSTMs and conventional convolutional models. However, an ST-GCN can be more computationally expensive due to the graph convolutions, especially for larger graph structures representing the human skeleton [19].

We utilized the ST-GCN architecture of Microsoft's ASL Citizen Dataset [5], the open-source code for which was developed by Alex Lu and Microsoft [1]. One of the issues we had with the SPOTER model was the WLASL dataset; since we installed it locally, we noticed that there were a lot of videos missing, resulting in some gesture classes having an uneven number of training videos to work with, and we believe that this impacted the misclassifica-

tion rate of our SPOTER model. As a result, we ended up using Microsoft's ASL citizen dataset, created through community sourced clips fo ASL gestures and an ST-GCN model architecture. Training this led to a validation accuracy around 82%, which is lower than that of the SPOTER model. But during inference time, we were able to achieve a much higher value sitting around 70%, which is far closer than that of the SPOTER model.

5.3 OpenAI GPT3.5 v. Meta Llama2

For our LLM, we were considering two different models - OpenAI's GPT3.5 model and Meta's Llama2. We chose these because we have access to both and have experience using both for past projects. Based on research, we were able to determine that this tradeoff is similar to that of the GRU v. LSTM tradeoff from earlier. From Diana Cheung, we were able to learn that Llama2 is overall a lighter model in comparison to GPT3.5, meaning that there is a chance that it is faster, but GPT3.5 does have higher performance and accuracy. She also mentions that the decision is heavily influenced by the risk tolerance of our use case requirements [3]. Additionally, we also realized that Llama2 does not necessarily have an easy to use API like that of GPT3.5, meaning that we would have to run Llama2 locally, which could impact the performance of our other components. GPT3.5 having a simpler library and API call structure made it the better choice.

6 SYSTEM IMPLEMENTATION

6.1 Human Pose Estimation

The first subsystem we developed is our Human Pose Estimation model using Mediapipe's holistic model and OpenCV. AS mentioned earlier, Medaipipe is an open source library provided by Google for on-device machine learning purposes, including vision and land marking for our use case. OpenCV is another open source library centered around providing user with computer vision technology, which we used for frame and video capture for individual frame analysis.

Our human pose detection model first begins by using OpenCV's frame reader to keep pulling frames form our camera for processing. This brings us to the Mediapipe model itself, which handles the human pose estimation. At its core, Mediapipe's holistic model integrates separate machine learning models that are each specialized in their respective domains: pose estimation, facial landmark detection, and hand tracking. The pose estimation model, known as BlazePose, operates on a lower, fixed resolution to efficiently capture the overall human form. It serves as the foundation upon which the other models build, providing critical information about the position and orientation of the body. Once the pose is detected, the Holistic model employs a multi-stage pipeline to refine the detection of hands and face. It generates regions of interest (ROIs) based on the pose landmarks, which are then re-cropped using a dedicated model to ensure high-resolution inputs for the subsequent stages. This re-cropping is essential because the initial pose estimation operates at a lower resolution, which is not sufficient for the detailed articulation required for hands and face. For the facial landmarks, the Holistic model applies a task-specific model that can detect up to 468 points on the face, providing a detailed map of facial features. Similarly, for hand tracking, the model identifies 21 landmarks for each hand, allowing for precise detection of hand movements and gestures. Additionally, we have 32 landmarks that are classified for the overall pose itself [15].

Initially, when we were using the SPOTER model, we only held on the hand and a few of the pose landmarks, as those were the main points of interest when considering sign language. However, for the ST-GCN, we utilized the entire holisitc suite, due to the nature of how they work (covered in section 6.2). As once we get the landmarks themselves, we process the landmarks such that they fit the data stricture of the appropriate model. During training time, we store these in npy files for later reference so that we can preprocess and train the model separately. Additionally, in order to improve real time recognition, we decided that we only would save the landmark data for frames that had at least one hand in the, as we did not want to influence either model with hand-less frames that are not relevant to the sign itself. For inferencing, we were able to determine that the overall latency of running human pose estimation and the classification model sequentially was negligibly similar to that of running them in parallel, hence allowing us to simply plug the processed HPE data during inference time.

6.2 Classification

Our classification model works using ST-GCN model that we mentioned earlier. An ST-GCN (Spatial Temporal Graph Convolutional Networks) model is designed for skeleton-based action recognition from sequences of body joint coordinates. The first step is to construct a spatial-temporal graph representation of the input skeleton data. The node set of this graph contains all the body joints across all frames of the input sequence. The edge set is divided into two subsets - spatial edges that connect joints within each frame based on the natural skeletal connectivity, and temporal edges that link the same joints across consecutive frames, representing their motion over time. This spatial-temporal graph explicitly models the spatial and temporal relationships present in the skeleton sequence [19].

The core of the ST-GCN model is the spatial-temporal graph convolution operation performed on the constructed graph. For each node (body joint), the convolution computes a weighted sum of the feature vectors from its spatial-temporal neighboring nodes, determined by spatial and temporal kernel sizes. The weight matrices for combining neighbor features are partitioned into subsets based on strategies like distance partitioning or spatial configuration partitioning. The spatial configuration strategy separates

neighbors into the root node itself, nodes closer to the skeleton's center (centripetal), and nodes farther from the center (centrifugal) to model concentric and eccentric body motions. Learnable edge importance weights scale the contributions of different spatial edges [19].

The ST-GCN has multiple layers of these spatial-temporal graph convolution units, with increasing output channels in higher layers. Residual connections and dropout are used for regularization. The final feature representation is fed into a softmax classifier to predict action class probabilities. The entire model is trained end-to-end using stochastic gradient descent to minimize the cross-entropy loss. Data augmentation techniques like random affine transformations and fragment sampling are applied to the input during training [19].

For action recognition on a test example, the spatial-temporal graph is first constructed from the input skeleton sequence. This graph is then fed through the trained ST-GCN model to extract feature representations, which are passed to the softmax classifier to obtain action class probabilities. The class with the maximum probability is output as the predicted action class [19].

For training, we utilized Microsoft's ASL Citizen database [5], which had been preprocessed using Mediapipe's Holistic model, as we mentioned earlier, which in total contains about 84,000 videos over 2,700 classes. Given our time frame resources, we focused on 10 common classes to train on. The reason for this is that, when looking at the provided metrics by Microsoft, as classification goes up, accuracy significantly drops. We can see this with the 32% accuracy that the 2700 class model gets. While this is good in a vacuum for research purposes, it did not hold up for our use case requirements. As such, we decided it would be better to focus on accuracy over class size for now and focused on 10 common classes and expand from there in the future, as it would just be a matter of improving the model.

During inference, we take the weights that we saved from training and instantiate a new model. This then gets skeletal data from human pose estimation passed though it to obtain a word classification for a given input. As more sequences come into the video itself, we collect these words and pass it onto the LLM API. One of the main challenges of inferencing however is the number of frames to include in an inference. When we are training and testing static videos, we know the number of frames that we are passing into the model. However, this is not the case in a real-time scenario, as we just have an endless stream of frames. As such, we have to decide how many frames to input into the model at a time for classification. Using a static count of frames occasionally works, but often results in signs being classified part way through the sign, resulting in misclassification. Using overlapping batches, where some frames from the previous batch are kept and the remainder of the batch being filled with new batches results in latency issues when inferencing. A solution that works is have a start and stop capability so that we can start and stop when we are recording a batch of frames be pushed into the model. This can either be done using a button that control when we start and stop recording. A more creative approach is start recording when a hand is in the frame and stop recording otherwise. The issue with both of these is that a fluent ASL speaker would have to break their normal signing style in order to either start and stop the recording or to put their hands in and out of frame.

6.3 LLM API

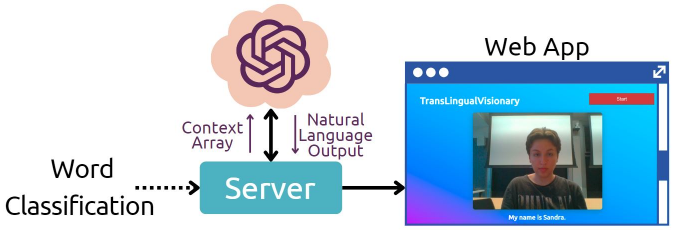

Figure 3: LLM. The output from the classification model is passed into OpenAI's API, where SLT takes place. The natural language English sentence that's generated is then passed to the web application that displays this text alongside the user's video.

The LLM utilizes OpenAI's GPT3.5 model in order to translate between ASL words and an English sentence. ASL grammar and sentence structure is heavily different than English, despite the similar vocabulary. As such, we decided to employ prompt engineering for GPT3.5 to generate an English sentence using the outputs of our classification model. After we get a sequence of comma-separated, translated words from the classification model, we can append that sequence to the prompt below:

You will receive a list of comma separated words directly transcribed from signed ASL speech. Your objective is to act as an ASL sign language interpreter and perform sign language translation. Sign language translation (SLT) translates between spoken and signed speech that use the same language, which tend to show high lexical similarity but low syntactic similarity. You should correctly interpret all the user's words in their given order to create full natural English sentences that maintain the user's intended meaning. Knowledge of ASL grammar and syntax rules should be used when constructing natural language English sentences. Parts of speech like copulas, articles, adverbs, pluralization, and tense markers should be inserted into the interpreted sentences when appropriate. Please return only the SLT sentence and no other text.

This gets passed into the GPT3.5 API, returning a translated English sentence. This sentence then gets placed at the bottom of a viewing screen, which is also presenting the landmarks of the human pose estimation model.

The LLM API architecture has also been attached to a webapp for further implementation, which was based on another ASL translator utilizing a different system architec-

ture [9]. The webapp utilizes Flask, a Python microframework for web app development, to allow users to start and stop recording while also viewing themselves. Additionally, given a sequence of ASL translated words, it can utilize the LLM API code to translate the sequence into an Enlgish sentence.. This can be tied the start and stop portion of our classification algorithm to allow users to generate translations for ASL signs.

7 TEST & VALIDATION

7.1 Latency Testing

The main goal of testing our latency is to meet the overall use case requirement of a three-second maximum delay. To test this, it suffices to measure the time it takes from right after the sign is finished to when the translated text is generated and ensure that not only the average time is below three seconds, but that the time three standard deviations above the average time is also within the three-second latency mark. Under the assumption that the distribution of the times follows a normal distribution, if the third standard deviation from the mean time is also underneath our use case threshold, we can also assume that 99.85% of our translation times over some arbitrary n number of runs will also be under three seconds. We determined this as a reasonable threshold since this would mean that only 15 runs out of 10 thousand runs will be above 3 seconds, which we believe holds to our use case requirements.

That said, we also need to verify our component latencies to reach this overall latency requirement. As we outlined in our design requirements, the component latencies chart out to the following:

- Image pre-processing: 5% $\approx$ 150 milliseconds
- FPGA Human Pose Detection: 10% $\approx$ 300 milliseconds
- Classification Model: 50% $\approx$ 1.5 seconds
- Prompt Generation and LLM: 30% $\approx$ 900 milliseconds
- Webapp Display: 5% $\approx$ 150 milliseconds

We would similarly test these to that of the overall latency test; we want to use a timer to calculate the time from data entering the component to data leaving said component, and make sure that the third standard deviation is underneath the unit component latency requirement. This virtually ensures that we do not have a bottleneck for any component.

We also want to test the latency between two different design choices that we were considering. The first option is our current design. The second is where we our Human Pose Estimation model on the Jetson or the computer alongside the RNN. As of right now, we believe that having the Human Pose Estimation model on the FPGA will result in a significant speed-up, but we want to verify that, so we want to compare the latency between the two pipeline architectures. If there is no significant speed-up, we also want to test our classification accuracy (which we will mention in the next section) to see if containing everything on the Jetson will result in a lower accuracy due to requiring smaller models.

We would also test the latency between using GPT3.5 and Llama2, which would involve simply determining the time it takes to send out and receive a response from either API using an arbitrary prompt. Running this over multiple prompts will allow us to do a chi-squared test to determine if there is a significant difference in latency times that we should consider when deciding on an LLM or not.

7.2 Accuracy Testing

Testing our classification models is a matter of accuracy. To achieve this, we want to split our data into training, validation, and testing sets. For this, we are currently looking at using a 70-20-10 split for our data. The 70% training split allows us to use the majority of our dataset for training, which boosts accuracy. The main choice comes with the 20% and 10% validation and testing splits. Having more validation splits allows us to focus more on improving our model's hyperparameters. Considering the large size of our dataset, a 10% testing split gives us enough data points overall to test our model's final generalization. This 10% testing set would be run through the HPE-Classification pipeline in order to get our inference accuracy. Using these data splits, we can calculate accuracy by evaluating the percentage of classified words to their true labels and then represent these accuracies using a precision matrix.

To make sure that we can optimize the parameters of our classification models, we are going to be using our aforementioned validation data. Since we are not directly training our model through reevaluating our model weights using this validation data, as we do with the training data, we can get a general idea of how our data is generalizing using this validation set. We can therefore use this to tune our hyperparameters to find parameters that produce the best possible accuracy for this validation data, which we are aiming for at least 90%. This value of 90% in validation accuracy allows us a buffer for our use case requirement of 85% for our testing classification. This testing set would be a separate set that we have not shown to our classification model giving us the best possible representation for how generalized our model is.

Additionally, we need to test the impact of distance. To do so, we are planning on creating our dataset where we sign various words at different distances. We want to make sure that the accuracy of word classification over these distances does not change significantly. To do this, we can simply calculate the accuracy per word per distance and utilize a chi-squared test to determine if there is a statistically significant difference between the accuracies at different distances.

Finally, as we mentioned before, we want to test the accuracy of our classification model when given different

HPE inputs. One of them with a hand-only detection input and the other with hand-and-face input. We will be calculating the verification accuracy on these as well given the same data to train to determine which model is better suited for translation.

As we mentioned earlier, we want to validate our words-to-sentence translation LLM component utilizing BLEU score, which is defined as the following equations [7]:

$$BLEU = \min(1, \exp(1 - \frac{\text{reference-length}}{\text{output-length}})(\prod_{i=1}^{4} \text{precision}_i)^{\frac{1}{4}}$$

where

$$\text{precision}_i = \frac{\sum_{\text{snt} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}} \min(m^i_{cand}, m^i_{ref})}{w^i_t = \sum_{\text{snt'} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}'} m^i_{cand}}$$

This score, in essence, tells us the geometric mean of the precision of our candidate translation corpus compared to the reference translation, taking into account n-gram matches up to length 4. By utilizing BLEU[7], we have an automated and universal methodology for evaluating the accuracy of our translation LLM, rather than solely relying on human evaluation and opinion. However, unlike for word classification where we are using a train-validate-test split, we are instead going to just be using a train-test split because there is no model to train - we are just modifying our prompt to get the best possible results out of our LLM. However, since the accuracy of our LLM is dependent on our classification model, we determined that for the classification validation set, we want to get a validation BLEU score[7] of around 45% and a testing BLEU score of about 40%. There is also a need to verify which LLM between GPT3.5 and Llama2 is more suited to our use case, and as such, we plan on calculating validation BLEU scores given the same overall prompt for each LLM. We can run a chi-squared test on this as well to determine if there is a significantly better option between either model to make our decision.

7.3 Results for Latency Test

As we can see in Figure 4, our overall latency is about 2.2 seconds, which is considerably faster than our use case requirement of 3 seconds. The main reason for this speedup is due to the latency of our human pose estimation model and our classification model being far lower than we expected. Our HPE model can receive an input and generate skeletal data in approximately 65 milliseconds. On the other hand, our ST-GCN model can classify a word in about 9 milliseconds. Both of these estimates are quite fast. The HPE model is this fast due to the fact that Mediapipe is built to be a robust system that can be deployed on edge or IoT devices, meaning it should be able to run faster on a computer. We also know that the structure of an ST-GCN is a convolutional neural network utilizing spatio-temporal graphs. Therefore, its architecture allows for efficient feedforward computation through techniques like sparse connectivity, weight sharing, parallelization, dimensionality reduction, and being optimized for grid-like data like video. This allows the ST-GCN to compute features from the skeletal data inputs very quickly during inference time.

Additionally, we can also note the frames per second we were able to achieve on both the computer and the FPGA. As aformentioned, we were unable to quantize, and thus accelerate, the HPE model n the DPGA, resulting in the approximately 3 frames per second that we achieved while running human pose estimation on it. However, we were able to find that using other HPE models, we can reach speeds of about 25 fps or greater, as using Xilinx's pretrained hourglass pose estimation model [17]. However this model uses far less datapoints than we need for either model, so we chose not to use it and stick with Mediapipe instead. As a result, wera it on a computer and was able to get a consisten 17 frames per second, which is far more usable for our use case.

7.4 Results for Accuracy Test

The main tests that we ran in terms of accuracy were in order to compare the SPOTER and ST-GCN models. As we can see below in Figure 5, we were able to reach higher validation accuracies with the SPOTER model than we were able to with the ST-GCN model. While normally this would indicate using the SPOTER model, the ST–GCN model had far etter inference numbers, as the SPOTER model get around 50%-55% on our own inference test set compared to the St-GCN's top-1 accuracy of 63.27%. While this may not met our use-case standard, our top-3 accuracy does, showing that the model is always almost right, but not necessarily completely right all the time. More specifically, the correct answer always ends up in the top 3 classifications, but not always as the top answer for classification. This could be because of the signs that we chose and the dimensionality of our HPE model inputs. A lot of the signs we chose were similar, such as *good*, *afternoon*, and *thank you* beign almost the same sign, resulting in overlap between our 10 classes. Additionally, our HPE model only gives us 2 dimensional landmarks, thus meaning that we are missing out on the third dimension of depth that could have helped in classifying these better, especially since an ST-GCN can somewhat understand depth due to its architecture. The high top-3 accuracy shows that the model is able to perceive the differences in the frame sequences and can generally distinguish them, but maybe not so much on a smaller scale when comparing similar signs.

Metric	Tests	Goal	Results
Recognition Rate	Calculate how often the model provides a sign classification when a user is signing. *	~95%	~98%
Word Classification Accuracy	Split data into training and validation sets and then calculate how often the model's output and the desired output is the same.	Training ~95% Validation ~85%	Training ~ 91.53% Validation ~ 82.05%
Inference Accuracy	Calculate accuracy of inference (how often the user's sign and the model's word is the same)	~80%	Top-1: ~63.27% Top-3: ~83.19%
BLEU Score	Calculate the BLEU score of the model's output sentence versus the desired output sentence	~40%	LLM: ~ 37.72% Overall: ~28.34%
Overall Latency	Run timer from beginning of HPE to classification output	~ 3 seconds	~ 2.2 seconds
Unit Latency	Measure latency via inter-component timestamps during live inferencing for various and signs	HPE: ~600 ms Classification: ~800ms	HPE: ~65 ms Classification: ~9 ms
FPS	Measure overall frames per second by calculating how long it takes a set of frames to go through the entirety of our pipeline.	Computer	FPGA
		~ 18 fps	Unaccelerated: ~ 3 fps Accelerated: ~ 25 fps

Figure 4: This table summarizes our primary results of the system in both accuracy and latency metrics

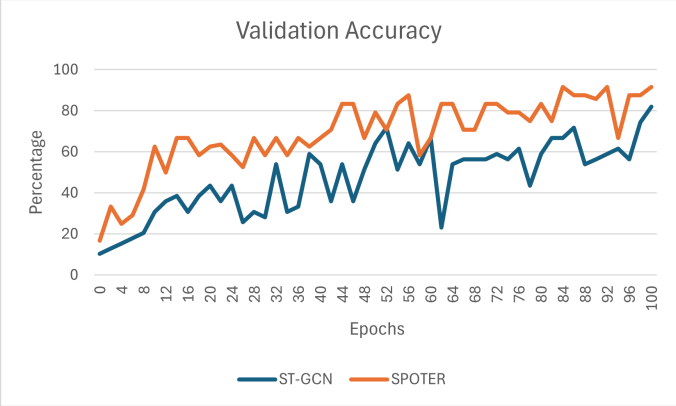

Figure 5: This graph details the difference of validation accuracy between our two classifications models: SPOTER and STGCN

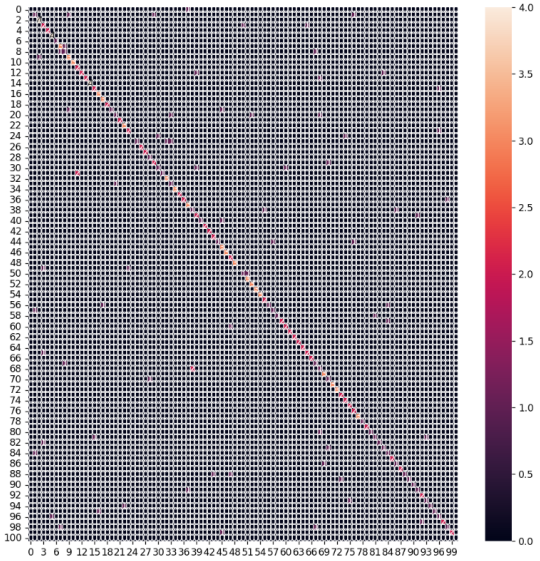

Figure 6: This figure illustrates the confusion matrix of 100 class SPOTER model

Another reason we chose to move away from the SPOTER model was that it got confused too often, as we can see in Figure 6. As we can see along the diagonal of the matrix, there are a lot of instances in which it did get confused, and a deeper dive into this showed us that the SPOTER model is very overconfident in its decision, leading to a subpar top-3 accuracy as well. While this could also be a similar sign issue, we believe it could also be how the points are normalized around the shoulders for the SPOTER model. Because of this, we have an issue where the 2 dimensional skeletons of signs that are around

the shoulders have similar frame structures, confusing the model further. We also believe it was a dataset issue with the WLASL dataset beng incomplete [6]. This could have also led to skewed dependencies within the model, resulting misclassification.

Taking a look at the BLEU score, we can see that the prompted LLM API on its own returns decent translations on its own during unit tests, almost reaching our BLEU score target. Not meeting our benchmark could be attributed to the generalization of modern LLMs, since their purpose is not strictly language translation. Being able to perform as well as it did is most probably due to the massive training set and model size of GPT3.5, but not finetuning the model for language translation has probably resulted in falling short of the desired BLEU score. However, our entire model does not achieve the same rate. This is most likely due to the top-1 accuracy issues of our model, which would be sending the wrong words to the LLM, resulting in sentence mistranslation. There are a good amount of times in which the LLM picks it up and is able to recover, but that is most likely due to the fact that the classification model has a limited number of classes which are all relatively topic-related.

As for classification distance, we were able to find that distance was not really an issue. Mediapipe is robust to the point in which it is able to still detect landmarks at a distance with minimal accuracy differential. The main source of variability came from the ST-GCN model, but even then, the accuracy differently was sub 5% as intended. This is most likely because the ST-GCN model is designed to be spatially aware, allowing it to maintain high levels of accuracy even when the subject is not in close proximity. The architecture of the model, which integrates spatial configuration and temporal dynamics, is adept at handling the variability in human movements. Since it leverages the connections between joints to predict activity rather than the joints themselves, it is less susceptible to distance-related accuracy issues. Furthermore, the robustness of the Mediapipe framework in detecting landmarks provides a stable input for the ST-GCN model, ensuring consistent performance across various distances.

8 PROJECT MANAGEMENT

8.1 Schedule

For this section please refer to our schedule labeled Fig. 8: Gantt Chart. Due to problems with quantization, we realized that FPGA was not a feasible option a few weeks after the Design Review. Since we had allocated enough slack time, Kavish had enough time to transition to developing the HPE model on the Jetson rather than the FPGA. Additionally, the testing and optimization of the classification model took longer than expected; however, with enough slack time initially allocated, we met our milestones before the deadline.

8.2 Team Member Responsibilities

Kavish Purani: Kavish was initially responsible for the development of the Human Pose Estimation model on the FPGA; however, after the change in design due to quantization difficulties, he transitioned to developing the Mediapipe model on the Jetson. After the HPE development, he was also responsible for the full integration of all the subsystems. His secondary responsibility was working with Neeraj to test and debug the classification model, and help him with implementing the integration with other submodules.

Neeraj Ramesh: Neeraj was primarily responsible for the development and testing of classification models. He first researched and trained the SPOTER model and later worked on developing the STGCN model. His secondary responsibilities included helping with the integration of the subsystems and the development of the Mediapipe model before FPGA and Jetson testing.

Sandra Serbu: Sandra was primarily responsible for developing the prompt generation code and interactions with the OpenAI API. Her secondary responsibilities included working with Kavish to integrate API call code and developing the web application that can overlay the camera stream with sentence outputs from the LLM.

8.3 Bill of Materials and Budget

Please refer to the Table 1 to look at the bill of materials. Nothing was bought and left unused. We didn't originally plan to purchase OpenAI API credits in our original system design but successful SLT was much more effective and simple for integration relative to Meta's open-source Llama2 LLM.

8.4 AWS Usage

This project did not use any AWS resources.

8.5 Risk Management

Some of the risks that we had initially anticipated were FPGA implementation of the HPE model and the accuracy of the classification model. We had planned that if the FPGA was not feasible for any reason, we would quikly switch to implementing the HPE model on the Jetson. This plan was very helpful when we realized that the Mediapipe model was not eaisly quantizable on the KV260 board. This did not impact our timeline and schedule by a large amount since we had planned for this in our slack time.

In the case of poor classification from our initial plan of RNN for both the GRU and LSTM, we planned to look into the feasibility of using a transformer and seeing if that architecture can give us a better response, considering that transformers have also been used for machine translation. This plan was also very helpful since our early preperation allowed us to pivot to the SPOTER (a transformer model) and intergate our project using that. It also allowed us

Table 1: Bill of Materials

Description	Model #	Manufacturer	Quantity	Source @	Cost
Jetson Nano	0022	Nvidia	1	inventory	$0.00
Logitech C920S (Camera)	-	Logitech	1	pre-owned	$0.00
Display (Monitor)	-	Dell	1	pre-owned	$0.00
OpenAI GPT3.5 API	-	OpenAI	1	Cloud	$2.00
					$2.00

enough slack time to notice new problems and also implement the STGCN model for our final demo.

Finally, when implementing the LLM SLT component of our design, there was risk involved with integrating and fine-tuning an open source LLM model that is only officially supported on very limited platforms. In the case of unsuccessful integration, we preliminarily worked with and tested OpenAI's free API components to ensure there was a simple and effective backup LLM model that could be accessed.

9 ETHICAL ISSUES

The underlying objective of our project serves to increase accessibility for the hearing impaired community. Systems that increase an individual's autonomy are powerful tools that broaden the scope of how we can meaningfully engage in the world around us. This reality shaped our use case as we took ethical considerations into account. We felt it's incredibly important for the users to see the correctness of their intended speech As a result, our output is displayed on screen in real-time so the user can see their communicated speech. Without a sanity check, a user could be rendered unable to meaningfully communicate if our system's translation doesn't maintain the user's intended meaning upon translation. Upon scaling a ASL-to-text system like TLV, there raises the risk of data security exposure if the translation isn't happening locally on a users device. There can be many sensitive contexts that users might choose to utilize TLV for communication and, if that sensitive information is necessarily passed over a network, the risk of being overheard must be taken into consideration. In our current implementation, TLV runs on local machines so there is little to no risk of a malicious actor overhearing sensitive information. As such, few data security have been taken to mitigate the risk of a malicious actor.

10 RELATED WORK

We have also looked at another experiment conducted by Nir David, Alexey Konev, and Jia Ying, in which they delve into various architectures that they use for translating ASL videos. This paper is what originally inspired us to use the WLASL dataset, as well as offered some overarching ideals for the general pipeline for translation. It also presented the issue with using raw images as inputs, resulting in their model training on extraneous unnecessary factors, and using HPE to counter that [4]. We are implementing our pipeline a bit differently, in that we are experimenting with variations of the pipelines that they used, their work provided us with a strong foundation to base our work.

As for overall models doing similar work, we know both the SPOTER and ST-GCN models and their associated papers work to classify videos into gestures, as we utilized both in our pipeline [19] [2]. As such, they hold a a lot of similarity in majority of their goals. The key distinction is that these works do not focus on a real time product and instead work on optimizing accuracy for set video inputs, thus strictly focusing on the model itself. That being said, we found somone named Simone Finelli try to do gesture to word real time translation [9]. However, we also add the LLM component on the end to get fully translated sentences, which neither Finalli or either of the classification models touch upon. We also differentiate from Finalli in our classification model itself, as he chose to use an I3D model, which does not necessarily rely on human pose estimates and takes in entire frames instead.

In addition to the demonstration by Peter Quinn, the research by Masaru Yamada on optimizing machine translation through prompt engineering provides useful insights for our project [18]. Yamada investigated how incorporating aspects like the purpose of the translation and target audience into prompts for ChatGPT can improve translation quality. His findings showed that adding this contextual information helps guide the model to produce more natural and human-sounding translations suited for the intended use case. This approach of using prompts to specify high-level goals, constraints, and users aligns well with our aim of generating accurate and natural translations for ASL. As we develop prompts for our translation models, Yamada's techniques on translation prompt engineering can inform effective ways to frame the task, provide examples, and indicate the target output style. His work demonstrates the power of prompts to direct large language models, which we can potentially leverage for specialized ASL translation.

11 SUMMARY

Our system utilized a combination of Human Pose Estimation, Classification model, and Large Language Model to perform real-time American Sign Language to English Language Translation. All of our subsystems met our initial design requirements and goals in terms of accuracy and

latency. However, our final integrated product struggled to completely meet the use case requirement. More specifically, due to problems with determining the accurate batch-size to capture each sign individually and lack of 3-D landmarks, we had relatively high confusion between different signs. This resulted in many similar classes being misclassified as the same sign during the inference stage of the system. If we had more time, we would work on improving the classification model by training on a large dataset with 3-Dimensional landmarks. In order to do so, we would build our own model based-off the STGCN model that we implemented above.

11.1 Future work

In addition to the listing above, our further work on this project would prioritize incorporating MediaPipe's face mesh landmarks into our classification algorithm to better inform the predicted speech of the user through their facial expression in tandem with their signs. Due to time and technical limitations, our final system is less extensive than we'd originally hoped despite the group's background knowledge. We would also build our own human pose estimation model which is quantizable on the FPGA to further accelerate the system. This would allow use to run the two models (pose estimation and classification) concurrently and overcome the latency bottleneck.

11.2 Lessons Learned

One of the big lessons we learned was that it is important to leave enough slack time when planning the project, since it is inevitable that the tasks will take longer than you expect them to. Additionally, successfully training and implementing a modified ML classification model posed more of a challenge than originally expected. There were a lot of unexpected variables that affected our models' results, which are difficult to anticipate early on. When working with the FPGA - in particular developing models on the Deep Processing Units using Vitis AI - it is important to check if your model is quantizable. Depending on the FPGA, the amount and type of models that the DPU can support are often quite limited.

Glossary of Acronyms

- ASL - American Sign Language
- BLEU - Bilingual Evaluation Understudy
- CNN - Convolutional Neural Network
- HOH - Hard of Hearing
- HPE - Human Pose Estimation
- LLM - Large Language Model
- NLP - Natural Language Processing
- RNN - Recurrent Neural Network
- RTSP - Real-Time Streaming Protocol
- SLT - Sign Language Translation
- SPOTER - Sign Pose-based Transformer
- STGCN - Spatio-Temporal Graphical Convolutional Networks
- SUS - System Usability Scale
- TLV - TransLingualVisionary (this project!)

References

[1] Lu Alex. *ASL Citizen Code*. `https://github.com/microsoft/ASL-citizen-code`. 2023.

[2] Matyáš Boháček and Marek Hrúz. "Sign Pose-Based Transformer for Word-Level Sign Language Recognition". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*. 2022, pp. 182–191.

[3] Diana Cheung. *Meta Llama 2 vs. OpenAI GPT-4*. 2023. URL: `https://medium.com/@meetdianacheung/meta-llama-2-vs-openai-gpt-4-785589efe15e#:~:text=GPT%2D4%20has%20higher%20multi`.

[4] Nir David. *Sign Language Video Translator*. 2022. URL: `https://medium.com/@oronird/sign-language-video-translator-8bc80480fbf5`.

[5] Aashaka Desai et al. "ASL Citizen: A Community-Sourced Dataset for Advancing Isolated Sign Language Recognition". In: *arXiv preprint arXiv:2304.05934* (2023).

[6] Dongxu. *WLASL: A large-scale dataset for Word-Level American Sign Language (WACV 20' Best Paper Honourable Mention)*. 2022. URL: `https://github.com/dxli94/WLASL`.

[7] *Evaluating Models*. URL: `https://cloud.google.com/translate/automl/docs/evaluate`.

[8] Farhad Fallahlalehzari. *FPGA vs GPU for Machine Learning Applications: Which one is better? - Blog - Company - Aldec*. URL: `https://www.aldec.com/en/company/blog/167--fpgas-vs-gpus-for-machine-learning-applications-which-one-is-better`.

[9] Simone Finelli. *simonefinelli/ASL-Real-time-Recognition*. 2024. URL: `https://github.com/simonefinelli/ASL-Real-time-Recognition`.

[10] Pablo Grill et al. *SpoterEmbeddings: Create embeddings from sign pose videos using Transformers*. `https://github.com/xmartlabs/spoter-embeddings`. 2023.

[11] E. J. Hwang J. Kim H. Lee J. H. Kim and J. C. Park. *Leveraging Large Language Models With Vocabulary Sharing For Sign Language Translation*. 2023. URL: https://ieeexplore.ieee.org/document/10193533.

[12] *Hand Landmarks Detection Guide*. 2023. URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker.

[13] *Holistic Landmarks Detection Task Guide*. 2023. URL: https://developers.google.com/mediapipe/solutions/vision/holistic_landmarker.

[14] *How fast should I sign?* URL: https://www.lifeprint.com/asl101/topics/how-fast-should-i-sign.htm#:~:text=So%2C%20yes%2C%20you%20need%20to.

[15] Mediapipe. *google/mediapipe*. 2023. URL: https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md.

[16] Kien Nguyen Phan et al. "Assessing Bicep Curl Exercises by Human Pose Application: A Preliminary Study". In: Mar. 2023, pp. 581–589. ISBN: 978-3-031-27523-4. DOI: 10.1007/978-3-031-27524-1_55.

[17] Michael Schmid. *Hardware Accelerated Human Pose Estimation — KV260, IAS Cam*. 2022. URL: https://www.hackster.io/michi_michi/hardware-accelerated-human-pose-estimation-kv260-ias-cam-d5ebb9.

[18] Masaru Yamada. "Optimizing Machine Translation through Prompt Engineering: An Investigation into ChatGPT's Customizability". In: 2 (2023), 195–204. URL: https://arxiv.org/ftp/arxiv/papers/2308/2308.01391.pdf.

[19] Sijie Yan, Yuanjun Xiong, and Dahua Lin. *Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition*. 2018. arXiv: 1801.07455 [cs.CV].

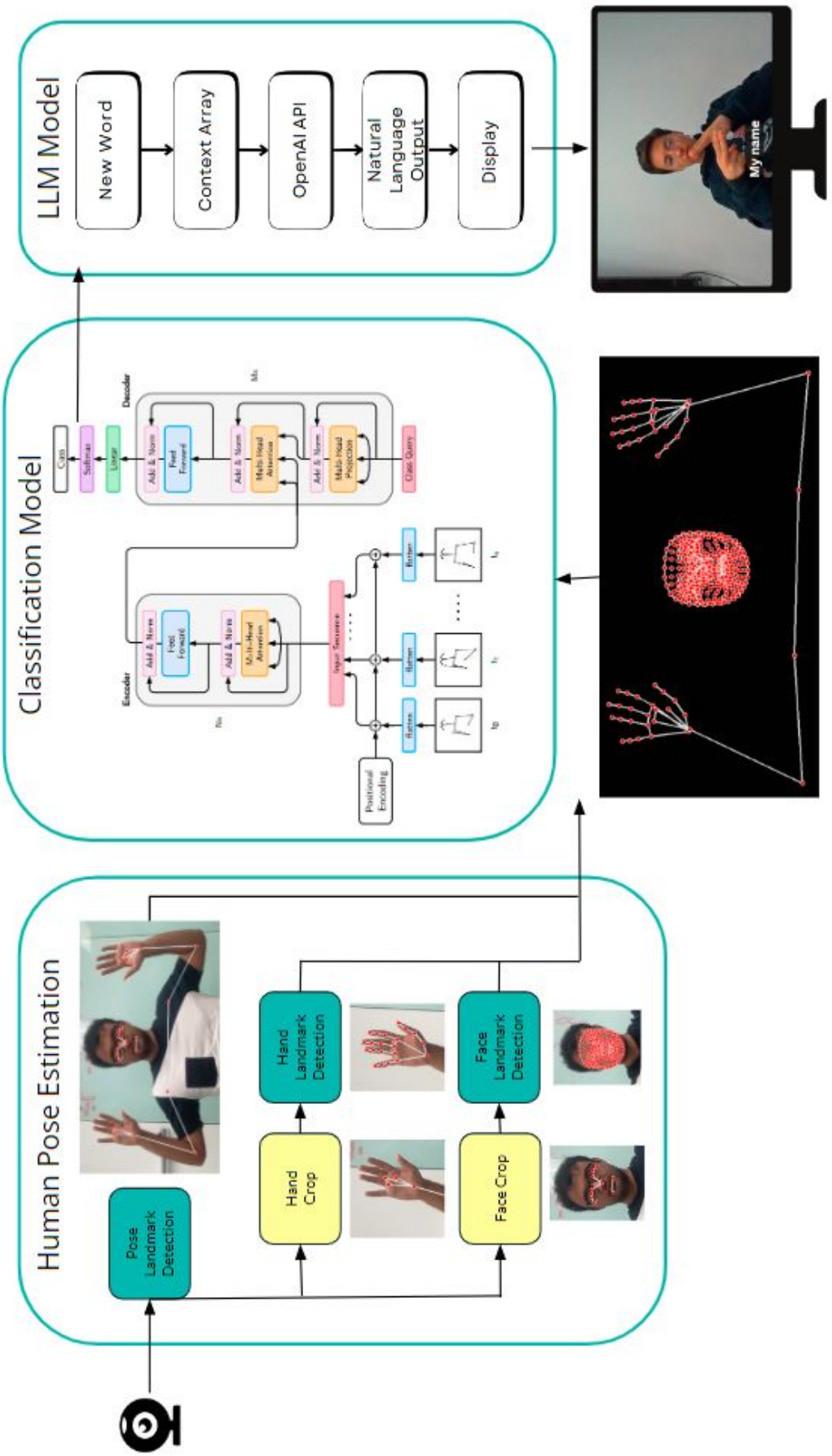

Figure 7: Full System Block Diagram

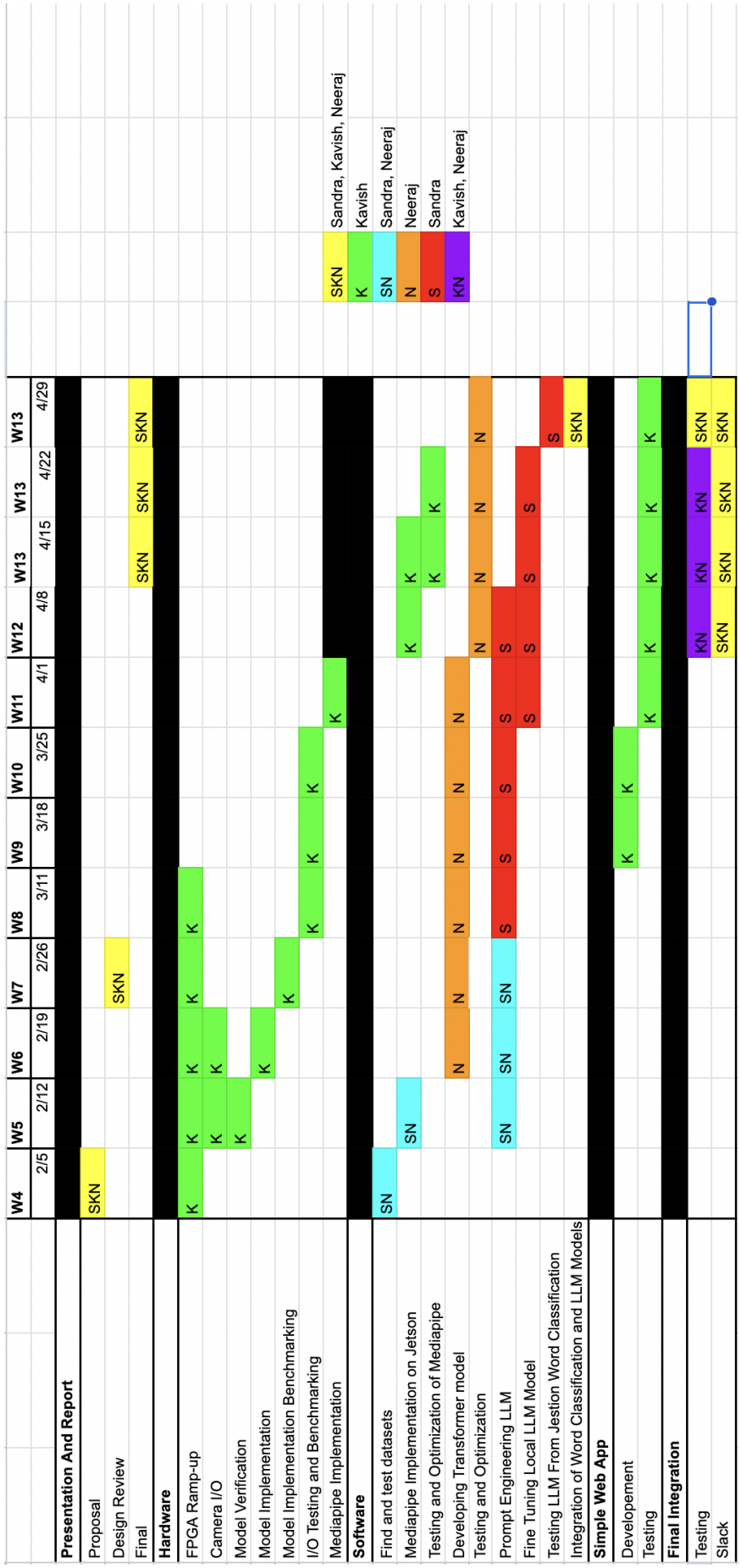

Figure 8: Gantt Chart