HUMAN POSE ESTIMATOR FOR FITNESS APPLICATIONS

TEAM EUREKA

Trisha Gumidelli - SE22UARI176
Likesh Koya - SE22UARI210
Tamidela Swathika Reddy - SE22UARI172
Basam Nanda Prabath Reddy - SE22UARI028
Sathwik Chitla - SE22UARI154

PROBLEM STATEMENT

In recent years, the fitness industry has witnessed a significant shift toward home-based and virtual workout programs. While convenient, these programs often lack real-time feedback on exercise form and posture, which are critical for effective and safe training. Improper posture during exercises can lead to injuries, inefficient exercises, and difficulty in correcting mistakes without a trainer’s guidance. Home-based fitness lacks real-time feedback, leading to improper posture, injuries, and ineffective workouts. Without personal trainers, users struggle to identify and correct mistakes. Existing solutions fail to provide affordable, personalized guidance.

PROJECT OVERVIEW

The “Human Pose Estimator for Fitness Applications” integrates image processing and Natural Language Processing (NLP) to deliver a highly interactive and adaptive fitness solution. Using real-time pose analysis, it detects body movements accurately, providing personalized feedback to improve posture and prevent injuries. The inclusion of NLP enhances user engagement through natural language command recognition and a chatbot offering tailored exercise guidance. This system caters to diverse applications, from general fitness and yoga to physical therapy, delivering precise corrections and personalized advice. By prioritizing safety, effectiveness, and accessibility, it addresses the limitations of current fitness technologies, enhancing the user experience.

KEY FEATURES

- Real-time pose detection and analysis by taking in a video input or real time (live) footage from the user
- Conversational interface for interactive fitness guidance- a chatbot to resolve any queries related to any workout routines or methods.

DATASET

Dataset Extracted from

- The dataset is loaded from Hugging Face's dataset hub using the load_dataset() function.

Dataset size

- The size of the dataset corresponds to the number of entries in the train split of the Hugging Face dataset.

Dataset format

- The dataset consists of entries with fields instruction (input prompt) and output (response/completion), processed into tokenised inputs, targets, and NSP labels.

MODEL AND METHODOLOGY

- For the natural language processing part in our project we added a chatbot which helps the user ask queries about workout regimes, workout positions and posture corrections.
- For this chatbot we used the BERT architecture which is a pre-trained language model architecture designed to understand the context of words in a sentence by looking at both their left and right surroundings simultaneously, making it bidirectional.
- The three major steps we have following in this process include: Preparing the Dataset; Building a model; Training the model.

PREPARING THE DATASET

The dataset preparation involves several key steps to ensure it is suitable for training the BERT model with both Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) objectives:

1. **Dataset Loading:**
    a. The dataset is loaded from Hugging Face's dataset hub using the load_dataset function.
    b. The train split is specifically used for training data.

2. **Tokenisation:**
    a. Each entry in the dataset contains two fields:
        i. instruction: The input prompt or query.
        ii. output: The expected response or completion.
    b. Both fields are tokenised using the BertTokenizer from the transformers library to convert text into tokenised sequences.

3. Example Generation:

- For each dataset entry:
    - A positive NSP example is created by pairing the instruction and output.
    - A negative NSP example is generated by pairing mismatched sentences from the dataset.

4. Masking for MLM:

- To support MLM training, 15% of tokens in the input sequence are randomly selected and replaced with:
    - The [MASK] token (80% of the time).
    - A random token from the vocabulary (10% of the time).
    - The original token (10% of the time).
- This creates a masked_indices field for MLM predictions.

5. Final Dataset Structure

- Each entry is a dictionary with:
    - masked_indices: Input tokens with masking.
    - target: Original token indices for MLM.
    - is_next: Binary NSP label (1 = true, 0 = false).
- The entries are combined into a pandas.DataFrame for training integration.

BUILDING THE MODEL

The BERT model is designed to perform both Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) tasks. Here's how the model is structured:

1. **Embedding Layer:**
    - A JointEmbedding module integrates token, positional, and segment embeddings to represent input sequences.
    - This prepares the input for the transformer encoder.

2. **Encoder Layer:**
    - A transformer-based Encoder processes the embeddings using self-attention and feedforward layers.
    - The output is a contextualized representation of the input sequence.

3. Prediction Layers:

- **Token Prediction Layer (MLM):** A linear layer maps the encoded representation to the vocabulary size for predicting masked tokens.
- **Classification Layer (NSP):** Another linear layer takes the encoding of the first token (typically [CLS]) and outputs a binary classification for NSP.

4. Forward Pass:

- Input sequences and attention masks are passed through the embedding and encoder layers.
- **Outputs include:** Token predictions for MLM and A binary NSP prediction based on the [CLS] token.

5. Masked Language Modeling (MLM):

- During training, 15% of the tokens in the input are masked.
- The model predicts the original tokens based on the surrounding context.
- Loss is computed using NLLLoss (Negative Log Likelihood Loss) between the predicted and original tokens.

TRAINING THE MODEL

The training process for the BERT model involves optimizing it for two tasks, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), using a well-defined pipeline.

1. **Data Loader Integration:**
    - The dataset, prepared with tokenized inputs, MLM targets, and NSP labels, is wrapped in a DataLoader for batching and shuffling during training.
    - Batches include input tokens, attention masks, masked token targets, and NSP targets.

2. Loss Functions:

a. MLM Loss:
- The model predicts masked tokens, and NLLLoss computes the loss by comparing predictions with the original tokens.

b. NSP Loss:
- A binary classification loss (BCEWithLogitsLoss) evaluates whether two sentences follow logically.

3. Training Loop:

- For each epoch:
    - The model processes batches of data to generate token predictions and NSP classifications.
    - The combined loss (MLM + NSP) is calculated and backpropagated.
    - The optimizer updates the model parameters to minimize the total loss.

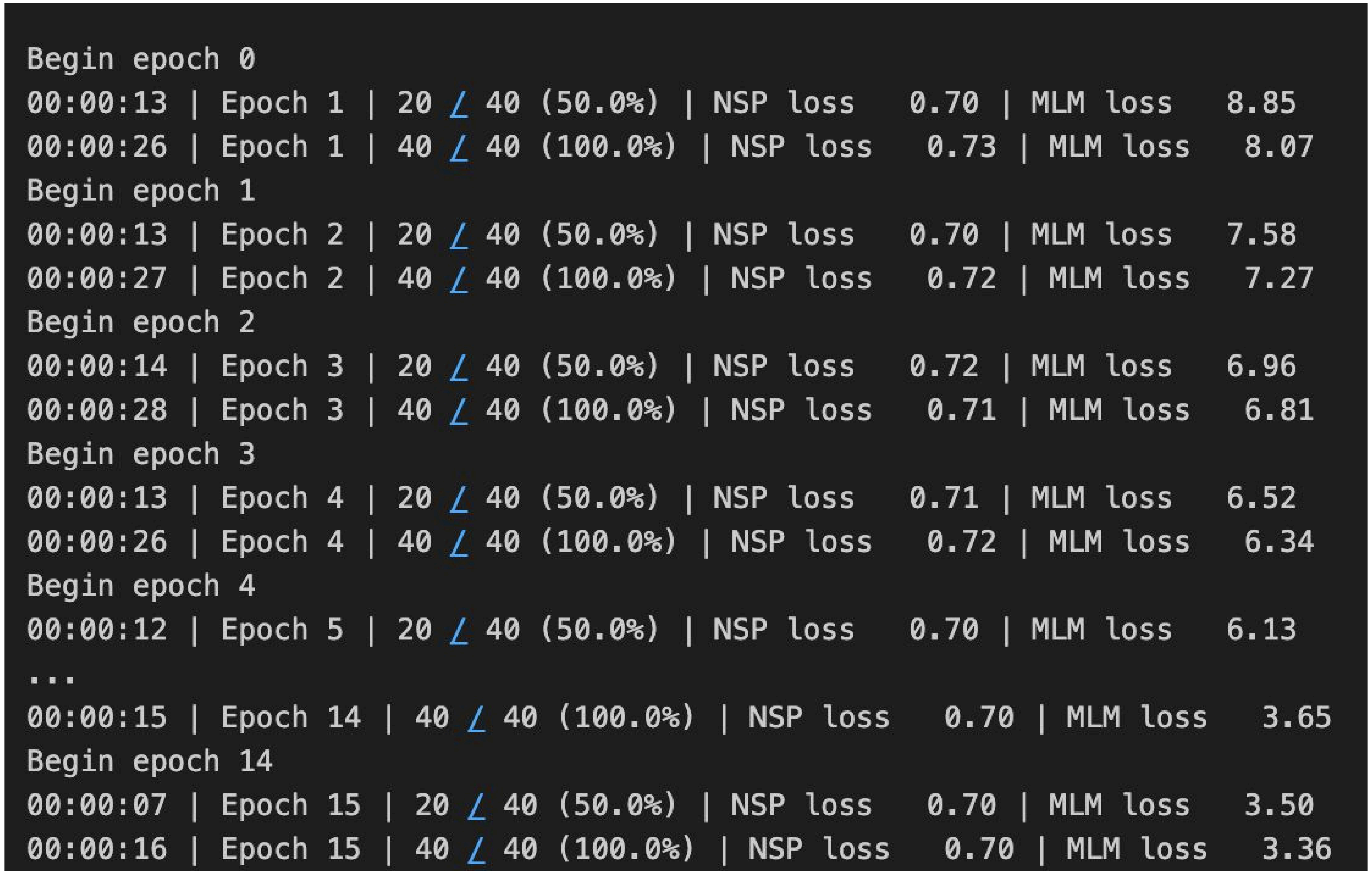
4. Optimization:

- The Adam optimizer is used for gradient updates, ensuring efficient convergence.

5. Model Improvements:

- Regular monitoring of the total loss helps track training progress.
- Techniques like gradient clipping can be added to handle exploding gradients during back propagation.

RESULTS OF THE MODEL



```
Begin epoch 0
00:00:13 | Epoch 1 | 20 / 40 (50.0%) | NSP loss   0.70 | MLM loss   8.85
00:00:26 | Epoch 1 | 40 / 40 (100.0%) | NSP loss   0.73 | MLM loss   8.07
Begin epoch 1
00:00:13 | Epoch 2 | 20 / 40 (50.0%) | NSP loss   0.70 | MLM loss   7.58
00:00:27 | Epoch 2 | 40 / 40 (100.0%) | NSP loss   0.72 | MLM loss   7.27
Begin epoch 2
00:00:14 | Epoch 3 | 20 / 40 (50.0%) | NSP loss   0.72 | MLM loss   6.96
00:00:28 | Epoch 3 | 40 / 40 (100.0%) | NSP loss   0.71 | MLM loss   6.81
Begin epoch 3
00:00:13 | Epoch 4 | 20 / 40 (50.0%) | NSP loss   0.71 | MLM loss   6.52
00:00:26 | Epoch 4 | 40 / 40 (100.0%) | NSP loss   0.72 | MLM loss   6.34
Begin epoch 4
00:00:12 | Epoch 5 | 20 / 40 (50.0%) | NSP loss   0.70 | MLM loss   6.13
...
00:00:15 | Epoch 14 | 40 / 40 (100.0%) | NSP loss   0.70 | MLM loss   3.65
Begin epoch 14
00:00:07 | Epoch 15 | 20 / 40 (50.0%) | NSP loss   0.70 | MLM loss   3.50
00:00:16 | Epoch 15 | 40 / 40 (100.0%) | NSP loss   0.70 | MLM loss   3.36
```

EXPERIMENTS

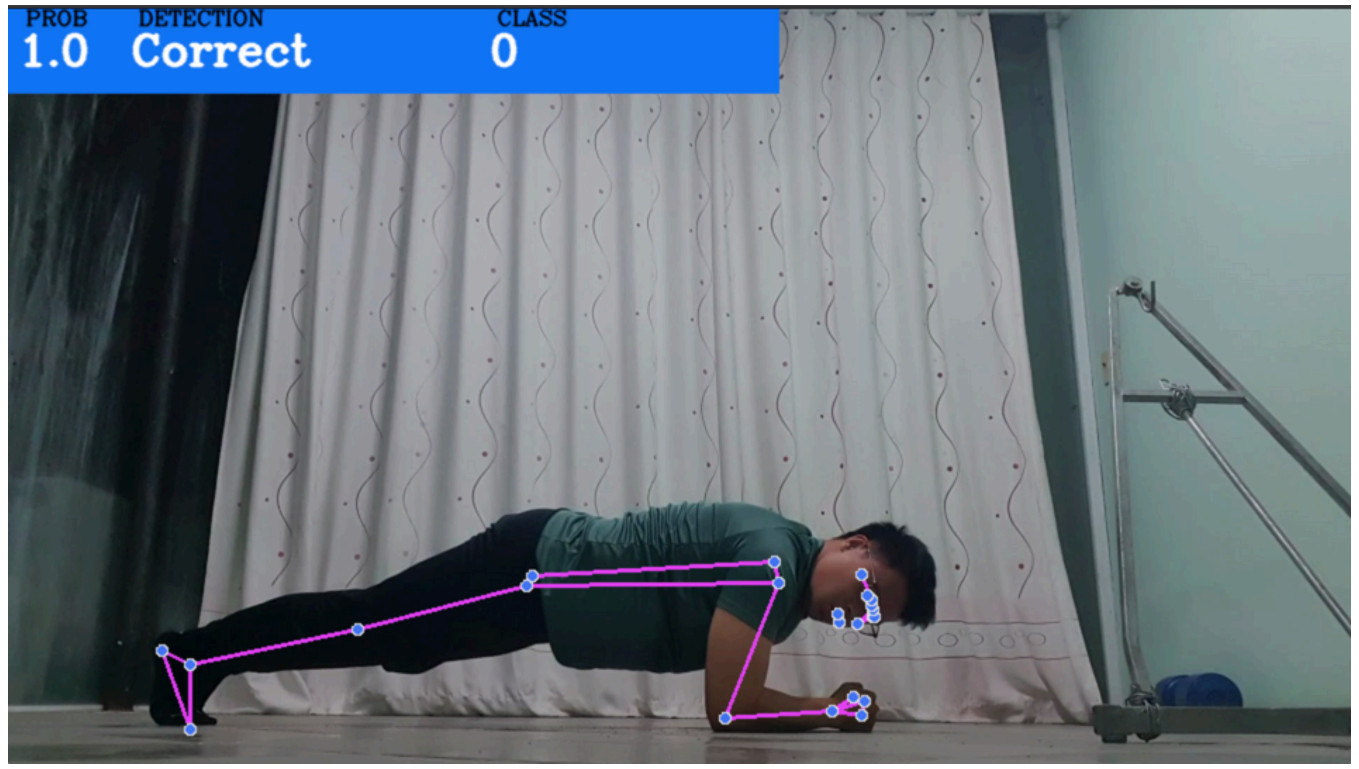
- We started of by trying to train our model using rule-based model for the training of our chatbot, with further training and testing we did not achieve our desired outputs so we shifted our model to the BERT architecture.
- Problems raised in rule-based model:

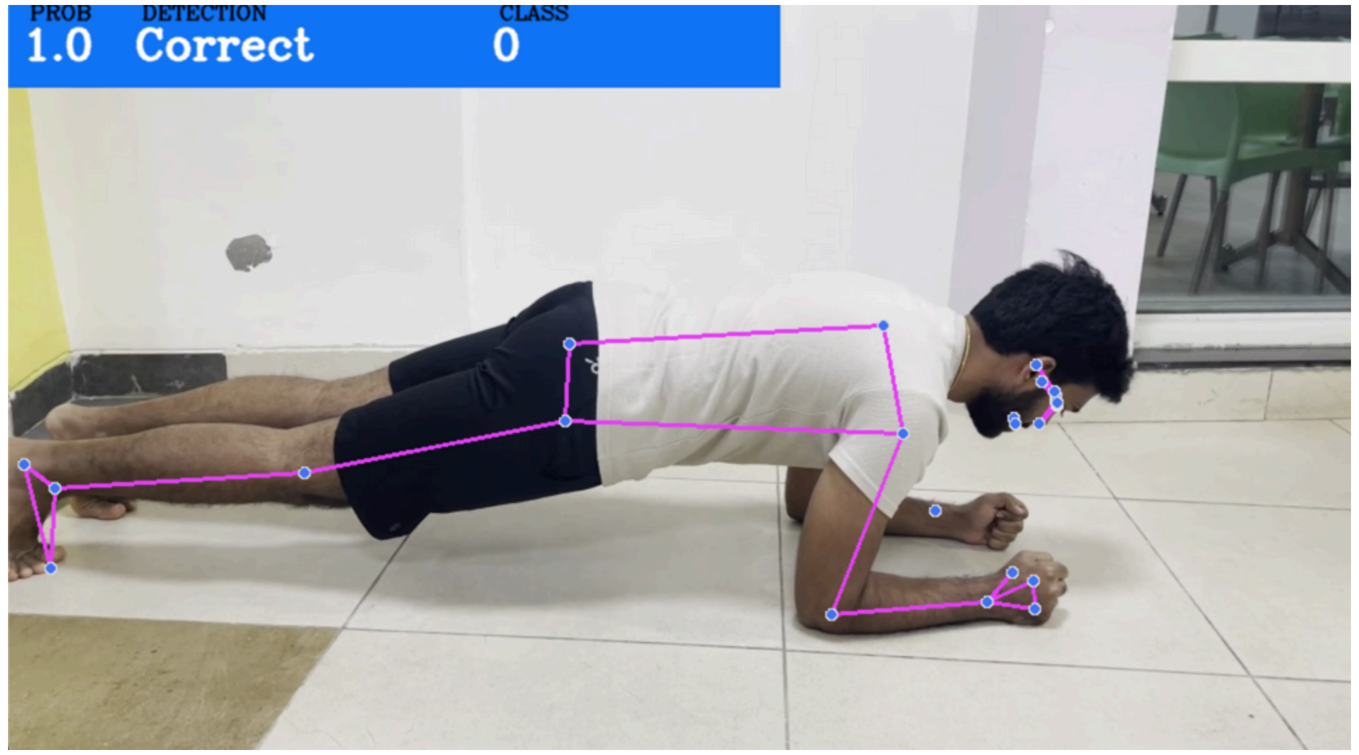
1. We could not define the rules for our model as the fitness domain we chose was large.
2. The trained model worked excellently on the trained patterns but failed to work on the test set.
3. As we went on further with our model, it was extremely hard for us to modify the rules based on the different scenario’s considered.
4. Overtime the chatbot did not improve its performance by learning from its past responses whichresulted in no improvement.
5. The chatbot could work efficiently only when the user queries were specified in a particular format.

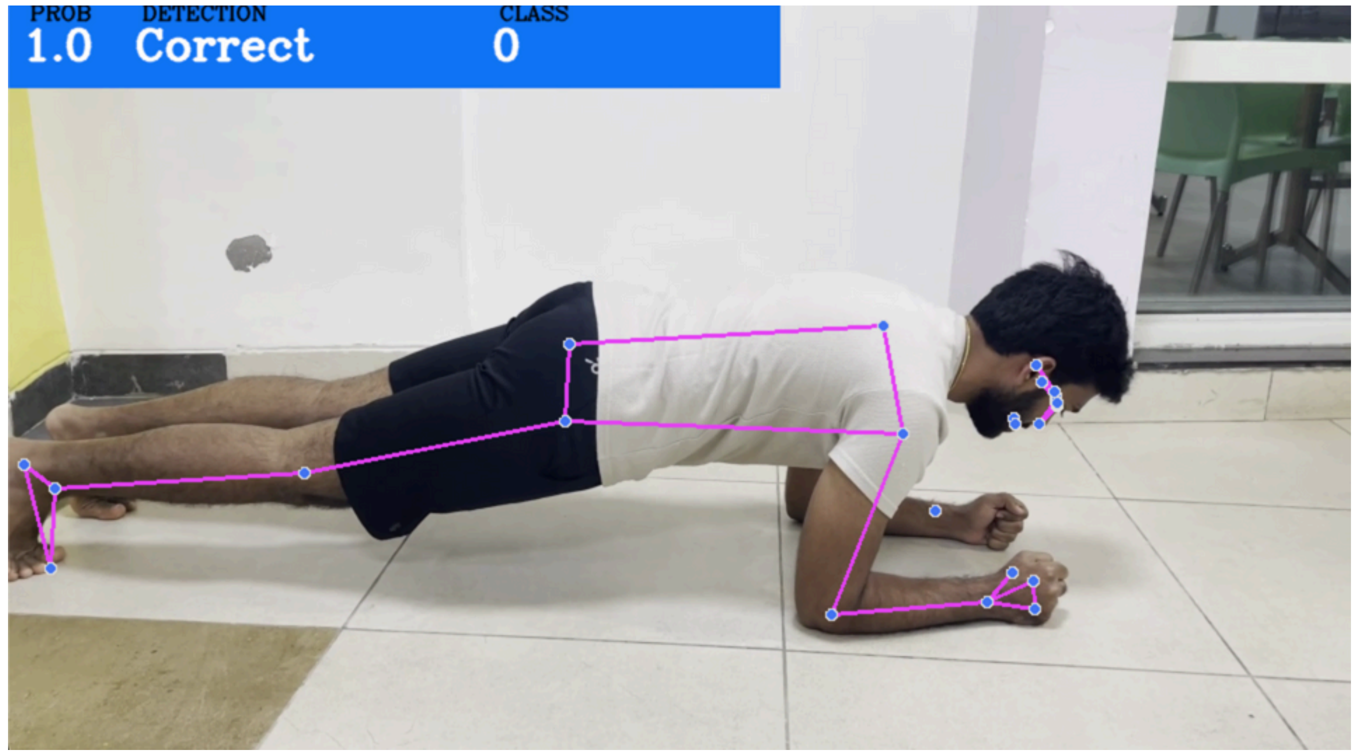
ANALYSIS AND CONCLUSION

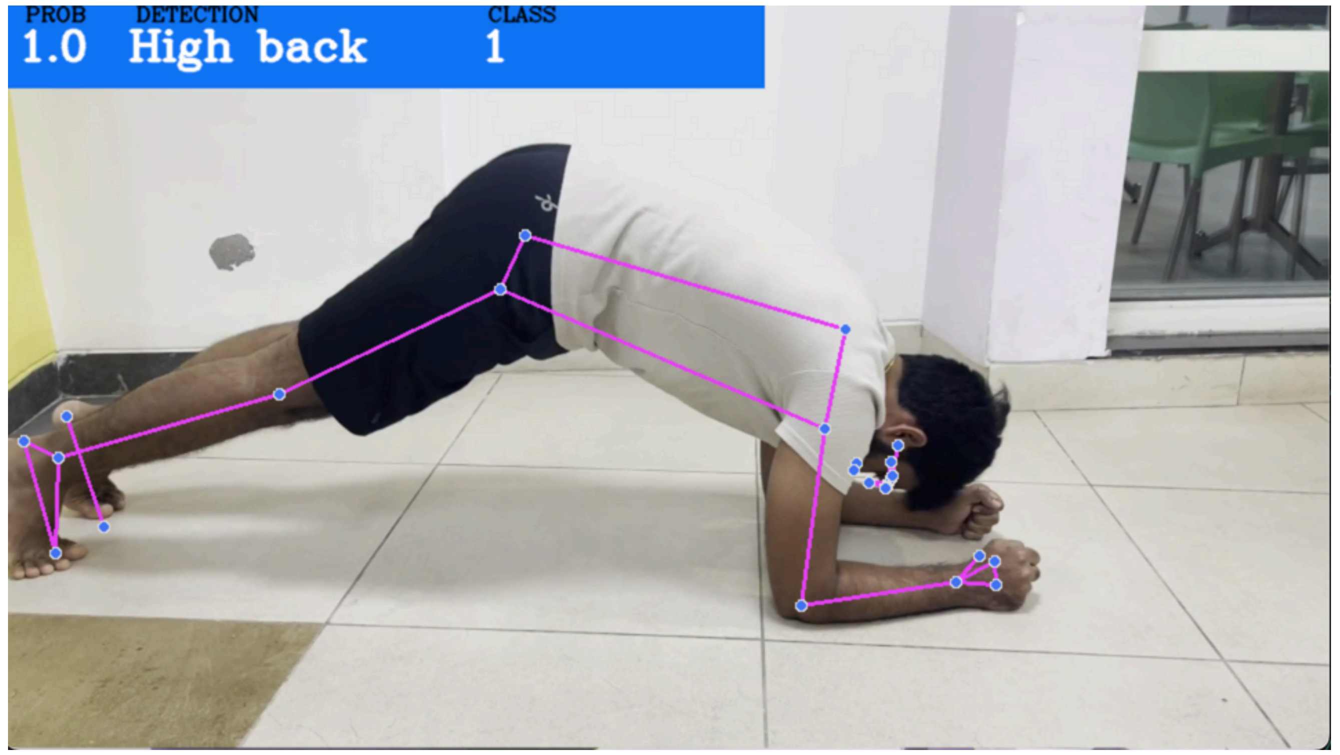
- The code implementation outlines a comprehensive pipeline for pretraining a BERT model, encompassing dataset preparation, model architecture, and training. The dataset is efficiently processed to create inputs for Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), ensuring the model learns both token-level semantics and sentence-level coherence.
- The model architecture is robust, featuring joint embeddings, a transformer encoder, and dedicated prediction layers for MLM and NSP. The training loop optimizes the model using a combination of NLLLoss for MLM and BCEWithLogitsLoss for NSP, leveraging the Adam optimizer for efficient parameter updates.
- This approach aligns well with BERT's original training methodology, making it suitable for general-purpose language understanding. The modular design of the dataset preparation, model, and training process allows for easy adaptation to different datasets and tasks. While the code is functional, enhancements like validation during training, gradient clipping, or mixed-precision training could further improve efficiency and performance.
- Overall, the code demonstrates a strong understanding of BERT's pretraining framework and provides a solid foundation for building and fine-tuning transformer-based language models.

POSTURE FEEDBACK









CHATBOT DEMO

```
/opt/anaconda3/envs/tf/lib/python3.9/site-
packages/transformers/tokenization_utils_base.py:1601: FutureWarning:
`clean_up_tokenization_spaces` was not set. It will be set to `True` by default.
This behavior will be depracted in transformers v4.45, and will be then set to
`False` by default. For more details check this issue:
https://github.com/huggingface/transformers/issues/31884
  warnings.warn(


Enter A sentence for NSP prediction (or type 'exit' to quit):

User: What is balanced diet?

NSP Result: A Balanced diet is the one that includes all nutrients that our body needs.

User: exit
```

TEAM COLLABORATION

- SE22UARI154,SE22UARI028--> In the DIP component we couldn’t find the dataset, they contributed in creating the required dataset. Data collection and preprocessing for the nlp component in our project.
- SE22UARI154,SE22UARI028,SE22UARI210--> dip component model building and training.
- SE22UARI172,SE22UARI176--> Deployment and final code for the dip component.
- SE22UARI210,SE22UARI172,SE22UARI176--> nlp BERT model building, training and testing.

- Problem Faced: We could build the dip and nlp components separately with through research and analysis but integrating both the components into a single working models requires multi-modal domain specific model which was hard for us to implement given the time, knowledge and resource constraints.

ACKNOWLEDGEMENTS

- https://github.com/NgoQuocBao1010/Exercise-Correction
- https://coaxsoft.com/blog/building-bert-with-pytorch-from-scratch
- https://huggingface.co/datasets/chibbss/fitness-chat-prompt-completion-dataset

THANKYOU!!