Sign Language Interpretation using Deep Learning

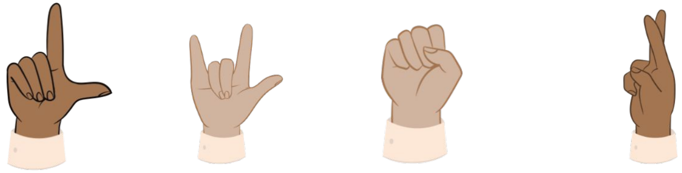
Data 255 - Deep Learning Project

San Jose State University

Dr. Taehee Jeong

Team 5

- Aparna Bharathi Suresh
- Aryama Ray
- Shravani Dattaram Gawade
- Sujata Deepraj joshi



Objective

- The Project aims to build an American Sign Language(ASL) Interpreter system using deep learning.
- Used Google's Isolated Sign Language Recognition dataset for training.
- Extract hand & facial landmarks using Google MediaPipe
- Our Goal is to develop a real-time ASL interpreter (Sign2GLoss2Text) to support individuals by utilizing deep learning to recognize hand gestures & convert them into text sentence

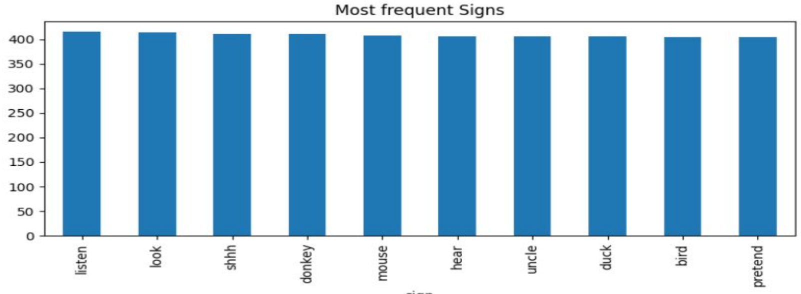
Google Isolated Sign Language Recognition Dataset (GISLR)

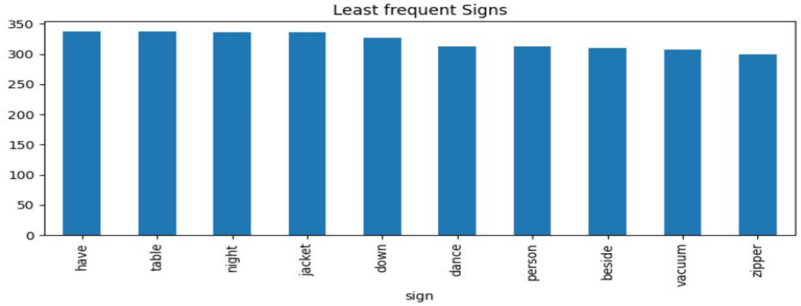
- CSV file (train.csv) that lists sign language labels
- JSON file that maps sign names (like "hello") to numbers (like 23)
- Landmark files (.parquet files) — these are like video recordings but instead of pixels, they have positions (x, y, z) of points on the hand, face, body in each frame

```
participants = 21
participant 16069 includes 4848 sequences
participant 18796 includes 3502 sequences
participant 2044 includes 4810 sequences
participant 22343 includes 4677 sequences
participant 25571 includes 3865 sequences
participant 26734 includes 4841 sequences
participant 27610 includes 4275 sequences
participant 28656 includes 4563 sequences
participant 29302 includes 4722 sequences
participant 30680 includes 3338 sequences
participant 32319 includes 4753 sequences
participant 34503 includes 4545 sequences
participant 36257 includes 4896 sequences
participant 37055 includes 4648 sequences
participant 37779 includes 4782 sequences
participant 4718 includes 3499 sequences
participant 49445 includes 4968 sequences
participant 53618 includes 4656 sequences
participant 55372 includes 4826 sequences
participant 61333 includes 4900 sequences
participant 62590 includes 4563 sequences
```

	path	participant_id	sequence_id	sign
0	train_landmark_files/26734/1000035562.parquet	26734	1000035562	blow
1	train_landmark_files/28656/1000106739.parquet	28656	1000106739	wait
2	train_landmark_files/16069/100015657.parquet	16069	100015657	cloud
3	train_landmark_files/25571/1000210073.parquet	25571	1000210073	bird
4	train_landmark_files/62590/1000240708.parquet	62590	1000240708	owie

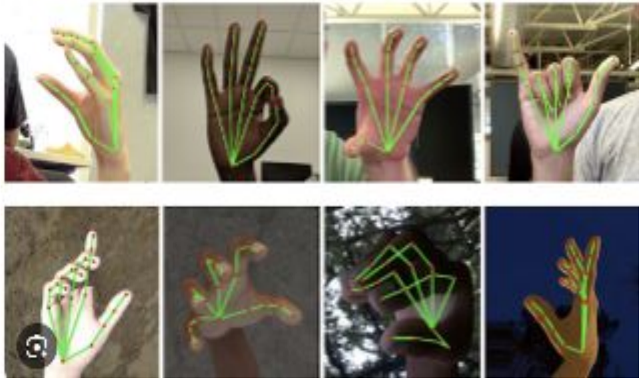
```
array(['blow', 'wait', 'cloud', 'bird', 'owie', 'duck', 'minemy', 'lips',
       'flower', 'time', 'vacuum', 'apple', 'puzzle', 'mitten', 'there',
       'dry', 'shirt', 'owl', 'yellow', 'not', 'zipper', 'clean',
       'closet', 'quiet', 'have', 'brother', 'clown', 'cheek', 'cute',
       'store', 'shoe', 'wet', 'see', 'empty', 'fall', 'balloon',
       'frenchfries', 'finger', 'same', 'cry', 'hungry', 'orange', 'milk',
       'go', 'drawer', 'TV', 'another', 'giraffe', 'wake', 'bee', 'bad',
       'can', 'say', 'callonphone', 'finish', 'old', 'backyard', 'sick',
       'look', 'that', 'black', 'yourself', 'open', 'alligator', 'moon',
       'find', 'pizza', 'shhh', 'fast', 'jacket', 'scissors', 'now',
       'man', 'sticky', 'jump', 'sleep', 'sun', 'first', 'grass', 'uncle',
       'fish', 'cowboy', 'snow', 'dryer', 'green', 'bug', 'nap', 'feet',
       'yucky', 'morning', 'sad', 'face', 'penny', 'gift', 'night',
       'hair', 'who', 'think', 'brown', 'mad', 'bed', 'drink', 'stay',
       'flag', 'tooth', 'awake', 'thankyou', 'hot', 'like', 'where',
       'hesheit', 'potty', 'down', 'stuck', 'no', 'head', 'food',
       'pretty', 'nuts', 'animal', 'frog', 'beside', 'noisy', 'water',
       'weus', 'happy', 'white', 'bye', 'high', 'fine', 'boat', 'all',
       'tiger', 'pencil', 'sleepy', 'grandma', 'chocolate', 'haveto',
       'radio', 'farm', 'any', 'zebra', 'rain', 'toy', 'donkey', 'lion',
       'drop', 'many', 'bath', 'aunt', 'will', 'hate', 'on', 'pretend',
       'kitty', 'fireman', 'before', 'doll', 'stairs', 'kiss', 'loud',
       'hen', 'listen', 'give', 'wolf', 'dad', 'gum', 'hear',
       'refrigerator', 'outside', 'cut', 'underwear', 'please', 'child',
       'smile', 'pen', 'yesterday', 'horse', 'pig', 'table', 'eye',
...
       'red', 'cow', 'person', 'puppy', 'cereal', 'touch', 'mouth', 'boy',
       'thirsty', 'make', 'for', 'glasswindow', 'into', 'read', 'every',
       'bedroom', 'napkin', 'ear', 'toothbrush', 'home', 'pajamas',
       'hello', 'helicopter', 'lamp', 'room', 'dirty', 'chair', 'hat',
       'elephant', 'after', 'car', 'hide', 'goose'], dtype=object)
```



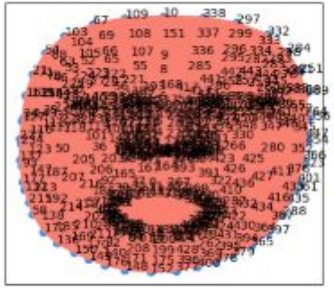


- **GISLR Dataset – Exploratory Analysis**

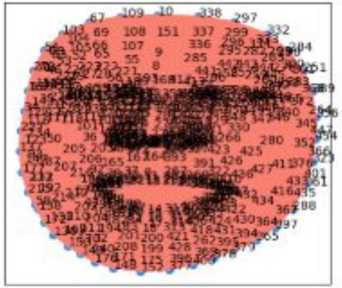
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57015 entries, 0 to 57014
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   frame           57015 non-null  int16
 1   row_id          57015 non-null  object
 2   type            57015 non-null  object
 3   landmark_index  57015 non-null  int16
 4   x               53193 non-null  float64
 5   y               53193 non-null  float64
 6   z               53193 non-null  float64
dtypes: float64(3), int16(2), object(2)
memory usage: 2.4+ MB
```



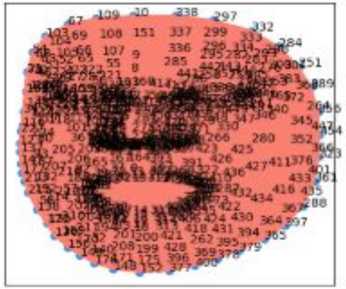
	frame	row_id	type	landmark_index	x	y	z
0	103	103-face-0	face	0	0.437886	0.437599	-0.051134
1	103	103-face-1	face	1	0.443258	0.392901	-0.067054
2	103	103-face-2	face	2	0.443997	0.409998	-0.042990
3	103	103-face-3	face	3	0.435256	0.362771	-0.039492
4	103	103-face-4	face	4	0.443780	0.381762	-0.068013



Frame no. 103



Frame no. 129



Frame no. 155

Preprocessing

1. Treating CSV & Parquet Files

CSV File

- CSV File & JSON File mapping (like “thank you” = 14).

	path	label
0	train_landmark_files/26734/1000035562.parquet	25
1	train_landmark_files/28656/1000106739.parquet	232
2	train_landmark_files/16069/100015657.parquet	48
3	train_landmark_files/25571/1000210073.parquet	23
4	train_landmark_files/62590/1000240708.parquet	164

Parquet Files

- Removing extra face points (keeping only important ones)
- Removing the z-coordinate (In many models (like MediaPipe), z is not true 3D depth. It’s just a relative number, not important)
- Converting from "rows for each point" ➜ to "one row per frame" with all (x, y) pairs.
- Removing of unnecessary columns like “row_id”, “type”

landmark index	x	y
0	0,12	0,45
0	0,30	0,50
2	0,42	0,53

Pivot →

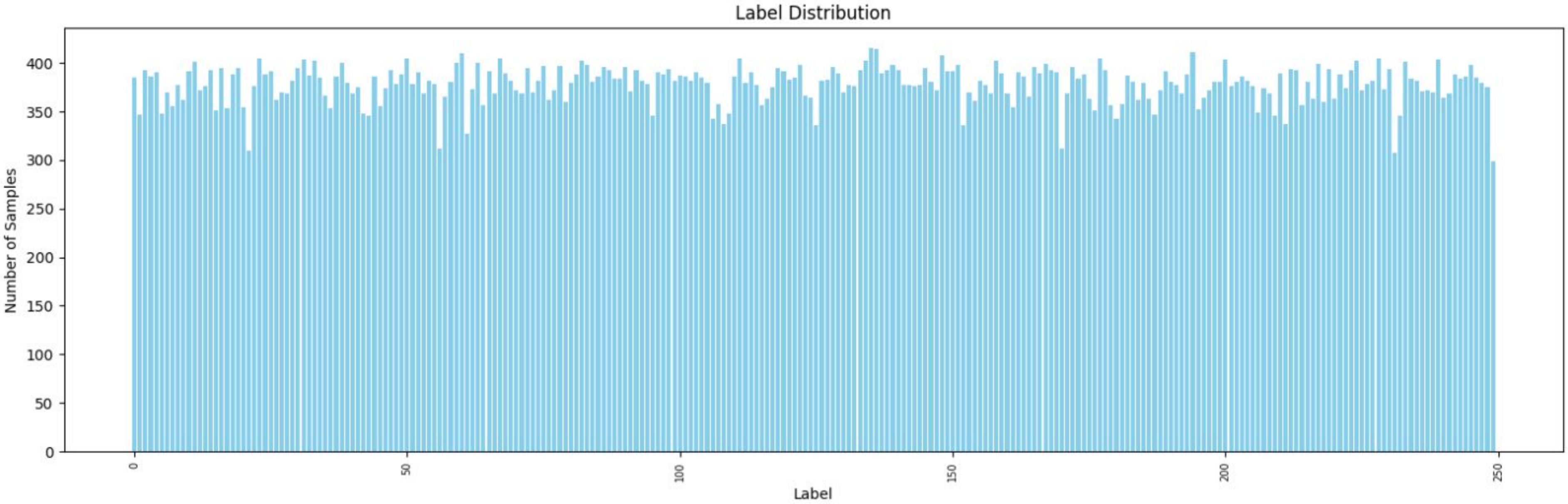
frame	x0	y0	y1	y1	y:
0	0,12	0,45	0,30	0,50	0,4
		0			

2. Normalization

- Applying normalization (with z-score normalization)

3. Padding Sequences

- Preparing for model training by ensuring that each sequence (a sign video) has a uniform length of 135 frames
- Evenly selecting 135 frames from longer sequences & pad shorter sequences with a value of -1 to reach the required length



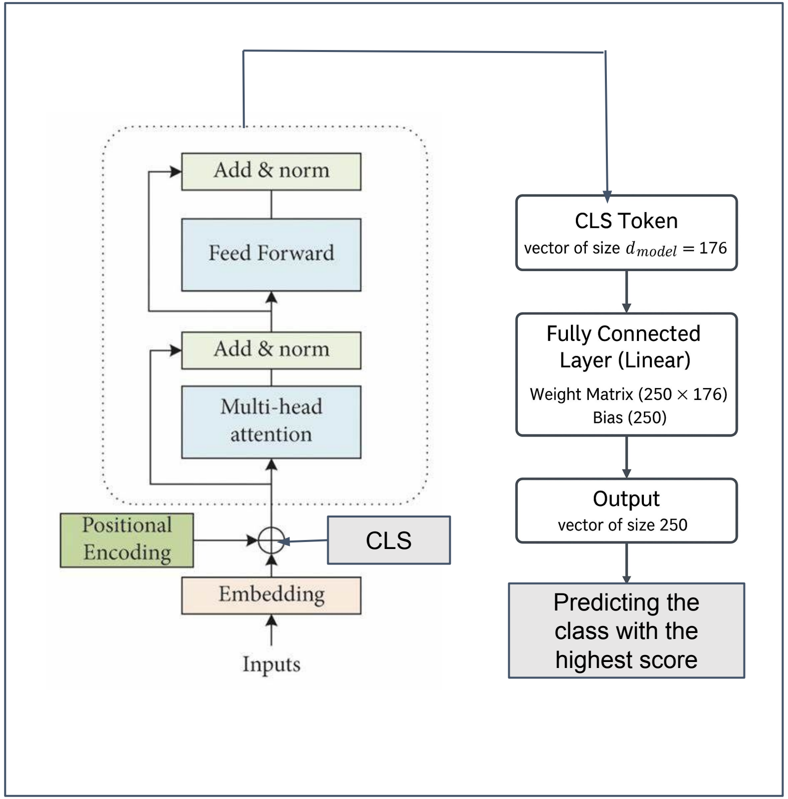
The label distribution is very balanced, with each class having a similar number of samples (around 350 - 420). This even spread ensures the model will learn all signs fairly without bias toward specific classes, leading to better overall accuracy & generalization.

Neural Network Architecture Design & Implementation

We have developed a Transformer Encoder-based model for recognizing ASL signs from landmark data sequences.

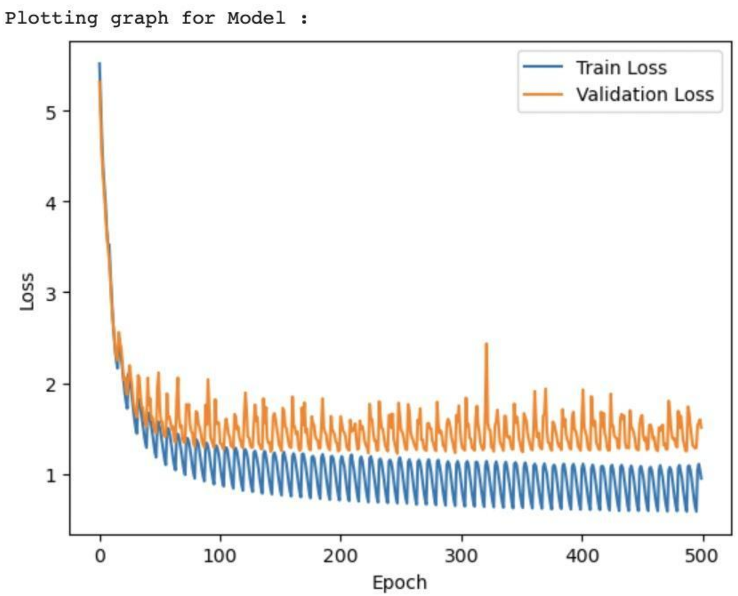
The model processes sequential inputs — sequences of hand & body keypoints — and learns to classify the entire sequence into an appropriate ASL sign labels.

- Input sequence (landmarks) → Embedding
- Adding Positional Encoding + CLS Token (summarizes entire sequence)
- Passing through Transformer Encoder stack
- Extracting only the CLS token output (after the Transformer)
- Passing CLS through a Fully Connected (Linear) layer
- Linear layer maps CLS to 250 ASL classes
- Model outputs final prediction (like 'Hello', 'Thank you', etc.)



Training Transformer Model

- Batch Processing - We loaded train data with batch size 128. Loading process takes about 20 minutes time.
- In Forward pass, we passed inputs through Embedding MsLP > Positional Encoding > Transformer Encoder > Output layer and generated prediction
- Loss Calculation - used CrossEntropyLoss with label smoothing and class weights
- Accuracy Calculation - Counted how many predictions match true labels
- Backward Pass - Calculated gradients using loss. Used Gradient Clipping (0.5) to avoid exploding gradients.
- Optimizer Step - Updated model parameters based on gradients using AdamW (handles weight decay correctly)



- Learning Rate Scheduler Step - Updated learning rate(0.0005 initial lr) dynamically after each batch using CosineAnnealingWarmRestarts Learning rate gradually decreases and periodically restarts.
- We ran the training for 500 epoch about 17 hours

Initial Model parameters

Model Parameter	Value
num_embed	4
d_model	176 (hidden size)
max_len	135 (maximum sequence length)
n_heads	4 (attention heads)
num_encoders	2 (transformer layers)
num_classes	250 (classification targets)
dropout	11.07%
activation	ReLU
batch_first	True (inputs shaped as batch × seq × features)

Hyper Parameters	
learning_rate	5.00E-04
weight_decay	0.1 (for regularization)
epochs	500
loss_function	CrossEntropyLoss
optimizer	AdamW
scheduler	CosineAnnealingWarmRestarts
gradient_clipping	0.5 (max norm)

Hyperparameter Tuning

- We performed hyperparameter tuning using Optuna framework.
- *Optuna* is an automatic hyperparameter optimization software framework
- Goal of a *study* is to find out the optimal set of hyperparameter values through multiple *trials* .
- We performed 20 trials to find the best hyperparameters
- With best parameters we trained the model with 300 epochs

```
Study statistics:
  Number of finished trials:  20
  Number of pruned trials:  0
  Number of complete trials:  20
Best trial:
  Value:  3.5336800104862935
  Params:
    num_embed_layers: 2
    n_heads: 11
    n_encoder_layers: 4
    dropout: 0.31891239569219343
```

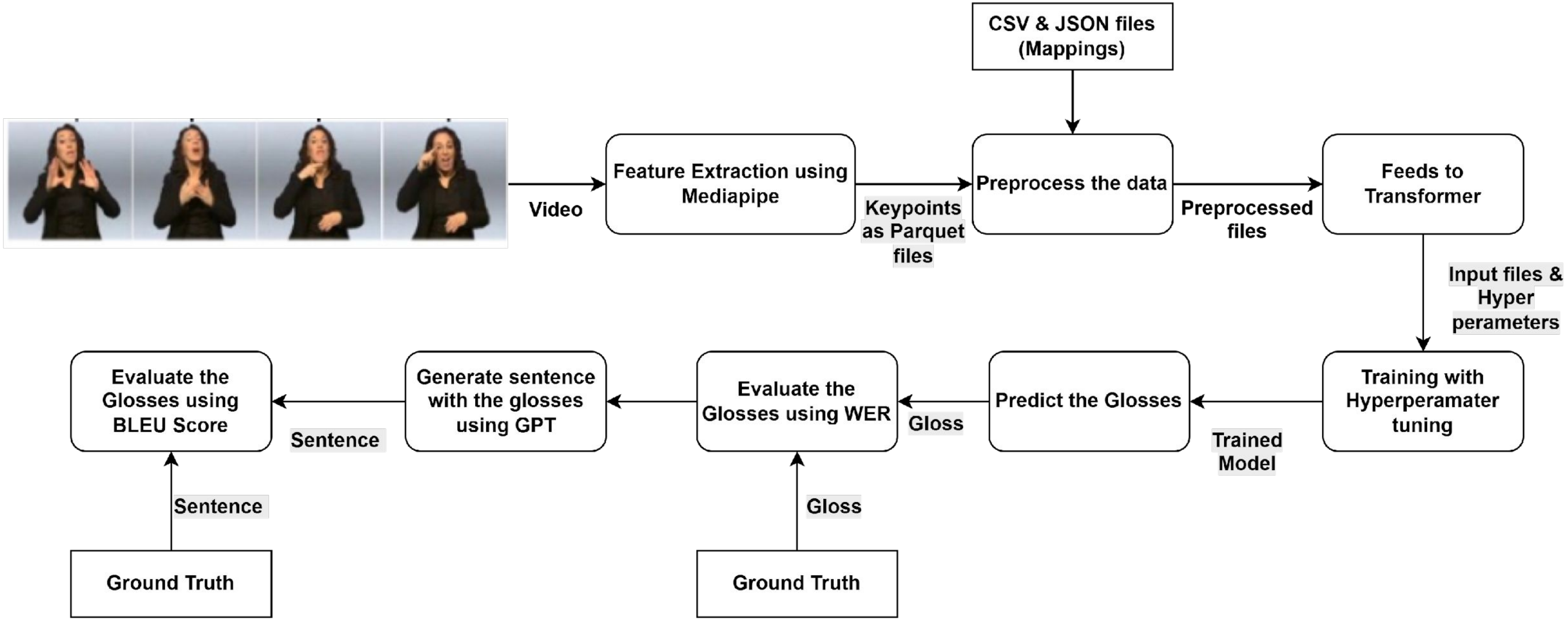
```
mean_acc_on_val_set = get_mean_classification_accuracy(val_loader)
mean_acc_on_train_set = get_mean_classification_accuracy(train_loader)

print(f'Mean Classification Accuracy on Train Set: {100*mean_acc_on_train_set:.4f}')
print(f'Mean Classification Accuracy on Validation Set: {100*mean_acc_on_val_set:.4f}')
```

```
Mean Classification Accuracy on Train Set: 72.8189
Mean Classification Accuracy on Validation Set: 62.7274
```

	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss
Base Model	0.757	0.947	0.651	1.51
Hyperparameter tuned Model	0.715	1.1	0.649	1.5

End-to-end Deployment



ASL Sentence Generation

- For Sentence generation we experimented different prompts with GPT-4.
- System predicted the Glosses from the Sign videos, then glosses were passed to GPT-4 for English spoken level sentence generation.

Prompt	(	(
	f"Create a short, creative, grammatically correct sentence	f"You are creating a meaningful, natural-sounding sentence suitable for a sign language performance.
	f"using ONLY these words: {',	f"Use ONLY these words: {',
	. join (predicted words) }.	.join(predicted _words)}. fMake the sentence visual, imaginative, and easy to express with body language.
	f"Do not add any extra words. You can rearrange them, repeat if needed,	f"Do NOT introduce new words, but you can rearrange or repeat words creatively if needed.
	f"but do not introduce new words."	f"Keep it short and vivid."
	)	)
Generated Sentence:	Fast Aunt emptied the milk.	Aunt fast empty milk.

Evaluation Metrics

Sign2Gloss:
Word Error Rate

- Measures how accurately the model predicts glosses (single words)
- WER =Number of words recognized correctly / total number of word
- Value range = 0 to infinity. ~ 0 = better prediction

Gloss2Text:
BLEU (Bilingual Evaluation Understudy)

- Score for comparing a generated sentence against a reference sentence (manually generated)
- BLEU-1 : for single word match(unigram)
- BLEU-2 : for matching 2 words (2-gram)
- BLEU-3 : for matching 3 words (3-gram)
- BLEU-4 : for matching 4 words (4-gram)

In our Project, for Gloss2test, we used BLEU-1 and BLEU-4 only.

Performance Evaluation : Sign2Gloss

For predicted Gloss word evaluation, we performed evaluation for single word prediction using jiwer.wer library . It gave WER =0.0

```
Summary of correct predictions:
   Sample Index Predicted Word Ground Truth Word
0          2056           wait              wait
```

Evaluation

```
!pip install jiwer
```

Show hidden output

```
from jiwer import wer

# Calculate WER
wer_score = wer(ground_truth_word, predicted_word)

print(f"Ground Truth: {ground_truth_words}")
print(f"Predicted Word: {unique_correct_words}")
print(f"Word Error Rate (WER): {wer_score:.4f}")
```

```
Ground Truth: ['wait']
Predicted Word: {'wait'}
Word Error Rate (WER): 0.0000
```

```
import editdistance

def compute_wer(references, hypotheses):
    """
    Compute Word Error Rate (WER) using Levenshtein distance.

    Args:
        references (list of str): Ground-truth words or glosses.
        hypotheses (list of str): Predicted words or glosses.

    Returns:
        float: WER = (insertions + deletions + substitutions) / (number of reference words)
    """
    if not references:
        return 0.0 if not hypotheses else float('inf')

    total_words = len(references)
    total_errors = editdistance.eval(references, hypotheses)

    return total_errors / total_words if total_words > 0 else 0.0
```

For evaluation of multiple Gloss word prediction , we used Levenshtein distance to calculate WER. For 5 words , WER = 0.7285

```
WER for the selected 5 predicted words: 0.7285

Detailed Predictions:
    Ground Truth Prediction
0        pajamas   scissors
1        pajamas   airplane
2            cry       pool
3            cry       pool
4           pool        cry
..           ...        ...
146         pool       pool
147          cry       pool
148      pajamas       pool
149         pool       pool
150       yellow       pool

[151 rows x 2 columns]
```

Performance Evaluation : Gloss2Text

ASL Sentence Construction

- ASL Gloss to sentence does not follow common English grammar. Instead, they follow:

- topic-comment structure: main subject + about the topic. [eg. STORE I GO]

- subject-verb-object structure: [eg. When talking about new information (BOY FIND TOY)]

For Sentence level evaluation we constructed reference sentence using ASL sentence construction rule.
GPT–4 Generated sentence is being evaluated with Reference sentence using BLEU score.

```
# Finally, convert set to list
predicted_words = list(predicted_words)

print("\nFinal 5 unique predicted words:", predicted_words)
```

```
[40] ✓ 1.6s
...
Sample 9166: Predicted unique word: fast
Sample 1747: Predicted unique word: milk
Sample 7219: Predicted unique word: empty
Sample 1085: Predicted unique word: aunt

Final 5 unique predicted words: ['fast', 'milk', 'empty', 'aunt']
```

Prompt-1 Output: BLEU-1 Score

```
Generated Sentence: Fast Aunt emptied the milk.
BLEU Score: 0.1269
```

Prompt-2 Output: BLEU-1 Score

```
[59] ✓ 0.3s
...
Generated Sentence: Aunt fast empty milk.
BLEU Score: 0.7071
```

Evaluation Comparison

For our project we referenced two papers – SLT model and Sign Spotter with LLM sentence generation.

Paper	Author	Year	#Citation
Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation	Camgoz, N. C., Koller, O., Hadfield, S., & Bowden, R.	2020	670
Using an LLM to Turn Sign Spottings into Spoken Language Sentences.	Sincan, O. M., Camgoz, N. C., & Bowden, R.	2024	2

Our approach of generating interpreted sentence from Sign video is different than previous SLT method. It cannot be directly compared.

		Dataset	WER	BLEU-1	BLEU-4
Sign2Gloss	Paper-1	Phoenix2014T	24.88	NA	NA
	Our Sign Transformer	GISLR	72.85	NA	NA
Gloss2Text	Paper-1	Phoenix2014T	NA	50.69	25.35
	Paper-2 [Spotter +GPT]	GDGS-20	NA	38.25	9.12
	Our Gloss2Sentence (using GPT-4)	GISLR	NA	70.71	13.41

	Paper 1 : Sign Language Transformers	Our work
Model	Heavy CSLR+SLT	**Two stage modular pipeline with Transformer Encoder (Gloss Prediction) and LLM (Sentence generation)**
Learning	Learned internally (CTC + gloss decoder)	Transformer encoder takes sequences of keypoints and **encodes them into rich hidden features** before predicting the gloss class for each sequence
Architecture	Full heavy Transformers + CNN features	**Lightweight keypoints + Custom Transformer**
Language Model	Trained decoder Transformer	**Zero-shot GPT-4 prompting (no extra training)**

	Paper 2: Spotter+GPT	Our Work
Model	I3D (heavy 3D CNN)	**Custom Transformer on keypoints** (small model)
Spotter Training	Supervised, large video datasets	Smaller, efficient data processing pipeline
End-to-End Efficiency	Computationally heavy	Lightweight , fast , scalable pipeline

Technical Novelty

- Our project **combines** the ideas from both papers end-to-end sign language transformer and intermediate gloss representation **along with** GPT language modeling but **avoids** their downsides (heavy 3D CNNs, rigid end-to-end training)

Technical Contributions

- **Two-stage modular pipeline:**
  - **Stage 1:** Predict glosses from Isolated Sign using a Transformer.
  - **Stage 2:** Generate spoken English sentences from glosses using **GPT prompting**.
- **Better gloss prediction quality:**
  - Transformer-based gloss prediction captures more complex temporal features than simple sliding window spotters.
- **Faster Training than Continuous SLT**
  - For Continuous SLT takes long videos (20s +) for training which make the training process longer and harder. Isolated Sign video clips are comparatively small. Since we are using only Transformer Encoder, our training time is less (in hours) compared to Continuous SLT training (which takes several days/weeks to train). However, we are achieving similar outcome of generating Spoken level Sentence.

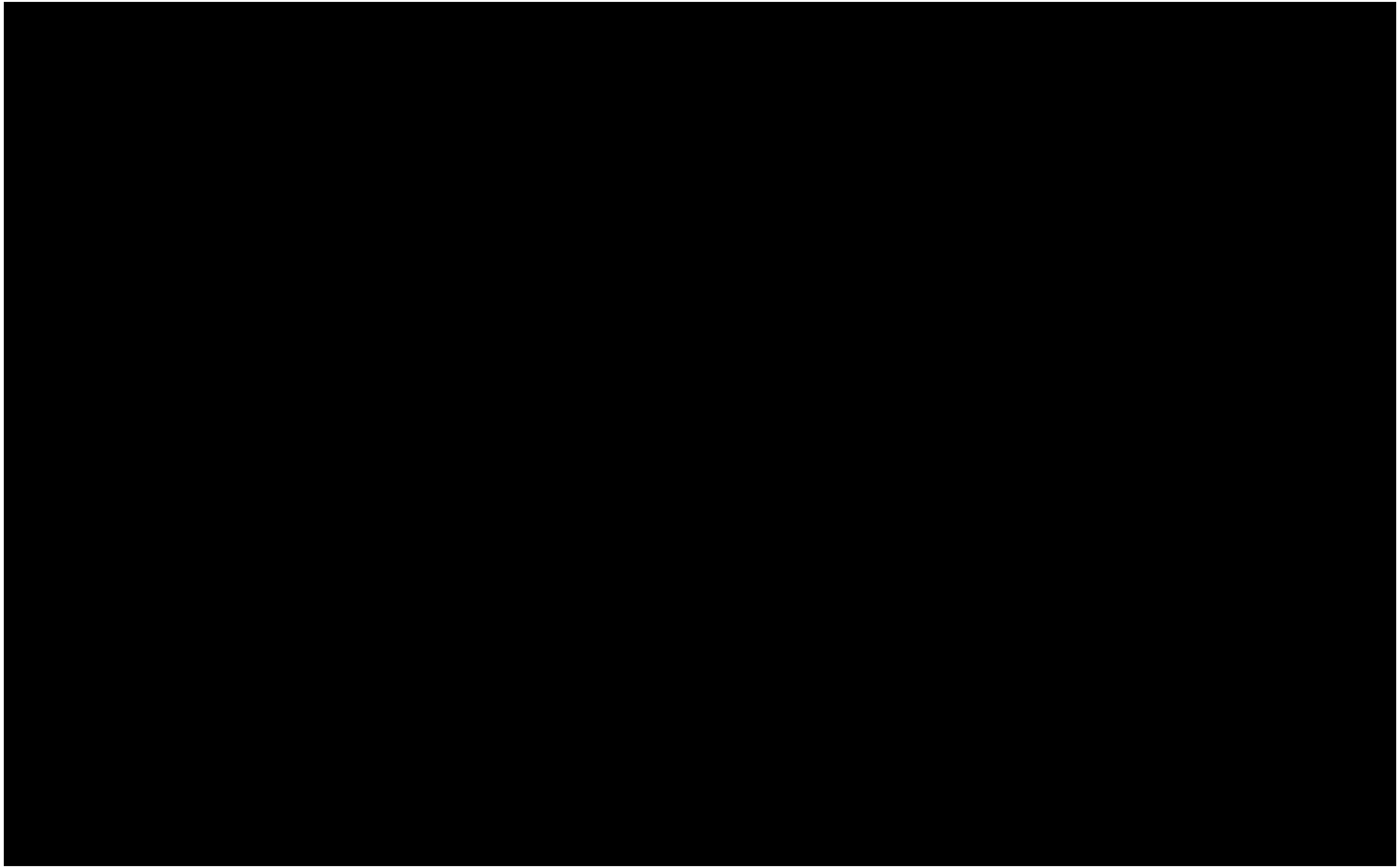
Community Contribution

- Every day, 33 babies are born with permanent hearing loss in the U.S. [Source : https://www.kdhe.ks.gov/887/Hearing-Loss-Facts]
- Many people, including those who are deaf, autistic, or have other disabilities, may struggle with vocal communication, making it important to value and support alternative forms of communication.
- Without sign language, deaf children are at risk of Language Deprivation Syndrome.
- According to Cheri Dowling, executive director of the American Society for Deaf Children , most deaf children are born to hearing parents and access to tools like Sign language interpreter would enable these parents to open effective communication channel with their kids. [Source : https://blogs.nvidia.com/blog/ai-sign-language/]
- Not only for people with special needs, Sign language has become an increasingly popular form of communication for people without hearing challenges as well.
- "According to the Modern Language Association, in 2021, American Sign Language (ASL) was the third most studied language at U.S. colleges and universities. " [Source : https://www.fraser.org/resources/blog/why-teaching-sign-language-can-benefit-young-children-]

❖ **Our project will enable Signer to communicate with non-signers. Without relying on human interpreter.**

❖ **Also, this project promotes ASL learning for all induvial.**

ASL Project - DEMO



References

Sincan, O. M., Camgoz, N. C., & Bowden, R. (2024, March 15). *Using an LLM to Turn Sign Spottings into Spoken Language Sentences*. arXiv.org. https://arxiv.org/abs/2403.10434

Camgoz, N. C., Koller, O., Hadfield, S., & Bowden, R. (2020). Sign Language Transformers: Joint End-to-End Sign Language Recognition and Translation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10020–10030. https://doi.org/10.1109/cvpr42600.2020.01004

https://github.com/optuna/optuna

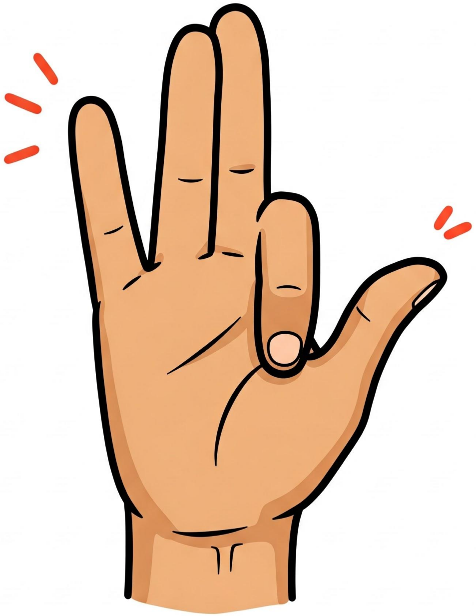
https://www.aslbloom.com/blog/asl-sentence-structure#:~:text=ASL%20uses%20a%20topic%2Dcomment,more%20similar%20to%20spoken%20English.

Images:

https://medium.com/neuralspace/word-error-rate-101-your-guide-to-stt-vendor-evaluation-5b68072fcbf7

Demo Video Link:

https://drive.google.com/drive/folders/1ME6mdPSpUxviKSLXS9e-DkVu9CqTmd_0?usp=sharing



“Thank you ”

Supporting Slide 0 : About the Frame Size

	path	participant_id	sequence_id	sign	label	num_frames
0	train_landmark_files/26734/1000035562.parquet	26734	1000035562	blow	25	23.0
1	train_landmark_files/28656/1000106739.parquet	28656	1000106739	wait	232	11.0
2	train_landmark_files/16069/100015657.parquet	16069	100015657	cloud	48	105.0
3	train_landmark_files/25571/1000210073.parquet	25571	1000210073	bird	23	12.0
4	train_landmark_files/62590/1000240708.parquet	62590	1000240708	owie	164	18.0

```
train.num_frames.describe()
```

```
count    94477.000000
mean        37.935021
std         44.177069
min          2.000000
25%         12.000000
50%         22.000000
75%         44.000000
max        537.000000
Name: num_frames, dtype: float64
```

```
train["sign"].value_counts()
```

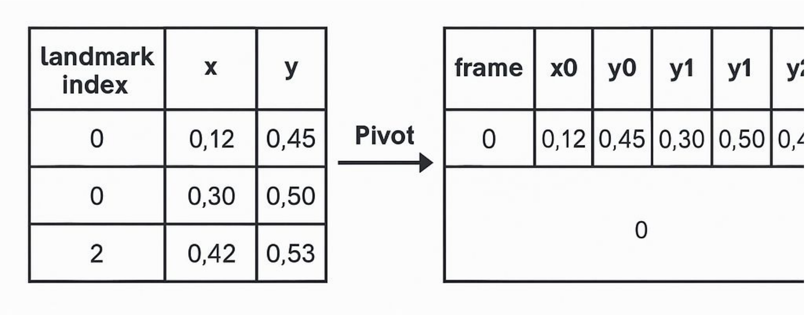
```
listen    415
look      414
shhh      411
donkey    410
mouse     408
         ...
dance     312
person    312
beside    310
vacuum    307
zipper    299
Name: sign, Length: 250, dtype: int64
```

Supporting Slide 1: Preprocessing Process - Step 1

- Reading the CSV and JSON to get the correct label number for each sign (like “thank you” = 14). Then replacing the sign names with numbers (so the model can understand them).

	path	label
0	train_landmark_files/26734/1000035562.parquet	25
1	train_landmark_files/28656/1000106739.parquet	232
2	train_landmark_files/16069/100015657.parquet	48
3	train_landmark_files/25571/1000210073.parquet	23
4	train_landmark_files/62590/1000240708.parquet	164

- Removing extra face points (keeping only important ones)
- Removing the z-coordinate (In many models (like MediaPipe), z is not true 3D depth. It’s just a relative number, not important)
- Converting from "rows for each point" ➜ to "one row per frame" with all (x, y) pairs.
- Removing of unnecessary columns like “row_id”, “type”



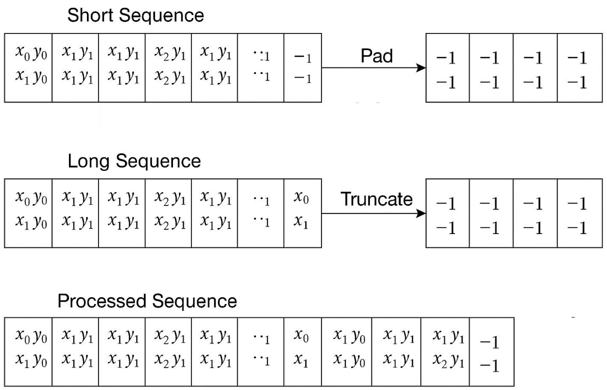
Supporting Slide 2: Processing Step 2 & Step 3

Step 2 -

- It processes the pre-extracted landmark data (from Step 1) -> applies normalization to ensure that all features are on a similar scale (Two-pass approach - in the first pass, it computes the mean & standard deviation of each feature (across all files) to get global statistics; in the second pass, it normalizes each file using these global statistics with z-score normalization)

Step 3 -

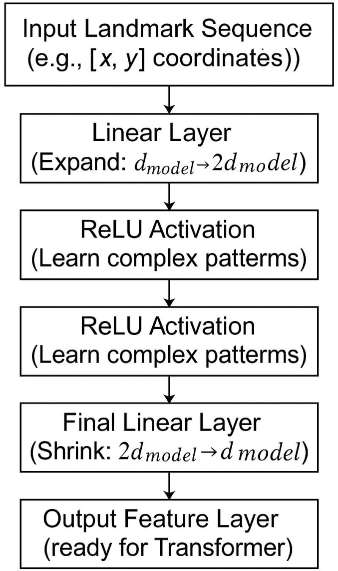
- The normalized landmark data is prepared for model training by ensuring that each sequence (a sign video) has a uniform length of 135 frames.("uniform sampling" to evenly select 135 frames from longer sequences & pad shorter sequences with a value of -1 to reach the required length).

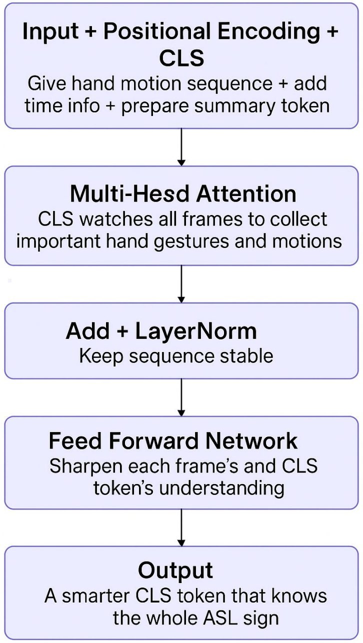


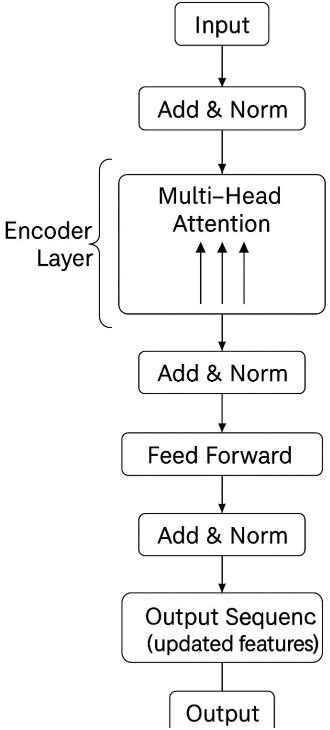
Supporting Slide 3

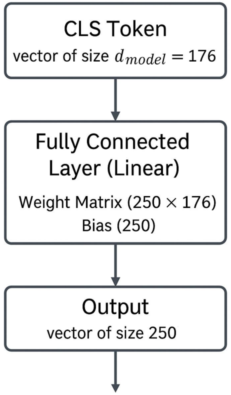
Embedding block - a multi-layer perceptron (MLP) network composed of several Linear, Layer Normalization & ReLU activation layers. A small neural network that transforms the input landmarks into richer features. It first increases the size of the features to make them more detailed, passes them through several layers to refine them & then reduces them back to the original siz

- **Positional encoding** (to understand the order of frames)
- **CLS token** (summarizes the entire sequence into vector for classification)
  [CLS] → Frame 1 → Frame 2 → Frame 3 → ... → Frame 135
- **Transformer ->** Multi-head Attention, feedforward networks









Supporting Slide 4

The **embedding block** is a multi-layer perceptron (MLP) network composed of several Linear, Layer Normalization, and ReLU activation layers. It starts by projecting the input features from `d_model` dimensions to a larger `2*d_model` space to enrich the feature representation, then continues through several intermediate layers, and finally reduces the feature size back to `d_model`. This design allows the model to extract complex spatial relationships within the frame before feeding the sequence into the Transformer. It ensures that the raw landmark inputs are mapped into a feature space that is easier for the Transformer to learn from.

After embedding, we add **positional encoding** to the sequence, which introduces information about the order of the frames into the model. Since Transformers are permutation-invariant, positional encoding is essential to let the model recognize temporal dynamics, such as the progression of hand gestures across frames. We also incorporate a learnable **CLS (classification) token**, which is designed to summarize the information from the entire sequence. By adding this token to the sequence, the model can focus its learning on generating a condensed, meaningful representation that can be used for final classification.

The core sequence processing is performed by a **stack of Transformer Encoder layers**. Each layer consists of a multi-head self-attention mechanism followed by a feed-forward network. The self-attention layers allow the model to relate each frame with every other frame, thus capturing both short-term interactions (e.g., slight finger movement) and long-term dependencies (e.g., hand moving from one location to another). The feed-forward sublayers and dropout regularization help in learning non-linear transformations and prevent overfitting. The number of Transformer layers (`num_encoders`) and heads (`n_heads`) are configurable to balance model complexity and performance.

Finally, the output corresponding to the **CLS token** is extracted after the Transformer stack. This vector, representing the full context of the sequence, is passed through a final **fully connected linear layer** that maps it to the output classes. The model is trained to predict the correct ASL sign label from the input landmark sequence.

This architecture is highly suited for ASL recognition because it flexibly models varying sequence lengths, captures detailed motion and spatial patterns, and offers robustness against noise and variations in signing style. Compared to traditional RNNs or simple CNNs, the Transformer Encoder model provides stronger global context modeling and parallel processing capabilities, making it ideal for this task.

Supporting Slide 5

Inside the Transformer Encoder, each token — including the special CLS token and all frame tokens — passes through two main operations repeatedly:-
Self-Attention and **Feed Forward Networks**, with Add & LayerNorm steps to stabilize learning.

First, in the **Multi-Head Self-Attention** block,

- Every token looks at every other token.
- The CLS token attends to all the frame tokens and gathers important information about the hand movements and gestures across the sequence.
- Frames also look at each other to understand local and global motion patterns.

After attention, we apply **Add & LayerNorm**, which helps the model learn more easily by stabilizing values and improving gradient flow.

Next, in the **Feed Forward Network**,

- Each token (CLS and frames) is individually refined.
- This step sharpens the features inside each token, allowing the model to better separate important signals from noise.

Again, an **Add & LayerNorm** is applied to stabilize learning.

This entire block — Attention + Feed Forward — is called a **Transformer Encoder Layer**, and it is repeated `num_encoders` times in the model.

By the end, the **CLS token** has collected a rich summary of the entire sequence, understanding both small and large hand movements, which is then used for final classification.

Supporting Slide 5

"Add" = Skip Connection (Residual Connection)

- After Self-Attention **OR** after Feed Forward Network,
- You **add the input back to the output**.

In simple terms:

- You don't just replace the original features with new ones.
- You **add the original + the new updated features** together.

Why?

- Helps the model **remember the original information**.
- Helps **gradient flow** during backpropagation (training becomes easier, prevents vanishing gradients).

It is called **Skip Connection** because the original input **skips** over the complex part (like attention or feedforward) and **directly connects** to the output. **You are skipping over** the intermediate transformation and **adding the input directly** to the output.

Supporting Slide 6

Normalization makes sure the **numbers inside the model are balanced**.

- It **rescales** and **re-centers** the outputs after complex operations (like Attention and Feed Forward).
- It **prevents numbers from becoming too big or too small**.

It **standardizes** the features:-

- Mean becomes around 0
- Variance becomes around 1

FFN is a **small neural network** that is applied **individually** to **each token** (each frame or CLS token). It consists of **two Linear layers** and an **activation function** (like ReLU) in between.

Linear (d_model → 2*d_model) -> Activation (ReLU) -> Linear (2*d_model → d_model)

Each token feature (like a 176-dimensional vector) is **expanded** to a bigger space (352 dimensions) -> Non-linearity (ReLU) is applied, allowing the model to **learn complex patterns ->** Then it is **compressed back** to the original size (176) -> Output is passed forward (after Add + LayerNorm)

Supporting Slide 7

Self-Attention is a mechanis where **every token (including CLS token and frames) looks at every other token** in the sequence & **decides how important each one is** to itself. Each token **"pays attention"** to all other tokens & **collects information** based on how important they are.

For every token (CLS, Frame 1, Frame 2, etc.):-

- Create three vectors:-
  - **Query (Q)**:- What you are looking for.
  - **Key (K)**:- What information you have.
  - **Value (V)**:- The actual information.

For each pair of tokens:-

- Compare Query & Key → **calculate similarity (attention score)**.

Multi-head:- Instead of doing **one** attention operation, the Transformer does **multiple attentions in parallel** — called **heads**.

Each **head** learns to focus on **different aspects** of the sequence.

- **One head** might focus on short-term movements (like small finger movements),
- **Another head** might focus on long-term movements (like full hand translation),
- **Another head** might focus on motion speed.

Split the input into multiple copies (one for each head) -> Each head does its own self-attention separately -> Outputs from all heads are **concatenated** (joined together) -> Pass concatenated output through a Linear layer to mix them back together.

Supporting Slide 8

Suppose you have:-

- Frame 1: Start of hand wave
- Frame 2: Middle of hand wave
- Frame 3: End of hand wave
- CLS token: wants to summarize

CLS token:

- Gives 0.8 score to Frame 2 (important middle motion),
- 0.6 score to Frame 1,
- 0.7 score to Frame 3.

Then, CLS **gathers weighted information** from Frame1, Frame2, Frame3 based on these scores.

That’s how CLS **builds a smart summary** of the entire hand movement!