TransLingualVisionary

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

***Abstract*– A system capable of detecting, interpreting, and displaying live ASL speech to English text. Using a camera to capture a live video stream, an FPGA to accelerate MediaPipe's human pose estimation, a Jetson Nano to classify words, and an LLM to generate english text, we will develop a novel platform that enables hearing impaired ASL users to better participate in virtual streaming environments. TransLingualVisionary will contribute to developing sign language processing systems using machine recognition and translation.**

***Index Terms*—Accessibility, ASL, FPGA, Gesture Recognition, LLM, NVIDIA Jetson Nano, MediaPipe, RNN, Video Processing**

1 INTRODUCTION

In today's rapidly advancing digital world, the necessity to bridge communication gaps has never been more critical, especially for communities that face inherent challenges in standard communication methods. Our team presents a pioneering solution aimed at dismantling these barriers for the hearing-impaired community through the development of "TransLingualVisionary" (TLV), a real-time American Sign Language (ASL) to English text translation platform.

The inspiration behind this project stems from the recognition of the difficulties faced by deaf or hard of hearing (HOH) individuals in participating in live digital environments, such as online meetings and live streams. The lack of widespread understanding and accessibility to ASL interpretation exacerbates these challenges, often necessitating reliance on translators for communication.

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 instantly transcribe a live video feed into written English text. A Kria KV260 FPGA edge device detects ASL from a video feed input and passes detected words to an AI model, hosted on a Jetson Nano, that will convert ASL-syntax sentences into natural English text. The output text will then be displayed on a web application, enabling efficient two-way communication.

Any deaf or hard-of-hearing individual who prefers communicating via ASL but interacts with many non-ASL users virtually regularly could benefit from TLV. We hope to mitigate the need for human translator assistance often required to communicate in visual/audio online environments for the hearing impaired.

2 USE-CASE REQUIREMENTS

The goal of this project is twofold: to increase accessibility in community involvement by expanding the range of virtual communication for ASL users and to reduce the digital isolation felt by many within the 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 [9], 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[5] 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.
- User Interface – The display must be accessible and simple to use. The user interface should display the output text with accurate overlay of video and translated text. The translation must be easily and clearly viewable on the web application, which will be measured through more qualitative methods.

3 ARCHITECTURE AND PRINCIPLE OF OPERATION

Physical design components include a Logitech C920S camera, a Kria KV260 FPGA, and an NVIDIA Jetson Nano Developer Kit. Overall system information flow can be easily seen in Figure 1, which displays which components interact within the system. The system architecture integrates three main computational modules: the FPGA, the Jetson, and the LLM text generator.

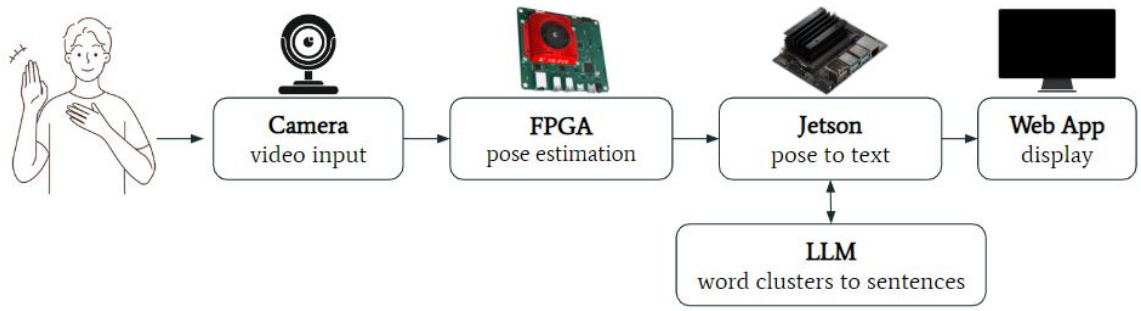

Figure 1: System diagram. Each component and its overarching tasks display the system's flow of information. User input is captured by an external camera that's then sent to the FPGA for human pose estimation. HPE vectors are sent to the Jetson Nano to generate English text. Captured video and generated text are then sent to the application display to be shown to the user.

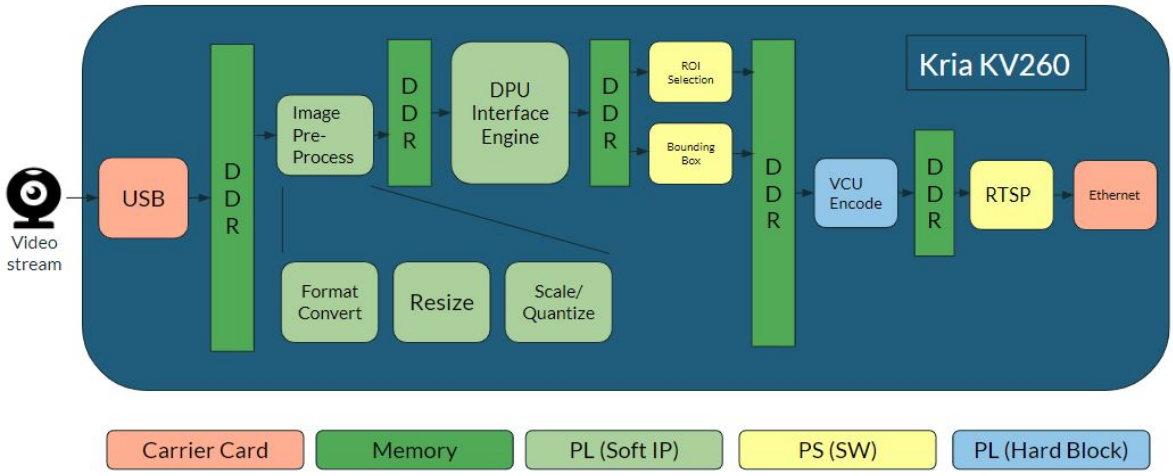

Figure 2: Zoom in of Kria KV260 FPGA block diagram. Displayed is the human pose estimation process running on the Kria KV260 FPGA. The video input stream is loaded to memory through a USB peripheral port. Programmable Logic (PL), reconfigurable logic, will be used to accelerate computation-intensive workloads. Processing Systems (PS) handle general-purpose computing tasks.

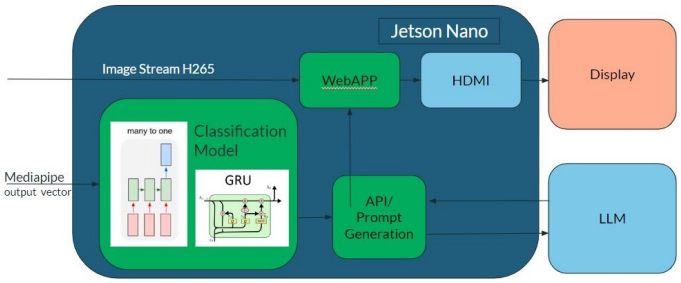

Figure 3: Zoom in of Jetson Nano block diagram. The output from the HPE model is passed into the Jetson, where our RNN classification model will generate a cluster of words. This will get passed to an LLM API to create a proper English sentence, which then gets displayed through our web app on a given display.

For the entire system to operate correctly, the camera will capture and send a live video stream of a user's upper body to the FPGA via USB port to accelerate MediaPipe's human pose estimation (HPE) models. HPE gesture and expression vectors will be sent to the Jetson Nano to perform word classification. The Jeston Nano will also deploy an API to run a RNN model on grammatically ASL word clusters and translate them to full english sentences. The Jetson will then output the video feed and generated text to a virtual web application for display.

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. ASL uses many non-manual markers to communicate aside from the hands, like facial expressions, body language, and gesture movement. The average English sentence is between 15–20 words long and, since ASL uses non-manual markers to express many words or phrases, 3 seconds would provide enough time to group words into approximately 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[5] 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 seperate components:

- ∼95% sign recognition rate. That is, the FPGA 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.
- Recognize 2000 signed words at ∼85% accuracy. The Jetson's word classification RNN must be able to correctly classify all 2000 words included in the training dataset with about 80% accuracy. Classifications will, ideally, 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[5] score of ∼40%. Given syntactically ASL word clusters, the LLM text generation must produce a natural english sentence with a BLEU[5] 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 FPGA's pose estimation model should be able to pass information that is equally useful for the word classification model on the Jetson Nano.

4.4 User Interface

Ease of use and accessibility of text is a qualitative metric that will be quantitatively measured through user feedback surveys, as described in Section 7.4. To achieve this, our display must receive a minimum SUS score[13] of 80% when gauging display accessibility and ease of use.

5 DESIGN TRADE STUDIES

5.1 Jetson v. FPGA for Human Pose Estimation

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 [6]. 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 RNN as well. As such, running human pose estimation on the Jetson might mean that we would have to reduce the size of our RNN, decreasing the number of parameters that we have and risking lower accuracy numbers. Being able to host human pose estimation on the FPGA gives us more flexibility with the size of the RNN model, possibly giving us a higher accuracy.

5.2 Mediapipe v. 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 de-

vices, 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 [11]. 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.

As mentioned, Mediapipe also offers numerous options for detection models, namely in what exactly we want to landmark. As of right now, we are looking through the difference between using a hand-only estimation model [7] and a hand-and-face estimation model (inspired by the holistic model given by Mediapipe) [8]. 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 RNN. 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. On the other hand, the hand-face model can capture these details and the position of the face gives us a relative body position, which could lead to higher classification accuracy, along with capturing important facial expressions. We plan to test this as well and lean with the HPE landmarks that give us the best verification accuracy.

5.3 RNN Architecture

Our RNN has a few design considerations in terms of overall architecture.

5.3.1 Input to Output Mapping

RNNs used in sequence transduction tasks like video-to-text in this case can model different input-output relationships. One-to-one mapping means every input frame is translated to an output word. Many-to-one compresses many frames down into one output. One-to-many is the reverse, generating descriptions from sparse inputs. Finally, many-to-many flexibly map arbitrary video segments to phrases.

For our use case of ASL translation, many-to-one sequence mapping proves most effective. This is because signs and sentences have differing temporal structures. Individual signs convey meaning over hundreds of milliseconds, containing motion and transition phases. In contrast, words last only briefly. Many-to-one networks handle this asymmetry. They aggregate information over the full sign before predicting an output, encapsulating the complete semantics. One-to-one mapping fails here as signs get fragmented across outputs. One-to-many risks incomplete context when attempting to describe mid-sign frames. Many-to-many provides flexibility but requires large data and model capacity. For accessible and robust ASL interfaces, many-to-one with encoded memory of the full sign performs best despite its simplicity. The consolidation of visual evidence informs more accurate word production without making fragile assumptions.

5.3.2 GRU vs LSTM

Recurrent neural network architectures like gated recurrent units (GRUs) and long short-term memory networks (LSTMs) offer complementary strengths and weaknesses when applied to sequence transduction problems such as translating sign language video into text. Both model types process input streams incrementally, encoding state and context about previous elements to inform future outputs. This allows the capture of the temporal structure critical for mapping motions into words.

A key difference lies in the complexity of internal memory modeling. LSTMs utilize more moving pieces - various gates control input, output, and forgetting alongside short and long-term cell states to preserve information over extended sequences. This mechanical depth empowers the representation of long-range dependencies often present in real-world sequence data. GRUs take a more minimalist approach, reducing the machinery down to a single update gate and context vector. This consolidated design still handles short-term dependencies but may falter at multi-phase sequences exceeding the capacity of a single-state vector. However, GRUs counteract with better generalization from fewer parameters, more efficient training, and overall faster inference times.

As such we are learning to use a GRU architecture for now since it is a lighter model. However, we are going to be testing both due to the similarity in developing them. There is a chance that the latency difference between the hidden state structures is negligible, we will most likely use an LSTM model instead due to its better long-term memory.

5.3.3 Tensorflow vs Pytorch

There are two predominant libraries that we have experience with when creating neural networks: TensorFlow and PyTorch. Both provide rich tooling for building and training neural network architectures. However, for implementing recurrent neural networks (RNNs), TensorFlow offers some advantages over PyTorch that motivated our choice.

A key benefit of TensorFlow is the high-level API which simplifies the configuration of complex RNN architectures [10]. Layers like LSTM and GRU can be cleanly stacked

with fully connected and output layers through simple function calls rather than manual tensor operations. This helps quick prototyping of different RNN arrangements. Another useful aspect is that distributed training of graph-based models like RNNs can be set up in TensorFlow. We utilize this during hyperparameter searches to parallelize model training. Finally, accessing low-level deployment optimizations in TensorFlow like quantization-aware training, pruning, and other model compression techniques is more straightforward. These allow our team to readily translate trained RNNs into efficient production implementations. Additionally, through Tensorboard, we can easily visualize the training of our model, which can also help with model debugging as well [10]. This combination of features makes it more scalable for our purposes and gives us more freedom to experiment with different RNN architectures.

5.3.4 MUSE-RNN

We are currently also exploring utilizing MUSE-RNN, an architecture that combines LSTM-based encoders and decoders with an additional set of multilinearly modulated latent states. This allows for the modeling of complex multimodal distributions in sequencing tasks, helping address uncertainty and noise. The multilinearly modulated units can capture subtle visual nuances and variations across repeated signs or gestures. Additionally, the latent states can account for inherent ambiguities in isolated sequences that rely heavily on contextual information [2]. By sampling from these latent variables, MUSE-RNN can represent multiple potential translations to mitigate risks during decoding. We believe this stochastic three-way architecture provides useful capabilities for the challenges of translating sign language video, where both subtleties in motion and ambiguity are present. The modular components also facilitate conditioning the sequential translation on other modalities beyond just the signing motions.

5.4 OpenAI GPT4 v. Meta Llama2

For our LLM, we were considering two different models - OpenAI's GPT4 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 GPT4, meaning that there is a chance that it is faster, but GPT4 does have higher performance and accuracy. She also mentions that the decision is heavily influenced by the risk tolerance of our use case requirements [1]. Since there might not be a large difference in latency between the models, we are leaning towards the GPT4 model, as there is more research backing its higher performance in comparison to Llama2. We intend to test both options in both accuracy and latency to verify which model better for our use case.

5.5 Ultra96 v. KV260

For our FPGA, we were considering one of two boards: the Ultra96 and the KV260.

The Ultra96 FPGA development board offers an affordable and accessible option for exploring FPGA-accelerated applications, with its Zynq UltraScale+ MPSoC. However, for more advanced computer vision pipelines, the KV260's dedicated AI engine and additional resources make it better suited. Specifically, the KV260 features a built-in Deep Processing Unit (DPU) - an AI accelerator optimized for CNN inference. This eliminates the need to develop custom CNN accelerators in programmable logic, streamlining deployment. The KV260 also provides higher memory bandwidth with its HBM2 stacks as well as faster communication channels like 100G Ethernet. These facilitate fast streaming of high-resolution image and video feeds.

In contrast, the Ultra96's more limited DDR memory and bandwidth can cause data movement bottlenecks for CV workloads. While the Ultra96's ease of development with IP libraries and lower cost are advantages, complex vision pipelines will be constrained in terms of performance versus the KV260. For local training and prototyping, the Ultra96 offers a usable platform. But for high-resolution, low-latency video analytics, the KV260's DPU, memory, and communication resources provide uncompromised acceleration without the development overhead. The KV260 is therefore better positioned for demanding computer vision applications - enabling rapid prototyping while also supporting seamless deployment thanks to its production-ready capabilities.

6 SYSTEM IMPLEMENTATION

As mentioned before there are three main computational modules within the project: the FPGA, the Jetson, and the Web App. The FPGA will be used to process the incoming video stream from the camera and run a human pose estimation model (MediaPipe) and send the outputs to the Jetson. The Jetson will run the classification model (RNN) and use GPT4 API with prompt generation to finalize the sentence structure. The Webapp will host a viewing application which is responsible for overlaying the video stream and text output and displaying them on the viewing platform.

6.1 FPGA

6.1.1 Carrier Card and Hard Blocks

The Kria KV260 FPGA comes with a built-in carrier card which has a working implementation of the USB and Ethernet interface. This allows us to connect the Logitech C920S camera through the USB port and capture the video stream directly into the DDR memory. Similarly, the encoded video stream and MediaPipe output vectors can be easily transmitted to the Jetson using the Ethernet port. There also exists built-in hard blocks like VCU Encoder

which can be utilized to encode the output video stream into a H265 video compression stream for fast and efficient data transfer. We will try to leverage the built-in capabilities as much as possible before attempting to modify the hard blocks for alternate use.

6.1.2 Programmable Logic (Soft IP)

Programmable logic is the logic that is directly implemented on the FPGA fabric. There are two main sections of the pipeline that rely on this: Image pre-processing and the CNN model running on the Data Processing Unit (DPU). We will primarily rely on Vitis AI tool to generate HLS for these two purposes. More specifically, there exists a Vitis model zoo which we can use to build the backbone of our model and write custom HLS for specific steps when necessary. Most common pre-processing step logic is available through Xilinx libraries (ex. Format Convert, Resize, Scaling etc.), thus we will mainly be focused on stitching these blocks together with deeper pipe-lining to achieve better performance.

6.1.3 Programmable Software

Programmable software is the logic that runs on the ARM Cortex-A53 processor, which we will boot with Ubuntu to have an easy development experience with a familiar Linux environment. This part of the logic is responsible for overlaying the outputs of MediaPipe with the images in the memory and passing them into the VCU Encoder to create a H265 stream. It then collects the H265 stream and runs the RTSP protocol to send the data through the Ethernet. These pieces of software will be written in C++ to maintain high throughput, but we will rely on the OpenCV library for image processing functions. As of right now, we have developed a testing script in Python utilizing both OpenCV and Mediapipe, inspired by that of Owen Talmo [14], which can generate landmarks in real-time to eventually be passed into the VCU Encoder.

6.2 Jetson

6.2.1 Classification Model

Our classification model will be a many-to-one GRU-RNN. We will be using the TenserFlow framework to develop this RNN. It will be trained on the outputs from the MediaPipe model, from which we will run the images and video clips from the WLASL dataset, which is a labeled dataset of videos containing a total of about 2000 common words in ASL [4].

6.2.2 Prompt Generation

The prompt generation module is a key component responsible for interfacing with the GPT-4 API to enhance the translation outputs. Specifically, it takes the raw translated words produced by the classification model and formulates prompt requests to be sent to the GPT-4 API. These prompts will provide context about the translation task and specify that the goal is to output an English-corrected version of the sentence. The prompt generation handles queuing up all of these prompt requests and throttling them appropriately to stay within API limits. As results start coming back from the powerful GPT-4 model, it collects them, stores them temporarily, and outputs the enhanced translations to the main web application controller and display.

6.2.3 Web Application

The web application's primary responsibility is to take the final output from the prompt generation module and overlay the text onto the video stream from the FPGA. It needs perform this process in a real-time environment and deal with any latency delays or mismatches between the video stream and the generated result. For testing purposes we will rely on OpenCV's image processing functions to easily view the outputs and debug the system. During this time, we will rely on the HDMI interface to view the image stream on a display. Eventually, for better user experience, we will switch to modern frameworks like React for our final product.

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
- RNN Classification: 50% $\approx$ 1.5 seconds
- Prompt Generation and LLM: 30% $\approx$ 900 milliseconds

- Webapp Display: 5% ≈ 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 itself 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 GPT4 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. 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 [5]:

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

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[5], 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[5] of around 45% and a testing BLEU score of about 40%. There is also a need to verify which LLM between GPT4 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 Hardware Testing

The main hardware test that we are going to conduct is the correctness of optimized operations that would apply to our Human Pose Estimation model. Since a Human POSE

estimation is a type of CNN, there is a lot of operation optimization that can be done to ensure that running HPE on an FPGA will give us a significant speed up. Validating correctness for this would just come down to comparing the output of each optimized operation with the output from a CPU for multiple different possible values. We need virtually 100% accuracy here, as we cannot afford to be miscalculating values when trying to optimize the necessary operations.

7.4 Website and User Interface Testing

The System Usability Scale is a staple universal method for evaluating the usability of a web app via surveying users. The average SUS score that marks above-average usability for a web app is 68% [13]. The scope of this project mainly focuses on the translation of ASL to text, as this is a product we intend to tie into APIs that we do not have access to yet, such as Zoom or Skype. As such, we want to spend more time optimizing our translation pipeline and want our web app to be usable to the extent that it works well for users in an accessible and usable manner, but not at the top of our priorities. Hence, we are targeting around 80% for our SUS score, which we will obtain by asking those who do know ASL to test our product and ask them to complete the System Usability Scale survey.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule is shown in Fig. 4.

8.2 Team Member Responsibilities

In terms of workload, Kavish is responsible for the FPGA development, Neeraj is responsible for Prompt generation and LLM communication, and Sandra is responsible for building the Web Application. All three team members will work on building and helping with the RNN model since that is the core of our project. Finally, all team members will come together and test the integration of individual components into the final product.

8.3 Bill of Materials and Budget

Please refer to the Table 1 to look at the bill of materials.

8.4 Risk Mitigation Plans

The main plan to mitigate a lot of our risks would be in the form of completing tests as soon as we can. The first big risk is in the FPGA vs Jetson choice. We want to make sure that the FPGA is feasible as soon as possible because, if not, we would have to pivot to the Jetson regardless, which would impact our RNN if we pivot too late.

Additionally, there is also a risk in whether our classification model will detect intricacies in hand gestures if the user is too far away, since for a hand-only pose estimation model, the landmark positions are relative to the image frame rather than relative to the person signing. This is something we also have to test, as we mentioned earlier, and have a pivot accounted for, being the hand-face estimation model.

Finally, in the case of poor classification from the RNN for both the GRU and LSTM, we are also looking 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 is a pivot we need to make early if necessary, so we are in the process of developing our RNN to test as soon as possible.

9 RELATED WORK

As seen in the Peter Quinn demonstration of implementing human pose estimation (including MediaPipe) models on the Kria KV260, his methods and steps are very similar to what we aim to achieve [12]. His work provided a solid foundation for our development as we aim to replicate his efforts and further improve our performance to meet our desired goals.

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 [3]. 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.

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 [15]. 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.

Table 1: Bill of Materials

Description	Model #	Manufacturer	Quantity	Source @	Cost
KRIA KV260 Vision AI Kit (FPGA)	KV260	AMD (Xilinx)	1	pre-owned	$0.00
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 GPT4 LLM	-	OpenAI	1	pre-owned	$0.00
Meta Llama2 LLM	-	Meta	1	pre-owned	$0.00
					$0.00

10 SUMMARY

Our design utilizes a Logitech C920S camera, a Kria KV260 FPGA, an NVIDIA Jetson Nano Developer Kit, and an off-site LLM to implement a live ASL-to-text system in virtual environments. Our real-time ASL translation project has the potential to profoundly impact accessibility for the hearing-impaired community in virtual spaces. By enabling effective communication across non-fluent hearing individuals and hearing-impaired ASL users in popular digital environments like video calls, live streams, and conferences, we can promote fuller inclusion to a wider array of individuals.

Risk mitigation may provide challenges as we progress in our system's development. In hosting and accelerating HPE on the FPGA, we're aware there may be unforeseen hurdles that block or limit the scope of the FPGA's abilities. Although we're confident in the selected FPGA's ability to accomplish effective HPE, if any insurmountable issues were to come up further down the line we would plan to remove the FPGA from our system altogether. In this case, HPE would need to run on the Jetson Nano in addition to its current tasks. This would substantially shift the structure and capacity of our current system plan and the Jetson's ability to perform all its required tasks. The FPGA has the potential to significantly improve system latency and space flexibility for Jetson Nano's RNN model but isn't integrally necessary to accomplish our project goals.

TransLingualVisionary seeks to increase accessibility to digital environments for those who are hard of hearing to promote an increased sense of community within the growing online space.

Glossary of Acronyms

- ASL - American Sign Language
- BLEU - Bilingual Evaluation Understudy
- DPU - Deep Processing Unit
- FPGA - Field Programmable Gate Array
- GRU - Gated Recurrent Unit
- HOH - Hard of Hearing
- HPE - Human Pose Estimation
- LLM - Large Language Model
- LSTM - Long Short-Term Memory
- RNN - Recurrent Neural Network
- RTSP - Real-Time Streaming Protocol
- SUS - System Usability Scale

References

[1] 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.

[2] Monidipa Das et al. "MUSE-RNN: A Multilayer Self-Evolving Recurrent Neural Network for Data Stream Classification". In: *2019 IEEE International Conference on Data Mining (ICDM)* (2019). DOI: https://doi.org/10.1109/icdm.2019.00021.

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

[4] 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.

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

[6] 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.

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

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

[9] *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.

[10] Vihar Kurama. *Pytorch vs. Tensorflow: Deep Learning Frameworks 2021 — Built In*. URL: https://builtin.com/data-science/pytorch-vs-tensorflow.

[11] 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.

[12] Peter Quinn. *PeterQuinn396/KV260-Pose-Commands*. 2024. URL: https://github.com/PeterQuinn396/KV260-Pose-Commands.

[13] *System Usability Scale (SUS)*. 2019. URL: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

[14] Owen Talmo. *OwenTalmo/finger-counter*. 2023. URL: https://github.com/OwenTalmo/finger-counter/tree/master.

[15] 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.

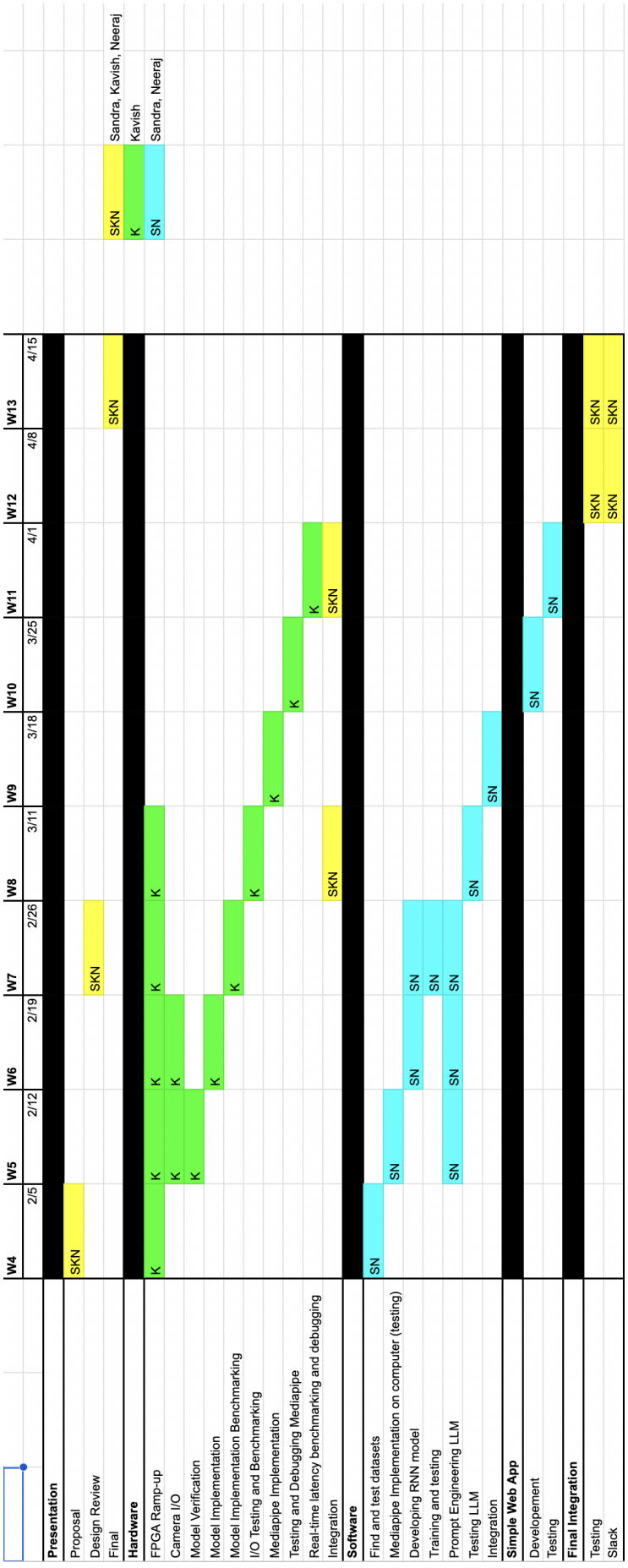

Figure 4: Gantt Chart